Aprendizaje e Inteligencia Computacional

Carlos A. Reyes-García, Eduardo F. Morales Manzanares,
Hugo J. Escalante Balderas, Alejandro A. Torres-García
Editores

ACADEMIA MEXICANA DE COMPUTACIÓN, A, C.

Aprendizaje e Inteligencia Computacional.
Autores: Carlos Alberto Reyes-García, Eduardo F. Morales Manzanares, Hugo Jair Escalante Balderas, Alejandro Antonio Torres-García

Segunda edición 2019.
Academia Mexicana de Computación, A. C.

ISBN: 978-607-97357-5-3.

Corrección de estilo: Luis Enrique Sucar-Succar. Diseño de portada: Mario Alberto Vélez Sánchez. Cuidado de la edición: Luis Enrique Sucar Succar.
Este libro se realizó con el apoyo del CONACyT, Proyecto No. I1200-28-2019.

Impreso en México.
Printed in Mexico.

Aprendizaje e Inteligencia Computacional.

Autores:
Carlos Alberto Reyes-García
Eduardo F. Morales Manzanares
Hugo Jair Escalante Balderas
Alejandro Antonio Torres-García

Agradecimientos

Los autores agradecen a la Academia Mexicana de Computación, en especial a los integrantes de la Sección Académica de Aprendizaje e Inteligencia Computacional que colaboraron con el desarrollo del libro. Agradecemos también a la Red Temática Conacyt en Inteligencia Computacional Aplicada (RedICA) por su apoyo y por facilitar el acceso a información útil para nuestro documento. Y en particular agradecemos a CONACyT el soporte otorgado a la AMEXCOMP y al desarrollo del presente libro por medio del Proyecto No. I1200-28-2019.

Prólogo

El presente libro ha sido desarrollado con el fin de proporcionar una compilación de las metodologías actuales más estudiadas y utilizadas en la búsqueda de alcanzar el tan perseguido sueño de crear las maquinas inteligentes. Con esta idea en mente, hemos conjuntado dos grandes áreas que aportan, de manera artificial, más de alguna capacidad considerada como parte de la inteligencia humana. Con el material incluido se cubren las áreas de Aprendizaje Computacional, tradicionalmente conocido como Aprendizaje Máquina, y el área de Inteligencia Computacional.

Para México es especialmente importante popularizar y extender el conocimiento y la utilización de las herramientas de inteligencia artificial, ya que es ampliamente conocida la tendencia mundial a volver *inteligentes* todo tipo de dispositivos, sistemas, vehículos, enseres, ciudades, carreteras y cosas en general que empleamos ahora. Es conocida también la intensa ofensiva intelectual que se combate entre las naciones más desarrolladas para lograr el predominio del ilimitado mercado para los productos inteligentes que se encuentra en constante expansión. Por lo que para nuestro país es una necesidad tener una reacción inmediata para integrarse a la comunidad de países que compitan en el desarrollo de estos demandados productos que integraran la inteligencia computacional como el verdadero valor agregado con el que se deberá competir. Con el presente libro, aun cuando sea un aporte mínimo, los autores pretenden contribuir con el impulso necesario para que se logre conseguir ese objetivo.

El material incluido en el libro ha sido escrito en un lenguaje accesible de tal modo que pueda ser asimilado por lectores no especializados, aunque si con un conocimiento básico de computación. El contenido ha sido distribuido de la siguiente manera: El capitulo 1 corresponde a información introductoria en la que se hace una breve reseña histórica del desarrollo de la inteligencia artificial y en particular de las áreas de aprendizaje e inteligencia computacio-

nal, además de incluir una descripción breve de las metodologías descritas en los siguientes capítulos. El capitulo 2 corresponde al tema de la Clasificación y en él se describe el problema, seguido de los algoritmos más utilizados para hacer clasificación. En el capitulo 3 se detallan los principales métodos de Regresión con énfasis en la formulación de la tarea y en los métodos de regresión lineal. La descripción y métodos mas importantes del Aprendizaje Bayesiano se encuentran en el capitulo 4 incluyendo *Naive Bayes*, Redes Bayesianas y Aprendizaje de Redes Bayesianas. El Capitulo 5 esta dedicado a la descripción del Aprendizaje por Refuerzo incluyendo métodos de solución y aplicaciones. La teoría básica de la Lógica Difusa se encuentra en el Capitulo 6 junto con una de sus principales aplicaciones como son los Sistemas de Control Difusos. En el capitulo 7 se detallan las Relaciones Difusas y una de sus principales aplicaciones como son las Redes Neurales Relacionales. Como complemento de la las metodologías difusas en el capitulo 8 se presentan los Mapas Cognitivos Difusos junto con los Sistemas Expertos Difusos. Para finalizar, en el capitulo 9 se hace referencia a Otras Metodologías de la Inteligencia Computacional como son las Redes Neuronales Artificiales y los Algoritmos Evolutivos, aunque la descripción es muy breve ya que en la serie de libros de AMEXCOMP se han escrito otros volúmenes cubriendo con mayor detalle tales metodologías.

Conscientes de que el material presentado no incluye exhaustivamente todo el que actualmente cubre las áreas del Aprendizaje y la Inteligencia Computacional y de menor manera a la Inteligencia Artificial completa, los autores esperan que el libro sea de interés y de utilidad para los lectores que se sientan atraídos en el aprender de estas áreas y a ellos está dedicado.

Felicitamos al Dr. Enrique Sucar Succar presidente de la Academia Mexicana de Computación por el impulso y apoyo brindado a los autores para la escritura de este libro que esperamos sea de gran utilidad para investigadores, académicos y estudiantes de todos los niveles.

Carlos A. Reyes García

Abreviaturas

MDL	Longitud de Descripción Mínima
NB	Naïve Bayes
BN	Redes Bayesianas
RL	Aprendizaje por Refuerzo
MDP	Procesos de decisión de Markov
SVM	Máquina de Soporte Vectorial
FIS	Sistema de Inferencia Difuso
FCM	Mapas Cognitivos Difusos
KNN	Algoritmo de k-vecinos más Cercanos
FRNN	Redes Neuronales Relacionales difusas
FCM	Fuzzy C-Means
AE	Algoritmos Evolutivos
BL	Aprendizaje Bayesiano
RNA	Red Neuronal Artificial
RLP	Regresión lineal localmente ponderada

Índice general

Capítulo 1

Introducción

C. Reyes-García, E. Morales, H. Escalante-Balderas, A. A. Torres-García

Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

En este libro hablamos de algunas de las técnicas más usadas en aprendizaje e inteligencia computacional. En particular, el libro consta de un capítulo introductorio, una primera sección dedicada a aprendizaje computacional, compuesta de seis capítulos, una segunda sección dedicada a inteligencia computacional, compuesta por cinco capítulos, y un capítulo final dedicado a dar una perspectiva de las áreas de aprendizaje e inteligencia computacional en México.

El capítulo introductorio hace un breve resumen de aprendizaje e inteligencia computacional, desde sus orígenes, hasta desarrollos más recientes. La primera parte está dedicada a tema de aprendizaje computacional. En esta parte se tocan los temas de segmentación (o *clustering*, clasificación, regresión, aprendizaje Bayesiano, aprendizaje por refuerzo y un capítulo final de otros enfoques de aprendizaje que por motivos de espacio sólo se mencionan brevemente.

La segunda parte está dedicada a inteligencia computacional. En particular, esta parte toca temas de lógica difusa y sistemas de control difusos, relaciones difusas y redes neuronales relacionales, mapas cognitivos difusos y sistemas expertos difusos, otras metodologías de la inteligencia computacional y sistemas híbridos.

Finalmente, se incluye un capítulo en donde se recapitula el aprendizaje e inteligencia computacional en México y cuáles son las perspectivas de estas

áreas a futuro.

1.1. Breve historia del área

Desde los inicios de la computación a finales de los 40s, y posteriormente con la creación del área de Inteligencia Artificial en 1956, los investigadores se han planteado la posibilidad de crear máquinas inteligentes. La inteligencia que un ente tiene o que puede desarrollar está relacionada con su capacidad de aprender. En 1950, Turing plantea, en lo que se conoce como la Prueba de Turing (Turing, 1950), que una máquina se podría considerar inteligente cuando lograra engañar a un interrogador al responder una serie de cuestionamientos. En ese mismo artículo establece que en lugar de crear una máquina que pasara esa prueba, era más sensato tener una máquina infantil (*Child Machine*) a la cual se le pudiera enseñar (Morales, 2013). Desde esos inicios, los científicos empezaron a crear sistemas básicos de aprendizaje computacional desde diferentes perspectivas, muchas de las cuales fueron sugeridas por Turing. Algunos intentaron emular los procesos dentro del cerebro, mediante redes neuronales artificiales. Posiblemente el primer artículo sobre aprendizaje computacional sea el de McCulloch y Pitts (McCulloch and Pitts, 1943) que muestra cómo emular compuertas lógicas proposicionales con modelos artificiales de redes neuronales. Un poco más tarde se empezaron a demostrar pruebas de convergencia en redes neuronales sencillas o perceptrones (e.g., Rosenblatt (Rosenblatt, 1958)). Esta línea de investigación ha seguido evolucionando hasta nuestros días en donde lo más actual son los sistemas que usan redes neuronales de "muchas" capas, llamado Deep Learning o aprendizaje profundo. Otros investigadores buscaron como aprender a jugar juegos sencillos mediante estrategias de búsqueda, aprendizaje de posiciones y ajuste de una función de recompensa. Una de las líneas de aprendizaje derivadas de esta otra línea de investigación es el aprendizaje por refuerzo (Sutton et al., 1998) en donde un agente computacional aprende cuál es la mejor acción a realizar en cada instante de tiempo para maximizar una función de recompensa. Recientemente se desarrolló un sistema que le ganó al campeón mundial de Go (Silver et al., 2016) usando una combinación de aprendizaje por refuerzo con aprendizaje profundo. También surgieron investigadores que desarrollaron algoritmos para aprender representaciones simbólicas, como Hunt (Hunt and Hoyland, 1963) y Feigenbaum (Feigenbaum, 1963). Uno de los algoritmos más utilizados en aprendizaje computacional, derivado de

ésta línea de investigación, es el de árboles de decisión y sus variantes, las cuales se han utilizado en una gran cantidad de aplicaciones reales. Por la misma época, otros investigadores estaban tratando de desarrollar sistemas inspirados en genética y evolución, e.g., Rechenberg (Rechenberg, 1971), Schwefel (Schwefel, 1975), Fogel (Fogel, 1966) y Holland (Holland, 1962). De aquí han surgido varios algoritmos que se engloban en lo que se conoce como estrategias evolutivas y algoritmos bio-inspirados utilizados principalmente para resolver problemas de optimización complejos en donde no es posible utilizar las técnicas matemáticas tradicionales. Además de aprender modelos u optimizar funciones, algunos investigadores trataron de formalizar el razonamiento humano en forma alterna a lo que se conocía en ese momento que era la lógica de predicados. Uno de estos investigadores fue Lofti Zadeh (Zadeh, 1965) que creó la lógica difusa en donde las expresiones, en esa lógica, toman no solo valores de verdad y falso, sino también valores intermedios. Esta línea de investigación, al igual que las anteriores, se han continuado desarrollando y refinando hasta nuestros días. Todas ellas forman parte del Aprendizaje e Inteligencia Computacional, y de las cuales hablaremos más ampliamente en las siguientes secciones.

1.2. Aprendizaje Computacional

El Aprendizaje Computacional (o *Machine Learning* en inglés) está dirigido al estudio y desarrollo de algoritmos con la capacidad de mejorar su desempeño con la experiencia. Otros nombres comúnmente utilizados para denotar el área son Aprendizaje Automático, Aprendizaje Maquinal y Aprendizaje de Máquina. El énfasis del Aprendizaje Computacional está en crear algoritmos que puedan automáticamente inducir modelos sin la intervención o asistencia humana. Dado que la capacidad de aprender es una característica distintiva de la inteligencia humana, no es de extrañarse que el Aprendizaje Computacional sea un área esencial dentro de la Inteligencia Artificial.

Para entender la utilidad práctica y relevancia del área podemos pensar en cómo se resuelven comúnmente muchos problemas. Generalmente las personas crean modelos o programas poder resolver problemas, sin embargo, algunos de ellos son difíciles de formalizar, no siempre se cuenta con expertos en el dominio o se tiene demasiada información. Por ejemplo, si queremos escribir un programa para reconocer personas, no es claro cómo programarlo, a qué expertos consultar o cómo limitar el número de fotos y videos de per-

sonas existentes. Lo que hacen los algoritmos de aprendizaje es que permiten generar automáticamente esos programas o modelos a partir de datos y con ellos resolver problemas. Esto claramente abre una gran cantidad de posibles aplicaciones que son difíciles de formalizar o programar por una persona y que requieren, escencialmente, para su desarrollo contar con datos.

El reciente éxito y publicidad que han recibido estos algoritmos se deben principalmente a tres factores: (i) disponibilidad de una gran cantidad de datos de todo tipo, (ii) máquinas más poderosas con altas capacidades de almacenamiento y mayores velocidades de procesamiento, y (iii) la facilidad de utilizar estas herramientas por las empresas para resolver problemas reales.

Algunos autores clasifican al área de aprendizaje computacional en términos de los modelos que inducen en tres categorias: (i) modelos geométricos (ii) modelos probabilístas y (iii) modelos lógicos. Otra posible clasificación se puede dar en términos de las tareas que abordan. Una caracterización burda, es en términos de aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Una clasificación un poco mñas detallada y que es la que vamos a seguir en este capítulo, es: (i) Sistemas que crean descripciones, resúmenes, identifican prototipos o casos anómalos a partir de datos, (ii) sistemas que segmentan/agrupan datos automáticamente, (iii) sistemas que crean automáticamente modelos para predecir salidas, ya sea para tareas de clasificación (salidas discretas) o de regresión (salidas continuas), (iv) sistemas que encuentran automáticamente dependencias entre variables, ya sean éstas probabilistas, funcionales, asociativas o causales, y (v) sistemas que aprenden a crear planes o estrategias de control para resolver problemas de decisión secuenciales. Vamos a hablar muy brevemente de cada una de estas.

1.2.1. Resumenes

Con el incremento acelerado en la generación de datos, es importante poder crear automáticamente resumenes que resalten la información más relevante. Los estadísticos entendieron rápidamente la utilidad de esto, y generaron medidas como la media o la moda y medidas de dispersión como la desviación estándar para poderse dar una idea rápida de la naturaleza de los datos. En aprendizaje computacional algunos investigadores han tratado de encontrar los datos prototípicos dentro de una gran cantidad de datos para poder hacer inferencias con estos, facilitanto el análisis y reduciendo la velocidad de procesamiento. Las personas que trabajan con análisis de texto también han

creado sistemas que automáticamente generan resumenes de textos para su fácil lectura y comprensión. También se pueden usar técnicas para reducir datos identificando unicamente los datos atípicos, los cuales son muchas veces los datos de mayor interes.

1.2.2. Segmentación

Los sistemas que segmentan automáticamente datos se conocen como algoritmos de agrupamiento o clustering. También es común referirse a ellos como algoritmos de aprendizaje no supervisado. La idea de estos algoritmos es poder agrupar automáticamente datos en conjuntos de tal forma que los elementos en cada grupo se parezcan, bajo algún criterio, entre si y no se parescan o se parescan poco, bajo ese mismo criterio, con los elementos de otros grupos. Por ejemplo, a partir de datos de clientes, estos se pueden agrupar bajo ciertos criterios para diseñar campañas publicitarias diferenciadas. Los grupos pueden ser disjuntos o tener traslapes, pueden estar organizados en jerarquías, pueden ser exhaustivos, la pertenencia a los grupos puede ser probabilísta, etc. La medida de similaridad se define generalmente por la proximidad que existe entre los elementos en un espacio multidimensional y existe una gran cantidad de medidas para agrupar datos. Si los datos son exclusivamente numéricos se pueden usar medidas de distancia, como la Euclideana. Si los datos no solo están descritos por números, se usan otras medidas de similaridad que dejan de cumplir con las propiedades de las métricas. También se les puede dar más peso a algún tipo variable o atributo que a otro(s), para que sus valores tengan más influencia en la creación de los grupos. Por otro lado, también existen diferentes estrategias para formar los grupos, algunas van juntando gradualmente elementos, otras parte de un grupo global al cual van separando en subgrupos, otros parten de particiones formadas originalmente de forma aleatoria las cuales las van modifican hasta cumplir con un cierto criterio, etc.

Uno de los algoritmos más conocidos de clustering es el de k-means. El algoritmo es relativamente simple y parte del número de particiones (k) que se quieren formar. El algoritmo selecciona aleatoriamente k elementos y calcula la similaridad de todos los datos hacia esos k elementos. Cada dato se asocia al elemento k más cercano con lo que se forman k grupos. Con los datos de cada grupo se obtiene un centroide o "promedio". Con los centroides resultantes se vuelve a calcular la similaridad de todos los elementos hacia esos centroides con lo que se forman, posiblemente, nuevos grupos. Este proceso

se repite hasta que los centroides no cambian. Existen muchas extensiones a este algoritmo y variantes de algoritmos que realizan agrupamiento.

Los datos se pueden agrupar desde diferentes perspectivas y de diferentes formas. Se han desarrollado medidas de calidad para los grupos que se forman tomando en cuenta la densidad de elementos, qué tan compactos son los grupos, etc. Algunos retos del área son cómo lidear con diferentes representaciones en los datos, cómo crear grupos con formas arbitrarias, como incluir restricciones y cómo hacerlos interpretables, entre muchos otros.

1.2.3. Predicción

Existen algoritmos que crean automáticamente modelos para predecir salidas, ya sea para tareas de clasificación (salidas discretas) o de regresión (salidas continuas). La tarea de predicción es posiblemente la que más atención a recibido en la literatura por su gran cantidad de posibles aplicaciones. A estos algoritmos también se les conoce como aprendizaje supervisado. La información de entrada dada a estos algoritmos son ejemplos, descritos generalmente con atributos o variables, asociados con una etiqueta. Cuando la etiqueta es nominal se conocen como algoritmos de clasificación, cuando es numérica se conoce como regresión o estimación.

Clasificación

La meta de los algoritmos de clasificación es inducir automáticamente, a partir de los datos, un modelo que dada la descripción de un ejemplo produsca una etiqueta de salida. Por ejemplo, crear un modelo que dada una descripción de atributos de un paciente prediga una enfermedad o dados los valores de variables de una línea de producción detecte una falla. En la tarea de clasificación existe una gran cantidad de algoritmos como árboles de decisión, máquinas de soporte vectorial y aprendizaje basado en instancias, entre otros. En este capítulo solo vamos a describir brevemente estos tres enfoques.

En un árbol de decisión cada nodo representa un atributo y sus ramas los posibles valores que ese atributo pueden tomar. En caso de atributos contínuos, estos se dividen automáticamente en dos rangos que son mayores o menores que cierto valor. Las hojas de los árboles son los valores de las clases que queremos predecir. El algoritmo selecciona de manera heurística el atributo que mejor particiona los datos en términos de las clases. Una vez que selecciona un atributo, el algoritmo se vuelve a aplicar a los datos

de cada una de sus ramas hasta que: (i) se queda con ejemplos de una sola clase y etiqueta la hoja con esa clase, (ii) se queda sin ejemplos y etiqueta la hoja con la clase mayoritaria de su padre, o (iii) se queda sin atributos y etiqueta la hoja con la clase mayoritaria de sus datos. Para clasificar un nuevo ejemplo, se recorre el árbol de acuerdo a los valores de los atributos del ejemplo y al llegar a una hoja se regresa la clase asociada a esa hoja. Existen varios medidas de selección de atributos, pero la más conocida se basa en ganancia de información la cual selecciona el atributo que reduce más la entropía o dispersión de las clases en los datos. Esto es, el atributo en donde las datos contenidos en sus ramas tienen más claramente definidas alguna de las clases.

Otro algoritmo muy usado en clasificación son las máquinas de soporte vectorial o de vectores de soporte. Una forma de construir un clasificador entre dos clases es tratar de encontrar un plano, o hiperplano en caso de más dimensiones, que mejor divida a las clases. En las máquinas de soporte vectorial el encontrar ese hiperplano se plantea como un problema de optimización en donde se busca el plano que divida a las clases y que esté más alejado de los ejemplos de las clases contrarias. Como en muchos problemas las clases no son linealmente separables (no existe un hiperplano que las divida perfectamente) se recurren a dos estrategias, una de ellas es usar variables de holgura, lo cual permite tener "excepciones" de algunos ejemplos, y la otra, y mucho más conocida e importante, es lo que se conoce como el "kernel trick", que consiste en crear una representación de más dimensiones que la original con la esperanza de que en ese nuevo espacio se pueda encontrar un mejor hiperplano.

Finalmente, están los algoritmos de aprendizaje basados en instancias, los cuales, en lugar de inducir automáticamente un modelo que sirva para predecir una clase, almacenan todos los datos y para clasificar un nuevo ejemplo, buscan cual o cuales son los datos más parecidos al ejemplo y los utilizan para predecir un valor. El algotitmo más conocido es el de k vecinos más cercanos. La idea es obtener del conjunto de datos los k elementos que sean más parecidos, bajo cierta medida, al ejemplo que se quiere clasificar, analizar las clases de esos k vecinos y regresar la clase mayoritaria. Al igual que en los algoritmos de clustering, existen diferentes medidas de similaridad que se pueden utilizar para encontrar los vecinos más cercanos. Muy relacionado a estos algoritmos está el razonamiento basado en casos, en donde cada ejemplo, llamado caso, se representa por la descripción de un problema, la solución empleada y los resultados obtenidos. Cuando se quiere resolver un

nuevo problema, se consultan las descripciones de los problemas en la base de casos, se obtienen los más parecidos y se utilizan sus soluciones, con posibles adecuaciones, para resolver el problema en cuestión.

La gran mayoría de los algoritmos de clasificación utilizan una representación de los ejemplos basada en pares artibuto-valor produciendo en muchos casos modelos proposicionales. Existen otros algoritmos que trabajan con representaciones más expresivas utilizando grafos o lógica de primer orden. Una de ellas es la programación lógica inductiva la cual recibe de entrada datos e información relacionales y produce modelos expresados en términos de relaciones. Esto permite inducir modelos que contengan funciones, variables y definiciones recursivas los cuales son más expresivos que los modelos generados por los algoritmos anteriores.

Algunos de los retos de los sistemas de clasificación son el ruido (valores erroneos y faltantes) que puede existir en los datos, una gran cantidad de datos, pocos datos pero muchos atributos, muchas posibles clases, entre otros.

Regresión

La meta de los algoritmos de estimación o regresión es inducir automáticamente, a partir de los datos, un modelo que dada la descripción de un ejemplo produzca una valor contínuo de salida. Por ejemplo, predecir el valor de una temperatura de salida dados los valores de las variables de un proceso industrial. Algunos de los algoritmos más usados son las redes neuronales, regresión lineal, regresión localmente pesada, processos gaussianos, etc.

Posiblemente el algotimo más conocido de regresión sea el de las redes neuronales. Éstas son descritas con más detalle en otro capítulo y aquí solo vamos a mencionar las ideas generales de la red más conocida que es la de feed-forward con retro-propagación. Esta red neuronal es un grafo dirigido organizado por capas, en donde todos los nodos de una capa están conectados hacia todos los nodos de la siguiente capa y reciben información de todos los nodos de la capa anterior. Cada conexión está asociada con un peso. A cada nodo le llega la suma de todas las salidas de los nodos de la capa anterior multiplicadas por sus respectivos pesos y a ese valor generalmente se le aplica una función contínua no lineal diferenciable que produce la salida de ese nodo. Esa salida, junto con las salidas de todos los nodos de ese capa se propagan de la misma forma hacia la siguiente capa de nodos, hasta producir una o más salidas finales. Los pesos de las redes se inicializan aleatoriamente y el aprendizaje consiste en ajustar los pesos de la red para que dados los

valores de ejemplos conocidos se produzcan las salidas deseadas. Este ajuste se hace gradualmente alterando los pesos en la dirección que produce el máximo descenso en la superficie del error. Para esto se obtiene el negativo de la derivada o gradiente del error con respecto a los pesos y se suma, en forma pesada, al peso actual. El error de salida se distribuye entre todos los pesos de las capas anteriores. Este proceso se realiza con todos los datos de entrenamiento varias veces hasta que los pesos producen resultados aceptables o cambian poco los pesos de la red.

Las redes nueronales fueron populares en los inicios de la Inteligencia Artificial, recuperaron popularidad cuando se difundió el algoritmo de entrenamiento que acabamos de describir y recientemente volvieron a recuperar su interés cuando se logró entrenar redes de "muchas" capas (entre 5 y 10). Aunque algunas de las ideas originales de entrenamiento de muchas capas se plantearon hace tiempo, la disponibilidad de grandes cantidades de datos y las velocidades de procesamiento de las máquinas actuales, fueron las que permitieron realizarlo. Estas redes se han utilizado con éxito en análisis de imágenes, procesamiento de lenguaje natural y en juegos.

Algunos de los retos del área son, como en todas las demás, la cantidad de ruido que exista en los datos, si se tienen datos numéricos y nominales, la posibilidad de realizar un sobre-ajuste en los modelos y el tiempo de entrenamiento de las redes.

1.2.4. Análisis de Dependencias

Algunos algoritmos encuentran automáticamente dependencias entre variables, ya sean éstas probabilístas, funcionales, asociativas o causales. Por ejemplo, con estos modelos se puede determinar si el valor de una variable depende de los valores de otras variables. Por ejemplo, cual es la probabilidad de que llueva dado que la presión atmosférica y la temperatura tienen ciertos valores, o qué tan frecuente ocurre y con que confianza, que las personas que compran leche y pan, también compren mantequilla y mermelada.

Una de las técnicas más usadas para capturar dependencias probabilístas son los modelos gráficos probabilístas y, en particular, las redes bayesinas, las cuales son descritas más ampliamente en otro capítulo. Las redes bayesianas describen una distribución de probabilidad conjunta de varias variabes y se representan como un grafo acíclico dirigido en el que cada nodo representa una variable aleatoria y cada arco una dependencia probabilísta. Las redes bayesianas permiten realizar razonamiento probabilísta apoyándose en la re-

gla de Bayes. Esto es, pueden evaluar cuáles son los valores más probables de un conjunto de variables, dados los valores de otras variables.

Los algotitmos de aprendizaje de redes bayesianas inducen tanto la estructura de la red como las dependencias entre las variables expresadas como tablas de probabilidad condicional de cada variable dados sus padres. El aprendizaje paramétrico o de los valores de las tablas de probabilidad condicional, generalmente se basa en obtener las frecuencias de los valores de las variables relacionadas. Las técnicas de aprendizaje estructural se basan en métodos usando medidas de ajuste y estrategias de búsqueda o en métodos basados en pruebas de independencia. Uno de los métodos más usadas, que forma la base de otros algoritmos de aprendizaje de redes bayesianas, es el de Chow y Liu. En este algoritmo se obtiene la información mutua entre todos los pares de variables, se ordenan estas medidas de mayor a menor, la de mayor información mutua representa un grafo inicial (con dos nodos) y se agregan las siguientes ramas mientras no se formen ciclos y hasta que se hayan incluido todas la variables. Determinar la dirección de las ramas y contruir grafos más sofisticados (poliárboles o redes multiconectadas) son algunas de las extensiones a este algoritmo. Dentro de los retos del área podemos mencionar el razonamiento con variables contínuas y mixtas y dominios en donde exista una gran cantidad de dependencias, lo cual complica el algoritmo de inferencia.

Otra forma de obtener relaciones entre variables es por medio de las reglas de asociación. Estos algoritmos buscan grupos de variables y valores que se repiten frecuentemente en los datos. Con estos grupos de variables y valores repetidos, construyen reglas que tienen tanto en el antecedente como en el consecuente subgrupos de pares atributo-valor. El algoritmo más conocido del área es el de Apriori. En este algoritmo, primero se buscan todos los pares atributo valor que cumplan con un mínimo de repeticiones o soporte en los datos. Con estos se buscan todas las parejas de pares atributo-valor con el soporte deseado. Esto continúa con tercias y así sucesivamente, hasta que no exista ningún subconjunto de pares atributo-valor más grande con el soporte requerido. Con cada subconjunto encontrado se crean todas las posibles reglas que cumplan con un nivel de confianza (soporte del subconjunto entre el soporte del antecedente de la regla) formadas por las combinaciones de antecedentes y consecuentes de los elementos de los subconjuntos. Esta área está muy relacionada con los algoritmos de patrones frecuentes y los de descubrimiento de subgrupos, los cuales se basan en buscar subconjuntos de pares atributo-valor frecuentes. Algunos de los retos de las reglas de asociación es

cómo encontrar todos estos conjuntos frecuentes de manera eficiente.

Tanto las redes bayesianas como las reglas de asociación se pueden modificar para resolver problemas de clasificación, en donde el interés está en predecir el valor de una sola de las variables (la clase).

Aunque se han desarrollado también algoritmos para encontrar dependencias funcionales y causales, estos no han gozado de la popularidad y desarrollo de las redes bayesianas y de las reglas de asociación.

1.2.5. Planeación

Finalmente, existen algoritmos que aprenden a crear planes o estrategias de control para resolver problemas de decisión secuencial. Por ejemplo, decidir una serie de acciones para cumplir una meta, como por ejemplo ganar un juego, o controlar algún dispositivo, como por ejemplo controlar un robot.

Posiblemente el más conocido en años recientes sea el aprendizaje por refuerzo o a veces también conocido aprendizaje por reforzamiento. En aprendizaje por refuerzo un agente aprende por prueba y error, cuál es la mejor acción a realizar en cada momento para maximizar el valor esperado de su recompensa total. Para aprender qué acción a realizar en cada estado (aprender una política), el agente selecciona en cada estado una de sus posibles acciones, con la cual se mueve, posiblemente, a otro estado y recibe una recompensa. La selección de acciones, que es inicialmente aleatoria, se va ajustando gradualmente tomando en cuenta la recompensa recibida y lo que se estima obtener de recompensa acumulada en otros estados. Uno de los algoritmos más conocidos es el de Q-learning, el cual ajusta gradualmente el valor de recompensa esperado de cada par estado-acción. El valor esperado se ajusta con la diferencia entre ese valor y la suma de la recompensa recibida con el valor esperado del siguiente estado.

Algunos de los retos de esta área son cómo lidear con espacios de estados y acciones contínuos, cómo aprender con muchas variables, qué hacer en ambientes no estacionarios y cómo reducir la velocidad de aprendizaje.

1.2.6. Otros Enfoques

Existen muchos otros algoritmos y estrategias de aprendizaje que se han desarrollado. Algunos que han perdido popularidad son basados en abducción, llamados aprendizaje basado en explicaciones, aprendizaje de macro-operaciones, usados en planeación, algoritmo de descubrimiento, y algoritmos

basados en analogías, entre otros.

Algunas de las variantes más utilizadas actualmente son los ensambles, en los cuales se inducen varios modelos a la vez y se combinan sus resultados para resolver alguna tarea.

También se han desarrollado algoritmos de aprendizaje por transferencia, en donde se utiliza información de una tarea para resolver otra de manera más eficiente y/o más efectiva.

Dentro de los algoritmos de clasificación se han desarrollado algoritmos de aprendizaje semi-supervisado, en donde se combinan datos etiquetados y no etiquetados (esto es, con y sin una clase asociada) para tratar de construir un mejor clasificador. También se han desarrollado algoritmos para tratar de predecir un conjunto de clases al mismo tiempo o algritmos multi-etiqueta.

Finalmente, existen algoritmos que buscan que un agente nunca deje de aprender, en lo que se conoce como life-long learning. Existe mucha expectativa del área y algunos investigadores creen que, dada la velocidad con la cual se está desarrollando, pronto se va a lograr crear un inteligencia artificial superior a la de los humanos.

Referencias

Feigenbaum, A. (1963). The simulation of verbal behavior. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, pages 297–309. Mc Graw Hill.

Fogel, L. J. (1966). *Artificial Intelligence through Simulated Evolution.* Wiley.

Holland, J. (1962). Outline for a logical theory of adaptive systems. *JACM*, 9:297–314.

Hunt, E. B. and Hoyland, C. I. (1963). Programming a model of human concept formation. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, pages 310–325. Mc Graw Hill.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent un nervous activity. *Bulletin of Mathetmatical Biophysics*, 5:115–133.

Morales, E. (2013). Educando al 'niño computacional' de Turing. In *Ciencias*, pages 32–39.

Rechenberg, I. (1971). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for the information storage and organization in the brain. *Phsycological Review*, 565(6):433–460.

Schwefel, H. P. (1975). *Evolutionsstrategie und numerische Optimierung*. PhD thesis, TU Berlin.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Sutton, R. S., Barto, A. G., Bach, F., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59:433–460.

Zadeh, L. (1965). Fuzzy sets. *Information Sciences*, 8:338–353.

Capítulo 2

Clasificación

E. Morales, H. Escalante-Balderas
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

Clasificar es una tarea que los seres humanos pueden realizar casi sin pensar. Clasificamos automáticamente todo lo que vemos: desde plantas y animales, hasta documentos y desechos. Más allá, los humanos tenemos excelentes habilidades para reconocer instancias de objetos, por ejemplo: reconocer una taza, autos, personas etc. Poder clasificar y/o reconocer no es tan sencillo para las máquinas, sin embargo, existen métodos que permiten generar modelos para esta tarea y que son muy efectivos. En este capítulo se describe la tarea, y se provee una descripción general de los principales algoritmos para clasificación así como medidas de evaluación.

2.1. Formulación de la tarea

El aprendizaje supervisado se refiere al área de aprendizaje computacional que, partiendo de ejemplos provistos por un usuario, permite diseñar modelos capaces de emular la tarea ejemplificada (Mitchell, 1997). Clasificación y regresión (ver Capítulo 3) son dos tareas de aprendizaje supervisado. En ambas tareas se busca encontrar un modelo que permita mapear entradas a salidas. En clasificación las entradas son los objetos de interés a la tarea (e.g., desechos, etc.) mientras que las salidas corresponden a las clases o categorias de interés (e.g., plástico, metal, vidrio, etc.).

Un método de aprendizaje supervisado requiere de ejemplos *etiquetados* de la tarea a resolver. Considerese por ejemplo que se desea desarrollar un

método de clasificación que permita filtrar correo no deseado (*spam*). Esta tarea se puede ver como un problema de clasifiación, donde se buscan clasificar mensajes de correo en las clases: genuino y no deseado. Para construir un modelo de clasificación, se requerirían entonces ejemplos de correo que pertenezcan a ambas clases. Un algoritmo de aprendizaje procesaría los datos y generaría un modelo capaz de hacer predicciones para nuevos mensajes. La entrada al modelo sería un mensaje desconocido y la predicción sería genuino o bien no deseado, dependiende de la recomendación del modelo.

Antes de procesar dichos mensajes, sería necesario realizar una codificación de los mensajes de manera que los algoritmos puedan procesar la información. Generalmente, los objetos de interés son representados por vectores númericos de tamaño fijo. En el caso de mensajes (y documentos en general), éstos se suelen representar por la llamada *bolsa de palabras*: se consideran vectores de tamaño d, igual al total de palabras existentes en el vocabulario; así, un mensaje está asociado a un vector de tamaño d cuyos elementos tienen el valor cero, excepto en los elementos del vector correspondientes a palabras que aparecen en el mensaje (dichos elementos tendrían un valor de 1, por ejemplo).

Más formalmente, sea $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ el conjunto de ejemplos provisto por el usuario para resolver una tarea de clasificación. Donde cada par $(\mathbf{x}_i, y_i)$ denota un ejemplo etiquetado (e.g., un mensaje de correo y su etiqueta correspondiente), y donde $\mathbf{x}_i \in \mathbb{R}^d$ es la codificación numérica del $i-$ésimo objeto y y_i es la clase del objeto, con $y_i \in \mathcal{C}$ y $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ es el conjunto de K clases asociadas al problema. Cuando $K = 2$ se dice que se trata de un problema de clasificación binaria (dos clases), y en dicho caso[1] $y_i \in \mathcal{C} = \{-1, +1\}$. Es decir, hay una clase negativa $(+1)$ y una clase positiva $(+1)$. Nótese que la codificación es dependiente de la tarea, para problemas que involucran imágenes podrían usarse los valores de intensidad de los píxeles, para problemas de procesamiento de voz/audio se pueden extraer coeficientes a partir del análisis de la señal, y así con otras tareas.

El conjunto de datos $\mathcal{D}$ usado para generar el modelo poviene de una distribución de probabilidad $P(\mathbf{X}, \mathbf{y})$ que es desconocida[2]. El problema de

[1] Nótese que suelen usarse otros indicadores para estas clases, sin embargo, la codificación mencionada permite una formulación matemática elegante y de la que se suele sacar provecho para obtener implementaciones eficientes de algoritmos.

[2] Si conocieramos la distribución, no habría necesidad de usar aprendizaje computacional para aproximar la función de clasificación, pues podríamos conocer la distribución $P(\mathbf{y}|\mathbf{X})$, que es justo la distribución que un clasificador intenta aproximar.

clasificación consiste en encontrar una aproximación f^* de la función $f : \mathbb{R}^d \rightarrow \mathcal{C}$, tal que $\hat{y}_j = f^*(\mathbf{x}_j)$ es igual (o lo más parecido posible) a y_j, $\forall(\mathbf{x}_j, y_j) \in \mathcal{T}$ asociado a la misma distribución $P(\mathbf{X}, \mathbf{y})$. Esto implica, que la función $f^*(\mathbf{x}_j)$ (*el clasificador*) debería clasificar correctamente cualquier ejemplo (conocido o no) que esté asociado al mismo problema (en general no se conoce la etiqueta verdadera de los ejemplos en $\mathcal{T}$).

En esta formulación, $\mathcal{T}$ representa el conjunto de todos los posibles objetos que pueden ser tomados de la misma distribución (e.g., cualquier mensaje de correo que pudiese ser generado), y para los cuales no se conoce la etiqueta. De igual forma f representa el clasificador perfecto, es decir aquel que se obtendría si se conociera la distribución $P(\mathbf{X}, \mathbf{y})$. Así, el clasificador $f^*(\mathbf{x}_j)$ provee una aproximación a $f(\mathbf{x}_j)$ que se construye usando una muestra $\mathcal{D}$ de datos etiquetados y asociados al problema de interés. Existen muchas formas de aproximar f, en la siguiente sección se revisan los principales enfoques, para una revisión más extensa se recomienda consultar las siguientes referencias (Hastie et al., 2001; Mitchell, 1997).

2.2. Principales algoritmos para clasificación

Una vez formulado el problema, surge la pregunta de cómo diseñar y/o generar la función de clasificación f^*. En esta sección se describen los principales métodos de clasificación, los cuales hacen diferentes suposiciones sobre la forma de f y/o el criterio que f^* debe cumplir. En cada método, se presenta la idea intuitiva detrás del clasificador y una descripción formal breve. De igual forma, se proveen referencias donde es posible consultar información adicional.

2.2.1. *k*-vecinos más cercanos

Uno de los métodos de clasificación más sencillos e intuitivos es el de $k-$ vecinos más cercanos (k-NN). Se basa en la comparación del objeto que se desea clasificar con todos los objetos almacenados en $\mathcal{D}$. Intuitivamente, la etiqueta de un objeto desconocido $\mathbf{x}_j$ se asigna analizando las etiquetas de los k ejemplos conocidos (i.e., en $\mathcal{D}$) que más se *parecen* a $\mathbf{x}_j$. En el escenario de clasificar spam, un nuevo mensaje de correo sería comparado con todos los almancenados en $\mathcal{D}$. De esta comparación se seleccionarian los $k-$ más parecidos, y la clase del mensaje sería la que más se repita dentro de estos

$k-$ mensajes vecinos.

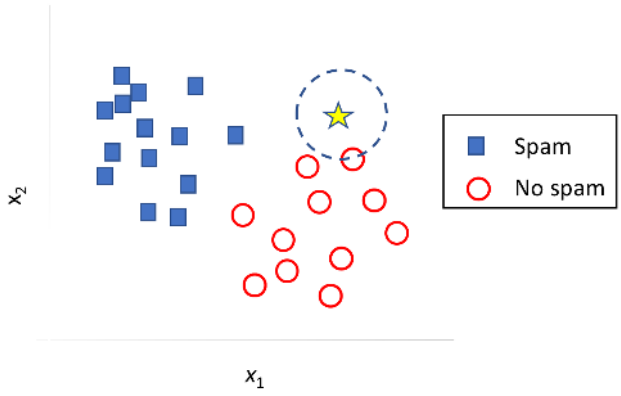


Figura 2.1 Clasificación mediante $1 - NN$.

La Figura 2.1 ilustra graficamente el proceso de clasificación de $k-$NN con $k = 1$ (i.e., $1-$NN). Se busca clasificar el ejemplo denotado con la estrella, de acuerdo a este ejemplo, el vecino más cercano al objeto a clasificar es de la clase No spam, por lo que esa sería la etiqueta del mensaje.

Formalmente, $k-$ NN clasifica un nuevo ejemplo $\mathbf{x}_j$ de acuerdo a la siguiente función de decisión:

$$f(\mathbf{x}_j) = \arg \max_{C_k \in \mathcal{C}} \sum_{i=1}^{k} \mathbf{1}_{v=f(\mathbf{x}_i^N)} \tag{2.1}$$

donde: $\mathbf{x}_1^N, \ldots, \mathbf{x}_k^N$ son los k vecinos más cercanos a $\mathbf{x}_j$ y $\mathbf{1}_{a=b} = 1$ si $a = b$ y 0 en caso contrario.

Para determinar cuáles son los vecinos más cercanos $\mathbf{x}_1^N, \ldots, \mathbf{x}_k^N$ a un ejemplo $\mathbf{x}_j$ usualmente se calcula la distancia entre el objeto a clasificar y todos los elementos de $\mathcal{D}$. Los k objetos con la menor distancia son los vecinos más cercanos. La forma particular de calcular la distancia depende del problema y aplicación, la forma más utilizada de calcular la distancia entre dos objetos es la Euclideana:

$$d(\mathbf{x}_j, \mathbf{x}_i) = \sqrt{(\mathbf{x}_j - \mathbf{x}_i)^2} = \sqrt{(\sum_{n=1}^{d} (x_{n,j} - x_{n,i})^2} \tag{2.2}$$

Es importante mencionar que el valor de k en $k-$ NN es un parámetro que debe ser ajustado por el usuario. Diferentes valores de k resultan en

aproximaciones diferentes, por ejemplo, valores altos de k (e.g., $k > 7$) son recomendables para problemas que son linealmente separables (veáse la Figura 2.1), mientras que valores pequeños son preferibles si la superficie de clasificación es no lineal. Por lo anterior es importante determinar el mejor valor de k para una tarea determinada. Generalmente se trata de un número impar, esto con la finalidad de evitar *empates* en la recomendación.

Una de las principales desventajas de este tipo de métodos es el hecho de que cada vez que se requiere clasificar un nuevo objeto, éste se tiene que comparar con todos los objetos existentes en la base de datos. De ahí que a este tipo de métodos se les conozca como algoritmos perezosos (*Lazy learning*). Lo anterior lo hace impráctico, aunque se ha probado que su desempeño es más que satisfactorio en numerosos dominios. Cabe mencionar que existen muchas variantes de $k-$ NN, algunas ponderando la contribución de cada vecino de acuerdo a su cercanía, otras aplicando una reducción a $\mathcal{D}$ antes de clasificar, entre otras. Para obtener más información respecto a este método se recomienda el Capítulo 8 de (Mitchell, 1997).

2.2.2. Máquina de soporte vectorial

Otro clasificador emblemático es la máquina de soporte vectorial (o SVM). Se trata de un clasificador que intenta aprender un híper plano que separe los ejemplos que pertenecen a dos[3] clases diferentes. Intuitivamente, se ubica a los ejemplos etiquetados en un plano $d-$dimensional, y se busca encontrar una linea en ese espacio (un híper plano en $d-1$ dimensiones) que separe los ejemplos que pertenecen a clases diferentes.

La Figura 2.2 ilustra el objetivo de este método (y en general, de cualquier clasificador lineal) cuando $d = 2$ y el híper plano es una línea. El híper plano está dado por $\mathbf{wx} + b = 0$, donde $\mathbf{w} \in \mathbb{R}^d$ es un vector de pesos y b el término de sesgo, estas variables se *aprenden* a partir del conjunto de datos etiquetados $\mathcal{D}$.

Para clasificar un nuevo ejemplo $\mathbf{x}_j$, basta determinar el lado del híper plano en que se localiza $\mathbf{x}_j$, esto es, la clase de dicho ejemplo estará dada por:

$$f(\mathbf{x}_j) = sign(\mathbf{w}\mathbf{x}_j + b) \tag{2.3}$$

En la SVM el híper plano que separa las clases es aquel que maximiza el margen de separación entre ejemplos de ambas clases. Para obtener dicho

[3]La extensión a multiples clases es sencilla pero no se discute en este capítulo.

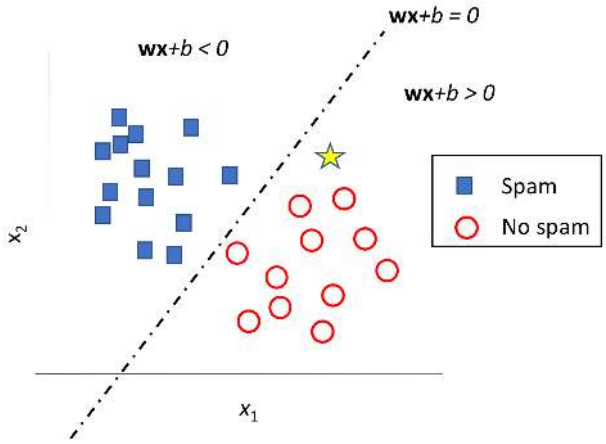


Figura 2.2 Clasificación lineal.

híper plano se resuelve el siguiente problema de optimización:

$$\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}|| \tag{2.4}$$

sujeto a:

$$y_i(\phi(\mathbf{x}_i) + b) \geq 1, \forall(\mathbf{x}_i, y_i) \in \mathcal{D} \tag{2.5}$$

Sin entrar en detalles, las restricciones de desigualdad en 2.5 forzan a que la solución (i.e., el vector de parámetros que resulta en el híper plano) clasifique correctamente cada ejemplo en $\mathcal{D}$. Por otro lado, la función objetivo en 2.4 encontrará el vector de pesos $\mathbf{w}$ que maximiza la distancia entre los extremos positivo y negativo del margen, esto se logra minimizando la norma del vector de pesos (la distancia entre el extremo $\mathbf{wx}+b = -1$ y $\mathbf{wx}+b = 1$ es $\frac{2}{||\mathbf{w}||}$, entonces minimizar $||\mathbf{w}||$ resulta en mazimizar el margen). El concepto de margen se ilustra en la Figura 2.3.

Existen métodos de optimización que garantizan resolver el problema anterior de forma directa y exacta. Sin embargo, mediante una formulación dual del problema en 2.4 es posible obtener una solución tipo *sparse*. En dicha formulación se busca:

$$\arg\max_{\alpha \in \mathbb{R}^n} W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.6}$$

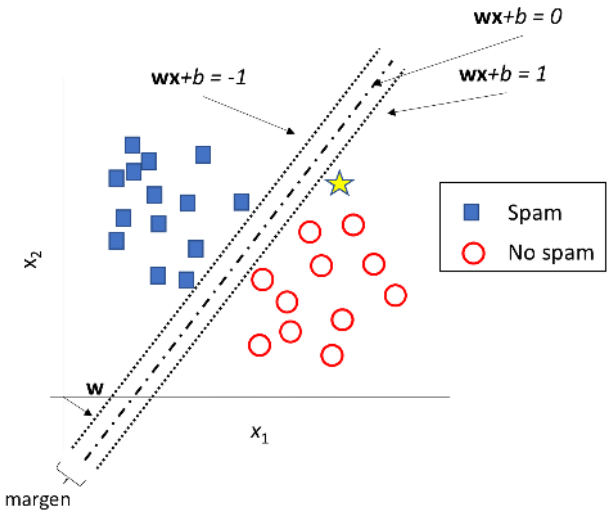


Figura 2.3 La máquina de soporte vectorial.

y la función de decisión se convierte en:

$$f(\mathbf{x}_j) = sign(\sum_{i=1}^{n} y_i \alpha_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b) \tag{2.7}$$

Así, el vector de pesos $\mathbf{w}$ es una combinación lineal de los ejemplos en $\mathcal{D}$, esto es: $\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i$. Lo interesante de esta formulación es que solo algunos de los valores de α serán diferentes de cero, por lo que la función de decisión depende únicamente de un subconjunto de ejemplos, los llamados vectores de soporte (por ello se dice que es una representación *dispersa*). Los vectores de soporte serán aquellos ejemplos que yacen en la frontera de decisión, los más cercanos al margen por ambos lados (positivo y negativo).

Este método garantiza obtener el híper plano óptimo cuando el problema es linealmente separable, como el caso de la Figura 2.3. En caso de que los datos no permitan encontrar un híper plano que separe los datos, veáse la Figura 2.4, la formulación 2.6 todavía permite encontrar el híper plano óptimo. Esto se realiza mapenado los datos originales $\mathbf{x}_i \in \mathbb{R}^d$ a otro espacio $\Phi(\mathbf{x}_i) \in \mathbb{R}^c$ de mayor dimensionalidad (i.e., $d << c$) donde el problema sea linealmente separable. Esto se ejemplifica en la Figura 2.5: en (a) no es posible separar los datos en la linea $\mathbb{R}$ con un umbral sin cometer errores; mientras que en (b), después de realizar un mapeo $\Phi(x_i) = x_i, x_i^2$, los datos (ahora en $\mathbb{R}^2$) se pueden separar con un híper plano.

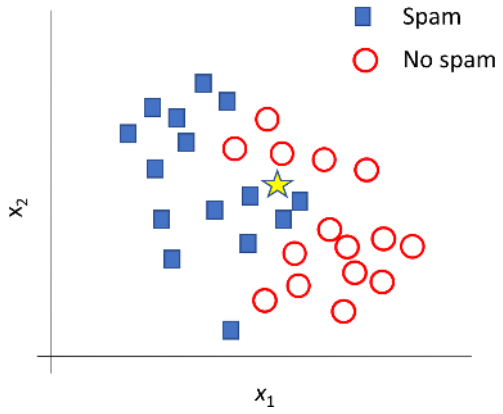


Figura 2.4 Ilustración de un problema de clasificación no lineal.

Así, la función de decisión de la SVM se transforma en:

$$f(\mathbf{x}_j) = sign(\sum_{i=1}^{n} y_i \alpha_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle + b) \tag{2.8}$$

Esta formulación requiere el calculo de un producto punto entre el resultado de mapear los datos originales a ese espacio de mayor dimensionalidad (el llamado espacio de características o *feature space*): $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$. En general, para casos no triviales como el que se ilustra en la Figura 2.5 (b), es muy costoso calcular el mapeo Φ. No obstante lo anterior, y gracias al denominado *kernel trick*, es posible estimar el resultado del producto punto $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ sin tener que calcular el mapeo Φ explícitamente!. Sea $k(\mathbf{x}_i, \mathbf{x}_j)$ una función llamada *kernel* que corresponde al resultado de un producto punto en algún espacio de características, i.e.: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$, y que cumple las propiedades de ser positivo definido y simétrico. Entonces:

$$f(\mathbf{x}_j) = sign(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) + b) \tag{2.9}$$

Existen diferentes formas de definir k, por ejemplo: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^p$ y $k(\mathbf{x}_i, \mathbf{x}_j) = exp\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{2\sigma^2}\right)$, son ambas funciones de kernel que calculan el producto punto de expansiones del espacio de entrada y que resultan en clasificadores que pueden lidiar con superficies de clasificación altamente no lineales.

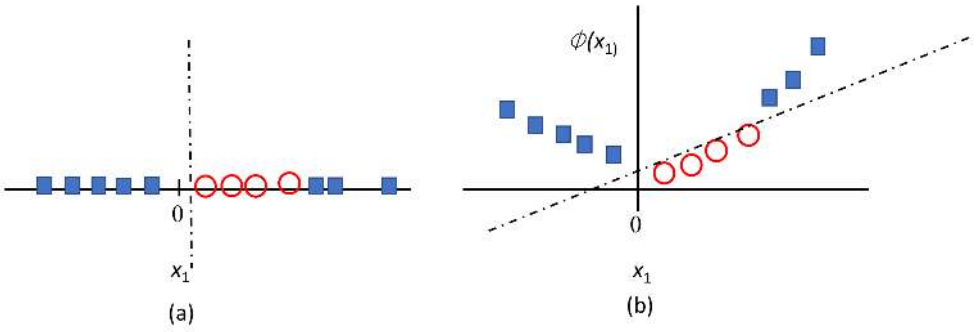


Figura 2.5 Mapeo de datos de el espacio de entrada original $x_i \in \mathbb{R}$ (a), al espacio $\Phi(x_i) \in \mathbb{R}^2$, con el mapeo $\Phi(x_i) = x_i, x_i^2$.

El problema de optimización asociado a la SVM se puede resolver exactamente, siempre y cuando los ejemplos de las clases consideradas sean *linealmente separables* (es posible que aun con el mapeo Φ los datos no sean linealmente separables). Cuando esto no sucede, aun es posible aplicar el SVM: se pueden introducir variables de holgura en 2.5 de manera que algunos ejemplos puedan caer incluso dentro el margen.

El clasificador SVM goza de fuertes fundamentos teóricos que permiten analizar en incluso anticipar su desempeño. Para un mejor entendimiento de este método se sugiere al lector revisar la siguiente referencia (Scholkopf and Smola, 2001).

2.2.3. Clasificador Bayesiano simple

Aunque el Capítulo 4 revisa en detalle métodos Bayesianos para clasificación, vale la pena mencionar aquí al clasificador Bayesiano simple o *ingenuo* (de *naïve* Bayes). Se trata de un clasificador probabilista que asume independencia entre los atributos descriptivos de los objetos dada la clase. Lo anterior hace que los parámetros del clasificador puedan estimarse fácil y eficientemente. La función de decisión que aproxima este clasificador es como sigue:

$$f(\mathbf{x}_j) = \arg \max_{C_k \in \mathcal{C}} \hat{P}(C_k|\mathbf{x}_j) \tag{2.10}$$

donde $\hat{P}(C_k|\mathbf{x}_j)$ es un estimado de la probabilidad de que el objeto $\mathbf{x}_j$ pertenezca a la clase C_k. La clase para la cual se maximice dicha probabilidad

será asignada al objeto. En el ejemplo de filtrado de spam, se calcularia para un nuevo mensaje cuál es la probabilidad de que el mensaje sea genuino, y la probabilidad de que éste sea no deseado. El mensaje será etiquetado con la clase que obtenga mayor probabilidad.

La probabilidad mencionada anteriormente podría estimarse de muchas formas, en el caso del clasificador Bayesiano simple ésta se obtiene haciendo uso del teorema de Bayes como sigue:

$$\hat{P}(C_k|\mathbf{x}_j) = \frac{\hat{P}(\mathbf{x}_j|C_k)P(C_k)}{P(\mathbf{x}_j)} \tag{2.11}$$

Eliminando el denominador (pues no hay dependencia con la clase), y asumiendo independencia[4] entre los atributos de $\mathbf{x}_j$, se tiene:

$$\hat{P}(C_k|\mathbf{x}_j) \approx \prod_{i=1}^{d} \hat{P}(x_{j,i}|C_k)P(C_k) \tag{2.12}$$

Esta formulación cuenta con dos términos: por un lado $P(C_k)$ toma en cuenta la probabilidad *a priori* de la clase C_k, esto es, ante la auscencia de información sobre $\mathbf{x}_j$, cuál es la probabilidad de que el objeto pertenezca a dicha clase. Este término puede calcularse de forma sencilla, por ejemplo contando la proporción de ejemplos que pertenecen a la clase con respecto al total.

Por otro lado, se tiene un componente que depende del objeto $\mathbf{x}_j$. Este componente comprende el producto[5] de la probabilidad de que se observe el valor característico $x_{j,i}$ en ejemplos en $\mathcal{D}$ que pertenecen a la clase C_k. En el ejemplo, esto podría corresponder a estimar la probabilidad de observar una palabra (e.g., *prestamo*) en un mensaje que es spam (C_k). Estos términos de probabilidad se pueden estimar de muchas formas, dependiendo de la forma de estimarlos, se obtiene una variante diferente del clasificador. Por ejemplo, para datos numéricos (valores en $\mathbb{R}$) se puede suponer una distribución Gaussiana y estimar la probabilidad consecuentemente. Para atributos que

[4]Es esta suposición lo que hace al clasificador Bayesian *ingenuo*, puesto que, en general, las características descriptivas de un objeto no son independientes entre si. Por ejemplo, considerese el ejemplo de filtrar spam, en ese escenario la ocurrencia de palabras en un mensaje no es necesariamente independiente (e.g., si aparece la palabra *premio* es más probable que aparezcan palabras como: *loteria, sorteo, dinero*, etc.).

[5]Generalmente se realiza un suavizado de las probabilidades para evitar que haya valores de cero en el producto.

toman valores binarios, se asume una distribución de Bernoulli, mientras que para variables que toman valores enteros y/o categoricos se puede asumir una distribución de probabilidad multinomial. Para mayores detalles de este clasificador se sugiere consultar el capítulo de aprendizaje Bayesiano dentro de este volumen.

Además de ser altamente efectivo, la ventaja de este clasificador es su simplicidad, y eficiencia (su complejidad al clasificador es casi constante: simplemente basta consultar las probabilidades y realizar un producto). Dentro de las limitantes, se encuentra la posibilidad de caer en errores de precisión, puesto que se requiere la multiplicación de números muy pequeños. Para lidiar con este problema se suelen usar logaritmos. Igualmente, es necesario recurrir a técnicas de suavizado para lidiar con el caso de que alguno de los términos en la multiplcación tomen el valor cero (en principio, esto ocasionaría que el valor de probabilidad sea cero). Finalmente, existen diversas mejoras al clasificador que intentan aliviar la suposición de independencia (e.g., TAN (Friedman et al., 1997)), o bien, extender sus capacidades.

2.2.4. Otros métodos de clasificación

Existen muchos otros métodos de clasificación que no se mencionan en este capítulo introductorio, entre éstos destacan:

- **Árboles de decisión.** Generan un árbol que umbraliza los valores de los descriptores de los objetos a clasificar. Cada nodo del árbol realiza un particionamiento de los ejemplos de entrenamiento, de manera que las hojas de los árboles tengan la mayor pureza posible (i.e., que los ejemplos que puedan pertenecer a una hoja pertenezcan a la misma clase). Entre los beneficios de este tipo de métodos es que se puede analizar e interpretar a los árboles generados. Más información en Capítulo 3 de (Mitchell, 1997).

- **Perceptron multicapa.** Un perceptron multicapa es un arreglo de perceptrones simples (clasificadores lineales del tipo $\mathbf{wx} + b$) a los que se les aplican transformaciones no lineales. A través de multiples capas de este tipo de unidades es posible aproximar funciones altamente complejas. Son mejor conocidos como redes neuronales artificiales, veáse el libro sobre redes neurales de esta misma serie para mayores detalles, así como el Capítulo 4 en (Mitchell, 1997).

- **Clasificadores basados en reglas.** Se trata de clasificadores formados por reglas de decisión del tipo: ***si*** *(condición 1, ... condición n)* ***entonces*** *clase 1* ***de otra forma*** *clase 2*. Donde cada condición está asociada a la umbralización de los valores que toman los descriptores de los objetos a clasificar. La generación de reglas busca maximizar la cobertura de los ejemplos en $\mathcal{D}$: cada regla debe poder clasificar la mayor cantidad de ejemplos de una clase en particular.

2.3. Evaluación

Como se ha podido ver en la sección anterior, diferentes métodos de clasificación hacen diferentes suposiciones, tienen sesgos y características distintivas. Con tantas variantes de algoritmos de aprendizaje es crítico evaluar objetivamente su desempeño. Al desarrollar/implementar un clasificador como parte de algún sistema de toma de decisiones, es crítico evaluar su desempeño. La evaluación nos dará evidencia necesaria para anticipar el correcto funcionamiento del sistema. Una evaluación sistemática es imprescindible para anticipar la efectividad del clasificador antes de que se implemente en algún sistema.

2.3.1. Estimación del desempeño en datos no vistos

Hasta el momento se ha dicho que se cuenta con datos para generar el modelo ($\mathcal{D}$), y que se busca que el clasificador desarrollado pueda clasificar correctamente cualquier ejemplo posible generado de la misma distribución ($\mathcal{T}$). Así, es necesario utilizar los datos disponibles, $\mathcal{D}$, para generar un *estimado* del desempeño que el método tendría en $\mathcal{T}$. Tentativamente, uno podría pensar que un buen estimado se puede obtener al evaluar el desempeño del clasificador f^* en $\mathcal{D}$, por ejemplo mediante:

$$D_{\mathcal{D}}(f^*) = \sum_{\forall(\mathbf{x}_i, y_i)\in\mathcal{D}} \mathbf{1}_{y_i = f^*(\mathbf{x}_i)} \tag{2.13}$$

Donde $D_{\mathcal{D}}(f^*)$ es una medida del desempeño del clasificador f^* en $\mathcal{D}$, y recordando que $\mathbf{1}_{a=b} = 1$ si $a = b$ y 0 en caso contrario. Aunque $D_{\mathcal{D}}(f^*)$ es un estimado, generalmente no es de gran utilidad, debido a que f^* fue construido precisamente para optimizar $D_{\mathcal{D}}(f^*)$. Así, dicho estimado será optimista y poco informativo.

Para evitar obtener estimados más confiables se suele recurrir a técnicas de muestreo: los datos disponibles se subdividen en diferentes particiones $\mathcal{D} = \mathcal{D}_1, \cup \mathcal{D}_2, \ldots, \cup \mathcal{D}_K$ y $\mathcal{D}_1, \cap \mathcal{D}_2, \ldots, \cap \mathcal{D}_K = \emptyset$. Algunas de estas particiones se usan para generar el modelo y otras para evaluarlo. El caso más común es cuando $K = 3$, resultando en las particiones $\mathcal{D}_1, \mathcal{D}_2$ y $\mathcal{D}_3$. Una de estas particiones (e.g., $\mathcal{D}_1$) se usa para generar el modelo (partición de entrenamiento); otra (e.g., $\mathcal{D}_2$) se usa para ajustar los valores de los parámetros del modelo (partición de validación); y la última ($\mathcal{D}_3$) se usa para estimar el desempeño del modelo (partición de prueba). Por ejemplo, en el caso de $k - NN$, los datos en $\mathcal{D}_1$ se usarían para obtener los vecinos más cercanos, mientras que el desempeño del clasificador en $\mathcal{D}_2$ podría usarse para evaluar el desempeño de $k - NN$ para diferentes valores de k (e.g., $k \in \{1, 3, 5, 7\}$), y así seleccionar el mejor valor, digamos $k = 5$, finalmente, el estimado de desempeño en $\mathcal{T}$ se obtendría de clasificar los datos en $\mathcal{D}_3$ usando $k = 5$, el mejor valor resultante del proceso de *selección de modelo*.

Otra variante muy común es la de validación cruzada: $\mathcal{D}$ se divide en K particiones, y se realizan K evaluaciones de desempeño, en cada una de éstas se usan $K - 1$ particiones para generar el modelo y 1 para evaluar su desempeño. El promedio de las K evaluaciones se presenta como el estimado de desempeño en $\mathcal{T}$. Otras variantes incluyen 5×2- validación cruzada y validación tipo *leave-one-out*.

2.3.2. Medidas de evaluación

En la sección anterior se introdujo la fórmula 2.13 para calcular la efectividad de clasificación de f^*. Sin embargo, en este punto es conveniente preguntarnos qué es lo que se desea evaluar de f^*?. Por definición, se busca que f^* aproxime lo más fiel posible a f, usando $\mathcal{D}$. En la sección anterior se explicó que la estimación debe hacerse usando técnicas de muestreo, aunque no se ahondó en cómo hacer la estimación. En esta sección se describen las principales medidas de evaluación usadas en clasificación.

Sean $\mathcal{D}_G$ y $\mathcal{D}_E$ las particiones de datos para generar y evaluar a un modelo de clasificación f^*, respecivamente. Las medidas más usadas para evaluación en salidas categóricas son las siguientes:

- Exactitud (*accuracy*).

$$ACC(f^*)_{\mathcal{D}_E} = \frac{1}{|\mathcal{D}_E|} \sum_{\forall(\mathbf{x}_i, y_i) \in \mathcal{D}_E} (\mathbf{1}_{y_i = f^*(\mathbf{x}_i)})$$

- Error (*0-1 loss*)

$$ERR(f^*)_{\mathcal{D}_E} = \frac{1}{|\mathcal{D}_E|} \sum_{\forall(\mathbf{x}_i, y_i)\in\mathcal{D}_E} (\mathbf{1}_{y_i \neq f^*(\mathbf{x}_i)})$$

$ACC(f^*)$ básicamente es la proporción de ejemplos en $\mathcal{D}_E$ que fueron clasificados correctamente. Mientras que $ERR(f^*)$ es inversamente proporcional (e.g., cuando se tienen 10 ejemplos de correo genuino y 990 ejemplos de correo no deseado).

A pesar de ser las médidas más utilizadas, estas medidas pueden ser engañosas[6] cuando se tienen *datos deslabanceados*, esto es, que el número de ejemplos etiquetados para las diferentes clases sea desproporcional. Por lo que muchas veces es necesario analizar con mayor detalle los resultados. Una medida comúnmente usada para evaluación en salidas categóricas en datos desbalanceados es el BER:

- Tasa de error balanceada (*Balanced error rate*):

$$BER(f^*)_{\mathcal{D}_E} = \frac{E_- + E_+}{2}$$

donde E_- / E_+ es la tasa de error en instancias de las clases negativa y positiva, respectivamente.

De igual forma, otras medidas comunmente utilizadas para calcular el desempeño de clasificadores, y que proveen más información que la exactitud, son:

- **Sensitividad o cobertura (Recall).** Tasa de verdaderos positivos.

$$Sens(f^*) = Rec(f^*) = \frac{TP}{TP + FN}$$

- **Especifidad.** Tasa de verdaderos negativos.

$$Esp(f^*) = \frac{TN}{TN + FP}$$

[6] Si del total de instancias $\mathcal{D}_E$, 90 % son de la clase 1 y 10 % de la clase -1. Un clasificador trivial que siempre predice la clase 1 tendrá $ACC = 0.9$ y/o $ERR = 0.1$.

- **Precisión.** Del total de predicciones positivas (resp. negativas) cuántas clasificó correctamente.

$$Prec_{+}(f^{*}) = \frac{TP}{TP + FP}$$

- **Medida** f_{β}. Compromiso entre precisión y cobertura, usualmente $\beta = 1$.

$$f_{\beta}(f^{*}) = \frac{2 \times Prec \times Rec}{Prec + Rec}$$

Donde:

- **TP:** número de verdaderos positivos. *Cuántos ejemplos que pertenecen a la clase positiva se clasificaron correctamente.*
- **FP:** número de falsos positivos. *Cuántos ejemplos que pertenecen a la clase negativa se clasificaron incorrectamente.*
- **TN:** número de verdaderos negativos. *Cuántos ejemplos que pertenecen a la clase negativa se clasificaron correctamente.*
- **FN:** número de falsos negativos. *Cuántos ejemplos que pertenecen a la clase positiva se clasificaron incorrectamente.*

Los conceptos anteriores son ilustrados en la tabla de contingencia mostrada en la Figura 2.6

2.4. Aplicaciones

Se ha mencionado que clasificación es una de las tareas insignia dentro de aprendizaje computacional, sin embargo, no se ha ahondado en aplicaciones de sistemas de clasificación. A continuación se describen brevemente algunas aplicaciones del ámbito cotidiano que se basan en sistemas de clasificación:

- **Clasificación de textos.** Se ha usado como ejemplo de trabajo un sistema para filtrar correo no deseado, en dicho ejemplo los objetos de interés son mensajes de correo electrónico y las salidas corresponden a si el mensaje es deseado o no. Este mismo esquema es altamente popular dentro del área de procesamiento de lenguaje natural para

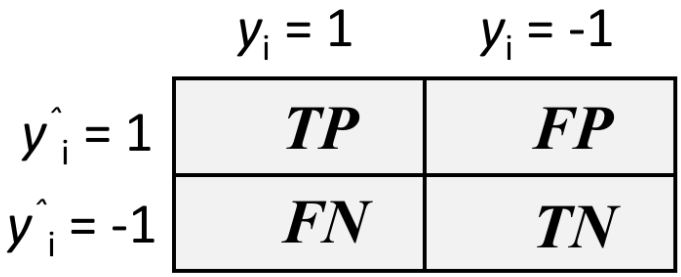


Figura 2.6 Matriz de confusión considerando 2-clases, y_i denota la etiqueta correcta para el ejemplo $i-$ésimo; mientras que $\hat{y}_i$ es la predicción del modelo: $\hat{y}_i = f^*(\mathbf{x}_i)$

la clasificación de textos. Por ejemplo, considerese la clasificación de noticias de acuerdo a su temática (e.g., deportes vs. política), o bien, la clasificación de tuits de acuerdo a su polaridad (e.g., positivo vs. negativo), o yendo más allá, reconocer tareas escritas por el estudiante A (escritas por estudiante A vs. escritas por alguien más). Todas estas tareas se pueden resolver por métodos de clasificación, y sistemas de este tipo son hoy en día ubicuos.

- **Reconocedores de rostros.** En este caso, nos interesa determinar la identidad de una persona a partir de una imagen del rostro. Aquí, los objetos corresponden a imágenes de rostro, caracterizadas por algún descriptor visual. Mientras que las categorías corresponderían a las identidades disponibles (e.g., *Juan, Maria, Pedro o Guadalupe,*). Problemas de este tipo se resuelven con métodos de clasificación, y están presentes en todos lados (e.g., en redes sociales, en sistemas de verificación de banca móvil, etc.).

- **Recomendadores de productos.** Considerese el problema de clasificar (filtrar) productos (e.g., películas o libros) que serán de interés para un usuario. Este problema, que genera utilidades millonarias, puede resolverse mediante el uso de un clasificador. Aquí los objetos corresponden a los usuarios, caracterizados por algún descriptor (e.g., edad, genero, localización geográfica, nivel de estudios, etc.), y las etiquetas corresponderian a los productos disponibles (e.g., todas las películas en

un catálogo).

A través de estos ejemplos, no es difícil imaginarse escenarios o aplicaciones similares en donde sistemas de clasificación tienen ingerencia, de ahí la importancia de esta tarea emblemática del aprendizaje computacional supervisado.

2.5. Discusión

En este capítulo se ha revisado la tarea de clasificación en el contexto de aprendizaje computacional. Se revisaron los algoritmos basicos de aprendizaje computacional, medidas de evaluación y se señalaron algunas de las principales aplicaciones. Cabe destacar que el presente capítulo intenta ser únicamente introductorio y que por ninguna manera intenta hacer una revisión exhaustiva. Se sugiere al lector consultar las referencias para descripciones más completas y un mejor entendimiento de los temas abordados.

Referencias

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Capítulo 3

Regresión

E. Morales, H. Escalante-Balderas
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

Regresión es otra tarea[1] de aprendizaje computacional supervisado, por tanto, requiere de ejemplos etiquetados para aprender un modelo que mapee entradas a salidas. A diferencia de la tarea de clasificación en la que las salidas corresponden a categorías (e.g., mensaje genuino vs. mensaje no deseado), en regresión nos interesa predecir una variable continua (e.g., la calificación que obtendrá un alumno en algún curso, a partir de información de su desempeño en cursos relacionados y otros antecedentes). Como se verá en este capítulo, la tarea tiene una formulación similar a la de clasificación, sin embargo posee características que la hacen diferente. El resto del capítulo presenta la formulación de la tarea, los principales métodos para generar modelos de regresión, medidas de evaluación y algunas aplicaciones.

3.1. Formulación de la tarea

Como se ha mencionado antes, el aprendizaje supervisado se refiere al área de aprendizaje computacional que estudia el desarrollo de modelos que, mediante el análisis de datos etiquetados generados por un usuario, sean capaces de realizar predicciones para datos no vistos. En el caso particular de regresión, se busca generar modelos que permitan mapear objetos a valores reales.

[1]Nótese que el análisis de regresión va mucho más allá que el aprendizaje supervisado, tiene sus raices en estadística.

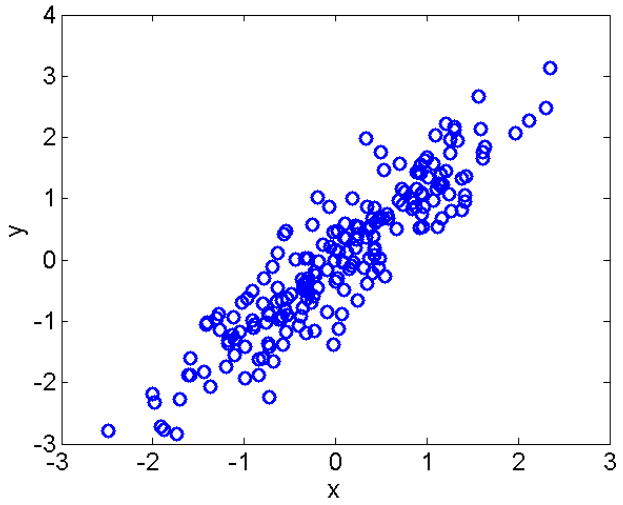


Figura 3.1 Problema de regresión.

Por ejemplo, considerese el problema de estimar la cantidad en kilogramos de grano que será producido por un agricultor, tomando en cuenta variables asociadas como el el área que fue sembrada, la cantidad de fertilizante utilizado, el número de días que se esperan de lluvia, etc. Se trata de un problema en el que se busca estimar una variable continua $y \in \mathbb{R}$ a partir de ciertas mediciones que caracterizan al fenomeno de interés.

Formalmente, sea $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ el conjunto de ejemplos provisto por el usuario para resolver una tarea de regresión. Donde cada par $(\mathbf{x}_i, y_i)$ denota un ejemplo etiquetado, y donde $\mathbf{x}_i \in \mathbb{R}^d$ es la codificación numérica del $i-$ésimo objeto y $y_i \in \mathbb{R}$ es el valor continuo asociado al objeto y que se desea estimar. Nótese que aunque es posible que $y_i \in \mathbb{R}^p$, es decir que se deseen predecir múltples variables, en el resto del capítulo nos enfocaremos en problemas donde se desea predecir una única variable continua. La Figura 3.1 ilustra los datos asociados a un problema de regresión, con $d = 1$.

Como es el caso de clasificación, el conjunto de datos $\mathcal{D}$ usado para generar el modelo poviene de una distribución de probabilidad $P(\mathbf{X}, \mathbf{y})$ que es desconocida. El problema de regresión consiste en encontrar una aproximación f^* de la función $f : \mathbb{R}^d \rightarrow \mathbb{R}$, tal que $\hat{y}_j = f^*(\mathbf{x}_j)$ es igual (o lo más parecido posible) a y_j, $\forall(\mathbf{x}_j, y_j) \in \mathcal{T}$ asociado a la misma distribución $P(\mathbf{X}, \mathbf{y})$. Así, la única distinción para abordar el problema es la naturaleza de la variable a predecir (denominanda variable dependiente).

3.2. Principales métodos para regresión

La presente sección describe los principales métodos para obtener un modelo de regresión, a partir de un conjunto de muestras etiquetadas $\mathcal{D}$. Primero se presentan métodos clásicos y posteriormente se extienden algunos métodos de clasificación vistos en el Capítulo 2 para abordar el problema de regresión.

3.2.1. Regresión lineal

Una de las formas más sencillas, aunque efectivas, es asumir que la función desconocida es una combinación lineal de los valores de las variables de entrada (denominadas variables independientes). Esto es, se asume que f es de la forma[2]:

$$y_i = f(\mathbf{x}_i) = \mathbf{w}\mathbf{x}_i + \epsilon_i \tag{3.1}$$

Donde ϵ_i es un error que está fuera de control del modelo, y puede ser causado por errores de medición y ruido, generalmente se asume que ϵ sigue una distribución Gaussiana, por el momento se omitirá el tratamiento de esta variable. La Figura 3.2 ilustra una aproximación a los datos mediante una función lineal. De acuerdo a la suposición 3.1, se busca entonces encontrar un estimado f^* que se aproxime lo más posible a f, tomando como referencia la muestra de datos $\mathcal{D}$. Así, se quiere minimizar la distancia entre $y_i = f(\mathbf{x}_i)$ y las predicciones de un modelo aproximado $f^*(\mathbf{x}_i) = \mathbf{w}\mathbf{x}_i$. Como se verá más adelante, es conveniente definir la siguiente medida de error:

$$Err(\mathbf{w}) = \frac{1}{2} \sum_{\forall \mathbf{x}_i \in \mathcal{D}} (\mathbf{w}\mathbf{x}_i - y_i)^2 \tag{3.2}$$

Así, se busca encontrar los valores del vector de pesos $\mathbf{w} \in \mathbb{R}^d$ que minimicen 3.2.

Una forma de resolver el problema es mediante el método de *ajuste por mínimos cuadrados*. Esto es, se busca minimizar $Err(\mathbf{w})$ calculando su derivada y encontrando el conjunto de valores $\mathbf{w}$ para los cuales la derivada parcial con respecto a $\mathbf{w}$ es cero. La derivada parcial de interés está dada por:

$$\frac{\partial Err(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2} \sum_{\forall \mathbf{x}_i \in \mathcal{D}} 2\Big(\mathbf{w}\mathbf{x}_i - y_i\Big)\mathbf{x}_i \tag{3.3}$$

[2]Nótese que se ha eliminado el termino de sesgo b del modelo lineal, esto no es ningún problema, pues b se puede incluir en $\mathbf{w}$ y se aumenta el espacio de entrada con una columna de unos.

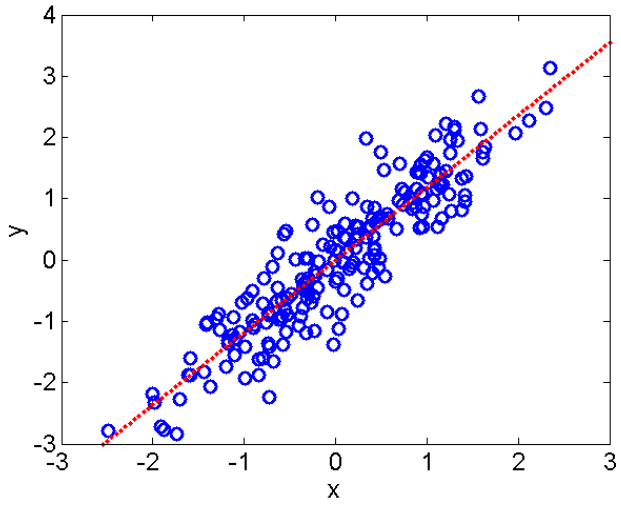


Figura 3.2 Problema de regresión y una aproximación.

En forma matricial, si $\mathbf{X}$ es la matriz con dimensiones $n \times d$ cuyas, filas corresponden a los valores de las variables de entrada $\mathbf{x}_i$ en $\mathcal{D}$, y $\mathbf{y}$ es el vector correspondiente con los valores de la variable dependiente y. Entonces:

$$\frac{\partial Err(\mathbf{w})}{\partial \mathbf{w}} = (\mathbf{X}\mathbf{w} - \mathbf{y})'\mathbf{X} \tag{3.4}$$

Donde $\mathbf{A}'$ indica la matriz transpuesta de $\mathbf{A}$. Igualando a cero y despejando se obtiene:

$$\mathbf{w} = ((\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}')\mathbf{y} \tag{3.5}$$

si $\mathbf{X}$ es una matriz cuadrada y positiva definida, entonces la inversión de la matriz es posible, cuando esto no es posible, aun se puede calcular una matriz pseudo inversa. Así, el vector de pesos que minimiza 3.2 se puede obtener de manera óptima mediante la ecuación 3.5. Nótese sin embargo que esta solución óptima para $\mathcal{D}$ no garantiza ser óptima para cualquier observación$\mathbf{x}_j$ que sea muestreada de la misma distribución de probabilidad, aunque puede ser una buena aproximación si se tienen suficientes datos y si la función de regresión es efectivamente lineal.

Otra opción para hallar los pesos de la función $f^*(\mathbf{x}_i) = \mathbf{w}\mathbf{x}_i$ es mediante optimización por gradiente descendiente, que aunque no garantiza obtener la solución óptima, permite resolver el problema cuando la inversión matricial de 3.5 no es posible, o bien, cuando es costosa (d y/o n son muy grandes).

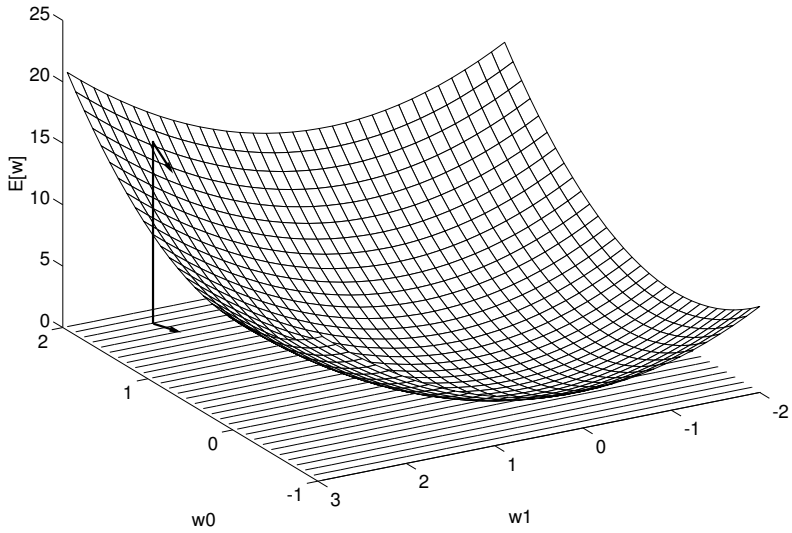


Figura 3.3 Comportamiento de $E(\mathbf{w})$ en dos dimensiones. La flecha indica la dirección que produce el máximo (resp. mínimo) incremento del error. Figura tomada de **?**.

Partiendo de la misma función de error en 3.2 y usando las derivadas parciales de $E(\mathbf{w})$ con respecto a $\mathbf{w}$ podemos aplicar el método de gradiente descendente para encontrar el vector de pesos. En términos generales, queremos encontrar el vector de pesos que minimice 3.2, esté método de optimización se basa en la idea de iniciar con un vector de pesos aleatorios para $\mathbf{w}$ e iterativamente se *mueve* dicho vector en la dirección que minimice el error $E(\mathbf{w})$. La dirección de cambio se obtiene mediante el gradiente. El gradiente nos especifica la dirección que produce el máximo incremento, por lo que el mayor descenso es el negativo de la dirección. Esto se ilustra en la Figura 3.3.

La regla de actualización de pesos es entonces:

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = -\alpha\nabla E$$

donde α es el factor de aprendizaje, intuitivamente indica la tasa de cambio, y la confianza en el error para ajustar el vector de pesos $\mathbf{w}$.

Ya hemos calculado en 3.3 las derivadas parciales necesarias, sin embargo, a continuación se presentan los gradientes correspondientes a nivel elemento

del vector de pesos $\mathbf{w}$:

$$\begin{aligned}\frac{\partial E(\mathbf{w})}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{\forall \mathbf{x}_i \in \mathcal{D}}(f^*(\mathbf{x}_i) - f(\mathbf{x}_i))^2 \\ &= \sum_{\forall \mathbf{x}_i \in \mathcal{D}}(f^*(\mathbf{x}_i) - f(\mathbf{x}_i))\frac{\partial}{\partial w_j}(f(x) - \mathbf{w}\cdot\mathbf{x}_i) \\ &= \sum_{\forall \mathbf{x}_i \in \mathcal{D}}(f^*(\mathbf{x}_i) - f(\mathbf{x}_i))(-x_{i,j})\end{aligned}$$

Por lo que:

$$\Delta w_j = \alpha \sum_{\forall \mathbf{x}_i \in \mathcal{D}} (f^*(\mathbf{x}_i) - f(\mathbf{x}_i))(-x_{i,j}) \tag{3.6}$$

Así, el procedimiento para encontrar el vector de pesos $\mathbf{w}$ consiste en:

1. Inicializar $\mathbf{w} \in \mathbb{R}^d$ con valores aleatorios
2. Repetir hasta que se cumpla criterio de paro:
   a) Calcular $\Delta w_j = \alpha \sum_{\forall \mathbf{x}_i \in \mathcal{D}}(f^*(\mathbf{x}_i) - f(\mathbf{x}_i))(-x_{i,j})$
   b) Actualizar $\mathbf{w}$: $w_j \leftarrow w_j + \Delta w_j, \forall j \in \{1, \ldots, \}$
3. Regresar $\mathbf{w}$

Como criterios de paro pueden considerarse un número fijo de actualizaciones del vector pesos, o bien algún criterio de convergencia (e.g., que los valores de $\mathbf{w}$ no cambien por cierto número de iteraciones).

3.2.2. Regresión lineal localmente ponderada

Una de las suposiciones más fuertes y que resulta ser una de las principales limitantes de regresión lineal, es precisamente suponer que la función desconocida se puede aproximar a través de una función lineal. Una extensión de la regresión lineal para lidiar con este tipo de problemas consiste en la regresión localmente ponderada (RLP). RLP construye una función que ajusta los datos de entrenamiento que están en la vecindad del objeto $\mathbf{x}_j$ que se desea clasificar. La idea intuitiva detrás de RLP se ilustra en la Figura 3.4

Básicamente, en RLP se trata de aproximar una función lineal en un vecindario alrededor de $\mathbf{x}_j$. Así, para clasificar a $\mathbf{x}_j$, es necesario primero determinar su vecindario. Esto es posible usando ideas como las vistas en la sección relacionada con el clasificador de $k-NN$. Una vez que se identifica el

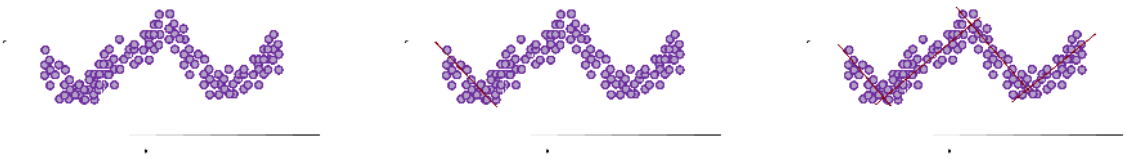


Figura 3.4 RLP. Izquierda: datos asociados a un problema de regresión altamente no lineal. Centro: RLP aproxima la función de manera local en un vecindario. Derecha: Mediante varias aproximaciones lineales y locales es posible aproximar la función no lineal.

vecindario, es posible aplicar un método de regresión lineal sobre los puntos que componen el vecindario de $\mathbf{x}_j$. Un esquema de RLP puede ser tan sencillo como aplicar cualquiera de los métodos descritos en la sección anterior, solo que en lugar de usar $\mathcal{D}$ para aproximar a la función f^* se usarían los vecinos más cercanos al punto que se desea clasificar: $\mathbf{x}_1^N, \ldots \mathbf{x}_k^N$. Alternativamente, se puede sacar provecho de información relevante a partir de los vecinos más cercanos. Por ejemplo, es posible obtener diferentes variantes de RLP si se minimizan diferentes medidas de error:

1. Minimizar el error cuadrado usando los k vecinos más cercanos.

$$E(W) = \frac{1}{2}\sum_{1}^{k}(f(\mathbf{x}_j) - f^*(\mathbf{x}))^2$$

2. Minimizar el error cuadrado usando todos los ejemplos pesados por su distancia a x_q.

$$E(W) = \frac{1}{2}\sum_{x\in\mathcal{D}}(f(\mathbf{x}_j) - f^*(\mathbf{x}))^2 K(d(\mathbf{x}_j, \mathbf{x}))$$

3. Minimizar el error cuadrado usando los k vecinos más cercanos pesados por su distancia a x_q.

$$E(W) = \frac{1}{2}\sum_{1}^{k}(f(\mathbf{x}_j)(f(\mathbf{x}_j) - f^*(\mathbf{x}))^2 K(d(\mathbf{x}_j, \mathbf{x}))$$

3.2.3. Extensión de métodos de clasificación

Además de las aproximaciones por mínimos cuadrados, descenso de gradiente y soluciones locales, es posible modificar algoritmos de clasificación para lidiar con salidas reales y convertirlos en métodos de regresión. En el resto de esta sección se describen tales enfoques.

KNN

El método de $k-$ vecinos más cercanos se puede utilizar casi de forma directa para abordar problemas de regresión. Básicamente, la predicción de la variable dependiente de $k-$ NN$para un ejemplo \mathbf{x}_j$ se obtiene de acuerdo a la siguiente función:

$$f^*(\mathbf{x}_j) = \frac{1}{k} \arg\max_{C_k \in \mathcal{C}} \sum_{i=1}^{k} f(\mathbf{x}_i^N) \tag{3.7}$$

donde: $\mathbf{x}_1^N, \ldots, \mathbf{x}_k^N$ son los k vecinos más cercanos a $\mathbf{x}_j$. Esto es, la predicción para el ejemplo $\mathbf{x}_j$ es el promedio de los valores de la variable dependiente de todos sus vecinos. Al igual que en el caso de clasificación, se puede modificar la función anterior para dar mayor peso a los valores y_i^N de vecinos que se encuentran más cerca de $\mathbf{x}_j$.

3.3. Evaluación

De manera similar a la clasificación, la evaluación de métodos de regresión tiene por objetivo determinar la efectividad de un modelo cuando se le presenten casos no vistos. Se busca que las predicciones de un modelo de regresión sean lo más similar posible a los valores verdaderos. Así, idealmente $\hat{y}_j = f^*(\mathbf{x}_j)$ debería ser igual a y_j para cualquier $\mathbf{x}_j \in \mathcal{T}$. Una forma de evaluar el desempeño de métodos de regresión es midiendo la "distancia" entre $\hat{y}_j$ y y_j. Hay muchas formas en que se puede hacer esto, las más comunes son:

- Root Mean-Squared Error:

$$RMSE(\hat{f}) = \sqrt{\frac{1}{T} \sum_{i=1}^{T} (y_i^T - \hat{y}_i^T)^2}$$

- Mean Absolute Error:

$$MAE(\hat{f}) = \frac{1}{T} \sum_{i=1}^{T} |y_i^T - \hat{y}_i^T|$$

Como en el caso de clasificación, es imprescindible particionar los datos disponibles en diferentes subconjuntos para obtener un buen estimado del desempeño del regresor en datos no vistos.

Capítulo 4

Aprendizaje Bayesiano

E. Morales, H. Escalante-Balderas
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

4.1. Aprendizaje Bayesiano

La incertidumbre es parte fundamental dentro del Aprendizaje Computacional. Por un lado, muchos sistemas de aprendizaje construyen sus modelos con base a medidas estadísticas sobre (parte de) los datos. Por otro, es deseable que los modelos tengan asociada una medida de calidad, siendo la probabilidad una medida ampliamente aceptada. En el Aprendizaje Bayesiano se utilizan principios del teorema de Bayes. Algunas de sus características son:

- Cada nuevo ejemplo puede aumentar o disminuir la estimación de una hipótesis. Esto le da flexibilidad e incrementalidad al sistema.
- Conocimiento *a priori* se puede combinar con datos para determinar la probabilidad de las hipótesis.
- Los resultados se presentan con probabilidades asociadas.
- Puede clasificar combinando las predicciones de varias hipótesis.
- Sirve de estandar de comparación de otros algoritmos.

Parte de su relevancia es que es un enfoque práctico que puede proveer de comprensión (y diseño) de otros algoritmos. Algunos de los problemas que

tiene es que se requieren conocer muchas probabilidades y es computacionalmente caro (depende linealmente del número de hipótesis).

Cuando se realiza un proceso de aprendizaje lo que normalmente se quiere saber es cuál es la mejor hipótesis (más probable) dados los datos.

Si $P(D)$ = probabilidad *a priori* de los datos (i.e., cuáles datos son más probables que otros), $P(D \mid h)$ = es la probabilidad de los datos dada una hipótesis, lo que queremos estimar es: $P(h \mid D)$, la probabilidad posterior de h dados los datos.

Esto lo podemos estimar con Bayes:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

Esto lo podemos usar para estimar la hipótesis más probable o MAP (*maximum a posteriori hypothesis*):

$$\begin{aligned} h_{MAP} &= argmax_{h \in H} \left(P(h \mid D)\right) \\ &= argmax_{h \in H} \left(\tfrac{P(D|h)P(h)}{P(D)}\right) \\ &\approx argmax_{h \in H} \left(P(D \mid h)P(h)\right) \end{aligned}$$

Ya que $P(D)$ es una constante independiente de h.

Si suponemos que las hipótesis son igualmente probables, nos queda la hipótesis de máxima verosimilitud o ML (*maximum likelihood*):

$$h_{ML} = argmax_{h \in H} \left(P(D \mid h)\right)$$

Por ejemplo, supongamos que se tienen dos hipótesis, el paciente tiene cancer o no. Sabemos por un lado que sólo el 0.008 % de la población tiene ese tipo de cancer. Por otro lado, sabemos que la prueba sobre cancer no es infalible y nos da resultados positivos correctos en el 98 % de los casos y resultados negativos correctos en el 97 % de los casos.

Con estos datos, lo que tenemos es lo siguiente: $P(cancer) = 0.008$ y $P(\neg cancer) = 0.992$
$P(\oplus|cancer) = 0.98$ y $P(\ominus|cancer) = 0.02$
$P(\oplus|\neg cancer) = 0.03$ y $P(\ominus|\neg cancer) = 0.97$

Si a un paciente le dieron un resultado positivo en la prueba, podemos calcular la probabilidad de que tenga cancer usando la regla de Bayes:

$P(cancer|\oplus) = P(cancer)P(\oplus|cancer) = 0.008 * 0.98 = 0.0078$
$P(\neg cancer|\oplus) = P(\neg cancer)P(\oplus|\neg cancer) = 0.992 * 0.03 = 0.0298$

Que al normalizar (para que la suma nos de 1), nos da:

$P(cancer|\oplus) = 0.21$

$P(\neg cancer|\oplus) = 0.69$

Por lo que a pesar de que la prueba salió positiva, sigue siendo más probable que no tenga cancer.

4.2. Diferentes vistas del Aprendizaje Bayesiano

Al aprendizaje bayesiano lo podemos relacionar con diferentes aspectos de aprendizaje (Mitchell, 1997). Vamos a ver los siguientes, pero la lista no es exhaustiva:

- Relación con Espacio de Versiones
- Clases contínuas con ruido
- Principio de Longitud de Descripción Mínima
- Clasificador bayesiano óptimo
- ...

4.2.1. BL y Espacio de Versiones

Una forma (impráctica) de un algoritmo Bayesiano es calcular todas las posibles hipótesis $P(h \mid D) = \frac{P(D|h)P(h)}{P(D)}$ y quedarse con la de mayor probabilidad. El problema radica en la cantidad de posibles hipótesis (puede ser gigantesco) y además necesitamos especificar los valores para $P(h)$ y para $P(D \mid h)$.

Si suponemos que no hay ruido y que todas las hipótesis son igualmente probables (i.e., $P(h) = \frac{1}{|H|} \forall h \in H$), $P(D \mid h) = 1$ sii D es consistente con h. Esto es:

$$P(h \mid D) = \frac{1}{\mid VS_{H,D} \mid}$$

donde, $VS_{H,D}$ es el subconjunto de hipótesis de H que es consistente con D (su espacio de versiones (Mitchell, 1982)).

Por lo mismo, toda hipótesis consistente es una hipótesis MAP. En este sentido, cualquier sistema de aprendizaje que nos de hipótesis consistentes

(suponiendo que no hay ruido y que todas las hipótesis son igualmente probables) nos está dando hipótesis MAP.

En general, en un sistema de aprendizaje lo podemos caracterizar suponiendo que las hipótesis más generales (o específicas) son más probables que las otras y en este sentido podemos sesgar al sistema de aprendizaje hacia estas hipótesis.

En general, podemos caracterizar varios algoritmos de aprendizaje con un enfoque Bayesiano, al caracterizar sus distribuciones de probabilidad $P(h)$ y $P(D \mid h)$.

4.2.2. BL, Variables Continuas y Ruido

Los métodos más usados para buscar funciones con variables continuas a partir de datos con ruido, son regresiones líneales, ajustes de polinomios y redes neuronales. La idea es aprender funciones $h : X \to \mathcal{R}$ lo más cercanas a f, en donde los datos están descritos por: $d_i = f(x_i) + e_i$, donde $f(x_i)$ es la función sin ruido y e_i es una variable aleatoria representando el error.

De nuevo lo que queremos es encontrar la hipótesis más probable:

$$h_{ML} = argmax_{h \in H} \left(p(D \mid h) \right)$$

Suponiendo que los datos son independientes entre sí dado h, la probabilidad se puede expresar como el producto de varias $p(d_i \mid h)$ para cada dato:

$$h_{ML} = argmax_{h \in H} \left(\prod_{i=1}^{m} p(d_i \mid h) \right)$$

Si suponemos el ruido con una distribución Gaussiana con media cero y varianza σ^2, cada d_i debe de seguir la misma distribución centrada alrededor de $f(x_i)$.

$$h_{ML} = argmax_{h \in H} \left(\prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2} \right)$$

$$h_{ML} = argmax_{h \in H} \left(\prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2} \right)$$

Podemos maximizar tomando su logaritmo (dado que es una función monotónica creciente):

$$h_{ML} = argmax_{h \in H} \left(\sum_{i=1}^{m} ln(\frac{1}{\sqrt{2\pi\sigma^2}}) - \frac{1}{2\sigma^2}(d_i - h(x_i))^2 \right)$$

Eliminando el primer término (que no depende de h):

$$h_{ML} = argmax_{h \in H} \left(\sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2 \right)$$

Lo anterior es igual a minimizar lo mismo pero con el signo contrario. Al cambiar signo y eliminar constantes que no dependen de h nos queda:

$$h_{ML} = argmin_{h \in H} \left(\sum_{i=1}^{m} (d_i - h(x_i))^2 \right)$$

Lo que nos dice que la hipótesis de máxima verosimilitud es la que minimiza la suma de los errores al cuadrado entre los datos observados (d_i) y los datos predichos ($h(x_i)$), siempre y cuando el error siga una distribución Normal con media cero. Hay que aclarar que lo anterior supone que el error está únicamente en la meta y no en los atributos.

4.2.3. BL y el Principio de Longitud de Descripción Mínima

Como el proceso inductivo no es seguro se necesita de alguna medida de calidad. Normalmente se hace con base en evaluaciones con los ejemplos de entrenamiento y prueba. Una alternativa es encontrar la hipótesis más probable dados los datos. El principio de Longitud de Descripción Mínima o MDL (*Minimum Description Length*) (Rissanen, 1978) está motivado al interpretar la definición de h_{MAP} con base en conceptos de teoría de información.

$$\begin{aligned} h_{MAP} &= argmax_{h \in H} \left(P(D \mid h) P(h) \right) \\ &= argmax_{h \in H} \left(log_2(P(D \mid h)) + log_2(P(h)) \right) \\ &= argmin_{h \in H} \left(-log_2(P(D \mid h)) - log_2(P(h)) \right) \end{aligned}$$

Lo cual puede pensarse como el problema de diseñar el mensaje de transmisión de información más compacto para transmitir la hipótesis y los datos dada la hipótesis.

MDL recomienda seleccionar la hipótesis que minimiza la suma de estas dos descripciones:

$$h_{MDL} = argmin_{h \in H} \left(L(h) + L(D \mid h) \right)$$

Si queremos aplicarlo a un árbol de decisión, tenemos que buscar una codificación para los árboles de decisión y una para los ejemplos mal clasificados junto con su clasificación. Este principio permite establecer un balance entre complejidad de la hipótesis ($L(h)$) y número de errores o calidad de la hipótesis ($L(D \mid h)$).

4.2.4. Clasificador Bayesiano Óptimo

En lugar de la hipótesis más probable, podemos preguntar, cuál es la clasificación más probable. Esto se puede obtener combinando las clasificaciones de todas las hipótesis aplicables pesadas por su probabilidad de clasificación.

$$P(v_j \mid D) = \sum_{h_i \in H} P(v_j \mid D, h_i) P(h_i \mid D) = \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D)$$

Donde v_j es el valor de la clasificación y la clasificación óptima será:

$$argmax_{v_j \in V} \left(\sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D) \right)$$

Por ejemplo, supongamos que tenemos 2 clases y 3 hipótesis (h_1, h_2, h_3), cuyas probabilidades dados los datos son $(0.4, 0.3, 0.3)$. Un nuevo ejemplo x se clasifica positivo por h_1 y negativo por h_2 y h_3. Su clasificación por la hipótesis MAP sería positivo, pero considerando todas las hipótesis sería negativo.

$P(h_1|D) = 0.4, P(\ominus|h_1) = 0, P(\oplus|h_1) = 1$
$P(h_2|D) = 0.3, P(\ominus|h_2) = 1, P(\oplus|h_2) = 0$
$P(h_3|D) = 0.3, P(\ominus|h_3) = 1, P(\oplus|h_3) = 0$

$$\sum_{h_i \in H} P(\oplus \mid h_i) P(h_i \mid D) = 0.4$$

$$\sum_{h_i \in H} P(\ominus \mid h_i) P(h_i \mid D) = 0.6$$

$$argmax_{v_j \in \{\oplus,\ominus\}} \left(\sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D) \right) = \ominus$$

Aplicar el clasificador Bayesiano óptimo puede ser muy costoso (muchas hipótesis). Una posibilidad es seleccionar una hipótesis (h) aleatoriamente de acuerdo con la distribución de probabilidad de las probabilidades posteriores de H, y usar h para predecir. Se puede mostrar que el error esperado es a lo más el doble del error esperado del clasificador Bayesiano óptimo.

4.3. Naïve Bayes

El clasificador ingenuo de Bayes o *Naïve Bayes* (Good et al., 1966) se usa para clasificar una instancia descrita por un conjunto de atributos (a_i's) en un conjunto finito de clases (V). El clasificador asigna una clase de acuerdo con el valor más probable dados los valores de sus atributos:

$$v_{MAP} = argmax_{v_j \in V} \left(P(v_j \mid a_1, \ldots, a_n) \right)$$

Usando Bayes:

$$\begin{aligned} v_{MAP} &= argmax_{v_j \in V} \left(\frac{P(a_1,\ldots,a_n|v_j)P(v_j)}{P(a_1,\ldots,a_n)} \right) \\ &= argmax_{v_j \in V} \left(P(a_1, \ldots, a_n \mid v_j) P(v_j) \right) \end{aligned}$$

$P(v_j)$ se puede estimar con la frecuencia observada en los datos de las clases, pero para $P(a_1, \ldots, a_n \mid v_j)$ generalmente tenemos pocos datos. El clasificador NB supone que los valores de los atributos son condicionalmente independientes entre sí dado el valor de la clase, por lo que la probabilidad condicional conjunta la podemos expresar como un producto de probabilidades: $P(a_1, \ldots, a_n \mid v_j) = \prod_i P(a_i \mid v_j)$. Por lo que:

$$v_{NB} = argmax_{v_j \in V} \left(P(v_j) \prod_i P(a_i \mid v_j) \right)$$

Los valores $P(a_i \mid v_j)$ se estiman con la frecuencia de los datos observados.

Algo importante de este tipo de clasificador es que no se hace una búsqueda de hipótesis, simplemente se cuentan frecuencias de ocurrencias de valores en los datos.

Por ejemplo, tomando la tabla de jugar golf, si queremos clasificar con un *Naïve Bayes* el siguiente ejemplo:

Ambiente=soleado, Temperatura=baja, Humedad=alta, Viento=si

Tenemos que calcular cuál sería la probabilidad de que la clase sea positiva para ese ejemplo y cuál de que sea la clase negativa para ese ejemplo y quedarnos con la más probable:

$$\begin{aligned} v_{NB} = \; & argmax_{v_j \in \{P,N\}} P(v_j)\,(P(Ambiente = soleado \mid v_j) \\ & P(Temperature = baja \mid v_j) P(Humedad = alta \mid v_j) \\ & P(Viento = si \mid v_j)) \end{aligned}$$

Tenemos que calcular la probabilidad *a priori* de cada clase: $P(Clase = P) = 9/14$ y $P(Clase = N) = 5/14$ y las probabilidades condicionales de cada atributo dada la clase usando los valores de los atributos del ejemplo:

$P(Viento = si \mid P) = 3/9 = 0.33$ y $P(Viento = si \mid N) = 3/5 = 0.60$

...

Entonces para la clase positiva: $P(P)P(soleado \mid P)P(baja \mid P)P(alta \mid P)P(si \mid P) = 0.0053$

y para la negativa: $P(N)P(soleado \mid N)P(baja \mid N)P(alta \mid N)P(si \mid N) = 0.0206$

Normalizando el último nos da: $\frac{0.0206}{0.0206+0.0053} = 0.795$.

Por lo que la clase más probable es la negativa.

4.3.1. Estimación de Probabilidades

Como ya vimos, podemos estimar probabilidades por frecuencia simple: $\frac{n_c}{n}$ simplemente contando el número de ejemplos con un valor particular entre el total de ejemplos. Esto sin embargo, no siempre da resultados adecuados, sobretodo cuando tenemos pocos datos.

Para estos casos podemos usar un estimador m (*m-estimate*):

$$\frac{n_c + m * p}{n + m}$$

donde p es una estimación *a priori* de lo que queremos estimar y m es una constante llamada "tamaño de muestra equivalente" (*equivalent sample size*). Una valor típico para p es suponer una distribución uniforme, por lo que: $p = \frac{1}{k}$ cuando existen k valores y m se usa como estimador de ruido.

Por ejemplo, vamos a suponer que queremos clasificar textos en diferentes categorías. Los ejemplos son textos asociados con una clase (e.g., me interesa

vs. no me interesa ó política, deportes, espectáculos, sociales, etc.). Suponemos que las palabras son independientes entre sí y de su posición en el texto (lo cual no es cierto, pero para clasificar textos no es tan relevante). *Vocabulario* = todas las palabras distintivas (eliminando palabras muy comunes y poco distintivas como artículos, puntuaciones, etc.).

Sea $doc(clase)$ = subconjunto de textos de esa clase, $P(clase) = \frac{|doc(clase)|}{Ejemplos}$, *Texto* = concatenación de todos los textos en $doc(clase)$, y n = número de palabras distintas en *Texto*. Para cada palabra (w) en *Vocabulario*: n_k = número de veces que aparece la palabra w en *Texto*.

Se calculan las probabilidades considerando el estimador m, $\frac{n_c+mp}{n+m}$ con probabilidad uniforme en las clases (Laplace) y $m = |Vocabulario|$. Entonces:

$$P(w|clase) = \frac{n_k + 1}{n + |Vocabulario|}$$

Para clasificar un nuevo documento (considerando sólo las palabras en el nuevo documento que tenemos en *Vocabulario*):

$$v_{NB} = argmax_{v_j \in V} \left(P(v_j) \prod_i P(a_i \mid v_j) \right)$$

Lo que estamos haciendo es multiplicando todas las palabras del documento considerando la probabilidad de que sea de cada clase. Como es de esperarse, las palabras comunes usadas en las distintas clases son suficientes para poder hacer una buena clasificación. Esto es, si en el texto aparece la palabra "gol", "marcador" y "partido" (entre muchas otras), lo más probable es que el texto pertenezca a la clase "deportes" y no a la clase "sociales". Con el estimador m podemos evitar tener ceros en la multiplicación con palabras que no aparecieron en el conjunto de entrenamiento.

4.4. Redes Bayesianas

Las redes bayesianas o probabilísticas son una representación gráfica de dependencias para razonamiento probabilístico (Pearl, 2000). Es un gráfo acíclico dirigido (DAG) en el cual cada nodo representa una variable aleatoria y cada arco una dependencia probabilística, en la cual se especifica la probabilidad condicional de cada variable dados sus padres. La variable a la que apunta el arco es dependiente de la que está en el origen de éste.

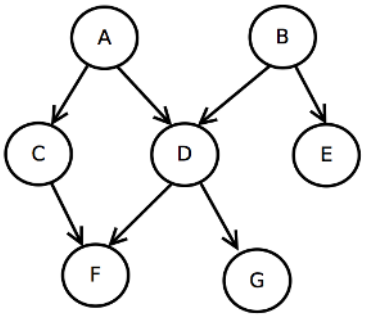


Figura 4.1 Ejemplo de una red Bayesiana.

Una Red Bayesiana representa la distribución de la probabilidad conjunta de las variables representadas en la red. Por ejemplo:

$$P(A, B, C, D, E, F, G) = P(G|D)P(F|C, D)P(E|B)P(D|A, B)P(C|A)P(B)P(A)$$

La topología o estructura de la red nos da información sobre las dependencias probabilísticas entre las variables (ver figura 4.1). La red también representa las independencias condicionales de una variable (o conjunto de variables) dada(s) otra(s) variable(s). Por ejemplo, *{E} es condicionalmente independiente de {A,C,D,F,G} dado {B}*.

Esto es: $P(E|A, C, D, F, G, B) = P(E|B)$. Esto se representa gráficamente por el nodo B separando al nodo E del resto de las variables.

En general, el conjunto de variables A es independiente del conjunto B dado C si al remover C hace que A y B se desconecten. Es decir, no existe una trayectoria entre A y B en que las siguientes condiciones sean verdaderas:

1. Todos los nodos con flechas convergentes están o tiene descendientes en C.

2. Todos los demás nodos están fuera de C.

Esto se conoce como *Separación–D*.

En una BN todas las relaciones de independencia condicional representadas en el grafo corresponden a relaciones de independencia de la distribución de probabilidad. Dichas independencias simplifican la representación del conocimiento (menos parámetros) y el razonamiento (propagación de las probabilidades).

4.4.1. Propagación de Probabilidades

El razonamiento probabilístico o propagación de probabilidades consiste en propagar la evidencia a través de la red para conocer la probabilidad *a posteriori* de las variables. La propagación consiste en darle valores a ciertas variables (evidencia), y obtener la probabilidad posterior de las demás variables dadas las variables conocidas (instanciadas). Los algoritmos de propagación dependen de la estructura de la red:

1. Árboles
2. Poli-árboles
3. Redes multiconectadas

Para entender más los conceptos de redes Bayesianas y cómo funcionan los mecanismos de propagación de probabilidades, se puede consultar el libro de Enrique Sucar sobre el tema (Sucar, 2015).

4.5. Aprendizaje de Redes Bayesianas

Las redes bayesianas son una alternativa para minería de datos, la cual tiene varias ventajas:

- Permiten aprender sobre relaciones de dependencia y causalidad.
- Permiten combinar conocimiento con datos.
- Evitan el sobre-ajuste de los datos.
- Pueden manejar bases de datos incompletas.

El obtener una red bayesiana a partir de datos es un proceso de aprendizaje el cual se divide, naturalmente, en dos aspectos:

1. **Aprendizaje paramétrico**: Dada una estructura, obtener las probabilidades *a priori* y condicionales requeridas.
2. **Aprendizaje estructural**: Obtener la estructura de la red Bayesiana, es decir, las relaciones de dependencia e independencia entre las variables involucradas.

4.5.1. Aprendizaje Paramétrico

El aprendizaje paramétrico, como lo indica su nombre, consiste en encontrar los parámetros asociados a una estructura dada de una red bayesiana. Ésto es, las probabilidades *a priori* de los nodos raíz y las probabilidades condicionales de las demás variables, dados sus padres. Para que se actualicen las probabilidades con cada caso observado, éstas se pueden representar como razones enteras y actualizarse con cada observación.

Para las probabilidades previas se puede hacer lo siguiente:

$$\begin{array}{ll} P(A_i) = (a_i + 1)/(s + 1) & i = k \\ P(A_i) = a_i/(s + 1) & i \neq k \end{array}$$

Esto es, actualizar con cada nuevo dato la probabilidad de cada variable padre dependiente del valor de ese dato para esa variable. De forma similar para las probabilidades condicionales, podemos actualizar las probabilidades con cada valor que tome la variable en el ejemplo.

$$\begin{array}{ll} P(B_j \mid A_i) = (b_j + 1)/(a_i + 1) & i = k \text{ y } j = l \\ P(B_j \mid A_i) = b_j/(a_i + 1) & i = k \text{ y } j \neq l \\ P(B_j \mid A_i) = b_j/a_i & i \neq k \end{array}$$

Donde s corresponde al número de casos totales, i, j los índices de las variables, k, l los índices de las variables observadas.

En algunos casos, existen variables que son importantes para el modelo pero para las cuales no se tienen datos (nodos no observables o *escondidos*). Si algunos nodos son parcialmente observables, se pueden estimar de acuerdo a los observables con el siguiente algoritmo:

1. Instanciar todas las variables observables.
2. Propagar su efecto y obtener las probabilidades posteriores de las no observables.
3. Para las variables no observables, *suponer* el valor con probabilidad mayor como observado.
4. Actualizar las probabilidades previas y condicionales de acuerdo a las formulas anteriores.
5. Repetir 1 a 4 para cada observación.

Existen otras formas más sofisticadas en las que las probabilidades se actualizan dando incrementos no sólo al valor mayor o más probable, sino a todos en proporción de las probabilidades posteriores. El aprendizaje se basa en el gradiente, lo cual es análogo al aprendizaje del peso en capas ocultas de redes neuronales. En este caso, se busca maximizar $P(D \mid h)$ siguiendo el gradiente del $ln(P(D \mid h))$ con respecto a los parámetros que definen las tablas de probabilidad condicional. Estos algoritmos suponen que se tienen algunos datos, a partir de los cuales es posible estimar una probabilidad (aunque por tener pocos datos se tenga que ajustar). Cuando no se tiene ningún valor para un dato, se puede usar el algoritmo EM o de *Expectation Maximization*, el cual se describe en el capítulo de Agrupamiento o *Clustering*.

4.5.2. Aprendizaje Estructural

Al igual que los mecanismos de razonamiento, las técnicas de aprendizaje estructural dependen del tipo de estructura de red y se dividen naturalmente en árboles, poliárboles y redes multiconectadas. Dentro de todos estos, una alternativa es combinar conocimiento subjetivo del experto con aprendizaje. Esto es, partir de la estructura dada por el experto, y validarla/mejorarla utilizando datos estadísticos.

Para aprender la estructura de la red Bayesiana, en el caso de **Naïve Bayes** sólo tenemos que aprender los parámetros, dado que la estructura ya está dada. Una forma de mejorar la estructura de un NB es añadiendo arcos entre los nodos o atributos que tengan cierta dependencia. Existen dos estructuras básicas:

1. TAN: Clasificador bayesiano simple aumentado con un árbol entre los atributos.

2. BAN: Clasificador bayesiano simple aumentado con una red entre los atributos.

Otra forma de aprender estructuras es realizando operaciones o modificaciones locales, medirlas y continuar hasta que no mejore la predicción. Dentro de las operaciones locales se puede hacer:

1. Eliminar un atributo,

2. Unir dos atributos en una nueva variable combinada,

3. Introducir un nuevo atributo que haga que dos atributos dependientes sean independientes (nodo oculto).

Se pueden ir probando cada una de las opciones anteriores midiendo la dependencia de los atributos dada la clase:

$$I(X_i, X_j \mid C) = \sum_{X_i, X_j} P(X_i, X_j \mid C) log(\frac{P(X_i, X_j \mid C)}{P(X_i \mid C)P(X_j \mid C)})$$

Podemos entonces pensar en un algoritmo de Mejora Estructural:

1. Obtener la información mutua condicional (IMC) entre cada par de atributos.

2. Seleccionar el par de atributos de IMC mayor.

3. Probar las 3 operaciones básicas (i) eliminación, (ii) unión, (iii) inserción.

4. Evaluar las 3 estructuras alternativas y la original, y quedarse con la "mejor" opción.

5. Repetir 2–4 hasta que ya no mejore el clasificador.

Para evaluar las estructuras resultantes se pueden usar datos de prueba o una medida basada en MDL.

Algoritmo de Chow y Liu

Uno de los algoritmos más utilizados de aprendizaje estructural de árboles está basado en el algoritmo desarrollado por Chow y Liu (Chow, 1968) para aproximar una distribución de probabilidad por un producto de probabilidades de segundo orden. La probabilidad conjunta de n variables se puede representar (aproximar) como:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i)P(X_i \mid X_{j(i)}))$$

donde $X_{j(i)}$ es la causa o padre de X_i.

Con esto se puede diseñar un algoritmo en donde la representación de una distribución conjunto de probabilidades se plantea como uno de optimización

donde lo que se desea es obtener la estructura en forma de árbol que más se aproxime a la distribución "real". Para esto, se utiliza una medida de la diferencia de información entre la distribución real (P) y la aproximada (P^*):

$$I(P, P^*) = \sum_x P(\mathbf{X}) log(\frac{P(\mathbf{X})}{P^*(\mathbf{X})})$$

donde el objetivo es minimizar I.

Se puede definir dicha diferencia en función de la información mutua entre pares de variables, que se define como:

$$I(X_i, X_j) = \sum_x P(X_i, X_j) log(\frac{P(X_i, X_j)}{P(X_i)P(X_j)})$$

Se puede demostrar (Chow 68) que la diferencia de información es una función del negativo de la suma de las informaciones mutuas (pesos) de todos los pares de variables que consituyen el árbol. Por lo que encontrar el árbol más próximo equivale a encontrar el árbol con mayor peso.

El algoritmo, es entonces relativamente sencillo:

1. Calcular la información mutua entre todos los pares de variables ($n(n-1)/2$).

2. Ordenar las informaciones mutuas de mayor a menor

3. Seleccionar la rama de mayor valor como árbol inicial

4. Agregar la siguiente rama mientras no forme un ciclo, si es así, desechar

5. Repetir (4) hasta que se cubran todas las variables ($n-1$ ramas)

El algoritmo no provee la direccionalidad de los arcos, por lo que ésta se puede asignar en forma arbitraria o utilizando semántica externa (experto).

En un ejemplo hipotéticos, supongamos que tenemos cinco atributos y queremos obtener la mejor distribución de segundo orden que representa la distribución conjunta de todas las variables. Lo que hacemos es calcular la información mutua entre todos los pares de variables y los ordenamos. Por

ejemplo:

No.	Var 1	Var 2	Info. mutua
1	A	B	.2856
2	E	B	.0743
3	E	C	.0456
4	C	B	.0074
5	E	D	.0060
6	D	A	.0052
7	D	B	.0017
8	E	A	.0003
9	C	A	0
10	D	C	0

Con estos resultados empezamos creando una liga entre A y B. Después entre E y B. Después entre E y C. El siguiente par incolucra a las variables C y B, pero si las incluimos formamos un ciclo, por lo que lo ignoramos. Finalmente añadimos una ligar entre E y D. Con ésto ya incluimos a todas las variables y podemos terminar el proceso. Después se tienen que asignar las direcciones de todos los arcos.

Rebane y Pearl (Rebane G., 1987) extendieron el algoritmo de Chow y Liu para poliárboles. Parten del esqueleto obtenido con Chow y Liu y determinan las direcciones de los arcos utilizando pruebas de dependencia entre tripletas de variables. Dadas 3 variables, existen 3 casos posibles:

1. Arcos divergentes: $X \leftarrow Y \rightarrow Z$.
2. Arcos secuenciales: $X \rightarrow Y \rightarrow Z$.
3. Arcos convergentes: $X \rightarrow Y \leftarrow Z$.

Los primeros dos casos son indistinguibles, pero el tercero es diferente, ya que las dos variables "padre" son marginalmente independientes. Con esta observación se define un algoritmo para crear poliárboles que sigue los siguientes pasos:

1. Obtener el esqueleto utilizando Chow y Liu
2. Encontrar tripletas de nodos que sean convergentes (tercer caso) -nodo multipadre-.

3. A partir de un nodo multipadre determinar las direcciones de los arcos utilizando la prueba de tripletas hasta donde sea posible (base causal).

4. Repetir 2-3 hasta que ya no se puedan descubrir más direcciones

5. Si quedan arcos sin direccionar utilizar semántica externa para obtener su dirección

Hay que aclarar que este algotimo es sólo para poliárboles, no garantiza obtener todas las direcciones y requiere la definición de un umbral.

Existen dos clases de métodos para el aprendizaje genérico de redes bayesianas, que incluyen redes multiconectadas. éstos son:

1. Métodos basados en medidas de ajuste y búsqueda.

2. Métodos basados en pruebas de independencia.

4.5.3. Aprendizaje de Redes Basados en Búsqueda

El aprendizaje de redes Bayesianas basadas en búsqueda, en general, generan diferentes estructuras y las evalúan respecto a los datos utilizando alguna medida de ajuste. Estos métodos tienen dos aspectos principales:

1. Una medida para evaluar qué tan *buena* es cada estructura respecto a los datos (e.g., usando alguna medida como BIC o MDL entre otras).

2. Un método de búsqueda que genere diferentes estructuras hasta encontrar la *óptima*, de acuerdo a la medida seleccionada.

Una posible medida para evaluar una estructura es el criterio de información bayesiano o BIC (Schwarz et al., 1978) que estima la probabilidad de la estrutura dado los datos la cual se trata de maximizar. Lo que busca es maximizar la probabilidad de la estructura dados los datos, esto es:

$$P(Es \mid D)$$

Donde Es es la estructura y D son los datos. La medida la podemos escribir en términos relativos al comparar dos estructuras, i y j como:

$$P(Es_i \mid D)/P(Es_j \mid D) = P(Es_i, D)/P(Es_j, D)$$

Considerando variables discretas y que los datos son independientes entre sí, las estructuras se pueden comparar en función del número de ocurrencias (frecuencia) de los datos predichos por cada estructura.

Otra medida popular, es la basada en el principio de longitud de descripción mínima o MDL. Esta medida estima la longitud (tamaño en bits) requerida para representar la probabilidad conjunta con cierta estructura, la cual se compone de dos partes:

1. Representación de la estructura,

2. Representación del error de la estructura respecto a los datos

Hace un compromiso entre exactitud y complejidad. La exactitud se estima midiendo la información mutua entre los atributos y la clase; y la complejidad contando el número de parámetros.

Se utiliza una constante, α, en [0, 1] para balancear exactitud contra complejidad. La medida de calidad está dada por:

$$MC = \alpha(W/Wmax) + (1 - \alpha)(1 - L/Lmax)$$

Donde W y $Wmax$ representa la exactitud y máxima exactitud, y L y $Lmax$ representan la complejidad y máxima complejidad del modelo. Para determinar estos máximos normalmente se considera un máximo en cuanto al número de padres máximo permitido por nodo.

La complejidad está dada por el número de parámetros requeridos para representar el modelo. Se puede calcular como:

$$L = S_i[k_i log_2 n + d(S_i - 1)F_i]$$

Donde, n es el número de nodos, k es el número de padres por nodo, S_i es el número de valores promedio por variable, F_i el número de valores promedio de los padres, y d el número de bits por parámetro.

La exactitud se puede estimar con base en el "peso" de cada nodo y el peso de cada nodo se estima con la información mutua con sus padres:

$$w(xi, Fxi) = \sum_{xi} P(xi, Fxi) log[P(xi, Fxi)/P(xi)P(Fxi)]$$

El peso (exactitud) total está dado por la suma de los pesos de cada nodo:

$$W = \sum_{i} w(xi, Fxi)$$

Para construir una red usando la medida MDL (o cualquier otra) se puede hacer mediante un algoritmo *hill-climbing*, iniciando con una estructura simple, por ejemplo un árbol construido con Chow-Liu (o compleja – altamente conectada), agregando (o eliminando) ligas que mejoren la medida de calidad hasta alcanzar un mínimo local. El algoritmo podría seguir los siguientes pasos:

1. Generar estructura incial - árbol (o multiconectada)
2. Calcular medida de calidad de la estructura inicial
3. Agregar (eliminar) / invertir un arco en la estructura actual
4. Calcular medida de calidad de nueva estructura
5. Si se mejora la calidad conservar el cambio, si no dejar la estructura anterior
6. Repetir 3 a 5 hasta que ya no haya mejoras.

También se pueden combinar los dos enfoques, partir de una escructura simple y de una compleja y realizar una búsqueda bi-direccional.

4.5.4. Métodos Basados en Pruebas de Independiencia

Los métodos basados en pruebas de independencia usan medidas de dependencia local entre subconjuntos de variables. El caso más sencillo de este tipo de algoritmos es el algoritmo de Chow y Liu (información mutua entre pares de variables). En general, se hacen pruebas de dependencia entre subconjuntos de variables, normalmente dos o tres variables. La desventaja es que pueden generarse muchos arcos "innecesarios", por lo que se incorporan formas para luego eliminar arcos. Hay diferentes variantes de este enfoque que consideran diferentes medidas de dependencia y diferentes estrategias para eliminar arcos innecesarios.

Posiblemente el algoritmo más popular es el de PC (Spirtes and Glymour, 1991). PC obtiene el esqueleto (grafo no dirigido) y después las orientaciones de los arcos. Para el esqueleto empieza con un grafo no dirigido completamente conectado y determina la independencia condicional de cada par de variables dado un subconjunto de otras variables $I(X, Y \mid \mathbf{S})$. Se puede obtener con una medida de entropia condicional cruzada y si el valor es menor a

un umbral se elimina el arco. La dirección se obtiene buscando estructuras de la forma $X - Z - Y$ sin arco en $X - Y$. Si X, Y no son independientes dado Z, orienta los arcos creando una estructura "V": $X \rightarrow Z \leftarrow Y$. Al terminar trata de otientar el resto basado en pruebas de independencia y evitando ciclos. El proceso viene descrito en el algoritmo 1.

Algorithm 1 Algoritmo PC para construir redes Bayesianas

Require: Set of variables **X**, Independence test I

Initialize a complete undirected graph G'
i=0
repeat
  for $X \in$ **X do**
    for $Y \in ADJ(X)$ **do**
      for $S \subseteq ADJ(X) - \{Y\}, \mid S \mid = i$ **do**
        if $I(X, Y \mid S)$ **then**
          Remove the edge $X - Y$ from G'
        end if
      end for
    end for
  end for
  i=i + 1
until $\mid ADJ(X) \mid \leq i, \forall X$
Orient edges in G'

Otro algoritmo muy popular por su simplicidad es el algoritmo K2 (Cooper and Herskovits, 1992). Como todos los algoritmos busca de manera heurística la estructura más probable dados los datos. Una desventaja es que requiere que todas las variables estén ordenadas lo cual le impone ciertas restricciones y la necesidad de conocer este orden de antemano. El algoritmo empieza con una variable suponiendo que no tiene padres y añade aquellos, siguiendo el orden de las variables, que al incluirlos como padres aumentan una medida (ver abajo). Después sigue con las siguientes variables dadas por el orden. El proceso viene descrito en el algoritmo 2.

La fórmula para decidir si se añade un padre o no es:

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} \alpha_{ijk}!$$

donde:

Algorithm 2 Algoritmo K2.

Require: Conjunto n de variables ordenadas y número máximo de padres por nodo (u)

for $i := 1$ hasta n **do**
  $\pi_i = \emptyset$
  $P_{old} := f(i, \pi_i)$ {Ver fórmula}
  $Bandera := true$
  while $Bandera$ and $|\pi_i| < u$ **do**
    sea $z \in Pred(x_i) - \pi_i$ que maximice $f(i, \pi_i \cup \{z\})$
    $P_{new} := f(i, \pi_i \cup \{z\})$
    if $P_{new} > P_{old}$ **then**
      $P_{old} := P_{new}$; $\pi_i := \pi_i \cup \{z\}$
    else
      $Bandera := false$
    end if
  end while
  escribe: 'Los padres de: ' x_i 'son:' π_i
end for

- π_i = padres de x_i
- $q_i = |\phi_i|$
- ϕ_i = posibles instanciaciones de los padres de x_i
- $r_i = |V_i|$
- V_i = posibles valores de x_i
- α_{ijk} = número de ejemplos en que el nodo x_i tiene el valor k y sus padres tienen el valor j
- $N_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$ = ejemplos donde los padres de x_i toman valores j

El encontrar la estructura óptima es difícil, ya que el espacio de búsqueda es muy grande (e.g., más de 10^{40} estructuras para 10 variables). Una alternativa es combinar conocimiento de expertos con datos, donde el experto da un estructura inicial la cual es validada o modificado por la información que nos proporcionan los datos. Otra alternativa es obtener una estructura inicial a

partir de datos y luego utilizar conocimiento del experto. Finalmente se puede hacer *transfer learning* (utilizar lo aprendido en un dominio parecido para facilitar el aprendizaje en el dominio actual).

4.6. Conclusiones

La incertidumbre es una parte intrínseca dentro de los algoritmos de aprendizaje y el aprendizaje Bayesiano considera a la incertidumbre dentro de su proceso de inducción. Una de las herramientas más utilizadas para representar conocimiento con incertidumbre son las redes Bayesianas. Estas permiten hacer inferencias probabilistas considerando la evidencia con que se cuente. Esto quiere decir que se pueden predecir los valores más probables de cualquier subconjunto de variables dados los valores de las otras variables. Esta es una flexibilidad poco común en los sistemas de aprendizaje en donde generalmente los modelos son unidireccionales: Dado un conjunto de entradas fijas, regresa los valores de las salidas.

En este capítulo vimos algunos aspectos del aprendizaje Bayesiana. Cómo éste se puede usar para buscar la hipótesis más probable dados los datos. También vimos algunos de los algoritmos principales para aprender redes Bayesianas.

Referencias

Chow, C. K.; Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467.

Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347.

Good, I. J., Hacking, I., Jeffrey, R., and Törnebohm, H. (1966). The estimation of probabilities: An essay on modern bayesian methods.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 2:203–226.

Mitchell, T. M. (1997). *Machine Learning: Capítulo 6.* McGraw Hill.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Rebane G., P. J. (1987). The recovery of causal poly-trees from statistical data. In *Proceedings, 3rd Workshop on Uncertainty in AI*, pages 222–228, Seattle, WA.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–658.

Schwarz, G. et al. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

Spirtes, P. and Glymour, C. (1991). An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72.

Sucar, L. E. (2015). Probabilistic graphical models. *Advances in Computer Vision and Pattern Recognition. London: Springer London. doi*, 10:978–1.

Capítulo 5

Aprendizaje por Refuerzo

E. Morales, H. Escalante-Balderas
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

5.1. Introducción

Uno de los enfoques más usados dentro de aprendizaje computacional es el aprendizaje supervisado a partir de ejemplos (pares entradas – salidas provistos por un usuario o por el medio ambiente), para después poder predecir la salida de nuevas entradas. Cualquier sistema de predicción puede verse dentro de este paradigma, sin embargo, ignora la estructura secuencial del mismo. En algunos ambientes, muchas veces se puede obtener sólo cierta retroalimentación o recompensa o refuerzo (e.g., gana, pierde).

El refuerzo puede darse en un estado terminal y/o en estados intermedios. Los refuerzos pueden ser componentes o sugerencias de la utilidad actual a maximizar (e.g., buena movida). En aprendizaje por refuerzo (RL) el objetivo es aprender cómo mapear situaciones a acciones para maximizar una cierta señal de recompensa (Sutton et al., 1998). Una de las promesas con este tipo de aprendizaje es: *programar agentes mediante premio y castigo sin necesidad de especificar cómo realizar la tarea.*

En comparación con otros tipos de técnicas de Aprendizaje Computacional en Aprendizaje por Refuerzo (RL):

1. No se le presentan pares entrada - salida.
2. El agente tiene que obtener experiencia útil acerca de los estados, acciones, transiciones y recompensas de manera activa para poder actuar

de manera óptima.

3. La evaluación del sistema ocurre en forma concurrente con el aprendizaje.

5.1.1. Aplicaciones

Existe una gran cantidad de aplicaciones que han usado RL. Aquí hacemos un breve recuento histórico de los principales desarrollos.

RL se empezó a utilizar desde los inicions de la computación. La primera aplicación en aprendizaje por refuerzo fué el programa para jugar damas de Arthur Samuel (Samuel, 1959). En ese trabajo se usó una función lineal de evaluación con pesos usando hasta 16 términos. Su programa era parecido a la ecuación de actualización de pesos que se usó en los primeros sistemas de redes neuronales, pero no usaba recompensa en los estados terminales, lo que hace que puede o no converger y puede aprender a perder. Samuel logró evitar ésto haciendo que el peso para ganancia de material fuera siempre positivo.

Una de las más aplicaciones más conocidas es el control del péndulo invertido. Controlar la posición x para que se mantenga aproximadamente derecho ($\theta \approx \pi/2$), manteniendose en los límites de la pista. $X, \theta, \dot{X}$ y $\dot{\theta}$ son continuas. El control es de tipo bang–bang. Boxes (Michie and Chambers, 1968) balanceaba el pendulo por más de una hora después de 30 intentos (no simulado). Se discretizó el espacio en "cajas". Se corría el sistema hasta que se caía el péndulo o se salía de los límites. Entonces se daba un refuerzo negativo a la última "caja" y se propagaba a la secuencia de "cajas" por las que había pasado.

Una de las aplicaciones que hizo mucho ruido en etapas más recientes fue el sistema que aprendió a jugar backgammon de Gerry Tesauro. TD-gammon (Tesauro, 1995) representó una función de evaluación con una red neuronal de una sola capa intermedia con 40 nodos, que después de 200,000 juegos de entrenamiento mejoró notablemente su desempeño. Para mejorar el desempeño añadió atributos adicionales a una red con 80 nodos escondidos, que después de 300,000 juegos de entrenamiento, logró jugar como los 3 mejores jugadores del mundo.

Un poco después se desarrolló un algoritmo de RL que actualiza las funciones de evaluación en un árbol de búsqueda en juegos. En ajedrez logró mejorar el puntaje de un programa de 1,650 ELO a 2,150 ELO después de

308 juegos en 3 días.

Dentro de las aplicaciones recientes podemos mencionar a Watson de IBM, quien se comvirtió en campeón en el juego de Jeopardy en 2011 (High, 2012). En este sistema se usó RL para aprender una función de valor que se usó para generar "apuestas" y poder ganar más puntos durante el juego.

Combinando RL con técnicas de Redes Neuronales Profundas o *Deep Learning*, se han logrado resultados muy interesantes. En una aplicación se entrenó un sistema de Deep Learning, Atari 2600, que aprendió como jugar 46 video juegos, superando en 29 de ellos a expertos humanos (Mnih et al., 2013).

Más recientemente se hizo un sistema que logró vencer al campeón mundial de Go (AlphaGo (Silver et al., 2016)). Este sistema se refinó para poder aprender jugando contra sí mismo (AlphaGo Zero (Silver et al., 2017)). En la última sección describimos con más detalles estos sistemas recientes.

5.2. Aprendizaje por Refuerzo

En RL un agente trata de aprender un comportamiento mediante interacciones de prueba y error en un ambiente dinámico e incierto. En general, al sistema no se le dice qué acción debe tomar, sino que él debe de descubrir qué acciones dan el máximo beneficio. En un RL estandar, un agente está conectado a un ambiente por medio de percepción y acción. En cada interacción el agente recibe como entrada una indicación de su estado actual ($s \in S$) y selecciona una acción ($a \in A$). La acción (posiblemente) cambia el estado y el agente recibe una señal de refuerzo o recompensa ($r \in \mathcal{R}$). Ver figura 5.1.

El comportamiento del agente debe de ser tal que seleccione acciones que tiendan a incrementar a largo plazo la suma de las recompensas totales. El objetivo del agente es encontrar una política (π, que mapea estados a acciones) que maximice a largo plazo el refuerzo acumulado. En general el ambiente es no-determinístico, por lo que al tomar la misma acción en el mismo estado puede dar resultados diferentes. Sin embargo, se supone que el ambiente es estacionario (las probabilidades de cambio de estado no cambian con el tiempo o cambian muy lentamente).

La figura 5.2 muestra un ejemplo en donde un robot tiene varias posibles acciones a realizar (izquierda, derecha, arriba y abajo). Las transiciones no son determinístas, como se ilustra en la parte superior, en donde ir a la derecha tiene un 80 % de posibilidad de pasar al siguiente estado y un 20 %

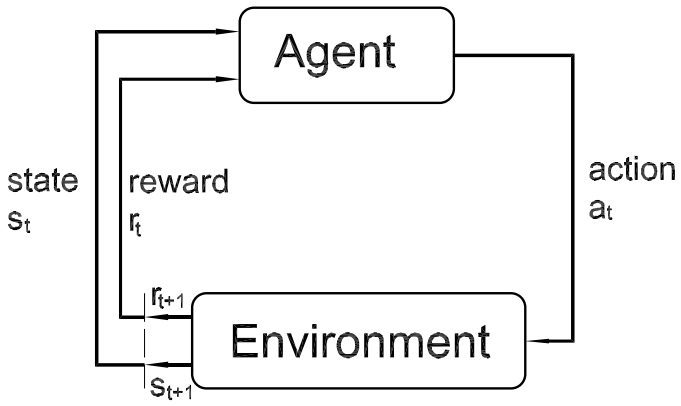


Figura 5.1 Aprendizaje por Refuerzo.

de quedarse en el mismo estado. El ambiente tiene estados por los que no se puede transitar (e.g., obstáculos) y estados que proporcionan premios o castigos.

Algunos aspectos que hay que considerar es que: (i) se sigue un proceso de prueba y error y (ii) la recompensa puede estar diferida. En el proceso de aprendizaje se tiene que hacer un balance entre exploración y explotación. Para obtener buena ganancia uno prefiere seguir ciertas acciones, pero para saber cuáles, se tiene que hacer cierta exploración. En general, el agente que está aprendiendo empieza haciendo un proceso más de exploración y con el tiempo se realiza más explotación. Muchas veces el balance entre exploración y explotación depende de cuánto tiempo se espera que el agente interactue con el medio ambiente.

5.2.1. Procesos de Decisión de Markov

En RL se tiene que decidir en cada estado la acción a realizar. Este proceso de decisión secuenial se puede caracterizar como un proceso de decisión de Markov o MDP. Un MDP modela un problema de decisión secuencial en donde el sistema evoluciona en el tiempo y es controlado por un agente. La dinámica del sistema está determinada por una función de transición de probabilidad que mapea estados y acciones a otros estados.

Formalmente, un MDP es una tupla $M =< S, A, \Phi, R >$ formada por:

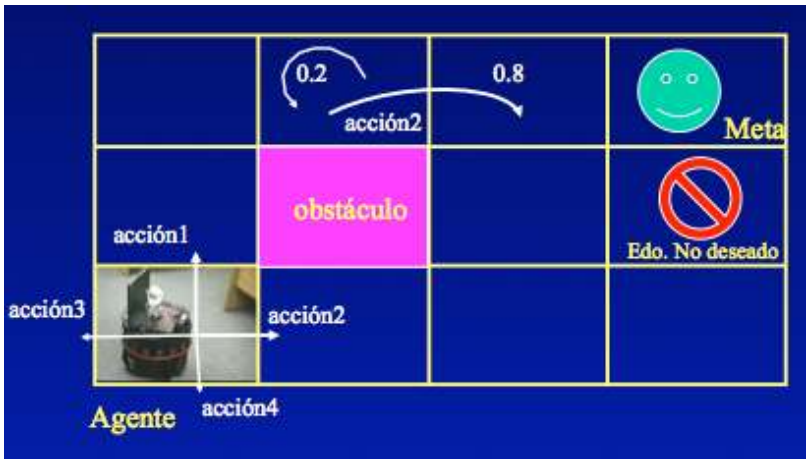


Figura 5.2 Ejemplo de problema.

- Un conjunto finito de estados S, $(s_i \in S, i = \{1, \ldots, n\})$
- Un conjunto finito de acciones A, que pueden depender de cada estado $(a_j(s_i), j = \{1, \ldots, m\})$
- Una función de recompensa (R), que define la meta y mapea cada estado–acción a un número (recompensa), indicando lo deseable del estado $(f(s, a) \Rightarrow \mathcal{R})$
- Un modelo del ambiente o función de transición de estados $\Phi(s'|s, a)$ $(\Phi : A \times S \rightarrow S)$ que nos dice la probabilidad de alcanzar el estado $s' \in S$ al realizar la acción $a \in A$ en el estado $s \in S$

Adicionalmente, en un problema de Aprendizaje por Refuerzo, se tenen que considerar los siguientes elementos Adicionales:

- Política (π): define cómo se comporta el sistema en cierto tiempo. Es un mapeo (a veces estocástico) de los estados a las acciones $(\pi(S) \rightarrow A)$
- Función de valor (V): indica lo que es bueno a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en un estado s $(V(s))$ o en un estado haciendo una acción a $(Q(s, a))$
- Las recompensas están dadas por el ambiente, pero los valores se deben de estimar (aprender) con base en las observaciones

Lo que hace RL es aprender las funciones de valor o la política mientras interactua con el ambiente.

5.2.2. Modelos de Recompensas

Se pueden definir diferentes modelos de recompensans. Dado un estado $s_t \in S$ y una acción $a_t \in \mathcal{A}(s_t)$, el agente recibe una recompensa r_{t+1} y se mueve a un nuevo estado s_{t+1}. Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ..., lo que queremos es maximizar lo que esperamos recibir de recompensa total acumulada (R_t). Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *contínuas*.

Existen diferentes modelos de recompensas:

- **Horizonte finito**: el agente trata de optimizar su recompensa esperada en los siguientes h pasos, sin preocuparse de lo que ocurra despues:

$$E(\sum_{t=0}^{h} r_t)$$

- Se puede usar como:
  - *política no estacionaria*: en el primer paso se toman los h siguientes pasos, en el siguiente los $h-1$, etc., hasta terminar. El problema principal es que no siempre se conoce cuántos pasos considerar
  - *receeding-horizon control*: siempre se toman los siguientes h pasos

- **Horizonte infinito** (la más utilizada): las recompensas que recibe un agente son reducidas geométricamente de acuerdo a un factor de descuento γ ($0 \leq \gamma \leq 1$):

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots = \sum_{t=0}^{\infty} \gamma^k r_t$$

  donde γ se conoce como la *razón de descuento* y lo que queremos maximizar es la recompensa total esperada:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

- **Recompensa promedio**: optimizar a largo plazo la recompensa promedio:

$$lim_{h\to\infty}E(\frac{1}{h}\sum_{t=0}^{h}r_t)$$

  El problema es que este equema no distingue políticas que reciban grandes recompensas al principio de las que no.

Los algoritmos de RL suponen que se cumple con la propiedad Markoviana (las transiciones de estado sólo dependen del estado actual) y las probabilidades de transición están dadas por:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

Entonces el valor de recompensa esperado es:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

Lo que se busca es estimar las funciones de valor. Esto es, qué tan bueno es estar en un estado (o realizar una acción). La noción de "qué tan bueno" se define en términos de recompensas futuras o recompensas esperadas.

5.2.3. Funciones de Valor

La política π es un mapeo de cada estado $s \in S$ y acción $a \in \mathcal{A}(s)$ a la probabilidad $\pi(s, a)$ de tomar la acción a estando en el estado s. El valor de un estado s bajo la política π, denotado como $V^\pi(s)$, es la recompensa total esperada estando en el estado s y siguiendo la política π:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=o}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\}$$

El valor esperado tomando una acción a en estado s bajo la política π ($Q^\pi(s, a)$):

$$\begin{aligned} Q^\pi(s, a) &= E_\pi\{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi\left\{\textstyle\sum_{k=o}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \end{aligned}$$

Las funciones de valor óptimas se definen como:

$$V^*(s) = max_\pi V^\pi(s) \text{ y } Q^*(s, a) = max_\pi Q^\pi(s, a)$$

Las cuales se pueden expresar como las ecuaciones de optimalidad de Bellman:

$$V^*(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \text{max}_{a'} Q^*(s',a')]$$

5.3. Métodos de Solución

Existen tres métodos principales de resolver MDPs:

1. Programación Dinámica
2. Monte Carlo, y
3. Diferencias Temporales o de Aprendizaje por Refuerzo

Vamos a ver brevemente cada uno de ellos.

5.3.1. Programación Dinámica

Si se conoce el modelo del ambiente, o sea, las transiciones de probabilidad ($\mathcal{P}^a_{ss'}$) y los valores esperados de recompensas ($\mathcal{R}^a_{ss'}$), las ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incognitas.

Para resolver este sistema, se puede hacer de forma incremental por medio de aproximaciones sucesivas. Consideremos primero como calcular la función de valor V^π dada una política arbitraria π. Esto se puede realizar de la siguiente forma:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \\ &= \textstyle\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en estado s bajo la política π.

Podemos hacer aproximaciones sucesivas, evaluando $V_{k+1}(s)$ en términos de $V_k(s)$.

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Con esto podemos entonces definir un algoritmo de evaluación iterativa de políticas, de la siguiente forma:

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa $V \approx V^\pi$

Lo que en realidad se busca es calcular la función de valor de una política para tratar de encontrar mejores políticas. Dada una función de valor, podemos probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^\pi(s)$. En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de cada estado, seleccionando aquella que parezca mejor de acuerdo a una política *greedy*. Entonces podemos calcular una nueva política $\pi'(s) = \text{argmax}_a Q^\pi(s,a)$ y continuar hasta que no mejoremos.

Esto sugiere, partir de una política (π_0) y calcular la función de valor (V^{π_0}), con la cual buscamos encontrar una mejor política (π_1) y así sucesivamente hasta converger a π^* y V^*. A este procedimiento se llama iteración de políticas y viene descrito a continuación:

$V(s) \in \mathcal{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in S$ (**Inicializa**)
Repite (**Evaluación de Política**)
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_{s'} \mathcal{P}^{\pi(s)}_{ss'}[\mathcal{R}^{\pi(s)}_{ss'} + \gamma V(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)

pol-estable $\leftarrow$ true (**Mejora de Política**)
Para cada $s \in S$:
$b \leftarrow \pi(s)$
$\pi(s) \leftarrow \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
if $b \neq \pi$, then *pol-estable* $\leftarrow$ false
If *pol-estable*, then stop, else evalúa nva. política

Iteración de políticas en cada iteración evalúa la política y requiere recorrer todos los estados varias veces. El paso de evaluación de política lo podemos truncar sin perder la garantía de convergencia, después de recorrer una sola vez todos los estados. A esta forma se le llama iteración de valor (*value iteration*) y se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1}(s) = max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]$$

Esta esquema se puede ver como expresar la ecuación de Bellman en una regla de actualización. Con esto podemos definir el algoritmo de interación de valor como sigue:

Inicializa $V(s) = 0$ para toda $s \in S$
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) \leftarrow max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$
$\Delta \leftarrow max(\Delta, |v - V(s)|)$
Hasta que $\Delta < \theta$ (número positivo pequeño)
Regresa una política determinística tal que:
$\pi(s) = argmax_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$

En caso de conocer el modelo del ambiente lo más recomendable es usar programación dinámica, debido a que los métodos que no tienen esta información son mucho más tardados.

5.3.2. Monte Carlo

Los métodos de Monte Carlo, sólo requieren de experiencia y la actualización de la función de valor se hace por episodio en lugar de hacerla en cada paso.

El valor de un estado es la recompensa esperada que se puede obtener a partir de ese estado. Para estimar V^π y Q^π podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas.

Por ejemplo, podemos usar un método Monte Carlo para estimar V^π:

Repite
Genera un episodio usando π
Para cada estado s en ese episodio:
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añade R a $recomp(s)$
    $V(s) \leftarrow$ promedio($recomp(s)$)

Si queremos estimar pares estado-acción (Q^π) corremos el peligro de no ver todos los pares (a pesar de visitar todos los estados, no necesariamente se visitan todas las acciones posibles de cada estado), por lo que se busca mantener la exploración. Lo que normalmente se hace es considerar sólo políticas estocásticas que tengan una probabilidad diferente de cero de seleccionar todas las acciones.

Para poder mejorar las políticas aprendidas con Monte Carlo, podemos alternar entre evaluación y mejoras con base en cada episodio. La idea es que después de cada episodio las recompensas observadas se usan para evaluar la política y la política se mejora para todos los estados visitados en el episodio.

Un algoritmo basaso en Monte Carlo para mejorar políticas es el siguiente:

Repite
Genera un episodio usando π con exploración
Para cada par (s, a) en ese episodio:
    $R \leftarrow$ recompensa después de la primera
        ocurrencia de (s, a)
    Añade R a $recomp(s, a)$
    $Q(s, a) \leftarrow$ promedio($recomp(s, a)$)
Para cada s en el episodio:
    $\pi(s) \leftarrow \text{argmax}_a Q(s, a)$

Monte Carlo se puede utilizar cuando no se tiene un modelo del ambiente y se pueden hacer muchas simulaciones. Sin embargo, las actualizaciones se hacen después de cada episodio.

5.3.3. Diferencias Temporales

Los métodos de diferencias temporales (TD) combinan las ventajas de los dos métodos anteriores: permiten hacer *bootstrapping* - estimar valores con base en otras estimaciones - (como DP) y no requieren tener un modelo del ambiente (como MC). Métodos tipo TD sólo tienen que esperar el siguiente paso para actualizar su estimación. Los métodos de TD usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida final), aprendiendo al existir cambios entre predicciones sucesivas.

Como no tienen un modelo del ambiente, no pueden simplemente propagar como lo hacen los métodos de programación dinámica. Los métodos de TD requieren hacer exploración en el ambiente, para poder aprender la mejor acción a realizar en cada estado. Existen diferentes esquemas de exploración. Los dos más usados son:

1. $\epsilon-$*greedy*: La mayor parte del tiempo se selecciona la acción que da el mayor valor estimado, pero con probabilidad ϵ se selecciona una acción aleatoriamente.

2. *softmax*: La probabilidad de selección de cada acción depende de su valor estimado. La más común sigue una distribución de Boltzmann o de Gibbs, y selecciona una acción con la siguiente probabilidad:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

donde τ es un parámetro positivo (temperatura).

Antes de ver los algoritmos de RL también es importante distinguir entre los algoritmos *on-policy* y los algoritmos *off-policy*. Los algoritmos *on-policy* estiman el valor de la política mientras la usan para el control. Se trata de mejorar la política que se usa para tomar decisiones. Por otro lado, los algoritmos *off-policy* usan la política y el control en forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser ϵ-*greedy*. Esto es, la política de comportamiento está separada de la política que se quiere mejorar. Esto es lo que hace el algoritmo de Q-learning y quedará más claro cuando lo describamos más adelante.

5.3.4. Algoritmos

Una de las ventajas principales de los algoritmos de RL es que son incrementales y fáciles de implmentar. Básicamente, actualizan las funciones de valor usando el error entre lo el algoritmo TD(0):

Inicializa $V(s)$ arbitrariamente y π a la política a evaluar
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        $a \leftarrow$ acción dada por π para s
        Realiza acción a; observa la recompensa, r,
            y el siguiente estado, s'
        $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
        $s \leftarrow s'$
    hasta que s sea terminal

Un algoritmo muy parecido, pero para las funciones de valor Q se llama SARSA (state - action - reward - state - action). La actualización de valores tomando en cuenta la acción es:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

y el algoritmo es prácticamente el mismo, el cual viene descrito a continuación:

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Selecciona una a a partir de s usando la política
        dada por Q (e.g., ϵ–greedy)
    Repite (para cada paso del episodio):
        Realiza acción a, observa r, s'
        Escoge a' de s' usando la política derivada de Q
        $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    hasta que s sea terminal

Uno de los desarrollos más importantes en aprendizaje por refuerzo fué el desarrollo de un algoritmo "fuera-de-política" (*off-policy*) conocido como

Q-learning. La idea principal de este algoritmo es realizar la actualización de la siguiente forma (Watkins, 89):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

En donde en lugar de hacer la actualización usando la política, se realiza usando el valor Q con valor máximo, de ahí su nombre de fuera-de-política. El algoritmo es muy similar a SARSA pero cambia su mecanismo de actualización de la función Q. Su descripción es la siguiente:

Inicializa $Q(s, a)$ arbitrariamente
Repite (para cada episodio):
    Inicializa s
    Repite (para cada paso del episodio):
        Selecciona una a de s usando la política dada por Q
        (e.g., ϵ–greedy)
        Realiza acción a, observa r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma max'_a Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    hasta que s sea terminal

5.4. Estrategias para Espacios Grandes

Uno de los problemas principales de RL es su aplicación a espacios grandes (muchos estados y acciones). Aunque los algoritmos convergen en teoría, en la práctica pueden tomar un tiempo inaceptable. Para esto, se han propuesto diferentes estrategias para lidear con espacios grandes:

1. Actualizar varias funciones de valor a la vez
2. Usar aproximación de funciones
3. Aprender un modelo y usarlo
4. Utilizar abstracciones y jerarquías
5. Incorporar ayuda adicional

Vamos a revisar muy brevemente cada una de ellas.

5.4.1. Trazas de Elegibilidad

Las trazas de elegibilidad están entre los métodos de Monte Carlo y TD de un paso. Los métodos Monte Carlo, por un lado, realizan la actualización considerando la secuencia completa de recompensas observadas. La actualización de los métodos de TD, por otro lado, se hace utilizando únicamente la siguiente recompensa. La idea de las trazas de elegibilidad es considerar las recompensas de n estados posteriores (o afectar a n anteriores).

Si recordamos, un modelo de recopensa, calcula la recomensa total como:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

Lo que se hace en TD es usar:

$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

donde $V_t(s_{t+1})$ reemplaza a los siguientes términos $(r_{t+2} + \gamma r_{t+3} \ldots)$. Sin embargo, hace igual sentido hacer lo mismo considerando más pasos:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

y, en general, para n pasos en el futuro.

En la práctica, más que esperar n pasos para actualizar (*forward view*), se realiza al revés (*backward view*), actualizando n pasos hacia atrás. Se puede probar que ambos enfoques son equivalentes. Lo que se hace es guardar información sobre los estados por los que se pasó en el episodio y se actualizan hacia atrás los "errores", descontados por la distancia. Para esto se asocia a cada estado o par estado-acción una variable extra, representando su traza de elegibilidad (*eligibility trace*) que denotaremos por $e_t(s)$ o $e_t(s, a)$. Este valor va decayendo con la longitud de la traza creada en cada episodio.

Para $TD(\lambda)$:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{si } s = s_t \end{cases}$$

y para SARSA se tiene lo siguiente:

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s, a) + 1 & \text{si } s = s_t \end{cases}$$

El algoritmo de SARSA considerando trazas de elegibilidad se llama SARSA(λ) y su descripción es la siguiente:

Inicializa $Q(s, a)$ arbitrariamente y $e(s, a) = 0\ \forall s, a$
Repite (para cada episodio)
    Inicializa s, a
    Repite (para cada paso en el episodeo)
        Toma acción a y observa r, s'
        Selecciona a' de s' usando una política derivada
            de Q (e.g., $\epsilon-greedy$)
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $e(s, a) \leftarrow e(s, a) + 1$
        Para todos s, a
            $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
            $e(s, a) \leftarrow \gamma\lambda e(s, a)$
        $s \leftarrow s'; a \leftarrow a'$
    hasta que s sea terminal

De la misma forma, podemos definir un algoritmo que contiene trazas de elegibilidad para Q-learning o Q(λ). En este caso, como la selección de acciones se hace, por ejemplo, siguiendo una política $\epsilon-greedy$, se tiene que tener cuidado, ya que a veces los movimientos son movimientos exploratorios. Se tiene que tener cuidado, porque no queremos propagar en caminos buenos "errores" negativos por acciones exporatorias. Se puede mantener historia de la traza sólo hasta el primer movimiento exploratorio, ignorar las acciones exploratorias, o hacer un esquema un poco más complicado que considera todas las posibles acciones en cada estado.

5.4.2. Aprendiendo Modelos

Otro esquema para tratar de hacer frente a los espacios de estado-acción grandes es usando un modelo. Lo que hacen los algoritmos de programación dinámica es utilizar un modelo del ambiente para propagar y actualizar incrementalmente las funciones de valor. Con un modelo podemos predecir el siguiente estado y la recomepensa dado un estado y una acción. La predicción puede ser un conjunto de posibles estados con su probabilidad asociada o puede ser un estado que es muestreado de acuerdo a la distribución de probabilidad de los estados resultantes. Lo interesante es que podemos utilizar los estados y acciones simulados para aprender. Al sistema de aprendizaje no le importa si los pares estado-acción son dados de experiencias reales o simuladas.

Dado un modelo del ambiente, uno puede seleccionar aleatoriamente un par estado–acción, usar el modelo para predecir el siguiente estado, obtener una recompensa y actualizar valores Q. Esto se puede repetir indefinidamente hasta converger a Q^*. El algoritmo Dyna-Q combina experiencias con planificación para aprender más rápidamente una política óptima. La idea es aprender de experiencia, pero también usar un modelo para simular experiencia adicional y así aprender más rápidamente. Como no se tiene un modelo inicialmente, éste se construye al mismo tiempo que se realiza la exploración. Esto es lo que hace el algoritmo de Dyna-Q, el cual viene descrito a continuación:

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$
DO forever
    $s \leftarrow$ estado actual
    $a \leftarrow \epsilon-greedy(s,a)$
    realiza acción a observa s' y r
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $Modelo(s,a) \leftarrow s', r$
    Repite N veces:
        $s \leftarrow$ estado anterior seleccionado aleatoriamente
        $a \leftarrow$ acción aleatoria tomada en s
        $s', r \leftarrow Modelo(s,a)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

El algoritmo de Dyna-Q selecciona pares estado-acción aleatoriamente de pares anteriores. Sin embargo, la planificación se puede usar mucho mejor si se enfoca a pares estado-acción específicos. Por ejemplo, enfocarnos en las metas e irnos hacia atrás o, más generalmente, irnos hacia atrás de cualquer estado que cambie de manera importante su valor. Este proceso se puede repetir sucesivamente, sin embargo, algunos estados cambian mucho más que otros. Lo que podemos hacer es ordenarlos y cambiar sólo los pares que rebasen un cierto umbral. Esto es precisamente lo que hace el algoritmo de *prioritized sweeping*:

Inicializa $Q(s,a)$ y $Modelo(s,a)$ $\forall s \in S, a \in A$ y $ColaP = \emptyset$
DO forever
    $s \leftarrow$ estado actual
    $a \leftarrow \epsilon-greedy(s,a)$

realiza acción a observa s' y r
$Modelo(s, a) \leftarrow s', r$
$p \leftarrow\mid r + \gamma \max_{a'} Q(s', a') - Q(s, a) \mid$
if $p > \theta$, then inserta (s, a) a $ColaP$ con prioridad p
Repite N veces, mientras $ColaP \neq \emptyset$:
    $s, a \leftarrow$ primero($ColaP$)
    $s', r \leftarrow Modelo(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    Repite $\forall \overline{s}, \overline{a}$ que se predice llegan a s:
        $\overline{r} \leftarrow$ recompensa predicha
        $p \leftarrow\mid \overline{r} + \gamma \max_{a} Q(s, a) - Q(\overline{s}, \overline{a}) \mid$
        if $p > \theta$, then inserta $\overline{s}, \overline{a}$ a $ColaP$ con prioridad p

5.4.3. Aproximación de Funciones

Hasta ahora hemos supuesto que se tiene una representación explícita en forma de tabla de los estados o de los pares estado-acción. Esto funciona bien para espacios relativamente pequeños, pero es impensable para dominios como ajedrez (10^{120}) o backgammon (10^{50}) los cuales tienen una gran cantidad de estados o para estados con una representación contínua que no queremos discretizar. Una alternativa es usar una representación implícita, i.e., una función. Por ejemplo en juegos, una función de utilidad estimada se puede representar como una función lineal pesada sobre un conjunto finito de atributos (F_i's):

$$V(i) = w_1 f_1(i) + w_2 f_2(i) + \ldots + w_n f_n(i)$$

En ajedrez se tienen aproximadamente 10 pesos, por lo que es una compresión bastante significativa ya que lo que tenemos que aprender son los pesos de esa función.

La compresión lograda por una representación implícita permite al sistema de aprendizaje, generalizar de estados visitados a estados no visitados. Existen muchas alternativas que se pueden usar para aprender funciones contínuas y en RL se han usado redes neuronales (NN), máquinas de soporte verctorial (SVM), árboles de decisión, procesos gaussianos, entre otros. Como en todos los sistemas de aprendizaje, existe un balance entre el espacio de hipótesis y el tiempo que toma aprender una hipótesis aceptable.

Vamos a ver el caso más simple, y sin embargo muy utilizado, que consiste en representar la función que se quiere aprender como una combinación lineal

de funciones base $\phi_1, \phi_2, \ldots, \phi_n$ de la forma $\sum_{i=1}^{n} w_i \phi_i$ donde los pesos w_i son los que tenemos que aprender. Muchos sistemas de aprendizaje supervisado tratan de minimizar el error cuadrado (MSE) entre la función objetivo y la función que se está aprendiendo. Si $\vec{w}_t$ representa el vector de parámetros de la función parametrizada que queremos aprender:

$$MSE(\vec{w}_t) = \sum_{s \in S} [V^{\pi}(s) - V_t(s, \vec{w}_t)]^2$$

Para ajustar los parámetros del vector de la función que queremos optimizar, las técnicas de gradiente ajustan los valores en la dirección que produce la máxima reducción en el error:

$$\begin{aligned} \vec{w}_{t+1} &= \vec{w}_t - \tfrac{1}{2}\alpha \nabla [v_{\pi}(s_t) - \hat{v}(s_t, \vec{w}_t)]^2 \\ &= \vec{w}_t + \alpha [v_{\pi}(s_t) - \hat{v}(s_t, \vec{w}_t)] \nabla \hat{v}(s_t, \vec{w}_t) \end{aligned}$$

donde α es un parámetro positivo $0 \leq \alpha \leq 1$ y $\nabla f(\vec{w})$ denota un vector de derivadas parciales.

El algoritmo básico es el siguiente:

Dada una política π a evaluar y
una función diferenciable para v
Inicializa los pesos (w) de la función de valor
de manera arbitraria
Repite (para cada episodio):
Inicializa S
Repite (Para cada paso del epidosio):
Selecciona $A \sim \pi(\cdot|S)$
Toma la acción A, observa R, S'
$\vec{w} \leftarrow \vec{w} + \alpha [R + \gamma \hat{v}(S', \vec{w}) - \hat{v}(S, \vec{w})] \nabla \hat{v}(S, \vec{w})$
$S \leftarrow S'$

Para representar la función v se han usado funciones lineales, polinomios, series de Fourier, coarse coding, tile coding, funciones de base radial, NN, regresión local pesada, procesos Gaussianos, entre muchos otros. Para los algoritmos *off-policy* no siempre se logra convergencia. Para mejorar el estimador del error, a veces se usa información con trazas de elegibilidad. En otra sección veremos cómo se pueden aprender estas funciones de valor usando redes neuronales y cómo se han usado con redes neuronales profundas para aprender a jugar Go.

5.4.4. Abstracciones y Jerarquías

Otra forma de simplicar el espacio de estados-acciones es utilizar abstracciones. Por ejemplo, se puede hacer una agregación de estados juntando estados "parecidos" y a todos ellos se les asigna el mismo valor, reduciendo el espacio de estados. Por ejemplo, *tile-coding*, *coarse coding*, *radial basis functions*, *Kanerva coding*, y *soft-state aggregation*, son algunas estrategias que siguen este esquema.

Otros autores han realizado abstracciones basadas en máquinas de estado finito. La idea es tener varias máquinas de estado finito que pueden realizar diferentes acciones y se usa el aprendizaje por refuerzo para decidir que máquina utilizar (por ejemplo, HAM (Parr and Russell, 1998) y PHAM (Andre and Russell, 2003)).

Una forma de simplificar un problema es mediante la definición de una jerarquías. Lo que se hace es dividir el problema en subproblemas, se aprenden políticas para los espacios de más bajo nivel y éstas se usan para resolver problemas de más alto nivel (e.g., MAXQ (Dietterich, 1998), HEXQ (Hengst, 2002)). Algo parecido se usa con Macros y Options, en donde se aprenden políticas de subespacios que se usan para resolver problemas mas grandes.

Finalmente, un área interesante es utilizar representaciones relacionales dentro de aprendizaje por refuerzo (RRL) (Džeroski et al., 2001; Zaragoza and Morales, 2010). Esto es, representar los estados y acciones por medio de relaciones, lo cual permite tener representación entre objetos, usar variables, y aprender políticas más generales. Estas representaciones se han utulizado para representar las funciones de valor y/o para representar los estados y las acciones.

5.4.5. Incorporar Información Adicional

En su forma tradicional, RL no utiliza prácticamente nada de conocimiento del dominio. Una forma de ayudar a RL a coverger más rápidamente es incorporardo información adicional. Se han propuesto varios esquemas, uno muy usado es el de *reward shaping* (Ng et al., 1999; Tenorio-Gonzalez et al., 2010). La idea de *reward shaping* es incorporar información adicional a la función de recompensa para guiar la exploración del agente. También se han utilizado soluciones conocidas como guías o trazas que se usan para aprender más rápidamente las funciones de valor o para aprender un directamente la política. La idea es que un agente, generalmente un humano, le proporciona

una traza de cómo realizar la tarea y está sirve de guía para RL para aprender más rápido.

Una forma de seguir estas trazas es usando lo que se conoce como RL invertido. El problema de RL invertido es cómo aprender la función de recompensa dado un cierto comportamiento observado. Si tenemos información de cierto comportamiento, por ejemplo, dado por un experto, nos gustaría poder recuperar su función de recompensa y usarla para aprender el comportamiento deseado. Esto se ha usado para aprender, por ejemplo, a volar un helicóptero a escala teniendo el modelo dinámico del helicóptero.

5.4.6. Aprender la Política

Hasta ahora hemos visto cómo aprender una función de valor. Con esta función podemos entonces dfinir una política. Una opción es en lugar de tratar de aprender la función de valor (V o Q) podemos tratar de aprender directamente la política (π). Normalmente se tiene definida una función parametrizada, que representa la política, que se quiere optimizar y que es diferenciable con respecto a sus parámetros. A este procedimiento se lo que se conoce como gradiente de política (*policy gradient*).

Para aplicar un gradiente se necesita una medida de desempeño $\eta(\Theta)$ con Θ representando los pesos de la función de la política. Los parámetros de la función se aproximan usando el gradiente:

$$\Theta_{t+1} = \Theta_t + \alpha\widehat{\nabla\eta(\Theta)}$$

Donde $\widehat{\nabla\eta(\Theta)}$ es un estimador del gradiente. Las ventajas de aprender directamente una política son que posiblemente es más fácil que aprender la función de valor y que se pueden aprender políticas estocásticas.

El desempeño se puede definir como:

$$\eta(\Theta) = v_{\pi_\Theta}(s_0)$$

donde v_{π_Θ} es la función de valor verdadera para la política empezando en un estado inicial. El teorema del gradiente de política dice que:

$$\nabla\eta(\Theta) = \sum_s d_\pi(s) \sum_a q_\pi(s,a)\nabla_\Theta\pi(a|s,\Theta)$$

donde $d_\pi(s)$ se define como el número esperado de pasos de tiempo en donde el estado $S_t = s$ en un episodio generado aleatoriamente empezando en s_0 y siguiendo la política π

Después de varias manipulaciones matemáticas, la actualización de los parámetros de la política quedan como:

$$\Theta_{t+1} = \Theta_t + \alpha\gamma^t G_t \frac{\nabla_\Theta \pi(A_t|S_t,\Theta)}{\pi(A_t|S_t,\Theta)}$$

Donde:

- γ^t es el factor de descuento multiplicado el número de veces en que llega al estado
- G_t es el retorno que se obtiene a partir de ese estado
- A_t es la acción seleccionada por la política

El teorema de política del gradiente se puede generalizar para incluir una comparación entre el valor de la acción y un valor base (*baseline*):

$$\nabla\eta(\Theta) = \sum_s d_\pi(s) \sum_a \left(q_\pi(s,a) - b(s)\right) \nabla_\Theta \pi(a|s,\Theta)$$

Con esto la actualización queda como:

$$\Theta_{t+1} = \Theta_t + \alpha \left(G_t - b(S_t)\right) \frac{\nabla_\Theta \pi(A_t|S_t,\Theta)}{\pi(A_t|S_t,\Theta)}$$

Un candidato natural para $b(S)$ es el estimado de la función de valor $\hat{v}(S_t, w)$, donde w son los parámetros de la función de valor.

Con estos elementos, podemos definir el algoritmo de REINFORCE (Mnih et al., 2015):

Inicializa los pesos de política (Θ) y de función de valor (w)
Repite:
  Genera un episodio $S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    siguiendo π
  Para cada paso del epidosio $t = 0, 1, \ldots, T-1$
    $G_t \leftarrow$ retorno desde el tiempo t
    $\delta \leftarrow G_t - \hat{v}(S_t, w)$
    $w \leftarrow w + \beta\delta\nabla_w \hat{v}(S_t, w)$
    $\Theta = \Theta + \alpha\gamma^t \delta \nabla_\Theta log\pi(A_t|S_t,\Theta)$

con $\nabla logx = \frac{\nabla x}{x}$

Aunque el algoritmo de REINFORCE aprende una política y una función de valor, la funciónn de valor sirve como base y no como crítica. También, como buen método Monte Carlo el aprendizaje tiende a ser lento. Se puede hacer un algoritmo de diferencias temporales, cambiando el paso de recompensa completa que usa REINFORCE por el de un solo paso:

$$\Theta_{t+1} = \Theta_t + \alpha \left(G_t - \hat{v}(S_t, w)\right) \frac{\nabla_\Theta \pi(A_t|S_t, \Theta)}{\pi(A_t|S_t, \Theta)}$$

$$= \Theta_t + \alpha \left(R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\right) \frac{\nabla_\Theta \pi(A_t|S_t, \Theta)}{\pi(A_t|S_t, \Theta)}$$

$$\Theta_{t+1} = \Theta_t + \alpha \delta_t \frac{\nabla_\Theta \pi(A_t|S_t, \Theta)}{\pi(A_t|S_t, \Theta)}$$

Con esto, podemos definir un algoritmo que actualiza todo el tiempo y que no necesita que se finalice un espisodio para hacerlo.

Inicializa los pesos de política (Θ) y de función de valor (w)
Repite:
Inicializa S (estado inicial del episodio)
$I \leftarrow 1$
Mientras S no es estado terminal:
$A \sim \pi(\cdot|S, \Theta)$
Toma acción A, observa S', R
$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$
$w \leftarrow w + \alpha^w I \delta \nabla_w \hat{v}(S, w)$
$\Theta \leftarrow \Theta + \alpha^\Theta I \delta \nabla_\Theta ln \pi(A|S, \Theta)$
$I \leftarrow \gamma I$
$S \leftarrow S'$

con $\nabla logx = \frac{\nabla x}{x}$.

5.4.7. Deep RL

Recientemente se ha hecho una combinación entre Q-learning y redes neuronales profundas convolucionales. Se aplicó inicialmente para aprender a jugar los video-juegos de Atari y posteriomente para aprender a jugar Go. La relevancia del trabajo de Atari es que se usó la misma estructura para

aprender todos los juegos, aunque cada juego se aprendió por separado. La idea de cómo aplicar *deep learning* es siguiendos las ideas de los algoritmos de la última sección. El sistema que se aplicó a los juegos de Atari, alcanzó niveles de expertos humanos en 29 de los 46 juegos.

Para ésto, aprendió cada juego de 50 millones de pantallas (como 38 días de experiencia por juego). Las pantallas se redujeron a arreglos de 84×84 y se apilaron los últimos 4 frames (i.e., Input $= 84 \times 84 \times 4$). La arquitectura de la red tenía 3 capas convolucionales, de 32 20×20, 64 9×9, y 64 7×7 mapas de atributos. La última capa estaba conectada a una capa de 512 unidades y de ahí a 18 unidades de salida (1 por cada posible acción).

La función de recompensa cambió de $+1/-1/0$ dependiendo de la puntuación del juego. Se hizo una exploración usando ϵ-greedy descendiendo linealmente durante el primer millón de pantallas y manteniéndose bajo después de ésto. La actualización de los parámetros es:

$$\delta = R_{t+1} + \gamma max_a \hat{q}(S_{t+1}, a, \Theta_t) - \hat{q}(S_t, A_t, \Theta_t)$$
$$\Theta_{t+1} = \Theta_t + \alpha\delta\nabla_\Theta \hat{q}(S_t, A_t, \Theta_t)$$

y el gradiente se calcula usando backpropagation.

Para mejorar el entrenamiento, usaron *mini batch* haciendo actualización después de 32 imágenes y un algoritmo de gradiente que acelera el proceso de aprendizaje. También usaron *experience replay* que consiste en guardar información de las tuplas (s, a, s', r) para hacer actualizaciones múltiples simuladas. Para evitar inestabilidad, crearon una red alterna con los pesos actuales como estimador $(\tilde{q}(S_{t+1}, a, \Theta_t))$ para actualizar los pesos de la nueva red:

$$\delta = R_{t+1} + \gamma max_a \tilde{q}(S_{t+1}, a, \Theta_t) - \hat{q}(S_t, A_t, \Theta_t)$$

AlphaGO

El juego de Go comparado con ajedrez, por un lado, tiene más movimientos legales por posición (≈ 250 vs. ≈ 35) y más movimientos por juego (≈ 150 vs. ≈ 80 en ajedrez). Por otro lado, es difícil definir una función de evaluación adecuada.

AlphaGo combina Redes Neuronales Profundas, *Monte Carlo Tree Search* (MCTS) (Coulom, 2006), Aprendizaje Supervisado y Aprendizaje por Refuerzo. Para actualizar las estimaciones que se usan en MCTS usa directamente las funciones de valor. Para el sistema, los autores definieron y entrenarom

varias redes: la red de política usando aprendizaje supervisado, la red de simulaciones Monte Carlo y la red de la función de valor.

Posteriormente, los autores buscaron simplifar el mecanismo de aprendizaje. En lugar de aprender de muchos juegos y aprender varias redes neuronales, AlphaGo Zero, aprendió unicamente de auto juegos, haciendo un tipo de iteración de políticas. Se evalua la polñitica actual y se mejora. AlphaGo Zero usa MCTS para seleccionar las acciones y una sola CNN. MCTS corre una simulación hasta una hoja del árbol de búsqueda actual (en lugar de hasta un estado terminal - juego completo). La entrada de la CNN son matrices de $19 \times 19 \times 17$, representando 17 planos de atributos binarios con el tablero actual y 7 tableros anteriores del jugador actual, 8 del contrincante y 1 indicando el color. La red aprende: $f_\Theta(s) = (p, v)$, con $p_a = Pr(a|s)$ y v = probabilidad de ganar desde la posición igual.

En el árbol de búsqueda cada nodo tiene la siguiente información: $\{N(s,a), W(s,a), Q(s,a), P(s,a)\}$, donde:

- $N(s,a)$ es cuántas veces se ha visitado ($N(s,a) = N(s,a) + 1$)
- $W(s,a)$ es la función de valor total ($W(s,a) = W(s,a) + v$)
- $Q(s,a)$ es la función de valor promedio ($Q(s,a) = \frac{W(s,a)}{N(s,a)}$)
- $P(s,a)$ es la función de probabilidad a priori (lo que aprende la red)

En cada paso se selecciona la acción: $a_t = \text{argmax}_a(Q(s_t,a) + U(s_t,a))$, donde:

$$U(s,a) = cP(s,a)\frac{\sqrt{\sum_b N(s,b}}{1 + N(s,b)}$$

La CNN tiene 41 capas convolucionales y después se divide en 2: Una parte genera 362 salidas ($19^2 + 1$) que da la probabilidad de movida en cada lugar en el tablero + *pasar* (no hacer nada) - $Prob(a|s)$ desde el punto de vista del jugador actual. La otra parte genera una sola salida que estima la probabilidad de que el jugador gane desde la posición actual (v),

La red se entrenó con gradiente decendiente con muchas técnicas modernas usando "batches" de ejemplos tomados aleatorios de 500 mil juegos con la mejor política actual. Cada 1,000 pasos de entrenamiento se prueba la nueva red con la vigente. Si resulta mejor que un cierto umbral se reemplaza.

La red se entrenó con 4.9 millones de auto juegos que tomó alrededor de 3 días en computadoras de alto desempeño. Cada movida de cada juego

se seleccionaba al correr MCTS 1,600 interaciones (como 0.4 segundos por movida). Los pesos de la red se actualizaban sobre 700,000 batches cada uno con 2,048 posiciones. En cada posición se ejecuta un MCTS guiado por la red neuronal. El MCTS regresa una política (π) con las probabilidades de hacer cada movida y se usa como mejorador de políticas.

La red se entrena para maximizar la similaridad entre la predicción p y la política π obtenida con MCTS y para minimizar el error entre la predicción de quien va a ganar v y el resultado actual z. La función de pérdida (*loss*) es:

$$l = (z - v)^2 - \pi^T \log p + c||\Theta||^2$$

5.5. Conclusiones

El aprendizaje por refuerzo ha tomado mucho interés en los últimos años debido a su uso en juegos, como Go, y en aplicaciones en robótica. En este capítulo dimos algunos de los conceptos más importantes del área, sin embargo, existe muchos más. Una referencia obligada en esta área es el libro de Richard Sutton y Andrew Barto (2018): Reinforcement Learning: An Introduction (segunda edición), MIT Press.

Referencias

Andre, D. and Russell, S. (2003). *Programmable reinforcement learning agents*. PhD thesis, University of California, Berkeley.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Dietterich, T. G. (1998). The maxq method for hierarchical reinforcement learning. In *ICML*, volume 98, pages 118–126. Citeseer.

Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine learning*, 43(1-2):7–52.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with hexq. In *ICML*, volume 2, pages 243–250.

High, R. (2012). The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*.

Michie, D. and Chambers, R. A. (1968). Boxes: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

Parr, R. and Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Sutton, R. S., Barto, A. G., Bach, F., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

Tenorio-Gonzalez, A. C., Morales, E. F., and Villaseñor-Pineda, L. (2010). Dynamic reward shaping: training a robot by voice. In *Ibero-American Conference on Artificial Intelligence*, pages 483–492. Springer.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Zaragoza, J. H. and Morales, E. F. (2010). Relational reinforcement learning with continuous actions by combining behavioural cloning and locally weighted regression. *Journal of Intelligent Learning Systems and Applications*, 2(02):69.

Capítulo 6

Lógica Difusa y Sistemas de Control Difusos

C. Reyes-García, A. A. Torres-García
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

6.1. Lógica Difusa

La Lógica Difusa es un enfoque de computación basado en *grados de verdad* en lugar del tradicional *verdadero/falso* (1 o 0) de la lógica Booleana en que se basan las computadoras actuales. La idea y el término de Lógica Difusa (Zadeh, 1988) establecido por el Profesor Lotfi Zadeh de la Universidad de California en Berkeley en 1963. La idea principal era visualizar a la Lógica Difusa como la forma en que realmente funciona el razonamiento, y tomando a la lógica Booleana como un caso especial de ésta. La lógica difusa incluye 0 y 1 como los casos extremos de verdad, pero también incluye los diversos estados intermedios de verdad. La lógica difusa parece más cercana a la forma en que trabaja nuestro cerebro cuyo procesamiento se basa en la percepción y no en la precisión. La lógica difusa es esencial para el desarrollo de capacidades similares a las capacidades cognitivas humanas en sistemas de software.

En 1965 Zadeh publicó su primer artículo llamado "FUZZY SETS", donde establece los fundamentos de la teoría de los Conjuntos Difusos (Zadeh, 1965). De acuerdo a Zadeh, la membresía de un elemento a un conjunto difuso es gradual. Los grados de membresía son valores en el intervalo [0, 1]. Es por esto que, la llave maestra de los conjuntos difusos son los valores de

MEMBRESÍA. Además, establece que los conjuntos difusos incluyen los conjuntos duros (McNeill and Freiberger, 1994), ya que un Conjunto Duro (Crisp) es un conjunto difuso con valores de 1 y 0.

¿POR QUÉ DIFUSOS? Zadeh dice que el término era muy concreto, inmediato y descriptivo. Él sabía que el término mismo crearía controversia. Y de hecho el término causo exactamente ese efecto en muchos y aun lo hace. La difusión describe Ambigüedad de Eventos. Mide el grado al cual un evento ocurre, no si ocurrirá. El azar describe la incertidumbre de la ocurrencia del evento. La difusión es un tipo de incertidumbre determinística. Mientras la ambigüedad es una propiedad de los fenómenos físicos (Kosko, 1998).

¿POR QUÉ LÓGICA? Tal como la lógica binaria se apoya en los conjuntos duros la difusión se apoya en conjuntos difusos, y la lógica difusa como lógica es realmente una parte pequeña del área. Lógica difusa no es lógica que es difusa, sino lógica que describe y filtra la difusión. La mayor parte de la teoría no es lógica de ninguna manera, es una teoría de conjuntos difusos, conjuntos que calibran vaguedad.

Un ejemplo clásico es el de tratar de clasificar individuos en el conjunto de personas Altas, considerando la altura de cada persona. En los conjuntos típicos o duros, un elemento pertenece o no pertenece al conjunto, por lo que, para la clasificación anterior, es necesario establecer un límite inferior desde el cual se pueda clasificar a las personas como pertenecientes o no al conjunto de personas altas (ej. 1.75 mts.). Mientras que en el conjunto difuso la membresía es gradual, por lo que en la tabla 6.1 se muestra una comparación de ambos enfoques.

Tabla 6.1 Comparación de los valores difusos y duros para las alturas dadas

Persona	Altura	Valores duros	valores difusos
Juan	(2.05m) es alto	1	1
José	(1.96m) es alto	1	1
Pedro	(1.80m) es alto	1	0.85
Luis	(1.75m) es alto	0	0.7
Beto	(1.70m) es alto	0	0.4

De la tabla anterior, se interpreta que alguien que mide 1.75m es alto a un grado de 0.7 e incluso uno que mide 1.70m es alto a un grado de 0.4, y alguien que mide 1.96 m es indudablemente alto y su pertenencia al conjunto alto es 1. Todos los que midan más de 1.96 seguirán siendo altos a un grado

de 1, y, por ejemplo, alguien que mida 1.60 no pertenece al conjunto alto, y su grado de pertenencia es 0.

En las subsecciones siguientes se describen los principales conceptos y algunos ejemplos prácticos que dan soporte a este enfoque para una mejor compresión del lector.

6.1.1. Conjunto Difuso

Un conjunto difuso A se define como una ***Función de Membresía*** *que mapea los elementos de un dominio o Universo de discurso X con elementos del intervalo [0,1]:*
$A : X \to [0, 1]$

Cuanto más cerca esté $A(x)$ del valor 1, mayor será la membresía del objeto x al conjunto A. Asimismo, los valores de membresía varían entre 0 (no pertenece en absoluto) y 1 (membresía total).

Representación: Un conjunto difuso A puede representarse como un conjunto de pares de valores: Cada elemento $x \in X$ con su grado de membresía a A. También puede ponerse como una "suma" de pares:

- $A = \{A(x)/x, x \in X\}$
- $A = \sum_i A(x_i)/x_i$ (Los pares en los que $A(x_i) = 0$, no se incluyen).

Observe el siguiente ejemplo, el Conjunto de alturas del concepto difuso "Alto" en Personas:
$A = 0.25/1.75 + 0.5/1.8 + 0.75/1.85 + 1/1.9$

A este tipo de representación se denomina Singleton. Mientras que, si el Universo es Continuo, entonces $A = \int_x A(x)/x$.

En este caso, **la suma** y **la integral** no deben considerarse como operaciones algebraicas. La fracción denota pares ordenados y la suma la función unión.

6.1.2. Características de un Conjunto Difuso

Altura de un Conjunto Difuso (*height*): El valor más grande de su función de membresia: $sup_{x \in X} A(x)$.

Conjunto Difuso Normalizado (normal): Si existe algún elemento $x \in X$, tal que pertenece al conjunto difuso totalmente, es decir, con grado 1. O también, que: $Altura(A) = 1$.

Soporte de un Conjunto Difuso (*support*): Elementos de X que pertenecen a A con grado mayor a 0: $supp(A) = \{x \in X | A(x) > 0\}$.

Núcleo de un Conjunto Difuso (*core*): Elementos de x que pertenecen al conjunto con grado 1: $Nucleo(A) = \{x \in X | A(x) = 1\}$. Lógicamente, $Nucleo(A) \subseteq supp(A)$.

α-Corte (α-*Cut*): Valores de X con grado mínimo α: $A\alpha = \{x \in X | A(x) \geq \alpha\}$.

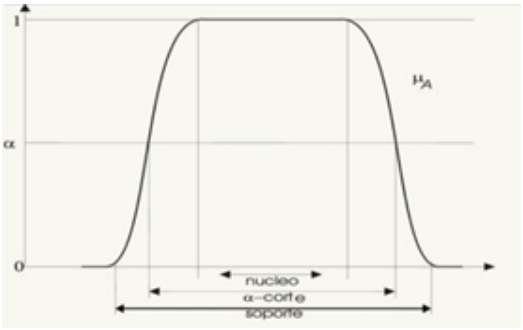


Figura 6.1 Características de un conjunto difuso

6.1.3. Distinción entre conjuntos clásicos y difusos

La difusión describe Ambigüedad de Eventos. Mide el grado el cual un evento ocurre, no si ocurrirá. El azar describe la incertidumbre de la ocurrencia del evento. La difusión es un tipo de incertidumbre **Determinística**. Ambigüedad es una propiedad de los fenómenos físicos (Kosko, 1992).

Una de las principales diferencias entre ambos tipos de conjuntos se da la intersección de un conjunto dado A con su complemento $\overline{A}$. Mientras que en los conjuntos tradicionales dicha operación resulta en $\emptyset$, en los conjuntos difusos esto no ocurre así.

CLÁSICOS	DIFUSOS
$A \cap \overline{A} = 0$	$A \cap \overline{A} \neq 0$
$P(A \cap \overline{A}) = P(0) = 0$	

6.1.4. Operaciones Difusas

Las operaciones en conjuntos difusos son:

Vacuidad: Un conjunto difuso está vacío si todos los candidatos tienen membresía 0 (EMPTYNESS). Por ejemplo: El Conjunto de océanos cuyo nombre comienza con X.

CONTENIMIENTO: En conjuntos difusos cada elemento debe pertenecer menos al subconjunto que al conjunto más grande.

	JUAN	JOSÉ	PEDRO	LUIS	BETO
MUY ALTO	1	0.95	0.8	0.4	0.01
ALTOS	1	1	0.85	0.7	0.4

INTERSECCIÓN: En conjuntos difusos es el grado de membresía que dos conjuntos comparten. Una intersección difusa es el valor menor o mínimo de la membresía de cada elemento en ambos conjuntos.

	JUAN	JOSÉ	PEDRO	LUIS	BETO
ALTOS	1	1	0.9	0.4	0.2
GORDOS	0.2	0.5	0.1	0.8	0.6
INTER	0.2	0.5	0.1	0.4	0.2

UNIÓN: La unión de conjuntos difusos es lo inverso de la intersección. Este es el valor más alto de los dos valores difusos.

	JUAN	JOSÉ	PEDRO	LUIS	BETO
ALTOS	1	1	0.9	0.4	0.2
GORDOS	0.2	0.5	0.1	0.8	0.6
UNIÓN	1	1	0.9	0.8	0.6

COMPLEMENTO: El complemento de un conjunto difuso es la cantidad que la membresía necesita para alcanzar 1.

6.1.5. Los números también son difusos

Aunque los números son el distintivo de la precisión, ellos también encierran difusión. Por ejemplo, considere el número CERO. En la figura 6.2, la línea significa que el número 0 pertenece 100 % al conjunto 0. Pero, ¿Qué pasa con los números cerca a cero, o casi cero? Ellos definen un espectro de números alrededor de cero y algunos pertenecen más al conjunto que otros.

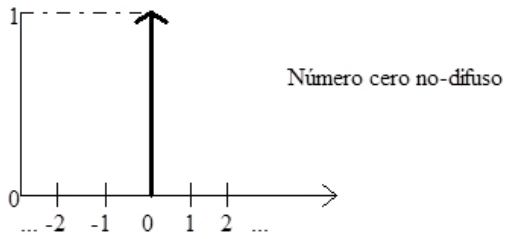


Figura 6.2 Gráfica de pertenencia al cero usando lógica clásica

El número cero difuso puede ser dibujado como una curva campana o un triángulo centrado en el número cero (ver figura 6.3). Entre más angosto se dibuje el triángulo, más se parecerá a la línea del cero clásico. Esa es otra sorpresa, *las matemáticas tal y como las conocemos son solo un caso especial de las matemáticas difusas* (Kosko and Toms, 1993).

Figura 6.3 Gráfica de pertenencia al cero usando lógica difusa

6.1.6. Geometría de los Conjuntos Difusos

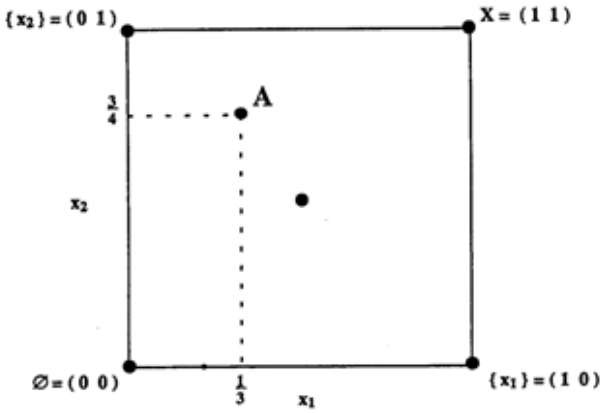
Un conjunto difuso define un punto en un hipercubo unidad, y uno no difuso define una esquina del hipercubo. Un hipercubo es un cubo de dimensión n ($n \geq 1$).

La geometría de los conjuntos difusos (Conjuntos como puntos) involucra:

DOMINIO: $X = \{X_1, X_2, X_3, \ldots, X_m\}$
RANGO: [0, 1]

MAPEO: $\mu_A : X \to [0, 1]$

Los vértices del cubo I^n definen conjuntos no difusos. El mayor grado de difusión se da en el punto medio (ver figura 6.4), donde no solo A es igual a No-A, sino que además (Kosko, 1992): $A = A \cup \overline{A} = A \cap \overline{A} = \overline{A}$



$\mu_A = (0.33, 0.75)$

Figura 6.4 Cubo difuso de un punto A y el punto medio (Kosko, 1990)

Los grados de membresía para cada una de las operaciones se calculan así;
$\mu_{A \cap B} = min(\mu_A, \mu_B)$
$\mu_{A \cup B} = max(\mu_A, \mu_B)$
$\mu_{\overline{A}} = 1 - \mu_A$

Ejemplos (por simplicidad se omite μ):
A = (1 .8 .4 .5)
B = (.9 .4 0 .7)
$A \cap B$ = (.9 .4 0 .5)
$A \cup B$ = (1 .8 .4 .7)
$\overline{A}$ = (0 .2 .6 .5)
$A \cap \overline{A}$ = (0 .2 .4 .5)
$A \cup \overline{A}$ = (1 .8 .6 .5)

Proposición:
A es propiamente difusa, IFF $A \cap \overline{A} \neq 0$ y IFF $A \cup \overline{A} \neq 1$

Una vez que ha sido localizado el conjunto o punto A, se tienen automáticamente todos los cuatro puntos, y los cuatro se encuentran igualmente cerca de su esquina más próxima.

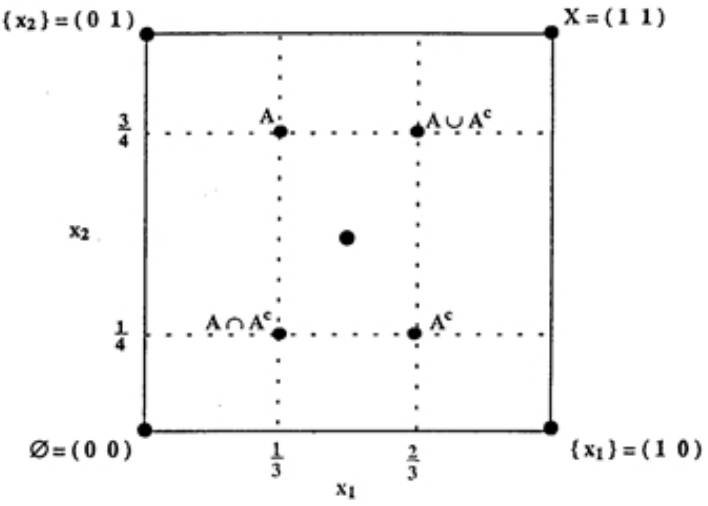


Figura 6.5 Cubo difuso con los valores para A, $\overline{A}$ y las operaciones de unión e intersección (Kosko, 1990)

6.1.7. Paradoja del Punto Medio

La lógica clásica y la teoría de conjuntos prohíben el punto medio por los mismos axiomas Aristotélicos;

1. Ley de la No Contradicción: A no puede ser a la vez B y No-B $(B \cap \overline{B}) = \emptyset$

2. La ley de la media excluida: (Ley de bivalencia): A debe de ser, ya sea B o No-B $(B \cup \overline{B}) = X$

Los fenómenos del punto medio incluyen:

- El vaso medio lleno y medio vacío.
- El Yin-Yang taoísta ☯.
- El mentiroso de Creta que decía que todos los cretanos son mentirosos. ¿Decía la verdad?

- El conjunto de todos los conjuntos que no son miembros de ellos mismos, de Bertrand Russell. Puede existir este conjunto?

- El barbero de Russell, cuyo letrero dice que el rasura a todos y solo aquellos hombres del pueblo, que no se rasuran ellos mismos. ¿Quién rasura al barbero?

Sea R la proposición de que el barbero se rasura a sí mismo, y no-R que no lo hace. Entonces, desde que R implica no-R y no-R implica R. Las dos proposiciones son equivalentes y tienen el mismo valor de verdad. Proposiciones equivalentes tienen el mismo valor de verdad.

$$\begin{aligned} t(R) &= t(no - R) \\ &= 1 \text{ - } t(R) \end{aligned}$$

Resolviendo para $t(s)$ da el punto medio de intervalo de verdad $[0, 1]$: $t(R) = 1/2$. El punto medio es equivalente a los vértices 0 y 1.

Los teóricos Fuzzy explican porqué los científicos y mucha gente a insistido en bivalencia por tanto tiempo, esto es por nuestra tendencia a redondear. El redondeo simplifica la vida y frecuentemente cuesta poco.

6.1.8. Cardinalidad de un conjunto difuso

El tamaño o cardinalidad de A, $M(A)$ permite saber qué tan grande es un conjunto difuso, y es igual a la suma de los valores fit (fuzzy unit) de A, es decir,

$$M(A) = \sum_{i=1}^{n} \mu_A(X_i).$$

Por ejemplo, si $A = (1/3, 3/4)$ entonces $M(A) = 1/3 + 3/4 = 13/12$ (ver figura 6.6).

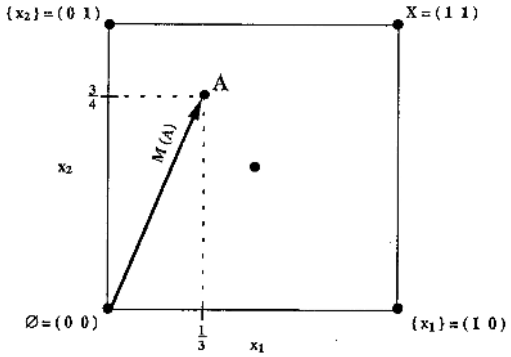


Figura 6.6 La cardinalidad $M(A)$ de A es la norma de Hamming difusa (l^1 norm) del vector dibujado desde origen hasta A (Kosko, 1990)

Como puede observarse el tamaño $M(A)$ de A iguala la distancia o norma difusa de Hamming (norma d)

$$M(A) = \sum_{i}^{n} |\mu_A(X_i) - 0| = \sum_{i=0}^{n} |\mu_A(X_i) - \mu_0(X_i)| = d(A, 0).$$

La distancia Euclidiana d^p entre los conjuntos difusos A y B está dada como sigue.

$$d^p(A, B) = \sqrt[p]{\|\mu_A(X_i) - \mu_B(X_i)\|^p},$$

donde $1 \leq p \leq w$ es la dimensión del hipercubo.

6.1.9. Entropía Difusa

La entropía difusa mide la difusión de un conjunto difuso. Entropía significa la incertidumbre o desorden en un sistema. Cuando el conjunto es difuso, el conjunto es incierto o vago a cierto grado. La entropía difusa mide este grado. La **entropía difusa** se puede medir con dos cuerdas.

1. Escoja un punto en el cubo difuso y llámelo A. $A = (2/3, 1/4)$.

2. Ate una cuerda roja de A a la esquina más cercana (línea punteada). Esta cuerda mantiene el seguimiento de que tan cerca se está de la esquina y que tan lejos del punto medio. Si A se aleja de la esquina más cercana, se acerca entonces a la esquina más lejana.

3. Ate una cuerda azul de A hasta la esquina más lejana (línea continua). El porcentaje de medida de entropía es solo el rojo sobre el azul.

4. Ate una cuerda azul de A hasta la esquina más lejana (línea continua). El porcentaje de medida de entropía es solo el rojo sobre el azul.

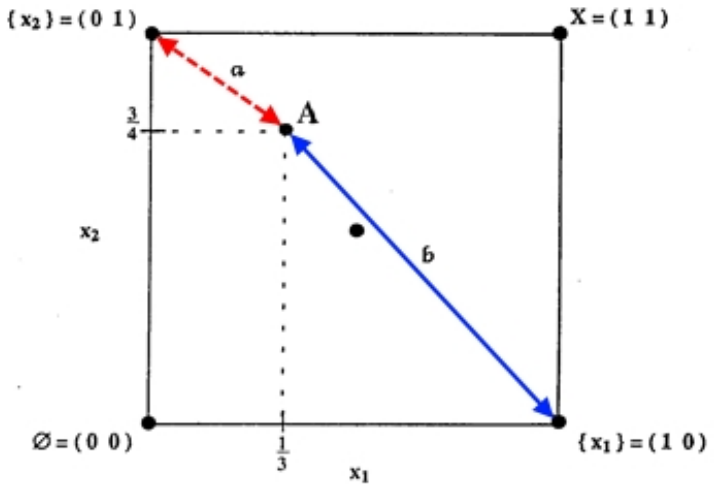


Figura 6.7 Ejemplo del cálculo de la entropía difusa (adaptado de (Kosko, 1990))

En la figura 6.7 se observa que al dividir la longitud roja (a) sobre la longitud azul (b) se obtiene un número entre 0 y 1 que describe la vaguedad de A. Entre mayor sea el número mayor es la **difusión**.

6.1.10. El Teorema de Entropía Difusa

La difusión se mide con una medida de entropía difusa. La entropía mide la INCERTIDUMBRE de un sistema o mensaje. Esa incertidumbre iguala su difusión.

La entropía difusa de A, $E(A)$ varía de 0 a 1 sobre el hipercubo unidad I_n. Solo los vértices del cubo tienen entropía cero. El medio punto del cubo únicamente tiene unidad o máxima entropía. La entropía difusa se incrementa suavemente al irse moviendo un punto de conjunto desde cualquier vértice hacia el punto medio. Lo más cerca el conjunto difuso A está al vértice más cercano A_{cerca}, lo más lejos A esté del vértice más lejano A_{lejos}.

Suponga que a denota la distancia $d'(A, A_{cerca})$, y que b denota la distancia de $d'(A, A_{lejos})$. Entonces la entropía difusa es igual a la relación de a con b:

$$E(A) = \frac{a}{b} = \frac{d'(A, A_{cerca})}{d'(A, A_{lejos})} \tag{6.1}$$

Usando la información de la figura 6.8, se tiene que $A(1/3, 3/4)$, $A_{cerca} = (0, 1)$ y $A_{lejos} = (1, 0)$.

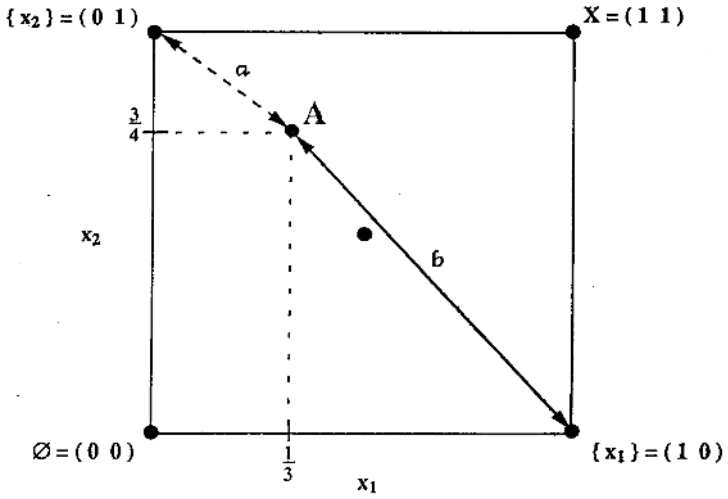


Figura 6.8 Ejemplo para el calculo de las distancias $d'(A, A_{cerca})$ y $d'(A, A_{lejos})$ (Kosko, 1990)

Con base en lo anterior,

$$a = \tfrac{1}{3} + \tfrac{1}{4} = \tfrac{7}{12} \text{ y } b = \tfrac{2}{3} + \tfrac{3}{4} = \tfrac{17}{12}$$
$$\therefore E(A) = \frac{\frac{7}{12}}{\frac{17}{12}} = \frac{7}{17}.$$

De la misma manera, por consiguiente cada uno de los puntos A, $A \cap \overline{A}$, $A \cup \overline{A}$ es igualmente cerca de su más cercano vértice. La distancia común es igual a. Y similarmente cada punto está igualmente lejos de su vértice más lejano. La figura 6.9 ilustra mejor lo anterior. Su distancia común, es igual a B. Así, el teorema de Entropía Difusa está dado como sigue,

$$E(A) = \frac{M(A \cap \overline{A})}{M(A \cup \overline{A})}. \tag{6.2}$$

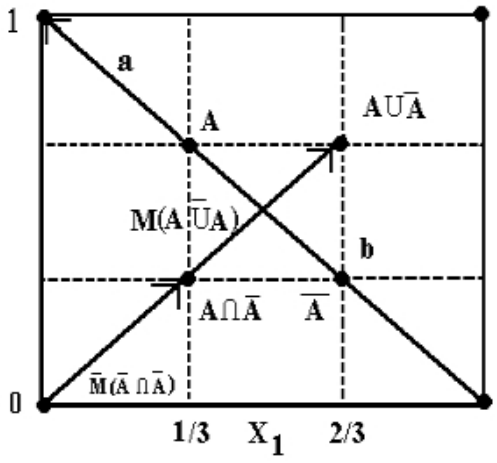


Figura 6.9 Geometría del teorema de entropía difusa (adaptado de (Kosko, 1990))

6.1.11. Teorema de Subconjuntos

Los conjuntos contienen subconjuntos (Kosko, 1992). En los conjuntos duros, A es un subconjunto de B, denotado $A \subset B$, sí y solo sí cada elemento de A es un elemento de B. El conjunto Potencia 2^B contiene todos los subconjuntos de B. Así, alternativamente, A es un subconjunto de B sí y solo sí A pertenece a 2^B

$$A \subset B \iff A \in 2^B$$

La relación del subconjunto corresponde a la relación IMPLICACION en lógica: Si A es un subconjunto de B, entonces la membresía en A implica la membresía en B.

$$x \in A \rightarrow x \in B$$

Si A y B son conjuntos difusos, la implicación difusa x *is* $A \rightarrow x$ *is* B forma una relación de implicación difusa.

En lógica clásica la verdad mapea del conjunto de estatutos a los dos valores verdad, $t : \{S\} \Rightarrow \{0, 1\}$. Considere la siguiente tabla de verdad de la implicación para las proposiciones bivalentes P y Q,

La implicación es falsa si y solo si el antecedente P es verdad y el consecuente Q es falso.

Lo mismo se aplica para subconjuntos, A es un subconjunto de B si y solo sí no hay elemento X que pertenece a A pero no a B.

P	Q	$P \Rightarrow Q$
0	0	1
0	1	1
1	0	0
1	1	1

$$A \subset B \Longleftrightarrow \mu_A(x) \leq \mu_B(x) \ \forall x$$

Esta relación es conocida como relación de la función de **Membresía Dominada**.
Si: $A = (\ 0.3\ 0.0\ 0.7)$ y $B = (\ 0.4\ 0.7\ 0.9\ )$
Entonces A es un subconjunto difuso de B.
Pero B no es un subconjunto difuso de A.

En este caso el conjunto difuso A, es o no es un subconjunto difuso de B. Lo cual no es una **Relación Difusa**.

Ahora; geométricamente el conjunto de potencia difuso (todos los conjuntos difusos) de $B(F(2^B))$ puede ser representado, para $B = (1/3, 2/3)$, como:

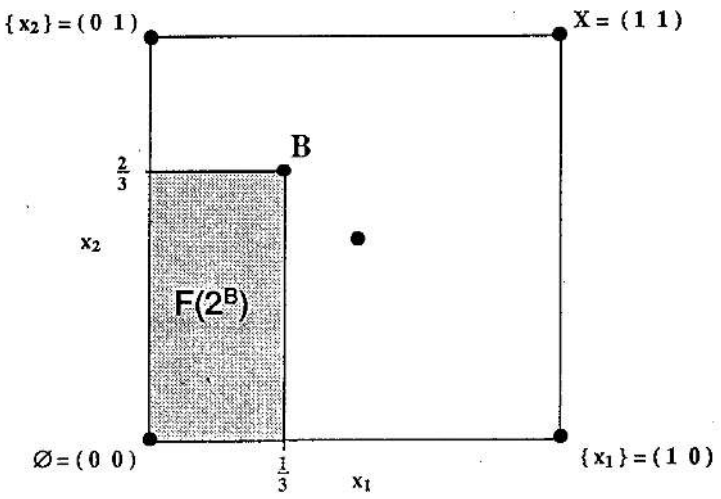


Figura 6.10 Conjunto potencia difuso $F(2^B)$ visto como un hiper-rectángulo dentro del cubo difuso (Kosko, 1990)

Si B no está vacío $F(2^B)$ tiene cardinalidad infinita. La figura 6.10 mues-

tra que $F(2^B)$ es un conjunto difuso. Y como se puede ver, el punto hipercubico A está fuera del hiper-rectángulo $F(2^B)$, aunque parte del conjunto es un subconjunto de $F(2^B)$.

6.1.12. Definición de Subconjuntos Difusos sobre $F(2^B)$

Algunos conjuntos A pertenecen a $F(2^B)$ en diferentes grados. Así la función de membresía $F(2^B)(A)$ puede ser igual a cualquier número en [0,1]. Lo que define los grados de subconjuntos.

Si $S(A, B)$ denota el grado al cual A es un subconjunto de B:

$$S(A, B) = \text{grado}(A \subset B) = F(2^B)(A)$$

Considerando la figura 6.11, por intuición, $S(A, B)$ debe acercarse a la unidad cuando A se aproxima al conjunto potencia $F(2^B)$. $S(A, B)$ debe decrecer y $1 - S(A, B)$ medida de superconjuntos debe crecer cuando A se aleja de $F(2^B)$.

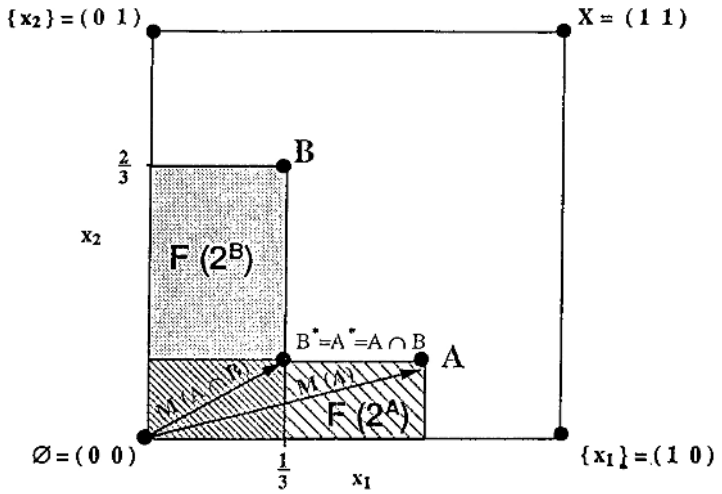


Figura 6.11 Descripción gráfica del teorema de subconjuntos difusos (Kosko, 1990)

Aquí el punto $B^* = A \cap B = A^*$ identifica el conjunto dentro del rectángulo $F(2^A)$ y el hiper-rectángulo $F(2^B)$ que tiene la cuenta máxima $M(A \cap B)$. Por lo que el teorema de subconjuntos difusos se define como sigue,

$$S(A, B) = \frac{M(A \cap B)}{M(A)}. \tag{6.3}$$

6.1.13. Variables lingüísticas

Una variable lingüística es una variable cuyos valores no son números si no palabras u oraciones en un lenguaje natural (Reyes-García, 1994).

Definición. Una variable lingüística se caracteriza por un quintuple $(X, T(x), U, G, \mu)$ donde X es el nombre de la variable; $T(X)$ es el conjunto de términos de X; U es el universo de discurso; G es la regla sintáctica que genera los términos de $T(X)$; y μ es una regla semántica que asocia el significado con cada valor lingüístico X, donde $M(X)$ denota un subconjunto difuso de U. Para una X en particular, el nombre generado por G es llamado un término.

Ejemplo:

Suponga que X es una variable lingüística llamada altura de personas con $U = [0, 250]$. Los términos de esta variable lingüística, que son conjuntos difusos, pueden ser llamados alto, bajo, muy alto, etc. La variable va a ser U es la altura en cms. de personas. $\mu(X)$ es la regla que asigna un significado al término.

$$\mu(x) = \begin{cases} 0 & \text{para } x \in [0, 150] \\ 1 + \{(x - 150)/10\}^2 & \text{para } x \in [150, 250] \end{cases}$$

$T(X)$ define el conjunto término de la variable X. Para el ejemplo, $T(altura) =$ {alto, muy alto, no muy alto, bajo, más o menos bajo}.

Donde $G(X)$ es un arreglo que genera los términos en el conjunto termino $T(X)$.

6.1.14. Modificadores lingüísticos

Un modificador lingüístico (Linguistic Hedge) es un operador que modifica el significado de un término. Si A es un conjunto difuso, entonces el modificador m genera el termino compuesto $b = m(A)$ (Pal and Mandal, 1992).

Los modelos matemáticos más frecuentemente usados como modificadores son:

- concentración:
$$\mu_{CON(A)} = (\mu_A(x))^2.$$
- dilatación:
$$\mu_{DIL(A)} = (\mu_A(x))^{\frac{1}{2}}.$$

- intensificación de contraste:

$$\mu_{INT(A)} = \begin{cases} 2 * (\mu_A(x))^2 & \text{para } \mu_A(x) \in [0, 0.5] \\ 1 - 2 * (1 - \mu_A(x))^2 & \text{en caso contrario} \end{cases}$$

Generalmente, los siguientes modificadores lingüísticos son asociados con los operadores matemáticos: Si A es un término (un conjunto difuso), entonces,

- Muy $A = CON(A)$,
- Más o menos $A = DIL(A)$,
- Más $A = A^{1.25}$,
- Poco $A = INT$ [más A y no (muy A)].

6.2. Sistemas de inferencia difusos

De acuerdo con (Jang et al., 1997), un sistema de inferencia difuso (FIS por sus siglas en inglés) es un esquema computacional popular basado en los conceptos de la teoría de conjuntos difusos, reglas IF-THEN difusas y razonamiento difuso. Los dos tipos de FIS más importantes son el de Mamdani y el de Sugeno. La principal diferencia entre ambos se encuentra en el consecuente de las reglas difusas. El FIS tipo Mamdani usa conjuntos difusos como consecuente de las reglas; mientras que el tipo Sugeno emplea funciones lineales de las variables de entrada como consecuente (Sivanandam et al., 2007). A continuación se describe el FIS tipo Mamdani debido a que es intuitivo y se adapta mejor a las definiciones lingüísticas propias de los humanos.

6.2.1. Sistemas de Inferencia Difuso tipo Mamdani

El método de inferencia difuso tipo Mamdani es la metodología difusa más conocida y uno de los primeros sistemas de control construidos usando la teoría de conjuntos difusos propuesta por Lofti Zadeh. Ebrahim Mamdani y Sedrak Assilian implementaron este método en 1975 como un intento para controlar una máquina de vapor y la presión de la caldera mediante la síntesis de un conjunto de reglas lingüísticas (tipo *IF-THEN* difusas) de control obtenidas a partir de la experiencia de los operadores humanos (McNeill and

Freiberger, 1994; Sivanandam et al., 2007). Un FIS de Mamdani puede ser implementado como sigue:

1. Definir los nombres o variables a controlar.

2. Escoger los conjuntos difusos para las variables de entrada y salida. Éstos son llamados funciones de membresía y se dibujan como curvas, trapezoides o triángulos.

3. Determinar un conjunto de reglas difusas. Tanto el antecedente como el consecuente de las reglas son difusos debido a que el FIS es tipo Mamdani.

4. Elegir un método de desdifusión, es decir, un método que permita la conversión de una cantidad difusa en una cantidad precisa. Los dos métodos más usados son el centroide ponderado y la media de los máximos.

En la figura 6.12 se puede observar la arquitectura de un FIS tipo Mamdani. Ésta consta de las siguientes partes: difusor, mecanismo de inferencia difuso, base de reglas difusas y desdifusor (Sivanandam et al., 2007).

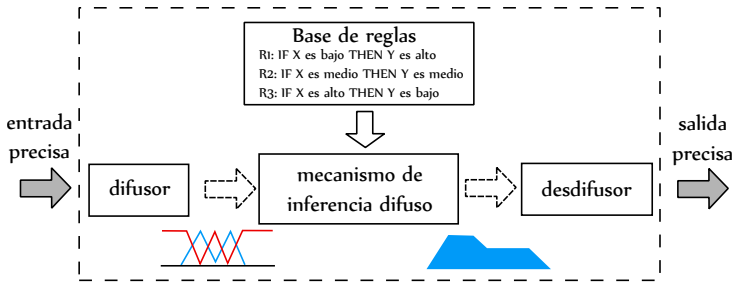


Figura 6.12 Arquitectura de un FIS tipo Mamdani (Adaptado de (Sivanandam et al., 2007))

A continuación se retoma la implementación de Mamdani y Assilian para ilustrar mejor el diseño de este tipo de FIS en aplicaciones de control. Asimismo, conviene mencionar que el control en las maquinas involucra el alcanzar una meta y permanecer allí, por ejemplo, en los termostatos, la meta es la temperatura que se ha seleccionado como deseable. La forma más común de

lograr lo anterior, en control clásico, es usando dispositivos llamados controladores **PID** (*Proportional-Integral-Derivative*) donde cada parte se refiere a una operación matemática.

El sistema de Mamdani y Assilian fue diseñado para mantener la presión de la caldera y la velocidad de pistón constante en una máquina de vapor; el sistema aprendía por sí mismo. Mamdani y Assilian no trataron de aplicar alguna fórmula para resolver su problema, sino que lo dividieron en reglas IF-THEN difusas. Debido a que el sistema usaba información de tres tipos, crearon tres escalas movibles; 2 sobre el estado de la caldera (input), y otro sobre un comando (output).

1. Error de presión: ¿Qué tan lejos está la presión deseada?
2. Cambio en el error de presión: ¿Qué tan rápido la presión se acerca o aleja de la deseada?
3. Cambio en calor: ¿Cuál es la respuesta correcta? (comando de salida)

Después establecieron siete propiedades lingüísticas para cada escala:

1. PB - Positive Big
2. PM - Positive Medium
3. PS - Positive small
4. ZE - Nil
5. NS - Negative small
6. NM - Negative Medium
7. NB - Negative Big

Todos estos conjuntos difusos sobrepuestos son del mismo tamaño y forma (triangular), y fueron espaciados uniformemente a lo largo de cada línea. Después se construyó la red de reglas usando estos conjuntos. Con esto no solo describieron el trabajo del sistema en términos simples. Si no que capturaron la experiencia de operadores diestros. El sistema completo está formado por 14 reglas como las siguientes:

Regla 1: IF EP es PS AND CEP es ZE THEN CC es NS

Regla 2: IF EP es ZE AND CEP es ZE THEN CC es ZE

Regla 3: IF EP es PS AND CEP es NS THEN CC es NS

El sistema de control completo es representado por la figura 6.13. Los cuadros en blanco no son necesarios ya que en práctica el sistema nunca estará en ellos.

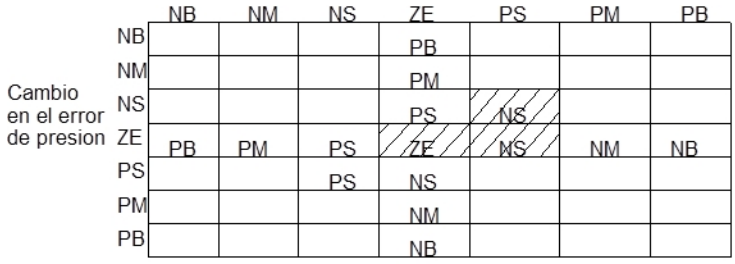


Figura 6.13 Reglas del FIS de Mamdani del primer sistema de control automático basado en reglas difusas

Las tres reglas previamente presentadas (o varias) pueden ser disparadas al mismo tiempo en diferentes grados. La acción final del sistema surge a partir de la consideración conjunta (desdifusión) de las salidas recomendadas por cada una de las reglas. Tal como puede verse en la figura 6.14.

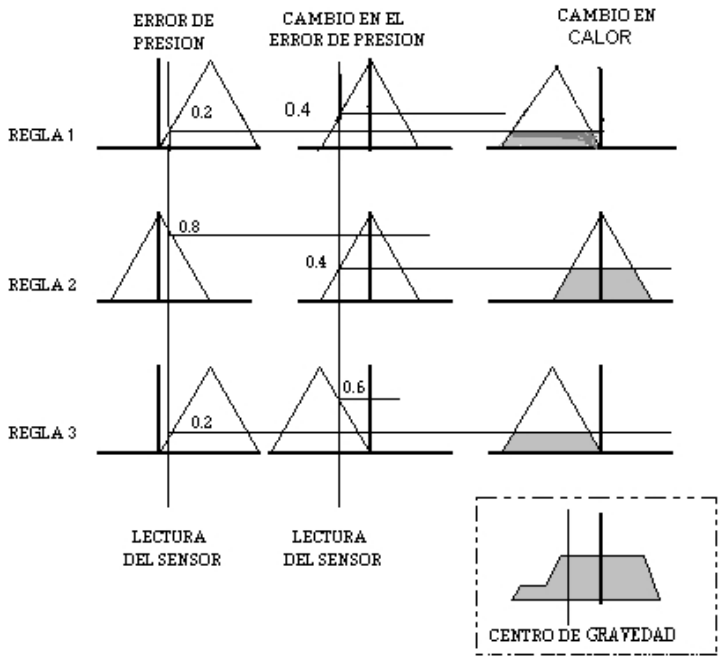


Figura 6.14 Ejemplo de un proceso de desdifusión a partir de tres reglas dadas

Los sistemas difusos son sistemas expertos, ya que se requiere del conocimiento de un experto humano para establecer las reglas, que forman el conocimiento del sistema.

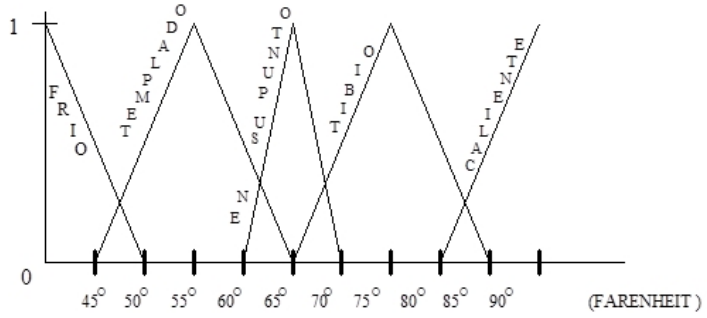
6.2.2. Construcción de un Sistema Difuso

Un sistema difuso se construye en cuatro pasos

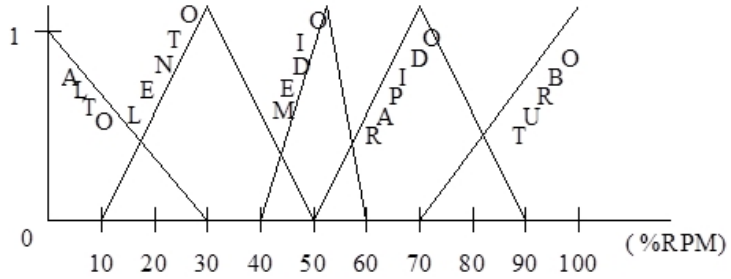
1. **Se escogen los nombres o variables** (X,Y, etc.). Por ejemplo, X=Temperatura, Y= Cambio en la velocidad de un motor para A. C.

2. **Se escogen los conjuntos difusos para variables de entrada y de salida**. Aquí se definen los subconjuntos difusos de los nombres X, Y. Éstos son llamados funciones de membresía y se dibujan como curvas, trapezoides o triángulos. Por ejemplo: los cinco conjuntos difusos sobre X son *frío, templado, en su punto, tibio* y *caliente*. Los conjuntos

más amplios son menos importantes. Para controles finos se necesitan conjuntos angostos.



Mientras que los conjuntos difusos para Y son: *alto, lento, medio, rápido,* y *turbo* para la velocidad del motor.

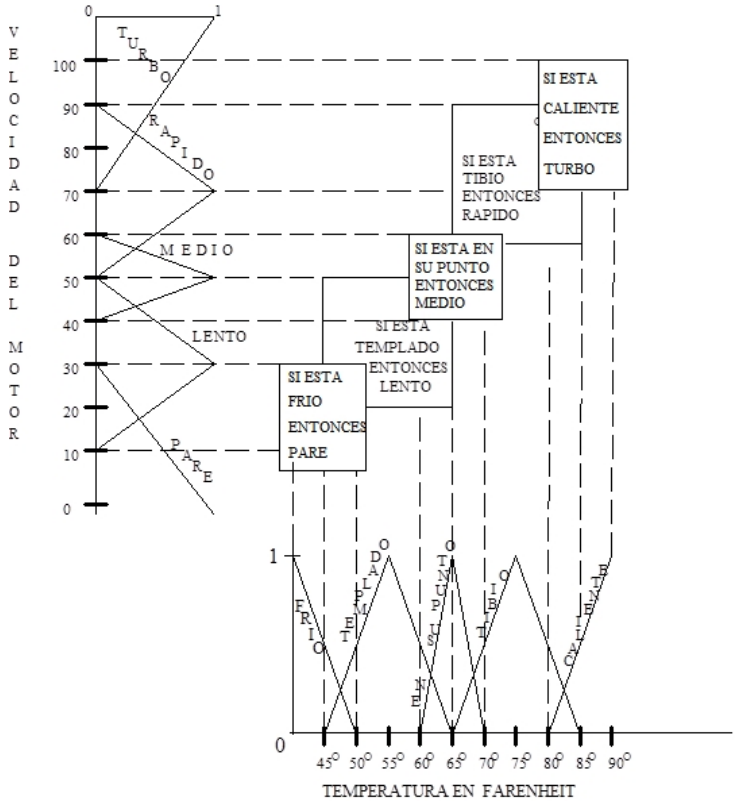


3. **Se establecen las reglas difusas**. Este paso asocia a los conjuntos de la velocidad del motor con los conjuntos de la temperatura. Se tiene que asignar un conjunto de velocidad a cada conjunto de temperatura. Esto da 5 reglas:

   REGLA 1: Si la temperatura esta fría, la velocidad del motor para.
   REGLA 2: Si la temperatura es templada, la velocidad del motor es lenta.
   REGLA 3: Si la temperatura está en su punto, la velocidad del motor es media.
   REGLA 4: Si la temperatura es tibia, la velocidad del motor es rápida.
   REGLA 5: Si la temperatura es caliente, la velocidad del motor es turbo.

Las reglas difusas parecen parches. En matemáticas, a este parche se llama el **producto de dos triángulos**.

Los cinco parches cubren muchas líneas (Un sistema lineal) que corre de la esquina baja izquierda a la superior derecha. El sistema no es lineal, pero se aproxima a uno lineal.



4. **Se selecciona el método de desdifusión**. Los métodos de desdifusión se utilizan cuando se requiere obtener un valor de salida duro (crisp) $y*$ de un sistema de inferencia difuso. La desdifusión es la conversión de una cantidad difusa en una cantidad precisa.

   Algunos de los métodos de Desdifusión en disponibles en Matlab son los siguientes

   - Centroide Ponderado (COA, Center of Area): Calcula el promedio

ponderado de un conjunto difuso.

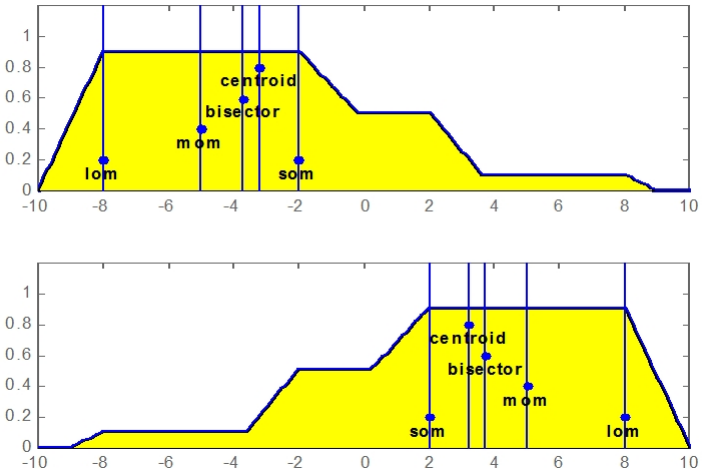
$$y^* = \sum \frac{\mu_A(y) * A_c}{\mu_A(y)}, \tag{6.4}$$

donde A_c es el centroide del conjunto A.

- Media de Máximos (Mean of Maxima, moM)

$$y^* = m + \frac{M}{2}, \tag{6.5}$$

donde m y M son los valores más pequeño y más grande de los máximos de y.



6.2.3. Conceptos Genéricos de Control Difuso

El concepto de un control difuso es una manera no convencional del diseño de un control retroalimentado. Debido a que el control difuso es realmente un sistema basado en conocimientos, la estrategia de control debe ser pensada en términos de reglas. Las reglas representan acciones de control y son establecidas en base a las experiencias de expertos en el manejo del sistema a controlar, o en base a nuestra intuición. Hay dos maneras principales de desarrollar las reglas con el conocimiento adecuado.

1. Utilización de conocimiento de sentido común.

2. Adquisición del conocimiento a través de simulación interactiva.

En el primer caso, el conocimiento de este tipo está disponible fácilmente, y puede ser estructurado dentro de un marco más formal. Ésta es la forma utilizada más ampliamente. Se deben tomar en cuenta las siguientes consideraciones:

1. Las reglas pueden ser desarrolladas para problemas relativamente pequeños que involucren pocas variables. Al incrementar su número también aumenta el esfuerzo para establecer reglas transparentes y significativas. Bajo estas circunstancias, usualmente se puede terminar con protocolos de control incompletos y/o inconsistentes, incluyendo, muchas veces, reglas conflictivas.

2. Las reglas se aplican fácilmente a sistemas que de algún modo cumplen con nuestra intuición sobre cómo se debe tomar la acción de control. El mismo tipo de reglas cualitativas no pueden ser completamente relevantes cuando se considere otra clase de sistemas como los que presente retrasos en las variables del sistema. Esto es por qué un sistema así no responde al control en la forma esperada.

El segundo enfoque es una forma poderosa y flexible de adquirir conocimiento. En este caso, el diseñador debe de interactuar con una serie de simulaciones de computadora y jugar con el sistema simulado a través de un modelo detallado que aprende o prueba el protocolo de control. Después de una serie de experimentos, con los cuales se refina el conocimiento del sistema de control, este se puede transformar en una estructura formal de reglas.

Referencias

Jang, J.-S. R., Sun, C.-T., and Mizutani, E. (1997). *Neuro-fuzzy and soft computing; a computational approach to learning and machine intelligence*. Prentice Hall.

Kosko, B. (1990). Fuzziness vs. probability. *International Journal of General System*, 17(2-3):211–240.

Kosko, B. (1992). Neural networks and fuzzy systems: a dynamic systems approach to machine intelligence. *Englewood Cliffs, NY Prentice Hall*.

Kosko, B. (1998). Neural networks and fuzzy systems. *Acoustical Society of America Journal*, 103:3131.

Kosko, B. and Toms, M. (1993). *Fuzzy thinking: The new science of fuzzy logic*. Hyperion New York.

McNeill, D. and Freiberger, P. (1994). *Fuzzy logic: The revolutionary computer technology that is changing our world*. Simon and Schuster.

Pal, S. K. and Mandal, D. P. (1992). Linguistic recognition system based on approximate reasoning. *Information Sciences*, 61(1):135–161.

Reyes-García, C. A. (1994). *On the design of a fuzzy relational neural network for automatic speech recognition*. PhD thesis, Florida State University.

Sivanandam, S., Sumathi, S., Deepa, S., et al. (2007). *Introduction to fuzzy logic using MATLAB*, volume 1. Springer.

Zadeh, L. (1965). Fuzzy sets. *Information Sciences*, 8:338–353.

Zadeh, L. A. (1988). Fuzzy logic. *Computer*, 21(4):83–93.

Capítulo 7

Relaciones Difusas y Redes Neuronales Relacionales

C. Reyes-García, A. A. Torres-García
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

7.1. Relaciones Matemáticas

La definición intencional de una relación R del conjunto x al conjunto y es como una oración abierta o incompleta, con dos espacios vacíos (Bandler and Kohout, 1987).

R = " _____ es el tío de _____"

Cuando el primer espacio es llenado insertando un elemento x de X, y el segundo espacio insertando un elemento y de Y, resulta una proposición que puede ser cierta o falsa.

Si es cierta, se dice que x está en la relación R a y, ${}_xR_y$. Mientras que, si es falsa, se dice que x no está en la relación R a y, ${}_x\neg R_y$.

El conjunto satisfacción R_s de una relación R consiste en todos los pares (x, y) de elementos para los cuales ${}_xR_y$, eso es

$$R_s = \{(x, y) \in X * Y |\ {}_xR_y\}.$$

Éste es claramente un subconjunto del producto cartesiano $X * Y$, que consiste en todos los pares posibles (x, y).

La relación también puede ser dada por medio de su matriz R_M, con componentes

$$R_{ij} \begin{cases} 1 \text{ si} & x_i \; R \; y_i \\ 0 \text{ si} & x_i \; \neg R \; y_i \end{cases} \tag{7.1}$$

La relación R puede ser representada por medio de una figura de flechas (*arrow picture*) R_A.

Ejemplo:

X = Goyo, Jaime, Karlos

Y = Ana, Lola, María, Sara

R = "_____ asiste a clases con _____"

R_s = (G,A),(G,M),(J,L),(J,M), **Conjunto satisfacción**.

Con base en los datos previos, se tiene que la matriz R_M es la siguiente,

$$R_M = \begin{array}{c|cccc} & A & L & M & S \\ G & 1 & 0 & 1 & 0 \\ J & 0 & 1 & 1 & 0 \\ K & 0 & 0 & 0 & 0 \end{array}$$

Mientras que la representación con flechas, R_A se observa en la figura 7.1.

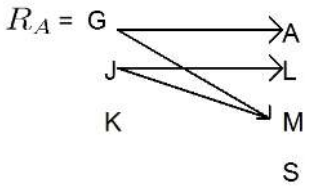


Figura 7.1 Representación gráfica de la relación R, *asistir a clases con*

Hay casos en que el conjunto X sea el mismo que el conjunto Y, los cuales son casos especiales particularmente importantes. Las relaciones de X a X son llamadas relaciones sobre X. Una representación de esta relación es por medio de una gráfica dirigida o digrafo R_D.

A continuación se presenta un ejemplo de un digrafo para la relación de X a X.

X = Ana, Beatriz, Clara, Diana, Elena ;

R = "A _____ le gusta convivir con _____".

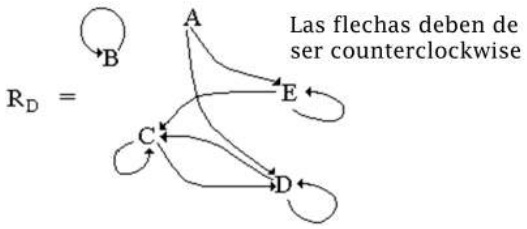


Figura 7.2 Representación gráfica de un diagrafo

7.1.1. Operaciones de inclusión

Sea $R(X \to Y)$ el conjunto de todas las relaciones binarias duras (CRISP) de X a Y. Mientras que la negación de $R \in R(X \to Y)$ es la relación $\neg R$, dada por

$$ {}_x\neg R_y \Longleftrightarrow \text{no es el caso que } {}_xR_y. \tag{7.2}$$

La conversa R^T de R que es una relación en dirección opuesta, de X a Y está dada por

$$ {}_yR^T_x \Longleftrightarrow_x R_y. \tag{7.3}$$

Esta relación también se conoce como inversa y se representa como R^{-1}.

La intersección $R \cap S$ y la unión $R \cup S$ reflejan semánticamente los conectivos lógicos AND y OR inclusivo entre las oraciones abiertas R y S.

Una relación R es una subrelación de T o está incluída en T, $R \subseteq T$, si y sólo si ${}_xR_y$ siempre implica ${}_xT_y$.

$$R \to T.$$

Ejemplo:

Dadas las siguientes dos relaciones R= *x es sobrino de y* y T= *x es pariente de y*, entonces en este caso, se tiene el hecho de que cuando *x sea sobrino de y* siempre implicará que *x es pariente de y* por lo que se puede decir que $R \to T$.

7.2. Productos Relacionales

Sea R una relación de X a Y y S una de Y a Z, hay varias operaciones binarias aplicables sobre ellas; cada una resultando en una relación producto del conjunto X al conjunto Z.

Producto Circulo o Circlet: $R \circ S$, para el cual x está en la Relación $R \circ S$ a z, si y sólo si hay al menos una y tal que ${}_xR_y$ y ${}_yS_z$:

$$ {}_x(R \circ S)_z \Longleftrightarrow \exists y \in Y \quad \ni \quad ({}_xR_y \wedge \ {}_yS_z). $$

Entonces ${}_x(R \circ S)_z$ si y sólo si hay un camino de x a z en la figura compuesta.

$$ (R \circ S)_{ik} = max_j(min(R_{ij}, S_{jk})). $$

Para la notación matricial R_M y S_M

$$ (R \circ S)_{ik} = \bigvee_j (R_{ij} \wedge S_{jk}). $$

Tabla 7.1 Otros productos relacionales

productos relacionales	símbolo
subtriángulo	$\lhd$
supertriángulo	$\rhd$
cuadrado	$\Box$

Producto Subtriángulo: x está en la relación $R \lhd S$ a z IFF, para cada y, ${}_xR_y$ implica que ${}_yS_z$. Para las matrices,

$$ (R \lhd S)_{ik} = min_j(R_{ij} \rightarrow S_{jk}), $$

Donde $\rightarrow$ es el operador de implicación material booleana dado en la tabla de verdad (ver tabla 7.2):

Tabla 7.2 Tabla de verdad del operador de implicación

P	Q	P $\rightarrow$ Q
0	0	1
0	1	1
1	0	0
1	1	1

Producto Supertriángulo: es un producto dual al anterior dado por:

$$ (R \rhd S)_{ik} = min_j(R_{ij} \leftarrow S_{jk}), $$

$$= (R^T \lhd S^T)^T_{ik}$$

Producto Cuadrado: es esencialmente la intersección de los dos anteriores, x está en la relación $R \square S$ con z, iff, para cada y, ${}_xR_y$ exactamente cuando ${}_yS_z$:

$$(R \square S)_{ik} = min_j(R_{ij} \leftrightarrow S_{jk}).$$

A continuación se describen algunos ejemplos de aplicación de productos relacionales en Sistemas Expertos.

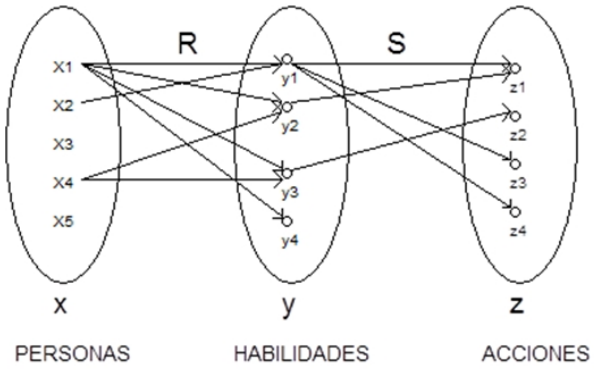
1.- Sea X es un conjunto de pacientes, Y un conjunto de sintomas, y Z es un conjunto de enfermedades. Entonces se tiene que,

- ${}_xR_y \Longleftrightarrow$ El paciente x muestra el síntoma y.
- ${}_yS_z \Longleftrightarrow$ El síntoma y es uno de los que caracterizan la enfermedad z.
- ${}_xR \circ S_z \Longleftrightarrow$ x tiene al menos uno de los síntomas de la enfermedad z.
- ${}_xR \lhd S_z \Longleftrightarrow$ Cada síntoma de x es uno de aquellos de la enfermedad z.
- ${}_xR \rhd S_z \Longleftrightarrow$ x tiene todos los síntomas de la enfermedad z, y quizás más.
- ${}_xR \square S_z \Longleftrightarrow$ x tiene exactamente los síntomas de enfermedad z y no otros.

2.- Sea X un conjunto de personas, Y un conjunto de habilidades, y Z es un conjunto de acciones. Entonces se tiene que,

- ${}_xR_y$ es x domina (o tiene la habilidad) y.
- ${}_yS_z$ es y es necesaria para llevar a cabo z.
- ${}_xR \circ S_z \Longleftrightarrow$ la persona x tiene al menos una habilidad necesaria para realizar la acción z.
- ${}_xR \lhd S_z \Longleftrightarrow$ las habilidades de x están entre aquellas necesitadas para realizar la acción z.
- ${}_xR \rhd S_z \Longleftrightarrow$ x tiene todas las habilidades necesarias para completar la acción z, y quizás más.

- ${}_xR\Box S_z \Longleftrightarrow$ x tiene exactamente las habilidades necesarias para realizar la acción z.



7.3. Relaciones Difusas

Una relación difusa es una que se establece entre dos elementos en algún grado entre 0 y 1. Así pues dados los siguientes conjuntos:
X es un conjunto de personas
Y es un conjunto de atributos o llaves
Z es un conjunto de objetos

Entonces se tiene que,

- ${}_xR_y$ se convierte en "sujeto x posee la llave y al grado μ_1".
- ${}_yS_z$ se convierte en "la llave y es necesaria para accesar el objeto z al grado μ_2".
- ${}_xR \circ S_z \Longleftrightarrow$ grado al cual sujeto x posee al menos una llave necesaria para accesar al objeto z.
- ${}_xR \triangleleft S_z \Longleftrightarrow$ grado al cual las llaves de x están entre aquellas necesarias para accesar al objeto z.
- ${}_xR \triangleright S_z \Longleftrightarrow$ grado al cual las llaves de x incluyen todas aquellas necesarias para accesar al objeto z y quizá más.

- ${}_xR\Box S_z \iff$ grado al cual las llaves de x son exactamente aquellas necesarias para accesar al objeto z.

7.3.1. Operadores de implicación

Para calcular los productos relacionales difusos, se puede observar la presencia de los operadores de implicación y de implicación doble. Por lo que, a continuación se ilustran algunas maneras de obtener las versiones difusas de dichos operadores.

Para el operador de implicación difuso se han propuesto las siguientes opciones para calcularlo,

- Gaines (G)
$$a \to b = min(1, \frac{b}{a}).$$
- Ł (Łukasewicz)
$$a \to b = min(1, 1 - a + b).$$
- KD (Kleene-Diens)
$$a \to b = (1 - a) \vee b = max((1 - a), b).$$
- Yager:
$$a \to b = \begin{cases} 1 & \text{si } y = 0 \text{ y } x = 0 \\ b^a & \text{en caso contrario.} \end{cases}$$

Mientras que la versión difusa del operador de doble implicación (también llamada bicondicional) se puede obtener como sigue,

- Ł (Łukasewicz) $\leftrightarrow$
$$a \longleftrightarrow b = min(1, 1 - |a - b|).$$

7.4. Red relacional difusa

La red neuronal relacional difusa (FRNN, por sus siglas en inglés) (Reyes-García, 1994), basada en la estructura propuesta por Pedrycz (Pedrycz,

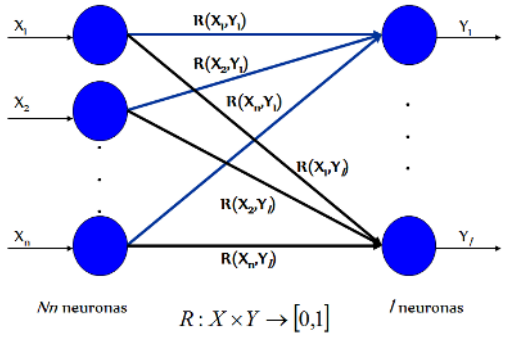


Figura 7.3 Arquitectura de una FRNN

1991). Entre sus ventajas está el permitir tener regiones de decisión no lineales y traslapadas. debido a que está basada en la teoría de conjuntos difusos.

La FRNN está formada por dos capas: la capa de entrada y la capa de salida. En la capa de entrada se tienen $N \times n$ neuronas, donde n es el número de características que describen al patrón de entrada y N es el número de propiedades lingüísticas asociadas a cada característica. Mientras que la capa de salida está formada por k neuronas, cada una asociada a una de las k clases.

Cada neurona de la capa de entrada de la FRNN está conectada con cada neurona de la capa de salida. Las conexiones son descritas por medio de pesos, o relaciones difusas, $R : X \times Y \rightarrow [0, 1]$ de las neuronas de la capa de entrada a las neuronas de la capa de salida. La figura 7.3 ilustra de mejor manera lo anterior. Al igual que otras redes neuronales, la FRNN tiene dos etapas, una para el aprendizaje y la otra para el procesamiento. Durante la etapa de aprendizaje se determinan las relaciones difusas entre un conjunto de ejemplos de entrada y las clases. Mientras que en la etapa de procesamiento se utilizan las relaciones difusas, obtenidas en la etapa de aprendizaje, para clasificar ejemplos desconocidos (no usados en el aprendizaje). A continuación se describen ambas etapas.

7.4.1. Etapa de entrenamiento

Esta etapa requiere como entrada un conjunto de ejemplos de entrenamiento, los cuales están descritos por n características. Además, se compone de tres módulos: el extractor de características lingüísticas, el estimador de salida

deseada y el entrenamiento de la red neuronal.

El extractor de características lingüísticas transforma las n características en $N \times n$ características, de tal manera que cada una de las n características se asocia con N propiedades lingüísticas, es decir, cada característica se representa con su valor de pertenencia a cada una de las propiedades lingüísticas. Los valores de pertenencia son obtenidos usando alguna función de pertenencia de las más comunes como: triangulares, trapezoidales, gaussianas entre otras.

Mientras que el estimador de salida deseada se encarga de calcular el grado de pertenencia de cada ejemplo del conjunto de entrenamiento a cada una de las k clases. Para definir el valor de pertenencia, primeramente, se calcula la distancia pesada del j-ésimo ejemplo a la l-ésima clase, de las k clases, como se muestra en la siguiente ecuación:

$$z_{jl} = \sqrt{\sum_{i=1}^{n} \left(\frac{F_{ji} - \mu_{li}}{\sigma_{li}} \right)^2} \text{ para } l = 1, \ldots, k.$$

donde F_{ji} es la i-ésima característica del j-ésimo ejemplo, μ_{li} y σ_{li} representan, respectivamente, la media y desviación estándar de la i-ésima característica de la l-ésima clase.

Una vez que la distancia pesada es calculada, el valor de pertenencia del j-ésimo ejemplo a la l-ésima clase se calcula como sigue:

$$\mu_l(F_j) = \frac{1}{1 + (\frac{z_{jl}}{f_d})^{f_e}} \tag{7.4}$$

donde f_e es el generador exponencial difuso y f_d es el generador denominacional difuso, los cuales controlan la cantidad de difusión (el valor de pertenencia).

Como se observa en la ecuación 7.4, una mayor distancia de una instancia a la clase es representada por un valor de pertenencia menor a esa clase. Esto significa que una instancia puede pertenecer con diferentes grados a más de una clase, dado que los límites de las regiones de decisión son difusos.

En cuanto al entrenamiento de la FRNN se refiere, éste tiene como objetivo modificar los pesos (relaciones difusas) de las conexiones de la capa de entrada con la capa de salida. Para ello, la salida de la red es obtenida y se calcula el error en la capa de salida durante un número finito de iteraciones (épocas). El error en cada iteración es representado como la diferencia entre la salida real y la salida deseada. Al final del proceso de entrenamiento, la

red neuronal es una matriz que contiene las relaciones difusas para poder mapear futuros ejemplos a las correspondientes clases.

Para lograr lo anterior se realiza una modificación iterativa de los pesos con base en los siguientes pasos (Reyes-García, 1994):

1. Calcular la salida de la FRNN usando la composición *máx-mín* (producto relacional *circlet*), usando la ecuación siguiente:

$$Y_l = \left[\bigvee_{i=1}^{n} (X(x_{ji}) \wedge R(a_{il}))\right] \bigvee V(v_l), \tag{7.5}$$

   donde $X_j = \{x_1, x_2, \ldots, x_n\}$ y $Y = \{y_1, y_2, \ldots, y_k\}$ son, respectivamente, el conjunto de neuronas en la capa de entrada y en la capa de salida; R representa la matriz de pesos o relaciones difusas de X a Y, y $V = \{v_1, v_2, \ldots, v_k\}$ representa el *bias* para cada clase. Este último representa el valor mínimo con el cual una instancia pertenece a una clase.

2. Calcular el incremento de los valores de las relaciones difusas usando las siguientes fórmulas,

$$\Delta a_l = \frac{\partial Q}{\partial a_l} = -\sum_{j:y_j>t_j} \frac{\partial f(x_j; a_l * v_l)}{\partial a_l} + \sum_{j:y_j<t_j} \frac{\partial f(x_j; a_l * v_l)}{\partial a_l}, \tag{7.6}$$

   donde,

$$\frac{\partial f(x_j; a_l * v_l)}{\partial a_l} = \begin{cases} 1 & \text{Si (a) } \bigvee_{i=1}^{n} (a_{il} \wedge x_{ji}) \geq v_l \\ & \text{(b) } \bigvee_{i=1, j\neq s}^{n} (a_{il} \wedge x_{ji}) \leq (a_{sl} \wedge x_{js}) \\ & \text{(c) } a_{sl} \leq x_{js} \\ 0 & \text{en otro caso} \end{cases} \tag{7.7}$$

   donde $\vee$ representa la unión difusa (máximo) y $\wedge$ la intersección difusa (mínimo). Se observa que la diferencial vale 1 sólo si se cumplen las tres reglas, de lo contrario, vale 0.

3. Calcular el incremento para el vector de *bias* tal como indica la ecuación 7.8.

$$\Delta v_l = \frac{\partial Q}{\partial v_l} = -\sum_{j:y_j>t_j} \frac{\partial f(x_j; a_l * v_l)}{\partial v_l} + \sum_{j:y_j<t_j} \frac{\partial f(x_j; a_l * v_l)}{\partial v_l}, \tag{7.8}$$

donde,

$$\frac{\partial f(x_j; a_l * v_l)}{\partial v_l} = \begin{cases} 1 & \text{Si } \bigvee_{i=1}^{n} (a_{il} \wedge x_{ji}) \leq v_l \\ 0 & \text{en otro caso} \end{cases} \tag{7.9}$$

En las ecuaciones 7.6 y 7.8 t_j representa el valor de la salida real, y y_j la salida deseada.

4. Actualizar los valores de las relaciones difusas y del *bias* con base en las ecuaciones 7.10 y 7.11.

$$a(m+1) = a(m) + \Psi_1(m) \left[\frac{\Delta a(m+1) + \eta \Delta a(m)}{Nn} \right] \tag{7.10}$$

$$v(m+1) = v(m) + \Psi_2(m) \left[\frac{\Delta v(m+1) + \eta \Delta v(m)}{Nn} \right] \tag{7.11}$$

donde Nn es el número de características difusas, m la época actual, Ψ_1 y Ψ_2 (tal que $\Psi_1(m) = \Psi_2(m) = \frac{1}{\sqrt{m+1}}$ son funciones no crecientes que controlan la influencia en el incremento tanto de Δa_l como de Δv_l, y η es la tasa de aprendizaje que especifica el nivel de modificación de los parámetros a aprender.

Como se ha mencionado, cuando el entrenamiento concluye, la salida es una matriz R que contiene los valores de las relaciones difusas que servirán para mapear nuevos ejemplos (no usados para entrenar la red) a las correspondientes clases.

7.4.2. Etapa de procesamiento

Cuando la etapa de aprendizaje se ha completado, la información que se obtuvo es utilizada para clasificar ejemplos desconocidos durante la etapa de procesamiento. Los módulos que forman esta etapa son, extractor de propiedades lingüísticas, clasificador difuso y módulo de toma de decisiones.

El primer módulo, el extractor de propiedades lingüísticas, es similar al descrito en la etapa de aprendizaje; sin embargo, en la etapa de procesamiento no se calculan los parámetros de las funciones de pertenencia. Los valores de pertenencia son calculados usando las funciones de pertenencia correspondientes en la etapa de entrenamiento.

La FRNN usa los valores de pertenencia calculados por el extractor de propiedades lingüísticas, de la etapa de procesamiento, y la matriz de relaciones R para clasificar nuevas instancias a sus correspondientes clases. La clasificación puede ser llevada en tres formas diferentes (Reyes-García, 1994):

- Mediante la composición máx-mín tal como se expresa en la siguiente ecuación:

$$Y(y_j) = \bigvee [X(x_{ji}) \wedge R(a_{il})] = max[min(X(x_{ji}); R(a_{il}))].$$

- Reemplazando el operador *min* de la expresión anterior por la media geométrica para obtener la salida *max-media geométrica*, tal como se expresa a continuación:

$$Y(y_j) = max[(X(x_{ji}) * R(a_{il}))^{\frac{1}{2}}].$$

- Usando el producto relacional cuadrado definido por:

$$Y(y_j) = min(X(x_{ji}) \longleftrightarrow R(a_{il})),$$

  donde $\longleftrightarrow$ es el operador de doble implicación calculado con base en la fórmula de Łukasiewicz (Bandler and Kohout, 1980). Por lo que la formula anterior se puede reescribir como sigue,

$$Y(y_j) = min(1 - |X(x_{ji}) - R(a_{il})|).$$

En todos los casos, se obtiene como salida un vector que contiene los grados de pertenencia de la instancia evaluada a las diferentes clases. El tipo de salida puede ser determinado a priori o mediante un algoritmo genético. Finalmente, el módulo de toma de decisiones asigna la instancia a la clase con el valor de pertenencia más alto. Con esto se puede evaluar si la clase asignada es igual a la clase deseada para determinar si la clasificación de dicha instancia fue correcta.

Referencias

Bandler, W. and Kohout, L. (1987). Relations, mathematical. *Systems and Control Encyclopedia*, pages 4000–4008.

Bandler, W. and Kohout, L. J. (1980). Fuzzy relational products as a tool for analysis and synthesis of the behaviour of complex natural and artificial systems. In *Fuzzy Sets*, pages 341–367. Springer.

Pedrycz, W. (1991). Neurocomputations in relational systems. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (3):289–297.

Reyes-García, C. A. (1994). *On the design of a fuzzy relational neural network for automatic speech recognition.* PhD thesis, Florida State University.

Capítulo 8

Mapas Cognitivos Difusos y Sistemas Expertos Difusos

C. Reyes-García, A. A. Torres-García
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE)

8.1. Mapas Cognitivos Difusos

Ron Axelord introdujo los mapas cognitivos como una manera formal de representar el conocimiento científico social y modelar la toma de decisiones en sistemas sociales y políticos (Axelrod, 1976). El objetivo principal de construir un mapa cognitivo en torno a un problema es poder predecir el resultado dejando que los temas relevantes interactúen entre sí. Estas predicciones pueden usarse para averiguar si una decisión tomada por alguien es consistente con toda la colección de afirmaciones causales declaradas.

El término mapa cognitivo difuso (FCM) fue acuñado por Bart Kosko en 1986 (Kosko and Toms, 1993). Para describir un modelo de mapa cognitivo con dos características significativas:

1. Las relaciones causales entre los nodos son difusas. En vez de usar solamente los signos para indicar causalidad positiva o negativa, un número se asocia a la relación para expresar el grado de relación entre dos conceptos.

2. El sistema es dinámico con retroalimentación, donde el efecto de cambio en un nodo de concepto afecta a otros nodos, lo que a su vez puede

afectar al nodo que inicia el cambio. La presencia de retroalimentación añade un aspecto temporal al funcionamiento de la FCM.

La estructura de FCM puede ser vista como una red neuronal artificial recurrente, donde los conceptos son representados por neuronas y relaciones causales por enlaces ponderados o bordes que conectan las neuronas.

Un mapa cognitivo difuso (FCM) es una representación gráfica del conocimiento o percepción de un sistema dado. El mapeo cognitivo difuso es una combinación de lógica difusa y mapeo cognitivo. El mapeo cognitivo se basa en teoría de grafos, que es también la base de la mayoría de los cálculos e índices. Un FCM se compone de factores (conceptos/nodos) que representan los elementos importantes del sistema mapeado. Las líneas dirigidas marcadas con valores difusos muestran la fuerza de las condiciones causales entre los factores. Un mapa cognitivo difuso es un modelo de la estructura del sistema. En sí, un mapa cognitivo difuso dibuja una figura causal. Liga hechos, cosas y procesos a valores, políticas y objetos. Permite predecir y simular cómo interactúan y actúan eventos complejos.

Utilizando un método originado en enfoques de redes neuronales, la influencia de los factores entre sí se puede calcular iterativamente. Una vez que la red se ha estabilizado, los resultados muestran las tendencias dentro del sistema. FCM también ofrece la posibilidad de ejecutar simulaciones y calcular los resultados de posibles escenarios (Kosko and Toms, 1993).

El mapeo cognitivo difuso FCM es una herramienta poderosa de simulación para formalizar el entendimiento de relaciones conceptuales y causales, mediante la combinación de herramientas de mapeo conceptual con lógica difusa y otras técnicas originalmente desarrolladas para redes neuronales. Los FCMs permiten la representación y formalización de dominios de conocimiento complejo y subjetivo como como en los siguientes campos.

En agricultura los FCM se han usado para caracterizar el comportamiento del rendimiento en cosechas algodón (Papageorgiou et al., 2009). También se han utilizado para categorizar el nivel de producción de coco con base en un conjunto dado de condiciones agroclimáticas, estudiando el impacto de las variaciones climáticas y los parámetros meteorológicos en el comportamiento del rendimiento del coco aprovechando las capacidades de razonamiento y simulación de los FCM (Jayashree et al., 2015).

En medicina con FCM se ha construido el modelo de cáncer de próstata utilizando datos clínicos reales y luego se aplica este modelo a la predicción del estado de salud del paciente (Froelich et al., 2012). Mientras que, en

educación los FCM se han usado como una herramienta para crear meta-conocimiento y explorar las implicaciones ocultas de la comprensión del estudiante (Cole and Persichitte, 2000).

En economía y finanzas se han utilizado para abordar el problema del diseño de una herramienta de metodología de gestión del conocimiento que actúe como mecanismo de apoyo a la decisión para las empresas financieras geográficamente dispersas. La investigación subyacente aborda el problema de la captura y representación de la información en las instituciones financieras con el fin de proporcionar una implementación del ciclo virtuoso del flujo de conocimiento (Xirogiannis et al., 2004).

Existen más áreas de aplicación como en entornos de apoyo a la toma de decisiones, planificación estratégica de sistemas de información, análisis de inversiones en acciones y otras aplicaciones financieras, apoyo a la toma de decisiones en procesos de producción industrial y otras caracterizadas por su información compleja.

8.1.1. Ejemplo de aplicación de un FCM

Un mapa cognoscitivo difuso o FCM dibuja una figura causal. Liga hechos y cosas y procesos a valores, políticas y objetos. Permite predecir cómo interactúan y actúan eventos complejos (Kosko and Toms, 1993). El ejemplo de la figura 8.1 ha sido tomado de (Hilera-González and Martínez-Hernando, 2000).

Interpretación:

- Si la Conflictividad Social se incrementa, entonces la Estabilidad del Gobierno decrece (ver figura 8.2).
- Si la Inversión Extranjera se incrementa, entonces el Empleo se incrementa (ver figura 8.3).

Las conexiones entre conceptos tendrán un valor en el intervalo [-1, +1], que representa el grado de causalidad o influencia de un nodo sobre otro. Este grado de causalidad, así como la o las conexiones entre ellos, es establecido por especialistas en el dominio del mapa, o por la colección automática de datos. La información difusa sobre la influencia de un concepto sobre otro, puede ser establecida a partir de un conjunto de valores lingüísticos, asociados con valores numéricos correspondientes. Un ejemplo es mostrado en la siguiente tabla.

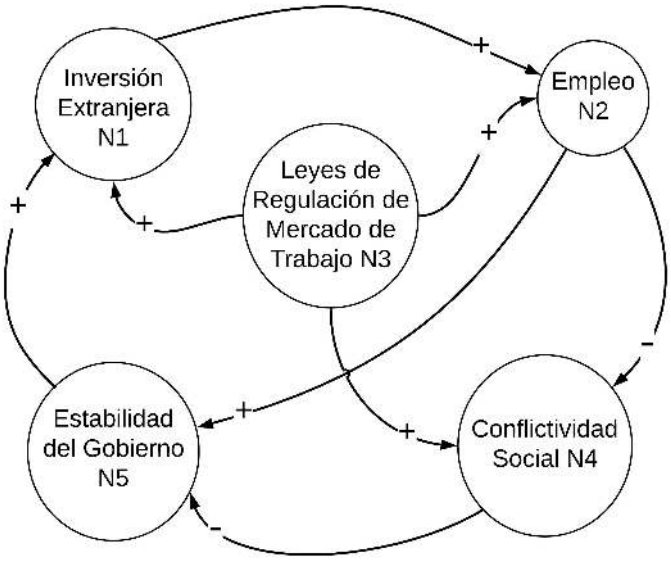


Figura 8.1 Ejemplo de mapa cognitivo

Figura 8.2 Relación entre la conflictividad social y la estabilidad del gobiernoAcá va una figura

Con base en este auxiliar, las opiniones de un experto humano pueden ser asignadas directamente a las conexiones del mapa convertidas en valores difusos.

Por otro lado, un mapa cognitivo difuso puede ser considerado como un tipo de red neuronal, donde los nodos representan neuronas, y las relaciones de causalidad las conexiones interneuronales. La red neuronal formada es una red monocapa similar a la red de Hopfield. El aprendizaje consiste en asignar directamente los pesos, que están representados por los valores propuestos por el experto humano para cada una de las relaciones causales. Una matriz

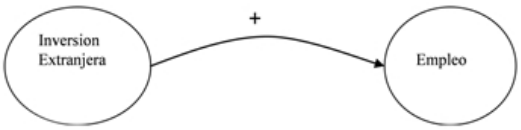


Figura 8.3 Relación entre la inversión extranjera y el empleo

Valores Lingüísticos	Valores Numéricos
Sin Efecto	0.0
Efecto Ligero	0.2
Efecto Moderado	0.4
Afecta	0.6
Efecto Considerable	0.8
Causa Directa	1.0

de pesos (W) para el mapa mostrado puede ser;

$$W = \begin{bmatrix} 0.0 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.0 & 0.2 \\ 0.8 & 0.4 & 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.8 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Después del aprendizaje se puede utilizar el mapa para simular el comportamiento del sistema ante la variación de los valores de evidencia de los conceptos.

Un caso hipotético de ejemplo (tomado de (Hilera-González and Martínez-Hernando, 2000)), puede ser el tratar de conocer el efecto que puede tener una fuerte inversión extranjera (N1=1) en plena crisis económica en un país con un gobierno muy inestable (N5=0), sin leyes para la regulación del mercado de trabajo (N3=0), con una gran conflictividad social (N4=1) y un gran nivel de desempleo (N2=0). En este caso, el vector que representa los valores actuales es:

$$\text{N} = (\text{N1}, \text{N2}, \text{N3}, \text{N4}, \text{N5}) = (1\ 0\ 0\ 1\ 0)$$

A partir de esto se efectúan una serie de multiplicaciones vector-matriz, hasta que el sistema alcance una situación de estabilidad. Para simplificar la evolución de los valores de activación de los nodos, se limitarán a valores 0 y 1, por lo que se usa el siguiente criterio. Si el valor resultante es negativo, el

valor del vector será 0, si es positivo el valor de salida es 1, y si es 0 la salida no cambia. De este modo se obtiene la siguiente secuencia de valores

$$\text{Red(t1)} = \text{N(t0)W}=(1\ 0\ 0\ 1\ 0) \begin{bmatrix} 0.0 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.0 & 0.2 \\ 0.8 & 0.4 & 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.8 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} = (0\ 0.6\ 0\ 0\ \text{-}0.8)$$

Aplicando el criterio anterior, obtenemos:

$$\text{Red(t1)} = (0\ 0.6\ 0\ 0\ \text{–}0.8) \Rightarrow \text{N(t1)} = (\ 1\ 1\ 0\ 1\ 0)$$

Repitiendo el proceso hasta alcanzar estabilidad, se obtiene:

$$\begin{gathered} \text{Red(t2)} = (0\ 0.6\ 0\ \text{-}1\ \text{–}0.6) \Rightarrow \text{N(t2)} = (1\ 1\ 0\ 0\ 0) \\ \text{Red(t3)} = (0\ 0.6\ 0\ \text{-}1\ 0.2) \Rightarrow \text{N(t3)} = (1\ 1\ 0\ 0\ 1) \\ \text{Red(t4)} = (0\ 0.6\ 0\ \text{-}1\ 0.2) \Rightarrow \text{N(t4)} = (1\ 1\ 0\ 0\ 1) \end{gathered}$$

A partir de la tercera iteración se puede observar que el sistema ya no cambia, lo que indica que ya se llegó a una situación de estabilidad. Interpretando estos resultados se puede decir, que gracias a la Inversión Extranjera (N1=1) se ha llegado a una situación de pleno empleo (N2=1), sin conflictos sociales (N4=0), donde el gobierno ha alcanzado gran estabilidad (N5=1), sin necesidad de leyes de regulación del mercado de trabajo (N3=0). Por lo que se puede deducir, a partir de los datos disponibles, que una fuerte inversión extranjera parece ser suficiente para sacar de la crisis a un país con las condiciones descritas inicialmente.

Si se desea colectar la opinión de varios especialistas, lo único que debe hacerse es aplicar la operación de unión difusa, es decir, seleccionar el valor máximo de los expresados por los diferentes expertos para cada una de las causalidades. Así, el valor final w_{ij} de la red neural, tomando en cuenta los valores propuestos por M expertos está representado por;

$$w_{ij} = \bigcup_{k=1}^{M} w_{ij}(k) = max(w_{ij}(1), w_{ij}(2), \ldots, w_{ij}(M)) \tag{8.1}$$

8.2. Sistema Experto Difuso

Un **Sistema Experto Difuso** es un sistema experto que usa Lógica Difusa en lugar de Lógica Booleana, es decir, es una colección de funciones de membresía y reglas que son usadas para razonar acerca de los datos. El desarrollo de la Lógica Difusa y su transformación en algo práctico tuvo que esperar al éxito en las pasadas décadas de los Sistemas Expertos, programas orientados a la consulta de temas muy especializados donde, los que introducen sus conocimientos como los que tratan de extraerlos suelen expresarse con un lenguaje humano basado en expresiones vagas.

Los Sistema Expertos Convencionales son Máquinas principalmente de razonamiento simbólico, a diferencia de los Sistemas Expertos Difusos, que están orientados hacia el procesamiento numérico y se basan en reglas difusas. Una regla difusa puede definirse como una sentencia condicional en la forma:

$$\text{SI } x \text{ es } A \text{ ENTONCES } y \text{ es } B.$$

Donde x e y son variables lingüísticas; mientras que A y B son valores lingüísticos determinados por conjuntos difusos en el universo de discurso X e Y, respectivamente.

La diferencia entre las reglas clásicas y difusas consiste básicamente en que una regla clásica IF-THEN utiliza lógica binaria, por ejemplo,

```
- SI la velocidad es > 100 ENTONCES distancia_de_frenado es larga
- SI la velocidad es <40 ENTONCES distancia_de_frenado es corta
```

La variable *velocidad* puede tener cualquier valor numérico entre 0 y 220 km/h, pero la variable *lingüística distancia_de_frenado* puede tomar tanto valor largo como corto. En otras palabras, las reglas clásicas se expresan en el lenguaje blanco y negro de la Lógica Booleana.

Sin embargo, también podemos representar las reglas de distancia de forma difusa:

```
- SI la velocidad es rápida ENTONCES distancia_de_frenado es larga
- SI la velocidad es lenta ENTONCES distancia_de_frenado es corta
```

Aquí la variable lingüística *velocidad* también tiene el rango (el universo del discurso) entre 0 y 220 km/h, pero esta gama incluye conjuntos difusos, tales como *lentos*, *medios* y *rápido*. El universo del discurso de la variable *lingüística distancia_de_frenado* puede estar entre 0 y 300 m., y puede incluir tales conjuntos difusos como *corto*, *medio* y *largo*. Así, las reglas difusas se

refieren a conjuntos difusos. Los sistemas expertos difusos combinan las reglas y consecuentemente reducen su número en por lo menos el 90 por ciento.

Otra marcada diferencia que presenta un Sistema Experto Difuso y un Sistema Experto Convencional, es precisamente, sobre el Proceso de Inferencia, el cual consiste básicamente de cuatro etapas, mismas que se describen a continuación:

1. DIFUSIÓN, las funciones de membresía definidas sobre las variables de entrada son aplicadas a sus valores actuales, para determinar el grado de verdad de cada premisa de la regla.

2. INFERENCIA, la inferencia difusa se puede definir como un proceso de mapeo de una entrada dada a una salida, utilizando la teoría de conjuntos difusos. Durante el mapeo el valor de verdad para el antecedente de cada regla es calculado, y aplicado a la parte del consecuente de cada regla. Este resulta en un subconjunto difuso para ser asignado a cada variable de salida para cada regla. Generalmente sólo MIN o PRODUCT son usados en reglas de inferencia. En la inferencia MIN corresponde a la interpretación tradicional de la operación lógica AND. En la inferencia PRODUCT es escalada por el grado de verdad de la premisa de la regla calculada.

3. COMPOSICIÓN, todos los subconjuntos difusos asignados a cada variable de salida son combinados para formar un subconjunto difuso por cada variable de salida. Generalmente MAX o SUM son usados, en la composición MAX el subconjunto difuso de salida es construido al tomar el punto máximo sobre todos los subconjuntos difusos asignados a variables por la regla de inferencia (en lógica difusa OR). En la composición SUM, el subconjunto difuso de salida es construido al tomar el punto de la suma de todos los subconjuntos difusos asignados a la variable de salida por la regla de inferencia.

4. DESDIFUSIÓN (opcional), se emplea cuando es útil convertir el conjunto de salidas difusas a un número preciso (número duro). Hay dos técnicas más comunes de desdifusión CENTROIDE PONDERADO y MÁXIMO. En el método del centroide el valor duro de la variable de salida es calculado al encontrar el valor de la variable del centro de gravedad de la función de membresía para el valor difuso. En el método del Máximo se toma el máximo valor de verdad de uno de los valores de las variables en cualquiera de los subconjuntos difusos.

Una muestra de las aplicaciones destacadas de los Sistemas Expertos Difusos se da a continuación. Se han aplicado Sistemas Expertos Difusos para la selección de la arquitectura, tecnología y características de un sistema demótico, modelado de fuerza de hilo de algodón, diagnóstico de enfermedades humanas, evaluación de desempeño de profesores, sistema experimental difuso para diabetes utilizando un mecanismo de veredicto difuso, diagnóstico de enfermedades infecciosas tropicales, asesor humano por internet, evaluación de la calidad del agua, estimación de la productividad en el trabajo de la construcción, diagnóstico enfermedades cardíacas, recuperación mejorada de petróleo, exploración integral de petróleo, diagnóstico médico, evaluación del rendimiento de capital intelectual, procesamiento de préstamos a pequeñas empresas, calificación de clientes para créditos bancarios, inversiones estratégicas, predicción de algunas características de calidad de jugo de uva concentrado, entre otras (Baghel and Sharma, 2013).

Beneficios de los Sistemas Expertos Difusos: Reducen el costo de desarrollo de aplicaciones, reducen los costos de ejecución, reducen los costos de mantenimiento. Estos beneficios se tienen porque los Sistemas Expertos Difusos son: más compactos (Requieren menos reglas), codifican conocimiento de alto nivel y en español (o cualquier otro idioma), son menos propensos a errores, manejan información vaga, incierta e imprecisa, y son mucho más fáciles de mantener.

Referencias

Axelrod, R. (1976). The analysis of cognitive maps. *Structure of decision*, pages 55–73.

Baghel, A. and Sharma, T. (2013). Survey on fuzzy expert system. *International Journal of Emerging Technology and Advanced Engineering*, 3(12):230–233.

Cole, J. R. and Persichitte, K. A. (2000). Fuzzy cognitive mapping: Applications in education. *International Journal of Intelligent Systems*, 15(1):1–25.

Froelich, W., Papageorgiou, E. I., Samarinas, M., and Skriapas, K. (2012). Application of evolutionary fuzzy cognitive maps to the long-term prediction of prostate cancer. *Applied Soft Computing*, 12(12):3810–3817.

Hilera-González, J. and Martínez-Hernando, V. (2000). *Redes neuronales artificiales: fundamentos, modelos y aplicaciones*. RA-MA.

Jayashree, L., Palakkal, N., Papageorgiou, E. I., and Papageorgiou, K. (2015). Application of fuzzy cognitive maps in precision agriculture: A case study on coconut yield management of southern india's malabar region. *Neural Computing and Applications*, 26(8):1963–1978.

Kosko, B. and Toms, M. (1993). *Fuzzy thinking: The new science of fuzzy logic*. Hyperion New York.

Papageorgiou, E. I., Markinos, A., and Gemptos, T. (2009). Application of fuzzy cognitive maps for cotton yield management in precision farming. *Expert systems with Applications*, 36(10):12399–12413.

Xirogiannis, G., Glykas, M., and Staikouras, C. (2004). Fuzzy cognitive maps as a back end to knowledge-based systems in geographically dispersed financial organizations. *Knowledge and Process Management*, 11(2):137–154.

Capítulo 9

Otras metodologías de la Inteligencia Artificial

9.1. Redes neuronales artificiales

Una red neuronal artificial (RNA) es un modelo de computadora diseñado para simular el comportamiento de redes neuronales biológicas y la forma en que el cerebro opera en procesos como el reconocimiento de patrones, procesamiento de lenguaje, y solución de problemas (Kröse et al., 1996).

Las redes neuronales están formadas por un conjunto de algoritmos orientados a identificar relaciones subyacentes en un conjunto de datos. Tienen la habilidad de adaptarse a entradas cambiantes de tal forma que la red produzca el mejor resultado posible sin la necesidad de rediseñar los criterios de salida.

Este tipo de arquitecturas se caracterizan por:

1. Tener un gran número de elementos de procesamiento muy simples similares a las neuronas.

2. Tener un gran número de conexiones con pesos asociados entre los elementos, que codifican el conocimiento de la red.

3. Control altamente distribuido y paralelo, y

4. Capacidad de aprender representaciones internas automáticamente al ajustar dinámicamente los pesos de las conexiones para alterar los resultados de la red.

La construcción de una red neural involucra las siguientes tareas:

1. Determinar las propiedades de la red, que son: la topología de la red (conectividad), los tipos de conexión, el orden de las conexiones, y el rango del peso.

2. Determinar las propiedades del nodo: el rango de activación y la función de transferencia o activación.

3. Determinar la dinámica del sistema: el esquema de inicialización de los pesos, la fórmula de cálculo de activación, y la regla de aprendizaje.

En resumen, las RNA aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos. Es así que las RNA pueden cambiar su comportamiento en función del entorno con el objetivo de producir por ellas mismas salidas consistentes a los nuevos ejemplos dados.

Las RNA generalizan automáticamente debido a su propia estructura y naturaleza. Estas redes pueden ofrecer, dentro de un margen, respuestas correctas a entradas que presentan pequeñas variaciones debido a los efectos de ruido o distorsión. Algunas RNA son capaces de agrupar datos, es decir, abstraer la esencia de un conjunto de entradas que aparentemente no presentan aspectos comunes o relativos.

Mayor información sobre redes neuronales se encontrará en el libro de Reconocimiento de Patrones y Procesamiento de Señales, también editado en colaboración con AMEXCOMP.

9.2. Algoritmos Evolutivos (AE)

Algoritmo Evolutivo es un término global usado para describir sistemas de solución de problemas de optimización y búsqueda que utilizan modelos computacionales que imitan procesos evolutivos como elementos clave para su procesamiento. Entre los algoritmos evolutivos más conocidos se encuentran los Algoritmos Genéticos, Programación Evolutiva, Estrategias Evolutivas y la Programación Genética.

Los AE mantienen una población de individuos que evolucionan de acuerdo a reglas de Selección y otros operadores, que se conocen como operadores operadores genéticos como la Cruza, Recombinación y Mutación.

El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad alrededor del mundo durante los últimos años. Se presentarán aquí los conceptos básicos que se requieren para abordarla, así como un sencillo ejemplo que permita a los lectores comprender cómo aplicarla al problema de su elección. Adicionalmente, se hablará acerca de los diversos ambientes de programación actuales basados en algoritmos genéticos y de las áreas abiertas de investigación. Mayor información sobre esta área se encuentra en el libro de Algoritmos Evolutivos, tambien editado en colaboración con AMEXCOMP.

Referencias

Kröse, B., Krose, B., Van der Smagt, P., and Smagt, P. (1996). *An introduction to neural networks*. University of Amsterdam, 8th edition.

Interacción Humano-Computadora y Aplicaciones en México se terminó el 30 septiembre de 2019.
A partir de 1 de diciembre de 2019 está disponible en formato PDF en la página de la Academia Mexicana de Computación:
http://www.amexcomp.mx