Universidad Nacional Autónoma de México
Programa de Maestría en Ciencias Matemáticas

USO DE ALGORITMOS GENÉTICOS EN ALGUNOS PROBLEMAS DE INFERENCIA BAYESIANA

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS MATEMÁTICAS

PRESENTA:
BRUNO RODRIGO GUTIÉRREZ DE LA PAZ

DIRECTOR DE TESIS:
DR. EDUARDO GUTIÉRREZ PEÑA
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS

Ciudad de México, Noviembre de 2018

Dedicado a la memoria de mis tías
Juanita y Susie

Agradecimientos

Este trabajo[1] que representa la culminación de mis estudios de maestría sólo fue posible con el invaluable apoyo de muchas personas. Mis papás y mi hermano, su motivación, compañía y excepcional apoyo cotidiano fue fundamental. Angie, su compañía y alegría me ayudó mucho. Mis maestros y amigos; y todos los equipos que me ayudaron con las complejas tareas de la maestría. María Emilia, quien siempre estuvo ahí, dispuesta a apoyarme.

De manera especial agradezco al Dr. Eduardo Gutiérrez quien dirigió esta tesis y con mucha paciencia me explicó bastantes temas con grandes cátedras sobre una hoja de papel en su escritorio. Su apoyo y consejos me ayudaron mucho, incluso fuera del ámbito académico. También agradezco al Dr. Ricardo Menchaca por todo su apoyo, en particular, su ayuda fue fundamental para abordar la parte de procesos de decisión de Markov.

A todos gracias.

Bruno Gutiérrez, Octubre de 2018.

[1]Realizado con el estímulo económico para ayudantes de investigadores del Sistema Nacional de Investigadores.

Índice general

Capítulo 1

Introducción

Algunos de los problemas actuales en estadística provienen de áreas como: procesos de decisión de Markov, *machine learning* (aprendizaje de máquina) y árboles de decisión. Para resolverlos, muchas de las técnicas tradicionales resultan poco eficientes debido a la gran cantidad de datos que hay que manejar, la cantidad tan grande de pasos que implicaría llevar a cabo, o la complejidad de las operaciones que hay que resolver. Es así que con mucha frecuencia se cae en el ámbito de los problemas con complejidad de tiempo polinomial no determinista (*NP problems*; véase el Apéndice), para muchos de los cuales no se ha encontrado una solución en tiempo polinomial. Esto implica que las soluciones ofrecidas son, en principio, imposibles de llevarse a cabo en la práctica. Por esta razón, es necesario explorar otras herramientas y técnicas que permitan resolver los problemas de una forma viable, o al menos aproximar a una solución aceptable. Esta tesis revisa en una técnica de búsqueda y optimización conocida como algoritmos genéticos, y hace énfasis en su implementación para resolver problemas de estadística.

Los algoritmos genéticos son un método basado en la teoría de la selección natural de Darwin, en donde las especies (a través de su proceso evolutivo) van mezclándose entre sí para adaptarse al ambiente y sobrevivir, ayudados por un factor genético clave: la mutación. Esta técnica ha mostrado su éxito en la solución de problemas dentro de áreas como la de las ciencias de la computación; sin embargo, es todavía un recurso relativamente poco explotado en los problemas de estadística.

En la literatura estadística tradicional encontramos pocas soluciones que impliquen el uso de esta herramienta de optimización; sin embargo, creemos que emplear los algoritmos genéticos para resolver problemas de estadística puede ser una he-

rramienta muy útil. A través de estas páginas se busca verificar esta creencia; y de resultar cierta, la invitación general sería la de hacer enriquecer la estadístca con el uso de los algoritmos genéticos. No obstante, el beneficio puede darse en ambos sentidos, ya que los conocimientos estadísticos también pueden emplearse para desarrollar mejoras en la teoría de los algoritmos genéticos. Podría ser el caso, por ejemplo, que utilizar información inicial (*prior*) para generar un algoritmo genético permita definir poblaciones iniciales que aprovechen información previa para incrementar la probabilidad de alcanzar el óptimo global.

Falta mucho por hacer y, en ese sentido, el propósito de este trabajo es recopilar algunas problemas actuales y sencillos de la estadística para mostrar una solución a través del aprovechamiento de los algoritmos genéticos; pese a eso, se pretende desarrollar un fundamento teórico sólido, ya que es una de las bases para su implementación en trabajos futuros.

En el Capítulo 2 se da una breve introducción a los algoritmos genéticos, desde su concepción, tomando como base el contexto biológico evolutivo, hasta las fases que conforman al algoritmo genético y que en resumen son: población inicial, selección, cruza, mutación y reemplazo. El capítulo concluye con un ejemplo donde se aplica un algoritmo genético para construir una frase o *string* en específico a partir de una cadena aleatoria de caracteres.

En el Capítulo 3 se discuten algunas ideas básicas de la estadística bayesiana. Primero se revisan muy brevemente los métodos estadísticos tradicionales para después explicar el método de inferencia bayesiana y la teoría de la decisión bayesiana. Por último se revisan algunos métodos bayesianos de clasificación, que para su estudio se separan en supervisados y no supervisados. Dentro de los métodos supervisados se encuentra CART, que es algoritmo basado en árboles de decisión. A este último, se le dedica una sección completa.

El Capítulo 4 se enfoca en la teoría de la decisión markoviana. Se introduce el concepto de red bayesiana y se explica la forma de inferir la estructura de estas redes cuando el problema lo amerita. En este contexto se exponen los problemas de decisión markovianos que pueden modelarse con una red de decisión. Un punto medular de este capítulo son los problemas de decisión secuencial, en los que el objetivo es maximizar la utilidad esperada a lo largo de la toma de decisiones sucesivas en ambientes de incertidumbre. Para resolver estos problemas se explora, entre otras técnicas, la programación dinámica y la de búsqueda directa dentro del espacio de

políticas a través de algoritmos genéticos. El siguiente tema que se expone es relativo a la toma de decisiones secuenciales cuando hay incertidumbre en el modelo que rige al problema; este tópico pertenece al campo del aprendizaje reforzado. Bajo estas mismas hipótesis se expone, por último, un método de búsqueda aleatoria simple que en algunos casos resulta ser muy eficiente para resolver este tipo de problemas.

En el Capítulo 5 se desarrollan dos aplicaciones. La primera consiste en crear un algoritmo genético sobre una población de árboles de clasificación para optimizar el método CART. La segunda aplicación resuelve de dos formas diferentes el problema del apostador que es un problema clásico de decisión secuencial en ambiente de incertidumbre. La primera solución se obtiene empleando la técnica de programación dinámica y la segunda buscando directamente una política óptima en el espacio de posibles políticas; esta búsqueda se realiza con un algoritmo genético.

Finalmente, en el Capítulo 6 se presentan las conclusiones de este trabajo. El Apéndice contiene una descripción general de los conceptos relacionados con la complejidad computacional.

Capítulo 2

Algoritmos Genéticos

2.1. Motivación

La teoría de la evolución de Darwin nos asegura que los individuos más aptos dentro de una especie son los que sobreviven; por otro lado, la mutación también juega un papel clave para que la especie pueda sobrevivir, pues a través ésta es que la especie adquiere rasgos que sus ancestros no tenían y, de este modo, podrían generarse individuos todavía más aptos.

Los algoritmos genéticos (AG) se inspiran en esta teoría evolutiva para optimizar una función determinada y de esa manera encontrar la solución de un problema. La idea es sencilla y análoga a la anterior. Veamos algunos conceptos antes de detallar el funcionamiento de los AG:

- **Cromosoma:** es una cadena de valores o un *string*. Cada cromosoma es una solución dentro del universo de posibles soluciones de un problema.

- **Gen:** es un valor o caracter dentro del cromosoma (usualmente toman los valores 0 o 1)

- **Población:** conjunto de cromosomas, que a su vez son un conjunto de posibles soluciones al problema.

En la Sección 2.2 se discute con más detalle el contexto biológico. Por ahora, los conceptos anteriores son suficientes para sentar una idea del funcionamiento de los algoritmos genéticos; que se da del siguiente modo:

1. **Se genera una población.** Una muestra de cromosomas por lo general es aleatoria dentro del universo de soluciones del problema. El número de individuos no debe ser muy alto porque el tiempo de cálculo podría crecer considerablemente, ni demasiado bajo como para que no haya suficiente diversidad en la población.

2. **Aptitud.** Se utiliza una función para medir el nivel de aptitud de cada individuo de la muestra (*fitness function*).

3. **Selección.** Se escogen dos padres dentro de la población. Hay diversos métodos para seleccionarlos dependiendo del problema que estemos atacando, pero por lo general se seleccionan dentro de los individuos más aptos.

4. **Cruza.** Se combinan las tiras genéticas de ambos padres para formar una descendencia. Usualmente su descendencia es de dos hijos. Así mismo hay varias técnicas para mezclar la información genética de los padres, siendo la más común la de seleccionar un punto intermedio en el cromosoma del padre 1 y del padre 2, y combinarlos a partir de ese punto.

   De ese modo, el hijo 1 tendría la primera parte del cromosoma del padre 1 y la segunda parte del cromosoma del padre 2, a su vez, el hijo 2 tendría la primera parte de la información del padre 2 y la segunda del padre 1. Véase la Figura 2.1.

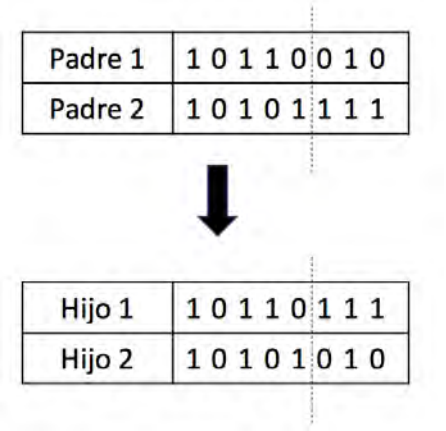


Figura 2.1: Mezcla de genes de los padres a los hijos. Figura tomada de (Sivanandam and Deepa, 2007)

5. **Mutación.** Se procede a mutar el cromosoma de los hijos. Esto se hace de manera aleatoria y con baja probabilidad de que ocurra. La mutación es parte importante de los AG pues es lo que impide que las soluciones se queden atrapadas en un máximo local; es decir, las mutaciones dan diversidad a la población ayudando a explorar todo el espacio de soluciones.
6. **Devolver los individuos a la muestra.** Por lo general, los nuevos individuos reemplazan a sus padres en la muestra original, pero hay varios métodos y el mejor depende del problema que se esté atacando.
7. **Volver al paso 2.** Cuando la solución a la que lleguemos sea la óptima, o el tiempo de búsqueda ha superado cierto límite, o han transcurrido varias generaciones sin obtener progresos significativos en la aptitud, entonces se detiene la búsqueda.
8. **Fin.**

Los AG han mostrado ser muy buenos resolviendo problemas *NP hard* (ver Apéndice) y otros problemas muy difíciles. Debemos recordar que son un método estocástico, por lo que no necesariamente debemos esperar encontrar "la mejor solución" sino debemos esperar encontrar "una buena solución".

Un sustento teórico de los AG lo encontramos en el Teorema de los Esquemas, que también se conoce como el "Teorema Fundamental de los Algoritmos Genéticos" (véase, por ejemplo Coello (2016) Capítulo 12). Sin entrar en detalles, este teorema nos dice que si tenemos un buen esquema, entonces su valor esperado incrementa conforme avanza el tiempo.

Por último, recordemos que el *No Free Lunch Theorem* nos dice que, en promedio sobre todos los problemas, todos los algoritmos de búsqueda funcionan igual (véase, por ejemplo Coello (2016) Capítulo 12). Saber esto es muy útil, pues nos asegura que dependiendo el problema, es el tipo de algoritmo que debemos utilizar. Hay que recordar que dentro de los algoritmos aleatorios, hay muchos métodos disponibles para llevar a cabo cada uno de los pasos anteriores (del 1 al 7), y dependiendo de nuestro problema se tendrá que estudiar cuál es la mejor selección.

2.2. Contexto biológico

En esta sección daremos un breve repaso sobre los conceptos provenientes del campo de estudio de la evolución y biología que también se emplean para referirse al

campo de estudio de los Algoritmos Genéticos. Lo explicado en esta sección proviene principalmente de Sivanandam and Deepa (2007) y parcialmente de Coello (2016).

2.2.1. La célula

Todas las células son compuestas por muchos y muy pequeños factores que trabajan juntos de manera compleja. El centro de todo el sistema es el núcleo de la célula y la información genética está contenida ahí.

2.2.2. Cromosomas

Dentro del núcleo se encuentran los cromosomas, en donde a su vez se encuentra almacenada toda la información genética. Cada cromosoma está compuesto por ácido desoxirribonucleico (ADN). Los cromosomas están divididos en muchas partes llamadas genes. Los genes poseen los códigos de las propiedades de la especie, es decir, dictan las características de cada individuo. Las posibilidades de los genes para alguna propiedad son llamados alelos y un gen puede tener diferentes alelos; así, cada una de las versiones de un gen es un alelo. El conjunto de todos los genes presentes en una población particular conforman a la reserva genética (*gene pool*); y si se toman en cuenta todos los posibles alelos de una reserva genética, se pueden determinar las posibles variaciones para las futuras generaciones.

El tamaño de la reserva genética ayuda a determinar la diversidad de los individuos en la población. El conjunto de todos los genes de una determinada especie se denomina genoma. Todos y cada uno de los genes tienen una posición única en el genoma, llamada locus. De hecho, la mayoría de los organismos vivos almacenan su genoma en una cantidad grande de cromosomas; pero en los AG todos los genes utilizables pueden ser almacenados en un mismo cromosoma. Esto implica que en el contexto de los AG, genomas y cromosomas son sinónimos.

2.2.3. Genética

Para un individuo en particular, el conjunto de todas las combinaciones de genes se llama genotipo, es "lo que potencialmente puede llegar a ser un individuo" (Coello, 2016). El fenotipo describe las características físicas que surgen al decodificar un genotipo. Un punto interesante de la evolución es que la selección siempre se hace sobre el fenotipo, mientras que la reproducción recombina el genotipo. Esta morfogénesis

juega un rol clave entre la selección y la reproducción.

En las formas superiores de vida (como es el caso de los humanos), el cromosoma contienen dos pares de genes. Esto se conoce como diploide. En caso de conflictos entre dos valores del mismo par de genes, el dominante determinará el fenotipo mientras que el otro gen, llamado recesivo, va a permanecer presente y podrá pasar a formar parte de la descendencia.

Los diploides permiten una amplia diversidad de alelos, lo que proporciona un mecanismo de memoria útil para el cambio en ambientes complejos o caóticos. Sin embargo, la mayoría de los AG comprenden cromosomas haploides porque éstos son mucho más simples de construir. En la representación haploide, sólo se almacena un lugar para cada gen, por lo que no se lleva a cabo el proceso de determinar cuál gen debería ser dominante y cuál recesivo.

2.2.4. Selección Natural

El surgimiento de las especies se basa en la preservación de variaciones favorables y el rechazo de desfavorables. Por variaciones nos referimos a las diferencias mostradas por individuos de una misma especie, e incluso cuando éstos son descendencia de los mismos padres.

Nacen más invididuos de los que pueden sobrevivir, por lo que existe una lucha continua por la vida, y los individuos con alguna ventaja tienen más oportunidades de sobrevivir. A esto se le conoce como la supervivencia del más apto. Como resultado, la selección natural juega un rol dominante en el proceso de supervivencia.

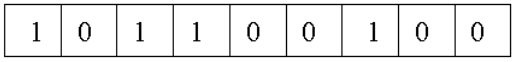
En el Cuadro 2.1 se da una lista de las diferentes expresiones que se tienen en común entre la evolución natural y los algoritmos genéticos. En la Figura 2.2 se muestran algunos ejemplos de los términos descritos en el cuadro.

2.3. Las fases de los algoritmos genéticos

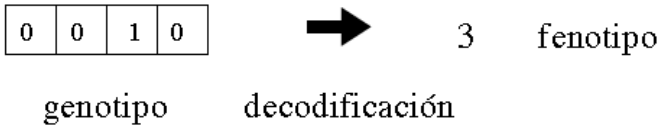
Lo expuesto en esta sección, así como las imágenes, proviene principalmente de Sivanandam and Deepa (2007). Algunas ideas secundarias provienen de Affenzeller et al. (2009). Al lector interesado en profundizar en estos puntos, se le recomienda dirigirse directamente a las fuentes originales.

Evolución Natural	Algoritmos genéticos
Cromosoma	*String* (cadena de caracteres o simplemente 'cadena').
Gen	Rasgo
Alelo	Posible valor del rasgo
Locus	Posición en la cadena
Genotipo	Estructura o cadena codificada
Fenotipo	Cadena decodificada

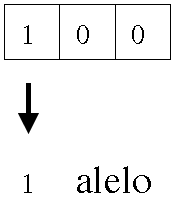
Cuadro 2.1: Expresiones en común de los AG y la evolución natural.



(a) Cadena o cromosoma.



(b) Genotipo (cadena codificada) y genotipo (cadena decodificada).



(c) Posibles valres del rasgo, en este caso son 0 y 1.

Figura 2.2: Algunos ejemplos que ilustran el uso de términos propios de la evolución natural, dentro del contexto de los AG. Figuras tomadas de Coello (2016).

Como se vio al principio de este capítulo, hay varias fases que dan forma a los algoritmos genéticos. Dentro de cada una de éstas hay una gama amplia de métodos que se pueden utilizar para seleccionar un AG adecuado para nuestro problema. El método más adecuado dependerá, sin duda, del tipo de problema que estemos atacando.

A lo largo de esta sección vamos a explorar las tácticas más comunes así como las condiciones bajo las que éstas son más efectivas.

2.3.1. Codificación

Los genes están representados por objetos. Para resolver cada problema en particular, se debe elegir determinado conjunto de objetos, que es de donde se irá seleccionando la información genética. Al proceso que consiste en delimitar este conjunto, se le llama codificación y se asocia con el genotipo.

Los tipos de objetos que pueden representar los genes son muy variados. Algunos ejemplos útiles en la práctica son los siguientes:

- **Codificación binaria.** Los genes toman algún valor dentro del conjunto $\{0, 1\}$; de este modo los cromosomas son cadenas (*strings*) de ceros y unos.

Cromosoma 1	1 1 0 1 0 0 0 1 1 0 1 0
Cromosoma 2	0 1 1 1 1 1 1 1 1 1 0 0

Figura 2.3: Codificación binaria.

  La ventaja de la codificación binaria es su simplicidad al sólo ocupar dos valores (alelos) distintos y puede ser útil para problemas en los que se requiere representar números enteros; sin embargo, para representar números reales, la cadena podría tornarse demasiado larga a medida que los números crecen.

  Aunque muchos problemas pueden resolverse utilizando esta codificación, dependiendo de la complejidad de los datos y la exactitud con que queramos representarlos, será el número de genes que tendrá cada cromosoma. En ese sentido, se dice que son pocos los problemas que se adaptan de forma natural a la representación binaria.

- **Codificación octal.** Los genes se toman dentro del conjunto $\{0, \ldots, 7\}$ y su uso es parecido a la codificación binaria, pero para algunos problemas puede ser más simple utilizar los elementos de este conjunto.

Cromosoma 1	03467216
Cromosoma 2	15723314

Figura 2.4: Codificación octal.

- **Codificación hexadecimal.** Los genes toman valores dentro del conjunto $\{0, 1, \ldots, 9, A, B, \ldots, F\}$ y la explicación de su uso es análoga a la anterior.

Cromosoma 1	9CE7
Cromosoma 2	3DBA

Figura 2.5: Codificación hexadecimal.

- **Codificación de permutación.** Esta codificación es útil únicamente para problemas de ordenación (como lo es el problema del viajero, que puede encontrarse en Kleinberg (2006)). Se enumeran los objetos a permutar y cada cromosoma se presenta como una cadena que contiene todos estos números. Lo que diferencia un cromosoma de otro es el orden en que se presenta cada gen.

Cromosoma A	1 5 3 2 6 4 7 9 8
Cromosoma B	8 5 6 7 2 3 1 4 9

Figura 2.6: Codificación de permutación.

- **Codificación de valores.** Hay problemas que son difíciles de adaptar a una codificación como las anteriormente vistas. En estos casos una solución bastante útil es emplear (o inventar) una cadena de caracteres para cada gen, que debería estar estrechamente relacionada con el problema a atacar. Un ejemplo concreto de esto es que cada gen sea un número real.

Es importante mencionar que al crear una codificación de valores, es posible que también tengamos que crear sus propias reglas de cruza y mutación.

Cromosoma A	1.2324 5.3243 0.4556 2.3293 2.4545
Cromosoma B	ABDJEIFJDHDIERJFDTYUPFEGTD
Cromosoma C	(atrás), (atrás), (der.), (adelante), (izq.)

Figura 2.7: Codificación de valores.

- **Codificación de árbol.** Cada cromosoma es un árbol de objetos, por ejemplo funciones o comandos de algún lenguaje de programación.

2.3.2. Aptitud

El nivel de aptitud de un individuo se mide a través de la *function fitness* o función de aptitud. Para hacer esto primero se debe decodificar el cromosoma de alguna forma adecuada para poder evaluarlo. La aptitud suele ser un número real.

Una vez asignado un valor numérico a la función de aptitud, en principio es fácil ordenar las soluciones y de este modo determinar cuáles son las mejores. Además, muchas veces podremos saber qué tan cerca está un individuo de la solución óptima. Si se conoce esta solución, en la práctica resulta conveniente asignarle a su aptitud el valor de cero.

2.3.3. Población

La población es el conjunto de soluciones con el que se trabaja en un determinado momento del tiempo. Hay dos puntos a considerar:

1. La población inicial, y

2. el tamaño de la población.

Usualmente se parte de una población generada aleatoriamente dentro de todo el espacio de búsqueda. Debemos generarla de buen tamaño, pues esto nos asegurará tener un buen número de genes para mezclar; el término en inglés es *gen pool* y lo traduciremos como reserva genética.

Aunque por lo general los elementos de la población inicial se eligen de manera aleatoria, se sabe de algunos procesos en los que es más recomendable tomar como población inicial a buenas soluciones dentro del universo de las soluciones conocidas. Un ejemplo podría ser el encontrar el máximo de una función con un comportamiento caótico en extremo.

Idealmente el tamaño de la reserva genética debe ser tan grande como se pueda, pues esto nos permitirá llegar más rápido a la solución ideal. A mayor tamaño de la población hay más exploración del espacio de búsqueda y es más difícil que el algoritmo se quede atrapado en un máximo local. El número ideal de individuos depende de la complejidad del problema. Así mismo, se debería procurar tener una población inicial en donde todos los alelos se encuentren representados, por esta razón, usualmente se genera la población inicial de manera aleatoria.

Por otro lado, y para sumar a las dificultades que enfrentamos al emplear AG, se ha demostrado que el tiempo de convergencia de un AG requiere de $\mathcal{O}(n \log n)$ evaluaciones, donde n es el tamaño de la población; por lo que una población grande puede aumentar mucho el costo computacional, la memoria requerida y el tiempo de convergencia. En términos generales una población de tamaño 100 puede funcionar bien.

2.3.4. Crianza

El proceso de crianza (*breeding*) engloba a los procesos de

1. Selección
2. Cruza
3. Reemplazo de viejos individuos por los nuevos

Los tres en su conjunto son la clave del funcionamiento de los AG. En las siguientes secciones se discuten estos procesos con más detalle.

2.3.5. Selección

Dentro de la población se seleccionan dos individuos que al combinar sus genes darán lugar a uno o varios nuevos individuos. Inspirados la teoría de Darwin, nos gustaría asegurar que los individuos a escoger estén dentro de los mejores. Obviamente la función de aptitud nos facilitará esta tarea, ya que se encarga de clasificar

a la población de acuerdo a su distancia al óptimo.

Una buena forma de elegir sería tomar a los dos individuos mejor valuados. Esto contribuye a una convergencia veloz. Sin embargo, el problema que podría surgir es el quedarnos atrapados en un óptimo local.

Por otro lado, una elección donde no estén los mejores dos individuos podría reducir innecesariamente la tasa de convergencia, pero ayudaría a que el algoritmo no se detenga antes de alcanzar el óptimo (véase la Sección 2.3.9). Varios métodos de selección se discuten a continuación:

2.3.5.1. Rueda de Ruleta

Es una forma muy popular de selección en los AG. Consiste en elegir a los padres de manera aleatoria, pero cada individuo tiene una probabilidad de ser elegido proporcional a su función de aptitud.

Ésta es una técnica de selección moderadamente fuerte, ya que los individuos mejor valuados tienen mayor probabilidad de ser elegidos; sin embargo, hay posibilidad de escoger individuos débiles.

2.3.5.2. Selección de acuerdo al Rango

Se ordena a la población de acuerdo al rango obtenido en la función de aptitud. El peor elemento ocupará el lugar 1 y el mejor elemento la posición N, sin embargo, en algunas ocasiones convendrá ordenarlos en orden inverso. Después se elige a los padres generando un par de números aleatorios (entre 1 y N) que favorezcan a los mejores individuos, o empleando el método de ruleta, en donde la probabilidad que tiene cada cromosoma de ser elegido será proporcional al rango que ocupa (y no al valor de su función de aptitud).

Este método tiene como objetivo disminuir el peso excesivo en la selección que pudieran tener algunos individuos mejor valuados, ya que se les selecciona de acuerdo a su rango y no de acuerdo a su puntaje de aptitud. Es útil cuando hay lejanía entre los resultados que arroja la función de aptitud; por ejemplo, supóngase que un solo individuo ocupa el 90 % de la ruleta y el resto el 10 %. En este caso, la selección de rango se ocupa para darle más probabilidad a la población de obtener variaciones genéticas.

La consecuencia es que el tiempo de convergencia aumenta pero la posibilidad de quedar atrapado en un máximo local disminuye.

2.3.5.3. Selección de Torneos

El objetivo de este método es, al igual que el anterior, reducir la presión o peso excesivo que pudieran tener algunos elementos de la población. Se escogen de manera aleatoria k individuos de la población y se ordenan por jerarquías. El ganador del k-torneo será el individuo mejor valuado. Existe una variante en la que, con probabilidad $0.5 < p \leq 1$ se elige al mejor individuo y, con probabilidad $1 - p$ se selecciona al peor.

Una ventaja que se tiene con respecto al método anterior es que, la selección de torneos es computacionalmente más eficiente, pues no tiene que ordenarse a toda la población.

2.3.6. Cruza o Recombinación

La cruza es el proceso de combinar los genes de dos padres para crear un nuevo individuo. Se espera que este nuevo elemento sea una mejor solución de la que ya eran sus progenitores. En este proceso no se crean nuevos genes, sino que se clonan los que ya existían (de los padres).

Varias técnicas de cruza son discutidas a continuación.

2.3.6.1. Cruza de punto único

Se escoge un punto (locus) entre los genes del cromosoma y se distingue con algún criterio a los progenitores; los nombramos *padre 1* y *padre 2*. En el nuevo individuo se copian los genes que están en el *padre 1* desde su locus 1 (primera posición), hasta el gen que ocupa la posición previa al punto seleccionado. El resto de la información genética proviene del *padre 2*, desde el punto seleccionado, hasta el final de la cadena del cromosoma. Véase la Figura 2.8.

Se puede obtener una mejor descendencia si se escoge un punto adecuado dentro del cromosoma para hacer este intercambio, pero también se puede elegir de manera aleatoria.

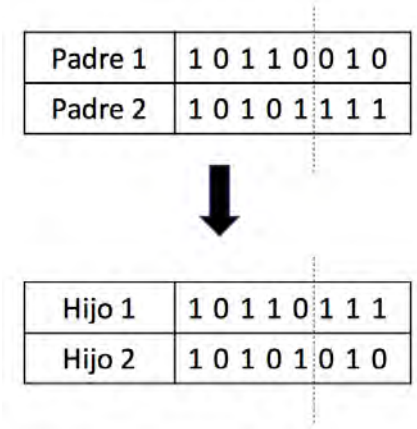


Figura 2.8: Cruza de punto único.

2.3.6.2. Cruza de puntos múltiples

Es la extensión natural de la cruza de punto único. Se escogen n puntos a lo largo del cromosoma y se copian los genes alternadamente del *padre 1* y del *padre 2* (ver Figura 2.9). De acuerdo con algunas investigaciones, la cruza de múltiples puntos es más adecuada para combinar las buenas cualidades presentes en los genes de los padres pues los copia de manera más uniforme (Reeves (1995)).

Tener un número grande de puntos de cruza hace que el algoritmo sea más complejo. Una buena opción es ir disminuyendo puntos a lo largo del recorrido del AG.

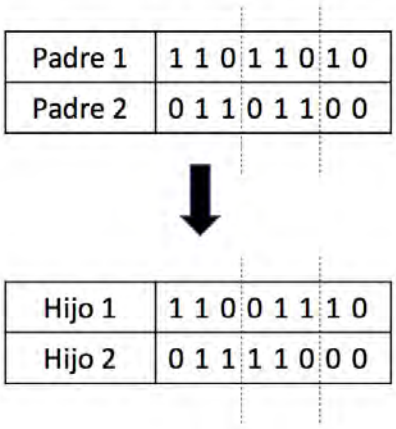


Figura 2.9: Cruza de dos puntos.

2.3.6.3. Cruza uniforme

Utilizando este método cada padre da de manera aleatoria un gen para el locus correspondiente en el cromosoma del hijo. Para esto, se crea aleatoriamente una máscara de ceros y unos del tamaño del cromosoma. Cuando en la máscara hay ceros, el *padre 1* otorga sus genes para en el locus correspondientes del hijo y análogamente, cuando hay unos, será el *padre 2* el que otorgue sus genes (ver Figura 2.10).

Padre 1	1 0 1 1 0 0 1 1
Padre 2	0 0 0 1 1 0 1 0
Máscara	1 1 0 1 0 1 1 0
Hijo 1	1 0 0 1 1 0 1 0
Hijo 2	0 0 1 1 0 0 1 1

Figura 2.10: Cruza uniforme.

2.3.6.4. Cruza de tres padres

Se seleccionan tres padres. Los genes del *padre 1* se comparan con los del *padre 2*; si son iguales el gen pasa a formar parte del hijo, pero si son diferentes el gen del *padre 3* formará parte del cromosoma del descendiente. Obviamente este método es útil para cromosomas binarios, pero en otra codificación difícilmente arrojará alguna ventaja. Véase la Figura 2.11.

Padre 1	1 1 0 1 0 0 0 1
Padre 2	0 1 1 0 1 0 0 1
Padre 3	0 1 1 0 1 1 0 0
Hijo	0 1 1 0 1 0 0 1

Figura 2.11: Cruza de tres padres.

2.3.7. Mutación

La mutación es aquello que nos permite, en un principio, explorar el espacio de búsqueda de manera uniforme y hacia el final del proceso explorar soluciones cercanas para hacer mejoras locales. Consiste en provocar un pequeño salto en la solución obtenida para crear diversidad en una población.

Las mutaciones se pueden presentar de diferentes formas, dependiendo de la representación de cada problema. Para la codificación binaria (Sección 2.3.1), una mutación puede consistir en cambiar el valor de cada gen con una pequeña probabilidad (usualmente es cercana a $1/L$, donde L es la longitud del cromosoma). Otra forma de mutación común, consisten en escoger una locus y cambiar aleatoriamente al gen que está en esa posición. En otros casos, se entiende a la mutación como el proceso de añadir un nuevo gen al cromosoma.

En los problemas de optimización combinatoria, pueden ocurrir mutaciones "ilegales" es decir, la mutación generada aleatoriamente da un elemento que no pertenece al espacio de soluciones. Para evitar esa situación, en la literatura (Sivanandam and Deepa (2007), Coello (2016)) podemos encontrar algunas alternativas; por ejemplo, cambiar el cromosoma "ilegal" por uno *cercano* que sea válido. En este contexto, el significado de la palabra *cercano* dependerá del problema que se esté abordando.

2.3.8. Reemplazo

En cada generación (idealmente) la reserva genética cambia; esto se logra con la eliminación de viejos individuos y con la inserción de nuevos. Una vez creada la descendencia, debemos determinar quiénes van a formar parte de la nueva generación eligiendo elementos dentro del conjunto de los recién creados y del conjunto de los que ya pertenecían a la vieja generación.

Hay varios métodos importantes a considerar:

2.3.8.1. Remplazo Generacional

Toda la población es reemplazada por los nuevos descendientes. Claramente, las reproducciones sólo pueden llevarse acabo entre miembros de una misma generación, lo que orilla a la desventaja de tener una supremacía de pocos individuos muy fuertes y en consecuencia, fácilmente se desemboca en paro prematuro, sin necesariamente

aproximarse a un óptimo global.

Esto se da porque se pierde mucha información genética de los individuos "débiles", y son éstos los que dan diversidad, los que se encargan de que el algoritmo explore todo el espacio de búsqueda.

2.3.8.2. Elitismo

Los mejores n ($n \geq 1$) individuos de la generación previa son conservados para la nueva generación.

2.3.8.3. Eliminación de los últimos n

Los n individuos más débiles son reemplazados por n descendientes.

2.3.8.4. Eliminación de n

En contraste con el método anterior, los n que se eliminan se escogen de manera aleatoria. Este proceso disminuye la velocidad de convergencia, pero da oportunidad de explorar más el espacio de búsqueda, evitando un paro prematuro.

2.3.8.5. Remplazo de Torneos

Se hacen conjuntos combinando a individuos nuevos con algunos de la generación anterior. Compiten entre sí y los ganadores serán los que pertenecerán a la nueva generación. Recordemos que una forma de competir es a través de la función de aptitud, así que bajo este criterio los mejores valuados serán los ganadores.

2.3.9. Fin de la búsqueda (Criterios de convergencia)

El proceso de hacer generaciones nuevas una y otra vez debe terminar en algún momento. El objetivo es llegar a una buena solución, pero no siempre se logra y en ese caso debemos determinar criterios de paro para el AG. Entonces entenderemos por *paro* al momento en que se finaliza el algoritmo sin llegar necesariamente a una solución deseada. A continuación se enumeran dichos criterios.

- **Generación máxima.** El AG se detiene después de alcanzar un número determinado de generaciones.
- **Transcurso de tiempo.** El algoritmo se detiene al alcanzar una determinada cota de tiempo.

- **Sin cambios en la aptitud.** El algoritmo para si no hay cambios en la función de aptitud de los mejores individuos durante un número específico de generaciones.

Cuando se llega a una buena solución antes del tiempo de paro de un algoritmo, diremos que el algoritmo converge. A continuación se enlistan algunas técnicas de convergencia.

2.3.9.1. El mejor individuo

El algoritmo se detiene cuando al menos un individuo de la población traspasa un umbral de aptitud, esto es, que su función de aptitud supera cierto valor. De este modo se puede decir que se consigue al menos una buena solución.

2.3.9.2. El peor individuo

Este criterio termina con la búsqueda cuando el individuo con la peor valuación en aptitud alcanza el umbral de aptitud. Esto asegura que toda la población alcanza la cota mínima de aptitud, por lo que todos los individuos serían buenas soluciones. El punto de este criterio no es asegurar una solución óptima sino un conjunto de buenas soluciones.

2.3.9.3. Suma de aptitud

Recordemos que es común valuar la aptitud de la solución ideal con 0. En ese sentido, utilizando el criterio de suma de aptitud, el algoritmo se detiene cuando la suma de la función de aptitud de toda la población es menor o igual a cierto umbral de convergencia. Esto nos asegura que la mayoría de las soluciones son buenas. Para este criterio debe considerarse el número de individuos en una población.

2.3.9.4. Aptitud mediana

El algoritmo converge cuando la mitad de la población supera el umbral de aptitud. De este modo se tiene un numero considerable de buenas soluciones.

2.4. Ejemplo

El siguiente es un ejemplo de un algoritmo genético programado en el lenguaje R. El código completo se encuentra en la siguiente liga: `https://github.com/`

`BrunoRGutierrez/Genetic_algorithm-`.

2.4.1. Resumen

A partir de una cadena aleatoria de 13 caracteres ASCII se busca llegar a la frase *"Hello, World!"* utilizando un algoritmo genético. La codificación empleada es de *valores*, pues los genes son los mismos caracteres ASCII. Para iniciar se crea de manera aleatoria una población de 20 cromosomas haploides, cada uno con longitud de 13 caracteres. Se diseña una función de aptitud en relación a la cercanía de cada individuo con la meta (*target*): *"Hello, World!"*; en donde el cromosoma que coincida con la meta tendrá una valuación de 0, y los que difieran de este tendrán una valuación positiva dependiendo con su distancia al mismo.

Se emplea un método de *selección de acuerdo al rango* para elegir a los dos individuos que serán los padres y sus genes se mezclan utilizando el método de *cruza uniforme*. La mutación se lleva a cabo sobre un gen aleatorio del nuevo cromosoma con una probabilidad de 2/3.

Para determinar si el nuevo individuo es insertado a la población se utiliza la técnica de *remplazo de torneos* entre el nuevo cromosoma y el cromosoma peor valuado de la generación anterior. La búsqueda converge con el criterio del *mejor individuo*, es decir, cuando la función de aptitud de un elemento de la población es 0.

2.4.2. Código

El siguiente código es un resumen más digerible del código original. En el código original varios de los pasos del AG se encuentran traslapados unos con otros procurando una mayor eficiencia del algoritmo. En este resumen, se han separado cuidadosamente los pasos con fines ilustrativos, pero siempre se invita al lector interesado a revisar el código completo y sin alteraciones en la liga antes citada.

En la primer parte del código, se define la meta y se crea una población aleatoria de individuos.

```
# Se crea la meta a la que se que se desea llegar
target <- "Hello, World!"
```

```
# Se define el tamano de la poblacion, en nuestro caso es de 20 individuos
GENSIZE <- 20

# Creamos el vector "population" en donde se va agregando cada individuo
population <- vector("list", GENSIZE)
# Se crea a cada individuo a traves de un ciclo for
i <- 1
for(i in 1:GENSIZE){
  # En el vector "cromo" se va a guardar el adn del nuevo individuo
  cromo <- vector("expression", nchar(target))
  # Se crea de forma aleatoria cada gen del individuo a traves de un for
  x <- 1
  for(x in 1:(nchar(target))){
  # se eligen un numero entero aleatorio dentro del intervalo [32,126].
      Despues estos numeros se convierten a caracteres ASCII.
    random_gen <- chr(sample(32:126,1))
    cromo[x] <- c(random_gen)
    x <- x + 1
  }

  # Convertimos el cromosoma de una lista a un string
  cromo <-paste(cromo,collapse="")
  # Valuamos al nuevo individuo
  valor <- fitness(cromo, target)
  # Agregamos al nuevo individuo con su valuacion a la poblacion
  population[[i]] <- list(cromo, valor)
  i <- i + 1
}
```

1. Aptitud.
   Se define la función de aptitud a través del entero que le corresponde a cada caracter ASCII

```
fitness <- function(source, target){
  fitval <- 0
  for (i in 1: nchar(target)) {
  # al valor de cada caracter de nuestro individuo, se le resta el valor
      de cada caracter de la meta. Esta diferencia se eleva al cuadrado
```

```
        para evitar tener numeros negativos. La suma acumulada es el valor
        de la funcion fitness
      fitval <- fitval + ((target,i,i) - (source,i,i)) ** 2
  }
  return( fitval )
}
```

2. Selección.
   Se elige a dos progenitores con una variable aleatoria que favorece a los mejor valuados.

```
random_padre <- function(population){
# Se multiplican dos uniformes (0,1) con el tamano de la muestra -1
  aleatorio = runif(1) * runif(1) * (GENSIZE - 1)
  # El numero anterior es un flotante en el intervalo (0,19). Lo
      convertimos a un entero del intervalo [1,20]
  aleatorio = 1 + floor( aleatorio )
  # Dentro de la poblacion ordenada de acuerdo a su valuacion, se
      selecciona al padre que se encuentra en la posicion del numero
      aleatorio.
return(population[[ aleatorio ]][[1]])
}

# Se seleccionan dos padres
padre1 <- random_padre(population)
padre2 <- random_padre(population)
```

3. Cruza.
   Se utiliza el método de cruza uniforme, por lo que se realiza una máscara de 13 caracteres compuesta de manera aleatoria por los números en el conjunto $\{1, 2\}$. El número i ($i = 1, 2$) en la posición j ($j = 1, \ldots, 13$) indica que el padre i otorga su j-ésimo gen a su descendencia, también en el lugar j.

```
cruza <- function(padre1, padre2){
  j <- 1
  for(j in 1: nchar(target)){
    xy <- genes_padres[[j]][[sample(1:2,1) ]]
```

```
    hijo_cromo[[j]] <- xy
    j = j + 1
  }
  hijo_cromo <- paste(hijo_cromo,collapse="")
  return(hijo_cromo)
}
```

4. Mutación.

   Se escoge uniformemente un gen del cromosoma. Con probabilidad 1/3 cambia al siguiente caracter en la lista ASCCI, con probabilidad 1/3 cambia al caracter anterior y con probabilicad 1/3 no muta.

```
mutate <- function(mut){
  alelo <- sample(1:nchar(target), 1)
  adn <- asc(substr(mut,alelo,alelo)) + sample(c(-1,0,1),1)
  gen <- chr(adn)
  substring(mut, alelo) <- c(gen)
  mutation <-mut
  return(mutation)
}
```

5. Remplazo.
   Se realiza un torneo del nuevo individuo con el peor de la generación anterior. Sólo el que tenga mejor valor de aptitud será injertado en la nueva generación.

```
if(hijo_cromo[[2]] < population [[20]][[2]]) {
    population[[20]] <- hijo_cromo
  }
```

6. Fin de la búsqueda.
   A través de un ciclo *while* se realizan del paso 3 al 6 hasta que el mejor individuo tenga valuación de función de aptitud igual a 0.

```
l <- 1
while(TRUE){
```

```
  #imprimimos al mejor elemento de la lista
  best <- paste(population[ [1]], collapse=', ' )
  print(c(l, best))

  #Si su valor de aptitud es ==0 se termina el ciclo.
  l <- l +1
  if(population [[1]][[2]] == 0){
    break
  }
}
```

Capítulo 3

Estadística Bayesiana

3.1. Introducción

La presente sección, así como las Secciones 3.2 y 3.3 se basan principalmente en Gutiérrez-Peña (2016) y parcialmente en Bernardo (1981); excepto las Secciones 3.3.4 y 3.3.5, que toman su fuente en Bernardo and Smith (2000). Al lector interesado en ahondar en los conceptos aquí explicados, se le recomienda dirigirse a las fuentes originales.

De manera muy general, puede decirse que la estadística es la disciplina que estudia los fenómenos inciertos (aleatorios), es decir, aquellos que no se pueden predecir con certeza. El estudio se lleva a cabo a partir del posible conocimiento previo sobre el fenómeno y de observaciones que se realizan sobre el mismo.

Encontramos dos casos de estudio para la estadística:

1. Se cuenta con todos los datos posibles del fenómeno bajo estudio (por ejemplo: censos).
   En este caso se tiene una población a la que se le conoce perfectamente bien, así que la estadística se encarga de analizar y explorar estos datos, para llegar a conclusiones sobre la población.

2. Se cuenta únicamente con una parte de todos los datos posibles (por ejemplo: encuestas).
   Sólo se dispone de una descripción aproximada de la población a estudiar. La estadística hace inferencias para poder llegar a conclusiones generales sobre la población.

En el segundo caso, si se tiene bajo estudio un fenómeno del cual se ha obtenido una muestra, entonces hay que hacer inferencias con base en la información con la que contamos, es decir la muestra, para llegar a conclusiones sobre el fenómeno de estudio. Pero, ¿cómo seleccionamos a la muestra?, y ¿cómo medir el grado de aproximación de nuestra inferencia con las características no observadas del fenómeno?

La solución es seleccionar a la muestra de manera probabilística, es decir, por sorteo. Para esto suponemos que el fenómeno está modelado por una variable aleatoria X y que cada observación x es el resultado del experimento aleatorio modelado por X. De esta forma, la probabilidad de que salga x en el experimento X es $\Pr[X = x]$.

Bajo estos supuestos, describir el fenómeno de estudio es equivalente a describir el modelo de probabilidad.

En ocasiones (inferencia no paramétrica) la propia forma funcional de $\Pr[X = x]$ se supone desconocida, pero en otras (inferencia paramétrica) resulta conveniente suponer, por ejemplo, que el modelo de probabilidad para la v.a. X es de la forma

$$\Pr[X = x] = p(x|\theta) \quad (\text{si } X \text{ es discreta})$$

donde $p(\cdot|\theta)$ tiene forma conocida pero el parámetro θ es desconocido. El punto clave que es describir al fenómeno de estudio, recae ahora en caracterizar el valor de θ.

Tiene sentido suponer que el fenómeno que nos interesa estudiar puede ser modelado por la familia

$$\mathscr{F} = \left\{ \{p(x|\theta) : \theta \in \Theta, x \in \mathscr{X}\} : \Theta \in \mathbb{R}^k, \mathscr{X} \in \mathbb{R}^p \right\} \tag{3.1}$$

en donde suponemos que conocemos la forma de $p(\cdot|\theta)$ pero el parámetro θ es desconocido.

3.1.1. Métodos estadísticos tradicionales

En la estadística clásica se especifica el modelo probabilístico (3.1) que describe el comportamiento del fenómeno de estudio, es decir, $p(x|\theta)$ determina las probabilidades de los posibles valores de X para un valor dado de θ. Para hacer inferencia sobre el valor desconocido de los parámetros, las técnicas tradicionales son:

- Estimadores puntuales: $\hat{\theta}$.

- Intervalos de confianza: $\theta \in (\theta_1, \theta_2)$.
- Pruebas de hipótesis: $H_0 : \theta \in \Theta_0$ vs $H_1 : \theta \in \Theta_1$.
- Pronóstico: Predecir los resultados futuros del fenómeno observado.

Entonces, el problema se reduce a hacer inferencias sobre θ con base en el valor observado de $X = x$, pues éste indirectamente nos da información sobre el valor de θ.

Por ejemplo, sea $\boldsymbol{x}$ el vector de observaciones correspondiente a una muestra aleatoria de una v.a. discreta X. Para este valor fijo de $\boldsymbol{x}$, $\Pr(\boldsymbol{x}|\theta)$ es una función de θ que indica qué tan probable es obtener $\boldsymbol{x}$ dado que está condicionado por el valor de θ. Así pues, el valor más "verosímil" de θ es el que maximiza la función $\Pr(\boldsymbol{x}|\theta)$. Para resaltar que la estamos tomando como función de θ, usualmente se denota $l_{\boldsymbol{x}}(\theta) = \Pr(\boldsymbol{x}|\theta)$ y se le llama *función de verosimilitud.*

Obsérvese que $l_{\boldsymbol{x}}(\theta) \geq 0$ para toda θ; sin embargo no es una función de densidad, ya que el resultado de integrar la función de verosimilitud sobre todos los valores de θ no siempre es 1.

Si el resultado observado es $X = x$, los valores de θ más "verosímiles" son aquellos que asignan una probabilidad alta de que se obtenga x por encima de los valores de θ que asignan a x una probabilidad baja de ser seleccionados. A esto se le conoce como el *principio de máxima verosimilitud.*

Definición 3.1.1. *El estimador máximo verosímil de θ es el valor de θ que maximiza la función de verosimilitud y se denota $\hat{\theta}$.*

En la estadística clásica el parámetro desconocido θ se considera una constante, no aleatorio. En ese sentido no podemos referirnos a él en términos de probabilidades. Por esta razón, en la terminología clásica se habla de: "verosimilitud", "confianza", "nivel de significancia", etc.

Paradójicamente, lo que se entiende de manera coloquial con la lista anterior, podría expresarse de manera intuitiva en términos de probabilidades. Por ejemplo, dado un intervalo de confianza para θ del 95 %, digamos (α_1, α_2), se podría entender erróneamente como $\Pr(\alpha_1 < \theta < \alpha_2) = 0.95$.

3.1.1.1. Interpretación subjetiva de la probabilidad

Para incluir la evidencia externa que tenemos de los fenómenos en el proceso de hacer inferencia, debemos presentarla en forma probabilística. Es común que las personas nos expresemos de manera coloquial con probabilidades para indicar nuestras creencias sobre la ocurrencia de algún fenómeno. Las frases siguientes ilustran la situación: “dejaré el paraguas porque me parece poco probable que llueva", “es muy probable que el gas dure 3 meses antes de pedir un nuevo tanque”, etc.

Sin embargo no podemos reportar cualquier número que queramos. Para ser tomadas en serio, las probabilidades que asignemos deben tener relación con la realidad y reflejar de manera honesta nuestro conocimiento sobre el fenómeno de estudio. Usualmente estas probabilidades son asignadas por expertos y/o con base en información (muestral) previa.

Al hacer inferencia acerca del valor desconocido del parámetro θ, generalmente se cuenta con algún tipo de información acerca de su valor, incluso antes de observar los datos. Las inferencias deben basarse tanto en los datos como en la información inicial.

Para combinar estas dos fuentes de información, en la teoría bayesiana se requiere la especificación de una distribución de probabilidad sobre θ que refleje el conocimiento que se tiene sobre su valor.

La teoría de la probabilidad ha sido estudiada de manera muy amplia para todo lo referente a azar y a la aleatoriedad. ¿Acaso no deberíamos apoyarnos en toda la herramienta probabilística para dar sustento a la inferencia estadística?

Sí deberíamos, y la respuesta es la estadística bayesiana.

3.2. Inferencia Bayesiana

3.2.1. El enfoque bayesiano

La idea es desarrollar una Teoría Estadística que nos permita estructurar la solución a cualquier problema de inferencia, basándonos en una serie de principios. Esta teoría es fundamental para darle a la estadística una estructura coherente y porque en otros enfoques pueden presentarse casos en los que:

1. No hay una solución razonable (véase Bernardo and Smith (2000), pag. 468).

2. Se presentan paradojas[1].

El Teorema de Bayes es una herramienta fundamental para lograr esta tarea.

Teorema 3.2.1 (Teorema de Bayes)**.** *Dados dos eventos A y B tales que* $\Pr(B) > 0$ *se cumple*

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$$

Si $\{A_i : 1 \leq i \leq M\}$ *es un conjunto exhaustivo de eventos mutuamente excluyentes, entonces*

$$\Pr(A_i|B) = \frac{\Pr(B|A_i)\Pr(A_i)}{\sum_{j=1}^{M} \Pr(B|A_j)\Pr(A_j)}$$

El teorema de Bayes es clave en los métodos bayesianos pues en él se basa el proceso de aprendizaje; veamos como se aplica para hacer inferencia:

Por lo general, se tienen datos sobre algún fenómeno denotados por X y cantidades desconocidas θ cuyo valor nos interesa; estas cantidades pueden ser parámetros del modelo, observaciones faltantes, mediciones que no podemos observar directamente o con suficiente precisión, etc.

Como se vio en la sección anterior, al igual que en los métodos tradicionales se postula un modelo de probabilidad (3.1) denotado por

$$p(x|\theta).$$

Dado este modelo, en los métodos bayesianos le damos a θ una distribución de probabilidad $p(\theta)$ (véase la Sección 3.2.2) que refleje nuestra incertidumbre inicial acerca de su valor. Por otro lado X es conocido, por lo que debemos condicionar en su valor observado x.

De este modo, nuestro conocimiento acerca del valor de θ queda descrito a través de su distribución final (Sección 3.2.4):

$$p(\theta|x).$$

[1]La estadística clásica propone una serie de métodos que no necesariamente son consistentes entre sí, por lo que no es difícil encontrar ejemplos de paradojas en las que utilizando el mismo procedimiento (aparentemente ideal) de prueba de hipótesis, se puede aceptar con una muestra la hipótesis nula y rechazarla con otra a pesar de que las dos muestras den la misma información sobre el valor del parámetro.

Las diferencias entre los métodos tradicionales para hacer estadística y los métodos bayesianos son:

- Tradicionales: Responden a la pregunta ¿Qué nos dicen los datos X acerca del parámetro θ?, ignorando toda evidencia externa.

- Bayesianos: ¿Cómo cambian nuestros juicios originales acerca del valor de la cantidad desconocida θ a la luz de los datos X?. Para contestarlo se puede tomar en cuenta cualquier evidencia externa.

3.2.2. La distribución inicial

La distribución *a priori* o inicial es la distribución de probabilidad que el investigador asigna sobre el parámetro desconocido θ, y se denota por $p(\theta)$. Para asignarla, el investigador se basa en su conocimiento hasta ese momento sobre el fenómeno de estudio.

El papel de esta distribución es tan importante que diferentes opiniones llevarán a distintas inferencias para un mismo conjunto de datos. Sin embargo, si éstas son razonables (congruentes con el objeto de estudio), su efecto sobre las inferencias disminuye conforme se analizan más datos.

Las familias conjugadas[2], en principio, son muy útiles para simplificar los cálculos. Es por eso que, en las ocasiones en las que tenemos una vaga idea de la forma que debería tomar la distribución inicial, resulta conveniente emplear algunas familias conjugadas que puedan ajustarse a nuestro modelo.

En otras ocasiones no es deseable incluir información inicial sobre el valor del parámetro, o en realidad no se cuenta con ella. Un ejemplo puede ser al hacer inferencia sobre el ganador de las elecciones políticas por parte de un organismo neutral. Dado que una distribución inicial pudiera favorecer a algún candidato, usualmente se emplea una distribución inicial *no-informativa*.

Las distribuciones iniciales no-informativas reflejan nuestra ignorancia sobre el valor del parámetro. Descartando los casos triviales, encontrar y hacer nuestros modelos de inferencia con estas distribuciones es una tarea para nada sencilla.

[2]Una familia de distribuciones $\mathcal{C}$ es conjugada para el modelo $p(x|\theta)$ si, para toda distribución inicial $p(\theta) \in \mathcal{C}$, ocurre que la correspondiente distribución final $p(\theta|x) \in \mathcal{C}$.

3.2.3. Distribución predictiva

Muchas veces lo que nos interesa es tener información de los valores x que pueden obtenerse de un determinado experimento aleatorio. Obsérvese que esta información puede obtenerse si se conoce la distribución inicial $p(\theta)$ y la probabilidad condicional $p(x|\theta)$. En efecto:

$$p(x) = \int_{\Theta} p(x, \tilde{\theta}) d\tilde{\theta} = \int_{\Theta} p(x|\tilde{\theta}) p(\tilde{\theta}) d\tilde{\theta}.$$

El resultado anterior describe una distribución de probabilidad que se llama *distribución predictiva* porque arroja información sobre los valores que podría tomar el experimento aleatorio en el futuro. Obsérvese que para hacer las predicciones no es necesario saber qué valor toma θ en $p(x|\theta)$.

3.2.4. Distribución final

La densidad de probabilidad $p(x|\theta)$ como función de x y θ expresa la relación que existe entre el resultado x obtenido en algún experimento aleatorio y el parámetro θ que aproxima la distribución del experimento.

Ahora bien, una vez obtenido el resultado aleatorio x, podemos mejorar la descripción que teníamos de θ, esto sería responder a la pregunta: dado que se observó x en el experimento, ¿cómo podemos ajustar la descripción que ya habíamos hecho de θ? La respuesta es la distribución condicional $p(\theta|x)$, que se obtiene mediante el teorema de Bayes (Teorema 3.2.1) y se le conoce como *distribución a posteriori* o *distribución final*.

En términos de variables aleatorias, el teorema de Bayes (*teorema* 3.2.1) toma la siguiente forma

Teorema 3.2.2. *Regla de Bayes. Sea x los resultados de un experimento aleatorio definido mediante el modelo $p(x|\theta)$ y sea $p(\theta)$ la distribución inicial de θ. La distribución final de θ es entonces*

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

donde $p(x)$ es la distribución predictiva de x.

El teorema anterior nos dice que si conocemos a $p(\theta)$ la distribución inicial de θ y la a función de verosimilitud $p(x|\theta)$, a través de la regla de Bayes se actualiza la información que se tiene de θ. Más aún, al tener una nueva observación del experimento, digamos y, la distribución final se convertirá en la nueva distribución inicial con el fin de ir actualizando, con cada observación, nuestra inferencia sobre θ. Este proceso continúa así de manera sucesiva.

De acuerdo con la sección anterior $p(x) = \int p(x|\tilde{\theta})p(\tilde{\theta})d\tilde{\theta}$ no depende de θ, por lo que es común escribir

$$p(\theta|x) \propto p(x|\theta)p(\theta).$$

En el enfoque bayesiano, la distribución final resume toda la información que tenemos hasta el momento sobre el parámetro desconocido θ. Aunque este método parece sorprendentemente simple, lo cierto es que en la práctica suele ser difícil encontrar la distribución final. Para ayudarnos a simplificar estos cálculos se pueden utilizar familias conjugadas; sin embargo, éstas son útiles sólo para ciertas verosimilitudes.

3.2.5. Distribución predictiva final

Hasta ahora las discusiones previas se han centrado básicamente en cómo hacer inferencias sobre el valor desconocido θ; sin embargo, muchas veces será necesario hacer inferencias sobre el posible resultado de un experimento aleatorio.

Si $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ representa una muestra de n observaciones hechas en determinado experimento aleatorio, y queremos hacer inferencias sobre el resultado que tendrá el experimento en el futuro, digamos de $X_{n+1} = Y$, podemos "predecir" el valor que tendrá y a partir de la información proporcionada por $\boldsymbol{x}$. Expresado de manera probabilística sería: $p(y|\boldsymbol{x})$.

Las reglas de la probabilidad nos dicen como llegar a la distribución anterior. En efecto:

$$\begin{aligned} p(y|\boldsymbol{x}) &= \int p(y|\theta, \boldsymbol{x})p(\theta|\boldsymbol{x})d\theta \\ &= \int p(y|\theta)p(\theta|\boldsymbol{x})d\theta \\ &= \mathbb{E}_{p(\theta|\boldsymbol{x})}\left[p(y|\theta)\right] \end{aligned}$$

y se le conoce como la *distribución predictiva final.*

Obsérvese que para lograr el resultado anterior es necesario conocer la función de verosimilitud y la distribución final de θ.

3.3. Teoría de la decisión

3.3.1. Estructura de un problema de decisión

Lo que estudiaremos aquí es el proceso normativo en el que deberían tomarse las decisiones que podría contrastar con la manera en que las personas usualmente las toman. En la vida tomamos muchas decisiones, gran parte de estas son triviales, es decir, cuyas consecuencias aparentemente no tiene un gran impacto; un ejemplo puede ser elegir qué película ver en el cine o el menú en un restaurante. Sin embargo hay decisiones importantes que requieren un análisis profundo, tal es caso de comprar una casa o elegir una carrera.

Lo que analizaremos en las siguientes páginas, es la forma en que deben tomarse las decisiones si se desea hacerlo de una forma *coherente*. De manera rápida, por *coherente* entenderemos a aquella toma de decisión que cumpla con los *principios de coherencia* expuestos más adelante.

Para empezar debemos considerar todas las posibles formas de actuar que se nos presentan. Debemos ser lo más exhaustivos posible, no basta con considerar una opción y su complemento, pues en la vida real la mayoría de las posibilidades a la hora de decidir aparecen como un gran abanico (muchas veces infinito). Mientras agotemos la mayor cantidad de opciones, entonces los resultados de analizar la decisión a tomar será más apegada a la realidad.

Observemos que, al tratarse de un problema de decisión, es natural que la elección de una alternativa excluya la elección de otra. Así, por ejemplo si alguien elige estudiar la carrera de matemáticas, es claro que excluye estudiar actuaría. Sin embargo el lector podrá argumentar que existen personas que estudian ambas carreras. En ese caso, la elección se hizo sobre un conjunto de cosas que excluía a otras alternativas, es decir, al escoger estudiar ambas carreras, se excluye la elección de sólo matemáticas y de sólo actuaría, ni que decir de medicina que también se excluye.

En consecuencia, el primer paso es plantear el conjunto de alternativas mutuamente excluyentes que llamaremos *espacio de decisión* o *conjunto de acciones potenciales*

y se denota por $\mathcal{A}$.

En principio, si se conocen con exactitud todos los escenarios posibles, siempre podríamos seleccionar la mejor opción analizando detalladamente cada una de las consecuencias. Lo cierto es que hay situaciones tan complejas, en donde las alternativas son muchísimas que un análisis exhaustivo es muy complejo. Tal es el caso de decidir la mejor jugada en una partida de ajedrez. Ese tipo de problemas no son de interés de este estudio.

Lo que aquí abordaremos será un análisis sobre la toma de decisiones *coherente* en un *ambiente de incertidumbre*, es decir, cuando nuestra elección depende de sucesos inciertos que generan consecuencias. Matemáticamente se han modelado estos problemas con la teoría de probabilidades. Así que también aquí la probabilidad va a jugar un papel fundamental para medir el grado de incertidumbre de las opciones.

Si bien es cierto que la realización de un suceso es desconocido, la persona que toma la decisión tiene una idea de qué tan plausible es cada opción. Esto puede medirse utilizando una función de probabilidad.

De acuerdo con lo anterior, una vez generado el *espacio de decisiones* se crea el *conjunto de sucesos inciertos* $\mathcal{E}$ modelado por una medida de probabilidad; en donde la elección de decisión en conjunto con la ocurrencia de cada suceso incierto da lugar a determinada(s) *consecuencia(s)* $\mathcal{C} = \mathcal{A} \times \mathcal{E}$. Obsérvese el siguiente *árbol de decisión* (*Figura* 3.1) en donde puede estructurarse nuestro problema.

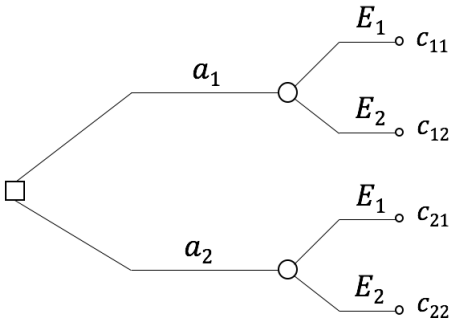


Figura 3.1: Árbol de decisión.

Recordemos que tanto $\mathcal{A}$ el conjunto de acciones como $\mathcal{E}$ el espacio de eventos relevantes pueden ser de tamaño infinito. En el ejemplo anterior la cardinalidad de ambos es igual a dos para facilitarnos la explicación.

Un factor importante del que depende nuestra toma de decisión es de las posibles consecuencias. De este modo y retomando el ejemplo anterior, si preferimos la consecuencia $c_{1,2}$ en vez de $c_{1,1}$, deberíamos de poder medir (o compararlo) de manera que sea un factor a tomar en cuenta a la hora de decidir. En ese sentido, denotaremos con el símbolo $\preceq$ el grado de preferencia entre una consecuencia y otra; así pues, en el ejemplo anterior se tendría que $c_{1,1} \preceq c_{1,2}$. Si ambas consecuencias son igualmente deseables, entonces se denota con el símbolo $\sim$. Así, si $c_{2,1}$ y $c_{2,2}$ fueran igual de deseables se escribiría $c_{2,1} \sim c_{2,2}$

En resumen, los elementos de un problema de decisión en un ambiente de incertidumbre son:

- $\mathcal{A} = \{a_1, \ldots, a_k\}$: Conjunto de acciones potenciales o espacio de decisión.
- $\mathcal{E} = \{E_1, \ldots, E_m\}$: Conjunto de eventos o conjunto de sucesos inciertos. Contiene todos los eventos relevantes al problema de decisión.
- $\mathcal{C} = \mathcal{A} \times \mathcal{E} = \{c_{i,j} | 1 \leq i \leq k, 1 \leq j \leq m\}$: Conjunto de consecuencias posibles.
- $\preceq$: Relación de preferencia entre las distintas consecuencias. Si preferimos c_2 a c_1 entonces $c_1 \preceq c_2$. Si ambas son igualmente deseables se escribe $c_1 \sim c_2$

3.3.2. Axiomas de Coherencia

Los axiomas parten de principios intuitivos que deben tomarse en cuenta al momento de hacer alguna decisión de manera racional. Algunos de estos se escriben a continuación, la lista completa de axiomas puede encontrarse en Bernardo and Smith (2000).

1. *Comparabilidad.* Para cada par de consecuencias en $\mathcal{C}$, una y sólo una de las siguientes condiciones puede ser cierta:

$$c_1 \preceq c_2,\ c_2 \preceq c_1 \text{ ó } c_1 \sim c_2.$$

2. *Transitividad.* Si $c_1 \preceq c_2$ y $c_2 \preceq c_3$, entonces $c_1 \preceq c_3$.

De los axiomas se deriva lo siguiente:

i. La información que el decisor tiene sobre la plausibilidad de los distintos eventos relevantes al problema de decisión debe ser cuantificada a través de una medida de probabilidad.

ii. De la misma manera, las preferencias del decisor entre las distintas consecuencias debe de cuantificarse a través de una función de utilidad. A cada una de las consecuencias c se le asigna un número $u(c)$ que mide la utilidad que c tiene para el decisor, de manera tal que

$$c_i \preceq c_j \text{ si y sólo si } u(c_i) \leq u(c_j). \tag{3.2}$$

iii. Debe tomarse la decisión que maximiza la utilidad esperada; es decir, debe elegirse la acción a_{i_*} tal que

$$\overline{u}(a_{i_*}) = \max_i \overline{u}(a_i)$$

donde

$$\overline{u}(a_i) = \sum_{j=1}^{m} u(a_i, E_j) \Pr(E_j) \quad (i = 1, \ldots k)$$

denota la utilidad esperada de la acción a_i.

Cualquier otra forma de actuación está en contradicción con los principios establecidos, y por lo tanto resulta inadmisible para quien los acepta.

3.3.3. Problemas de decisión estadísticos

Hacer inferencia sobre el valor del parámetro desconocido θ puede ser descrito como un problema de decisión. En este contexto los elementos del problema son los siguientes:

1. El espacio de acciones potenciales disponibles: $a \in \mathcal{A}$

2. Los posibles estados de la naturaleza contenidos en el espacio parametral: $\theta \in \Theta$, que describe los eventos inciertos que pueden afectar nuestras decisiones.

3. El espacio de las consecuencias: $\mathcal{C} = \mathcal{A} \times \Theta$

4. Una forma de cuantificar la incertidumbre sobre el espacio de estados de la naturaleza Θ. Para ser congruente con los axiomas de coherencia, debe hacerse a través de una medida de probabilidad: $p(\theta)$.

5. Una forma de cuantificar las consecuencias en $\mathcal{C}$. Debe hacerse a través de la función de utilidad: $u(a, \theta)$.

En la literatura estadística, en vez de trabajar directamente con la función de utilidad, es más común emplear una función de pérdida $\mathrm{L} : \mathcal{A} \times \Theta \to \mathbb{R}$, que a su vez queda definida por la correspondiente función de utilidad descrita en (3.2) de la siguiente manera:

$$\mathrm{L}(a, \theta) = B(\theta) - Au(a, \theta) \tag{3.3}$$

para alguna constante $A > 0$ y una función $B : \Theta \to \mathbb{R}$ cuyo valor esperado existe. En este caso, la pérdida esperada es

$$\mathrm{L}^*(a) = \int_\Theta \mathrm{L}(a, \theta) p(\theta) d\theta. \tag{3.4}$$

En la función de pérdida esperada (3.4) podemos darnos cuenta que para resolver el problema de decisión, es necesario tener una descripción completa del comportamiento de la distribución inicial $p(\theta)$.

En problemas de inferencia estadística, por lo general se cuenta con información adicional en la forma de una muestra $X_1, \ldots, X_n \sim p(x|\theta)$. Estos datos pueden incorporase a la inferencia si se combinan con la distribución inicial vía el teorema de Bayes, para obtener la distribución final:

$$p(\theta|\boldsymbol{x}) = \frac{p(\theta)p(\boldsymbol{x}|\theta)}{\int_\Theta p(\tilde{\theta})p(\boldsymbol{x}|\tilde{\theta})d\tilde{\theta}}. \tag{3.5}$$

Para ser congruentes con los axiomas de coherencia, deberá tomarse la decisión que minimiza la pérdida esperada; si se considera la información de la muestra $\boldsymbol{x}$, esta decisión es

$$a^*(\boldsymbol{x}) = \arg\min_{a \in \mathcal{A}} \mathrm{L}^*_{\boldsymbol{x}}(a), \tag{3.6}$$

donde

$$\mathrm{L}^*_{\boldsymbol{x}}(a) = \int_\Theta \mathrm{L}(a, \theta) p(\theta|\boldsymbol{x}) d\theta. \tag{3.7}$$

3.3.4. Estimación puntual

En los casos en los que $\theta \in \Theta$ es una cantidad desconocida, y que el conjunto de acciones potenciales cumple con $\mathcal{A} = \Theta$ y $\mathcal{E} = \sigma(\Theta)$, entonces la elección de $a \in \mathcal{A}$ es un estimador del verdadero valor de θ. A este tipo de problemas de decisión se les denomina de *estimación puntual*.

Un problema de estimación puntual queda completamente definido una vez que $\mathcal{A} = \Theta$ y $L(a, \theta)$ son especificadas. La intuición nos dice que para el caso unidimensional, medidas de tendencia central como la media, mediana o moda de $p(\theta)$ podrían ser estimadores puntuales de θ. Sin embargo, estos valores podrían estar muy lejanos y por lo tanto ser malos estimadores. La teoría nos proporciona un criterio para saber cuándo y por qué se justifica que una funcional sea un estimador puntual (Bernardo and Smith (2000)).

Definición 3.3.1 (Estimador de Bayes.)**.** *Un estimador de Bayes de θ con respecto a la función de pérdida $L(a, \theta)$ y la distribución inicial $p(\theta)$ es una $a \in \mathcal{A}$ que minimiza la pérdida esperada $\int_\Theta L(a, \theta)p(\theta)d\theta$.*

Proposición 3.3.1. *Las siguientes son formas de estimadores de Bayes.*

(i) Si $\mathcal{A} = \Theta = \mathbb{R}^k$, $L(a, \theta) = (a-\theta)^t \boldsymbol{H}(a-\theta)$, y $\boldsymbol{H}$ es simétrica definida positiva, entonces el estimador de Bayes satisface

$$\boldsymbol{H}a = \boldsymbol{H}E(\theta).$$

Si $\boldsymbol{H}^{-1}$ existe, entonces $a = E(\theta)$; por lo tanto el estimador de Bayes con respecto a la pérdida con forma cuadrática es la media de $p(\theta)$, siempre que ésta exista.

(ii) Si $\mathcal{A} = \Theta = \mathbb{R}$ y $L(a, \theta) = c_1(a - \theta)1_{(\theta \leq a)}(a) + c_2(\theta - a)1_{(\theta > a)}(a)$, el estimador de bayes con respecto a la perdida lineal es el cuantil tal que

$$\Pr(\theta \leq a) = c_2/(c_1 + c_2).$$

Si $c_1 = c_2$ el lado derecho de la igualdad es $1/2$ por lo que el estimador de Bayes con respecto a la pérdida con valor absoluto es la mediana de $p(\theta)$.

(iii) Si $\mathcal{A} = \Theta \subset \mathbb{R}^k$ y $L(a, \theta) = 1 - 1_{(B_\epsilon(a))}(\theta)$, donde $B_\epsilon(a)$ es la bola de radio ϵ en Θ centrada en a. El estimador de Bayes maximiza a

$$\int_{B_\epsilon(a)} p(\theta)d\theta.$$

Cuando $\epsilon \to 0$, la función a ser maximizada tiende a $p(a)$ y el estimador de Bayes con respecto a la pérdida cero-uno es una moda de $p(\theta)$, siempre que ésta exista.

3.3.5. Prueba de hipótesis

Las pruebas de hipótesis son un problema de decisión en el que debe elegirse entre dos hipótesis alternativas H_0 y H_1. Esto puede ser descrito como un problema de decisión con un conjunto de sucesos inciertos descrito del siguiente modo

$$\mathcal{E} = \Omega = \{\omega_0 = [H_0 : \theta \in \Theta_0], \omega_1 = [H_1 : \theta \in \Theta_1]\}$$

junto con $p(\omega)$, donde $\theta \in \Theta = \Theta_0 \cup \Theta_1$ es el parámetro desconocido en el modelo $p(x|\theta)$. En este caso se tiene que el espacio de acciones potenciales es de la forma

$$\mathcal{A} = \{a_0, a_1\}$$

donde

- elegir a_0 implica "actuar como si H_0 fuera cierto", y rechazar H_1,
- elegir a_1 implica "actuar como si H_1 fuera cierto" y rechazar H_0.

La función de pérdida es $L(a_i, \omega_j) = L_{i,j}$ con $i, j \in \{0, 1\}$, en donde $L_{i,j}$ refleja las cuatro posibles consecuencias como en la tabla siguiente. Usualmente se tiene que $L_{0,0} = L_{1,1} = 0$.

$L(a, \omega)$	ω_0	ω_1
a_0	$L_{0,0}$	$L_{0,1}$
a_1	$L_{1,0}$	$L_{1,1}$

A modo de ejemplo, analicemos el caso en que los valores de la función de pérdida están dados como se ilustra en la siguiente tabla

$L(a, \omega)$	ω_0	ω_1
a_0	0	k_0
a_1	k_1	0

donde $k_0 > 0$ y $k_1 > 0$. Entonces la pérdida esperada es

$$L^*_{\boldsymbol{x}}(a_0) = L_{0,0}p(\theta_0|\boldsymbol{x}) + L_{0,1}p(\theta_1|\boldsymbol{x}) = k_0 p(\theta_1|\boldsymbol{x})$$

$$L^*_{\boldsymbol{x}}(a_1) = L_{1,0}p(\theta_0|\boldsymbol{x}) + L_{1,1}p(\theta_1|\boldsymbol{x}) = k_1 p(\theta_0|\boldsymbol{x})$$

Recordemos que debemos elegir la acción que minimice la pérdida esperada, por lo que debemos rechazar H_0 si y sólo si

$$L^*_{\boldsymbol{x}}(a_0) > L^*_{\boldsymbol{x}}(a_1),$$

es decir, si y sólo si

$$\frac{p(\theta_1|\boldsymbol{x})}{p(\theta_0|\boldsymbol{x})} > \frac{k_1}{k_0}$$

o de manera equivalente, si y sólo si

$$\frac{p(\boldsymbol{x}|\theta_1)}{p(\boldsymbol{x}|\theta_0)} > \frac{k_1 p(\theta_0)}{k_0 p(\theta_1)}.$$

3.4. Métodos bayesianos de clasificación

La presente sección (basada en Gutiérrez-Peña (2004)) expone una idea general de los métodos bayesianos de clasificación tanto supervisados como no supervisados. Para el caso de la clasificación supervisada, se detallan una serie de pasos bien fundamentados en la teoría bayesiana que se pueden usar para derivar reglas de clasificación que tienen su base en modelos específicos. Al lector interesado en ahondar más en estos temas se le sugiere remitirse a la fuente original, en donde se incluyen algunos otros ejemplos y se cita una gran cantidad de autores para profundizar en cada uno de los temas aquí mencionados.

De manera tradicional, se describe al término "clasificación" como un proceso de aprendizaje; pero también se refiere al mismo, con al menos otra connotación: un conjunto de métodos entre los que destacan el análisis discriminante y el análisis de conglomerados (*clustering*). Hoy en día encontramos que el término "clasificación" toma un punto de vista más amplio pues se alude a él (en el sentido general) como *aprendizaje estadístico* (*Statistical Learning*).

La clasificación se refiere a la agrupación de objetos similares y, para poder hacer esta agrupación, necesitamos hacer juicios de similitud. Encontramos principalmente dos tipos de clasificación: predictiva y descriptiva. Ejemplos de éstas son el análisis discriminante y el análisis de conglomerados, respectivamente. En ambos casos el objetivo es aprender de los datos. Para fijar ideas, veamos los siguientes ejemplos.

- *Análisis discriminante* (clasificación predictiva). Se tiene un conjunto de datos u observaciones que están clasificadas en dos o más grupos; estas muestras reciben el nombre de datos de entrenamiento (*training data*). La tarea del análisis discriminante consiste en asignar un grupo a cada nueva observación.

- *Análisis de conglomerados* (clasificación descriptiva). El nombre en inglés de este método es *cluster analysis* y consiste en agrupar objetos dentro de clases de acuerdo a su "similitud", sin tener una división inicial de los objetos.

En ambos casos aplica la siguiente descripción en términos matemáticos: Se tiene un "objeto"que se codifica a través de mediciones específicas, mismas que se registran dentro de un vector multidimensional de variables; en ese sentido, podemos referirnos al objeto como un vector multidemsional de mediciones. La meta de la clasificación es encontrar una función que mapee estos objetos (relativos a algún problema en específico), hacia un conjunto de índices que identifican a cada una de las clases. A estos índices se les denota como *etiquetas de clase*.

Como ya se mencionó, los problemas de aprendizaje pueden dividirse en supervisados o no supervisados. En áreas como ciencias de computación o ingeniería, la primera de estas tareas de aprendizaje se conoce como *clasificación supervisada, reconocimiento de patrones supervisado* o *predicción de clase*, mientras que es más común referirse a la segunda como *clasificación no supervisada, reconocimiento de patronoes no supervisado* o *descubrimiento de clase.*

Véase el siguiente análisis: Sea Y una variable que denota la respuesta para un conjunto de variables predictoras $\boldsymbol{X} = (X_1, \ldots, X_p)$. Las predicciones se basan en una muestra de entrenamiento de casos previamente "resueltos" $(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)$, en donde los valores conjuntos de todas las variables son conocidos. En el aprendizaje supervisado, el "maestro" designa una respuesta $\hat{y}_i$ para cada $\boldsymbol{x}_i$ en la muestra de entrenamiento. De este modo, el maestro proporcionan la respuesta correcta y también un error asociado con la respuesta del "estudiante". Usualmente se caracteriza a este error por alguna función de pérdida $L(\hat{y}, y)$, En contraste, para el aprendizaje no supervizado se tiene que aprender "sin un maestro".

Suponiendo que $(Y, \boldsymbol{X})$ son variables aleatorias representadas por alguna densidad de probabilidad conjunta $p(y, \boldsymbol{x}) = p(y|\boldsymbol{x})\, p(\boldsymbol{x})$, entonces se puede caracterizar de manera formal al aprendizaje supervisado como un problema de estimar la densidad, en donde nos ocupamos de determinar propiedades de la densidad condicional $p(y|\boldsymbol{x})$ basados en la muestra de entrenamiento $(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)$. Usualmente

la propiedad de interés es el parámetro de "ubicación" μ que minimiza la pérdida esperada de cada valor $\boldsymbol{x}$,

$$\mu(\boldsymbol{x}) = \text{argmin}_{\hat{y}}\ E_{Y|\boldsymbol{x}}\{L(\hat{y}, Y)\}.$$

Por otro lado, en el aprendizaje no supervisado, sólo tenemos un conjunto de n observaciones $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ del vector p-dimensional $\boldsymbol{X}$ con densidad conjunta $p(\boldsymbol{x})$. El objetivo es inferir directamente las propiedades de esta densidad sin la ayuda de un 'maestro'. En problemas de dimensión pequeña existen varios métodos no paramétricos para estimar de manera directa la densidad $p(\boldsymbol{x})$. Sin embargo, en dimensiones altas, muchas veces debemos conformarnos con estimar un modelo de mezclas bastante burdo, en el que cada componente representa una distinta clase de observaciones; véase la Sección 3.4.2.

Debido a su naturaleza, los metodos bayesianos necesariamente están basados en modelos. Por fortuna, los avances numéricos y computacionales recientes como el desarrollo de las técnicas en Monte Carlo con Cadenas de Markov han hecho factible ajustar y analizar cada vez más modelos complejos. El objetivo de la presente sección es proporcionar un panorama general de los métodos de clasificación bayesianos, tanto supervisados como no supervisados, enfatizando en ideas generales antes que en problemas específicos o técnicos. Sin embargo, con el fin de motivar e ilustrar las ideas principales, encontramos conveniente centrarnos en el análisis discriminante (Sección 3.4.1) y en el análisis de conglomerados (Sección 3.4.2). Por otro lado, no se hace ningún intento por discutir la selección de las variables y el modelo, o criterios de evaluación del rendimiento.

3.4.1. Clasificación supervisada

En el aprendizaje supervisado, el objetivo es predecir el valor de las mediciones de un resultado con base en las mediciones de un número posiblemente grande de entradas. Estas mediciones pueden ser tanto cuantitativas como cualitativas. Usualmente, dentro de la literatura estadística en vez de "mediciones", emplearemos el término "*variables*". Las variables cualitativas también se conocen como categóricas; y en este contexto a veces son llamadas factores. Esta distinción en los tipos de resultados permiten nombrar convenciones según el tipo de tarea predictora: *regresión* cuando predecimos resultados cuantitativos, y clasificación cuando predecimos resultados cualitativos. Las mediciones de los resultados usualmente se nombran *respuestas* (o variables dependientes) y a las mediciones de las entrada se les llama *predictores* (o variables independientes). En otros campos, también es común emplear los términos

atributos o *características* para las variables, y *target* para las respuestas categóricas.

3.4.1.1. La clasificación como un problema de decisión

El problema de clasificación supervisada puede ser tratado como un problema de decisión estadístico. De acuerdo con la Sección 3.3.3 se tienen los siguientes elementos:

1. Espacio de acciones potenciales $a \in \mathcal{A}$.
2. Espacio de estados de la naturaleza: $\theta \in \Theta$.
3. Distribución inicial: $p(\theta)$, definida sobre Θ.
4. Función de pérdida: $L(a, \theta)$.

Regresión

En este caso la respuesta es una variable cuantitativa (continua), y se tiene que $\mathcal{A} \equiv \mathbb{R}, \Theta \equiv \mathbb{R}, a \equiv \hat{y}, \theta \equiv y$, y $L(\hat{y}, y) = (\hat{y} - y)^2$. Entonces

$$L_x^*(\hat{y}) = \int (\hat{y} - y)^2 p(y|\boldsymbol{x})\, dy, \tag{3.8}$$

que alcanza su mínimo cuando

$$\hat{y} = \hat{y}^*(\boldsymbol{x}) \equiv E(Y|\boldsymbol{x}). \tag{3.9}$$

Vamos a dejar de lado los problemas de regresión por ahora, y dedicar el resto de este apartado a la construcción de reglas de clasificación bayesiana. Para ahondar más en los métodos de regresión bayesianos, se recomienda remitirse a la fuente original.

Clasificación

En clasificación la respuesta que se obtiene es una variable cualitativa (o categórica), y por lo tanto $\mathcal{A} \equiv \{1, 2, \ldots, C\}, \Theta \equiv \{1, 2, \ldots, C\}, a \equiv \hat{y}$ y $\theta \equiv y$. En otras palabras, suponemos que la variable de entrada $\boldsymbol{x}$ proviene de una de las C

clases distintas, etiquetadas con $1, 2, \ldots, C$. Es común en los problemas clasificación utilizar funciones de pérdida de la forma $L(\hat{y}, y) = l_{\hat{y}y}$, en donde $l_{\hat{y}y} \geq 0$ denota la pérdida incurrida al escoger la clase $\hat{y}$ cuando la verdadera clase es y. Supóngase, por ejemplo que $l_{\hat{y}y} = 0$ si $\hat{y} = y$, y $l_{\hat{y}y} = 1$ si $\hat{y} \neq y$. Entonces $L_x^*(\hat{y}) = 1 - p(\hat{y}|x)$, que se minimiza cuando

$$\hat{y} = \hat{y}^*(\boldsymbol{x}) \equiv \text{argmax}_{\hat{y}} \; p(\hat{y}|\boldsymbol{x}), \tag{3.10}$$

en donde

$$p(y|\boldsymbol{x}) = \frac{p(y)\, p(\boldsymbol{x}|y)}{\sum_{j=1}^{C} p(j)\, p(\boldsymbol{x}|j)}. \tag{3.11}$$

En otras palabras, para la pérdida 0-1, la solución al problema de clasificación supervisado está dada en términos de la distribución final cuando se cumple (3.10).

3.4.1.2. Análisis discriminante

Hasta aquí se ha supuesto que se conoce la distribución condicional de las clases *i.e.* $p(\boldsymbol{x}|y)$. Bajo condiciones ideales, la ecuación (3.10) logra llevar a la mínima tasa de clasificación errónea; por esa razón se le conoce como *clasificador bayesiano.* Sin embargo, en las aplicaciones no se conoce la forma funcional de $p(\boldsymbol{x}|y)$; pero hay que recordar que en los problemas de aprendizaje supervisado, se tiene una muestra de entrenamiento $(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)$, y es partir de ésta que, en principio, se puede estimar $p(\boldsymbol{x}|y)$.

Para este fin, resulta útil suponer que las variables predictoras de cada clase son independientes entre sí, es decir, que $p(\boldsymbol{x}|y) = \prod_{l=1}^{p} p(x^l|y)$; aunque siendo realistas esta afirmación raramente se cumple en la práctica. Sin embargo, las densidades condicionales de clases frecuentemente se aproximan al producto de las densidades marginales de cada variable predictora de esa misma clase. Este proceso es conocido como *clasificador bayesiano ingenuo* (*naïve Bayesian classifier.*)

Modelos paramétricos

Una aproximación simple para estimar $p(\boldsymbol{x}|y)$, $y = 1, 2, \ldots, C$, es suponer que

$$p(\boldsymbol{x}|y) = f_y(\boldsymbol{x}|\boldsymbol{\theta}_y), \tag{3.12}$$

en donde las densidades $f_y(\boldsymbol{x}|\boldsymbol{\theta}_y)$ son conocidas salvo por el parámetro finito-dimensional $\boldsymbol{\theta}_y$.

Desde un punto de vista clásico, cada $\boldsymbol{\theta}_y$ se estima tomando como base las observaciones de la y-ésima clase $\mathbf{x}_{(y)} = \{\boldsymbol{x}_i : y_i = y; i = 1, \ldots, n\}$. A la estimación obtenida se le denota $\hat{\boldsymbol{\theta}}_y$. La regla de clasificación se obtiene de la ecuación (3.10) al reemplazar $p(\boldsymbol{x}|y)$ por su estimador $\hat{p}(\boldsymbol{x}|y) = f_y(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_y)$ en (3.11). Los "pesos iniciales" $p(y)$, $y = 1, 2, \ldots, C$, generalmente son "estimados" de acuerdo a la proporción de observaciones en la muestra, es decir $\hat{p}(y) = n_y/n$, donde n_y es la cantidad de observaciones en el conjunto $\mathbf{x}_{(y)}$. Esta manera pseudo-bayesiana de proceder no considera la incertidumbre al estimar $\boldsymbol{\theta}_y$, y por tanto puede llevar al error de estimar tasas muy optimistas.

Desde una perspectiva bayesiana, la incertidumbre adicional que surge en relación al valor desconocido de cada parámetro $\boldsymbol{\theta}_y$ debe describirse en términos de una distribución inicial $\pi_y(\boldsymbol{\theta}_y)$ definida sobre el espacio parametral $\boldsymbol{\Theta}_y$. Sean $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C\}$ el vector de parámetros desconocidos y $\mathbf{X} = \{\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(C)}\}$ la colección de todas las muestras de entrenamiento, entonces podemos reformular el problema de decisión del siguiente modo:

1. *Espacio de acciones potenciales*: $\mathcal{A} = \{1, 2, \ldots, C\}$.
2. *Espacio de estados de la naturaleza*: $\Omega = \bigcup_{y=1}^{C} (\{y\} \times \boldsymbol{\Theta}_y)$
3. *Distribución inicial*: $p(y, \boldsymbol{\theta}) = p(y)p(\boldsymbol{\theta}|y)$, con $p(\boldsymbol{\theta}|y) = \pi_y(\boldsymbol{\theta}_y)$. Cuando no se dispone de información sobre las clases verdaderas, es común asignar el valor de $p(y) = 1/C$ para cada $y = 1, 2, \ldots, C$.
4. *Función de pérdida*: $L(\hat{y}, y) = l_{\hat{y}y}$, tal como se describió antes.

Sea $\boldsymbol{x}$ una nueva observación por clasificar. La distribución final está dada por

$$p(y, \boldsymbol{\theta}|\boldsymbol{x}, \mathbf{X}) = p(y|\boldsymbol{x}, \mathbf{X})\, p(\boldsymbol{\theta}|y, \boldsymbol{x}, \mathbf{X}).$$

Obsérvese que una vez que se conoce la forma de la función de pérdida, todo lo que se necesita para calcular la pérdida esperada final es $p(y|\boldsymbol{x}, \mathbf{X})$, que puede obtenerse de la siguiente forma

$$p(y|\boldsymbol{x}, \mathbf{X}) = \frac{p(y|\mathbf{X})p(\boldsymbol{x}|y, \mathbf{X})}{\sum_{j=1}^{C} p(j|\mathbf{X})p(\boldsymbol{x}|j, \mathbf{X})}, \tag{3.13}$$

en donde

$$p(y|\mathbf{X}) = \frac{p(y)p(\boldsymbol{X}|y)}{\sum_{j=1}^{C} p(j)p(\mathbf{X}|j)}, \tag{3.14}$$

y

$$p(\boldsymbol{x}|y, \mathbf{X}) = \int f_y(\boldsymbol{x}|\boldsymbol{\theta}_y)\pi_y(\boldsymbol{\theta}_y|\mathbf{X})\, d\boldsymbol{\theta}_y$$

es la distribución predictiva final para muestras de la y-ésima clase.

Aquí,

$$p(\boldsymbol{X}|y) = p(\mathbf{x}_{(y)}|y) = \int f_y(\mathbf{x}_{(y)}|\boldsymbol{\theta}_y)\pi_y(\boldsymbol{\theta}_y)\, d\boldsymbol{\theta}_y,$$

y

$$\pi_y(\boldsymbol{\theta}_y|\mathbf{X}) = \pi_y(\boldsymbol{\theta}_y|\mathbf{x}_{(y)}) = \frac{\pi_y(\boldsymbol{\theta}_y)f_y(\mathbf{x}_{(y)}|\boldsymbol{\theta}_y)}{\int \pi_y(\tilde{\boldsymbol{\theta}}_y)f_y(\mathbf{x}_{(y)}|\tilde{\boldsymbol{\theta}}_y)\, d\tilde{\boldsymbol{\theta}}_y} \tag{3.15}$$

es la distribución final de $\boldsymbol{\theta}_y$ dados los datos de entrenamiento.

En el caso de la pérdida 0-1 explicada en la Sección 3.4.1.1, la regla de clasificación resultante, toma la forma

$$\hat{y} = \hat{y}^*(\boldsymbol{x}) \equiv \text{argmax}_{\hat{y}} \;\; p(\hat{y}|\boldsymbol{x}, \mathbf{X}), \tag{3.16}$$

Una forma de interpretar el procedimiento anterior es la siguiente: primero, se hace uso de la muestra de entrenamiento para ajustar la probabilidad de cada clase a través de (3.14), y para inferir los valores desconocidos del los parámetros $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C$ se emplea (3.15). Luego, la nueva observación $\boldsymbol{x}$ se clasifica de acuerdo a (3.16). Observe que la distribución final (3.13) es similar a (3.11) que se emplea para clasificar en el caso en que las densidades condicionales de clase son totalmente especificadas. De hecho, en lugar del clásico estimador *plug-in* $f_y(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_y)$, aquí la aproximación bayesiana "estima" a la densidad condicional de clase (que es desconocida) empleando la densidad predictiva final $p(\boldsymbol{x}|y, \mathbf{X})$. De manera similar, los "pesos iniciales" se ajustan empleando (3.14) en lugar de las proporciones de la muestra $\hat{p}(y) = n_y/n$. Esto proporciona una regla de clasificación completamente bayesiana.

Modelos no paramétricos

Los modelos paramétricos suelen ser demasiado restrictivos como para capturar algunas de las características más importantes de los conjuntos de datos que surgen en las aplicaciones. En un intento de enfrentar este problema, los métodos no

paramétricos consideran a la propia densidad condicional de clase $p(\boldsymbol{x}|y)$ como un parámetro desconocido. Una vez encontrada la estimación $\hat{p}(\boldsymbol{x}|y)$ se procede de manera similar que en el punto anterior pero utilizando $\hat{p}(\boldsymbol{x}|y)$ en lugar de la estimación (paramétrica) *plug-in* $f_y(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_y)$.

Aunque hay varios métodos no paramétricos para hacer aproximaciones, el presente trabajo se centra en el método de los k vecinos más cercanos (*k-nearest-neighbours method*). Se calcula $p(y|\boldsymbol{x})$ de la siguiente manera

1. Se emplea alguna métrica previamente definida para medir la distancia $\delta_i = ||\boldsymbol{x} - \boldsymbol{x}_i||$ entre el i-ésimo vector predictor y el vector al que deseamos predecirle una etiqueta de clase.

2. Definimos a $\delta_{(i)}$ como la i-ésima distancia ordenada, así pues $\delta_{(1)}$ es la mínima distancia observada y $\delta_{(n)}$ es la máxima. Sean $y_{(1)}, \ldots, y_{(n)}$ las etiquetas de clase correspondientes, es decir, están ordenadas de la misma forma que las $\delta_{(i)}$.

3. Se escoge un valor para k $(k = 1, 2, \ldots, n)$, y se predice y utilizando la clase que sea más frecuente en $y_{(1)}, \ldots, y_{(k)}$.

En otras palabras, el procedimiento anterior consiste en encontrar los k puntos (datos) más cercanos a $\boldsymbol{x}$ y predecir y con base en las etiquetas de clase de los k puntos más cercanos. Así, efectivamente se estima $p(y|\boldsymbol{x})$ usando la muestra pero de manera diferente a como se emplea en la Sección 3.4.1.1.

Existen métodos bayesianos para obtener versiones no paramétricas de (3.13), de donde se deriva una regla de clasificación bayesiana no paramétrica, análoga a (3.16). Así también existe una versión probabilística de los k vecinos más cercanos, pero esos temas escapan de los objetivos de este trabajo.

3.4.1.3. Regresión logística

El modelo de regresión logística surgió en un contexto paramétrico con la idea de poder modelar directamente la distribución final $p(y|\boldsymbol{x})$ utilizando funciones lineales sobre $\boldsymbol{x}$. Para clarificar la idea de este método, supongamos por el momento que sólo se tienen $C = 2$ clases. Generalmente, se supone que $p(y|\boldsymbol{x})$ es tal que

$$\log\left\{\frac{p(1|\boldsymbol{x})}{1-p(1|\boldsymbol{x})}\right\} \equiv \log\left\{\frac{\Pr(Y=1|\boldsymbol{x})}{\Pr(Y=2|\boldsymbol{x})}\right\} = \alpha + \boldsymbol{\beta}^T\boldsymbol{x}, \tag{3.17}$$

en donde $\alpha \in \mathbb{R}$ y $\boldsymbol{\beta} \in \mathbb{R}^p$ son parámetros desconocidos. De este modo, los logaritmos son tratados como funciones lineales de $\boldsymbol{x}$. Existen otros modelos logísticos no paramétricos de la forma

$$\log\left\{\frac{p(1|\boldsymbol{x})}{1-p(1|\boldsymbol{x})}\right\} = g(\boldsymbol{x}),$$

donde $g(\cdot)$ es una función suave desconocida, pero no será discutida en el presente trabajo. Una vez estimado $\hat{p}(\boldsymbol{x}|y)$ con base en (3.17), se puede obtener una regla de clasificación de (3.10) básicamente de la misma manera que se obtiene en la Sección 3.4.1.1.

Para extender la noción anterior a más de dos clases, hay que considerar a un conjunto de $C-1$ logaritmos, del siguiente modo

$$\log\left\{\frac{p(y|\boldsymbol{x})}{p(C|\boldsymbol{x})}\right\} \equiv \log\left\{\frac{\Pr(Y=y|\boldsymbol{x})}{\Pr(Y=C|\boldsymbol{x})}\right\} = \alpha_y + \boldsymbol{\beta}_y^T\boldsymbol{x}; \qquad y = 1, 2, \ldots, C-1.$$

3.4.2. Clasificación no supervisada

En la Sección 3.4.1 ya se explicó que el objetivo del aprendizaje supervisado es inferir el valor de un resultado con base en los datos de una muestra. En contraste, la meta del aprendizaje no supervisado es describir los patrones y relaciones que existen entre un conjunto de datos; sin que haya una muestra previa como guía.

Entre los métodos de aprendizaje no supervisado, los más populares son los referentes al *análisis de conglomerados*, también conocido como *segmentación de la muestra*. En el análisis de conglomerados, se tiene una muestra de n observaciones $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ que se agrupan en K conjuntos mutuamente excluyentes, de acuerdo a qué tan cercana es la relación que tienen las observaciones entre sí. La K puede o no estar previamente especificada.

Cuando se tienen datos de observaciones continuas, la forma más común para hacer aproximaciones es suponer un modelo para el que se tiene una mezcla (por lo general de distribuciones normales multivariadas), en donde cada moda de la mezcla estimada corresponde a un agrupamiento de los datos. Al hacer la suposición de que las variables son no correlacionadas con las distribuciones de la mezcla, se llega a un método conocido como *análisis de perfil latente*; y si es el caso en que se tienen datos multivariados categóricos, entonces recibe el nombre de *análisis de clases latentes.*

De aquí se puede hacer la suposición, poco realista de que las llamadas *variables de manifiesto* son independientes dentro de cada *clase latente*; pero no abordaremos en ese estudio. Dentro del análisis de clases latentes, la variable latente es categórica, y es ésta la que determina la estructura de los datos. Existen métodos que se obtienen cuando la variable latente es continua y se les conoce como *análisis de rasgo latente* pero también salen de los alcances de este trabajo.

3.4.2.1. Análisis bayesiano de conglomerados

El análisis bayesiano de conglomerados (*Bayesian cluster analysis*) parte de la siguiente formula general, en donde se especifica una densidad inicial conjunta con respecto al número de conglomerados K, y a los parámetros de las densidades en las componentes. El modelo toma la forma de una mezcla:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{j=1}^{K} w_j \, p_j(\boldsymbol{x}|\boldsymbol{\theta}_j).$$

Sea $\boldsymbol{z}_i = (z_{i1}, \ldots, z_{iK})$ un vector indicador, tal que $z_{ij} = 1$ si la observación $\boldsymbol{x}_i$ pertenece al conglomerado j, y $z_{ij} = 0$ en otro caso; para $i = 1, \ldots, n$; $j = 1, \ldots, K$. El conjunto de vectores indicadores $\mathbf{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)$ permite representar cualquier grupo de datos. Así que la distribución inicial se puede escribir como

$$\pi(k, \mathbf{z}, \boldsymbol{\theta}) = \pi(k)\pi(\mathbf{z}|k)\pi(\boldsymbol{\theta}|k, \mathbf{z}).$$

Esto se combina con la versomilitud

$$\prod_{i=1}^{n} \prod_{j=1}^{K} \{p_j(\boldsymbol{x}_i|\boldsymbol{\theta_j})\}^{z_{ij}} \tag{3.18}$$

para obtener la densidad final $\pi(k, \mathbf{z}, \boldsymbol{\theta}|\mathbf{x})$. Observe que lo anterior será cero a menos que $\boldsymbol{z}$ corresponda a una agrupación verdadera de los K conglomerados. Las probabilidades finales marginales de agrupaciones particulares,

$$\pi(\mathbf{z}|\mathbf{x}) = \sum_{k} \int \pi(k, \mathbf{z}, \boldsymbol{\theta}|\mathbf{x}) \, d\boldsymbol{\theta},$$

proporcionan la base para escoger la "mejor" agrupación de los datos. En algunas aplicaciones de este problema de decisión, proceder así reproduce criterios familiares de agrupamiento; sin embargo, la gran mayoría de los cálculos que hay que resolver

representan un verdadero problema analítico. Los avances en computación han hecho posible el desarrollo del análisis bayesiano de mezclas y han impulsado el desarrollo de procedimientos sofísticados de métodos bayesianos no supervisados.

3.4.2.2. Aprendizaje no supervisado como aprendizaje supervisado

Existe una técnica interesante que transforma el problema de estimar $p(\boldsymbol{x})$ en un problema supervisado de estimar una función. Este método es prometedor puesto que en la literatura hay un mayor desarrollo en las técnicas de aprendizaje supervisado, en comparación con las tradicionales de aprendizaje no supervisado. A continuación describimos este enfoque.

Sea $p(\boldsymbol{x})$ la densidad desconocida que se desea estimar con base en la muestra $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, y sea $p_0(\boldsymbol{x})$ una función de densidad que se emplea como referencia. Por ejemplo, $p_0(\boldsymbol{x})$ puede ser la distribución uniforme sobre el rango de las variables. A partir de $p_0(\boldsymbol{x})$ se simula una muestra de tamaño n. Estos dos conjuntos de datos se agrupan y se les asigna peso de 1/2 tanto a la muestra de $p(\boldsymbol{x})$ como a la muestra simulada a partir de $p_0(\boldsymbol{x})$. Lo que se obtiene es una muestra aleatoria de la mezcla $p_m(\boldsymbol{x}) = \{p(\boldsymbol{x}) + p_0(\boldsymbol{x})\}/2$. Si asignamos el valor de $Y = 1$ para cada punto muestral de $p(\boldsymbol{x})$ y $Y = 0$ para los puntos de $p_0(\boldsymbol{x})$, entonces

$$\mu(\boldsymbol{x}) = E(Y|\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{p(\boldsymbol{x}) + p_0(\boldsymbol{x})}$$

puede ser estimada como un problema de aprendizaje supervisado empleando a la muestra agrupada $(y_1, \boldsymbol{x}_1), \ldots, (y_{n+n_0}, \boldsymbol{x}_{n+n_0})$ como datos de entrenamiento. La estimación $\hat{\mu}(\boldsymbol{x})$ que se obtiene, se invierte para proporcionar un estimador de $p(\boldsymbol{x})$:

$$\hat{p}(\boldsymbol{x}) = p_0(\boldsymbol{x}) \frac{\hat{\mu}(\boldsymbol{x})}{1 - \hat{\mu}(\boldsymbol{x})}.$$

Existe una versión generalizada de la técnica de regresión logística (Sección 3.4.1.3) que se adecua muy bien a esta aplicación; en esta, los logaritmos $\lambda(\boldsymbol{x}) = \log\{p(\boldsymbol{x})/p_0(\boldsymbol{x})\}$ son estimados directamente y haciendo un despeje se llega a

$$\hat{p}(\boldsymbol{x}) = p_0(\boldsymbol{x}) \exp\{\hat{\lambda}(\boldsymbol{x})\}.$$

3.5. Métodos basados en árboles (CART)

El contenido de esta sección fue tomado de Hastie et al. (2009). Los métodos basados en árboles (*tree-based methods*) son un tipo de aprendizaje supervisado que ha cobrado relevancia en los últimos años. Estos métodos particionan el espacio de características o mediciones en rectángulos, y a cada rectángulo se le asigna el valor de una constante o de algún otro modelo simple. A continuación se describe un método de regresión y clasificación que se basa en esta idea y recibe el nombre de CART por las siglas en inglés de *Classification And Regression Tree*.

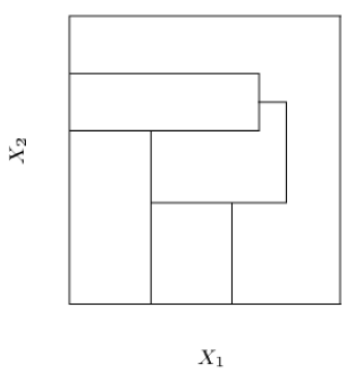
Para ejemplificar, consideremos el problema de regresión con entradas X_1 y X_2 en el intervalo unitario, y respuesta continua Y. En la Figura 3.2a se muestra una partición del espacio en regiones que están divididas por líneas paralelas a los ejes coordenados. En cada partición se puede modelar Y con una constante distinta. Sin embargo, la forma de conseguir determinadas regiones a partir de un árbol (como las de la Figura 3.2a), puede ser una tarea muy difícil o incluso imposible.

Para simplificar la tarea de estudiar los árboles, nos centraremos en el estudio de árboles que particionan recursivamente el espacio de manera binaria, como se muestra en la Figura 3.2b. Para esto, primero se divide el espacio en dos regiones; y se modela la respuesta obteniendo la media de Y en cada región. Se escoge la variable y el punto de división que mejor se ajuste. Después, una de las regiones o las dos, vuelven a dividirse en otras dos regiones, y este proceso continua hasta que se cumpla alguna regla de paro.

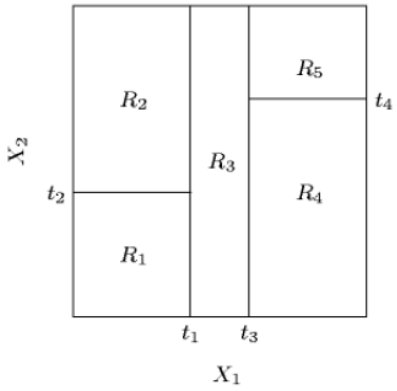
Por ejemplo, en la Figura 3.2b, primero se divide en el punto $X_1 = t_1$. Luego, la región $X_1 \leq t_1$ se divide en $X_2 = t_2$ y la región $X_1 > t_1$ se divide en $X_1 = t_3$. Finalmente, la región $X_1 > t_3$ se divide en $X_2 = t_4$. Como resultado de este proceso se crean las cinco regiones $R_1, R_2, \ldots, R_5$ que se aprecian en la figura. El correspondiente modelo de regresión predice Y con la constante c_m, para la región R_m, del siguiente modo:

$$\hat{f}(\boldsymbol{X}) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}. \tag{3.19}$$

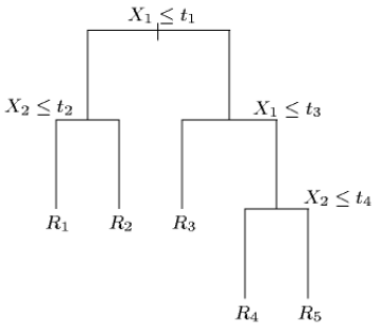
Este mismo modelo puede ser representado por el árbol binario en la Figura 3.2d. Iniciando con todos los datos en la raíz del árbol, a cada nodo las observaciones se separan a la izquierda si cumplen la condición o a la derecha si no la cumplen. Una *hoja* es un nodo terminal del árbol, que en este caso corresponde a alguna de las



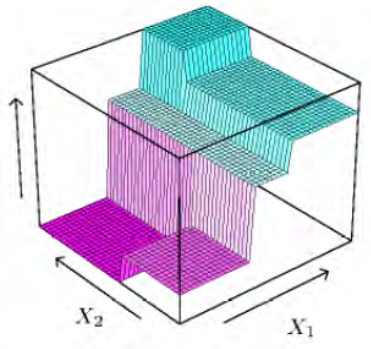
(a) Partición que no puede ser obtenida a partir de divisiones binarias recursivas.



(b) Partición del espacio hecha por divisiones binarias recursivas (tal como se hace en CART).



(c) Árbol correspondiente a la partición (b).



(d) Gráfica de las predicciones de la partición (b).

Figura 3.2: Regresión basada en árboles binarios. (*Imagen tomada de Hastie et al. (2009), pág. 268*).

regiones $R_1, \ldots, R_5$. La Figura 3.2d es una gráfica de la superficie de regresión para este modelo. Con fines ilustrativos, se designa $c_1 = -5, c_2 = -7, c_3 = 0, c_4 = 2$ y $c_5 = 4$.

La representación de árboles binarios es popular sobre todo entre los médicos, ya que ésta simula la forma en que suelen pensar los doctores; por ejemplo, al clasificar a los pacientes que llegan a la unidad de urgencias de un hospital. El árbol separa a la población dentro de estratos con respuestas altas o bajas, sobre la base de las características del paciente.

3.5.1. Árboles de regresión

Pasemos ahora a la cuestión de cómo hacer crecer un árbol de regresión. Los datos constan de N observaciones, cada una con p entradas y una respuesta, es decir, $(\boldsymbol{x}_i, y_i)$ para $i = 1, 2, \ldots, N$, donde $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. El algoritmo necesariamente tiene que elegir las variables que van a dividir los datos y los puntos de división de las mismas; y también, determina la forma del árbol. Supongamos, para empezar, que se tiene una partición en M regiones $R_1, R_2, \ldots, R_M$, y se modela la respuesta como una constante c_m en cada región:

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m I(\boldsymbol{x} \in R_m). \tag{3.20}$$

Adoptando como función de pérdida a $(y_i - f(\boldsymbol{x}_i))^2$, se puede demostrar que para la suma de cuadrados $\sum (y_i - f(\boldsymbol{x}_i))^2$ el mejor $\hat{c}_m$ es el promedio de las y_i en la región R_m:

$$\hat{c}_m = \text{Promedio}(y_i | \boldsymbol{x}_i \in R_m). \tag{3.21}$$

Se utiliza un *algoritmo voraz* (*greedy algorithm*)[3]. Empezando con todo el conjunto de datos, se considera una variable divisora j y un punto de división s para definir el par de semi-planos

$$R_1(j, s) = \{\boldsymbol{X} | X_j \leq s\} \quad \text{y} \quad R_2(j, s) = \{\boldsymbol{X} | X_j > s\}. \tag{3.22}$$

[3]Un algoritmo es *voraz* si construye una solución a través de pequeños pasos, de modo que en cada paso selecciona la opción que optimice algún criterio específico, pero sin tomar en cuenta lo que está más allá del estado actual. En otras palabras, el algoritmo voraz consiste en ejecutar una regla de decisión local a cada paso, con la intensión de llegar a un óptimo global al final de su ejecución.

Entonces, se encuentra la variable j y el punto de división s que resuelva

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right], \tag{3.23}$$

donde para cada j y s, la minimización interna se resuelve con

$$\hat{c}_1=\text{Promedio}(y_i|\boldsymbol{x}_i\in R_1(j,s)) \;\text{ y }\; \hat{c}_2=\text{Promedio}(y_i|\boldsymbol{x}_i\in R_2(j,s)). \tag{3.24}$$

Una vez encontrada el mejor punto de división (j,s), se dividen los datos en dos regiones y se repite el proceso en cada una de ellas. Este proceso es repetido en todas las regiones resultantes.

¿Qué tanto se debe dejar crecer el árbol? Un árbol muy grande podría sobreajustar los datos, mientras que un árbol muy pequeño podría no capturar una estructura importante. El tamaño ideal del árbol es un problema que depende de la complejidad del modelo y de los datos que se empleen.

La estrategia más popular para resolver este problema es dejar crecer un árbol T_0, y sólo detener el proceso de división cuando se alcanza algún número mínimo de datos en los nodos. Después, este árbol grande se poda utilizando un método que se conoce como *podado de costo-complejidad* (*cost-complexity pruning*). A continuación se describe dicho método.

Se define un sub-árbol $T\subset T_0$ que puede ser cualquier árbol obtenido a partir de podar a T_0; esto es, colapsando cualquier cantidad de nodos internos (no hojas). Luego se vuelven a indexar las m hojas que queden; donde m representa la región R_m y N_m la cantidad de objetos en la hoja m. Sea $|T|$ el número de nodos terminales en T. Haciendo

$$\begin{aligned}\hat{c}_m&=\frac{1}{N_m}\sum_{x_i\in R_m}y_i,\\ Q_m(T)&=\frac{1}{N_m}\sum_{x_i\in R_m}(y_i-\hat{c}_m)^2,\end{aligned} \tag{3.25}$$

definimos el criterio de costo-complejidad como

$$C_\alpha(T)=\sum_{m=1}^{|T|}N_mQ_m(T)+\alpha|T|. \tag{3.26}$$

La idea es encontrar para cada α, un sub-árbol $T_\alpha \subset T_0$ que minimice a $C_\alpha(T)$. El parámetro de ajuste $\alpha \geq 0$ balancea la preferencia entre el tamaño del árbol y la calidad en el ajuste de los datos. Los valores grandes de α dan como resultado pequeños árboles T_α, e inversamente para valores pequeños de α.

3.5.2. Árboles de clasificación

Si el objetivo es clasificar las respuestas que toman los valores $1, 2, \ldots, K$, los únicos cambios necesarios en el algoritmo anterior son con respecto a los criterios para los nodos divisores y para podar el árbol. Para regresión se emplea la medida $Q_m(T)$ de error cuadrático definida en (3.25), sin embargo esta medida no es adecuada para tareas de clasificación. Para un nodo terminal m que representa a la región R_m con N_m observaciones, sea

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \tag{3.27}$$

la proporción de las observaciones que pertenecen a la clase k en el nodo m. Las observaciones del nodo m serán clasificadas dentro de la clase $k(m) = \arg\max_k \hat{p}_{km}$, la clase que tiene más observaciones dentro del nodo m. Algunas de las medidas $Q_m(T)$ más comunes son las siguientes:

- Tasa de clasificación errónea (*Misclasification error*):

$$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.$$

- Índice Gini (*Gini index*):

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

- Entropía cruzada o desviación (*cross-entropy or deviance*):

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

Las tres medidas son similares, pero la entropía cruzada y el índice Gini son diferenciables, y por lo tanto más adecuados para una optimización numérica. En general, es más adecuado emplear el índice Gini o la entropía cruzada para hacer crecer el árbol. En cambio, para dirigir el podado de costo-complejidad se suele emplear la tasa de clasificación errónea; aunque en realidad, cualquiera de los tres métodos puede utilizarse. La elección anterior se debe a que el índice Gini y la entropía cruzada son más sensibles a los cambios en las probabilidades de los nodos, lo que los hace más eficientes para desarrollar la tarea de crecimiento del árbol.

3.5.3. Dos observaciones importantes

Cuando se trabaja con el método CART, se debe tener en cuenta lo siguiente:

- Aunque la división descrita en este algoritmo se hace de manera binaria, también se puede realizar una división múltiple. Sin embargo, la división múltiple fragmenta los datos demasiado rápido, dejando muy pocos datos para el siguiente nivel. Como las divisiones múltiples se pueden alcanzar a través de divisiones binarias, en general se recomiendan las segundas.

- Uno de los problemas más grandes al trabajar con árboles, es su alto grado de variabilidad[4]. Es común que con pequeños cambios en los datos se obtengan divisiones muy diferentes. La razón de esta inestabilidad es la naturaleza jerárquica del proceso: el efecto de un error en la primera división se propaga hacia abajo, alterando a todas las divisiones subsecuentes. En la Sección 5.1, se aprovecha esta característica de los árboles para diseñar un algoritmo genético para optimizar el método CART.

[4]Existe una técnica llamada *bosques aleatorios* (*random forests*) con la cuál se reduce la varianza de los árboles de regresión y clasificación. Para lograr esto, se crean B nuevas muestras de entrenamiento Z_b de tamaño N, a partir una selección uniforme y con reemplazo de la muestra de entrenamiento original (*bootstrapping*). En cada muestra Z_b ($b = 1, \ldots, B$) se hace crecer un árbol T_b. Para tareas de regresión se utiliza el promedio de la regresión que arrojan los B árboles; y para tareas de clasificación se elige la clase que más veces se repita. Como cada árbol generado en la nueva muestra es identicamente distribuido, la esperanza del promedio de los árboles es igual a la esperanza de cualquiera de ellos; con la ventaja de que al aumentar el número B de árboles, disminuye la varianza del promedio. Para ahondar más en este tema, puede revisarse Hastie et al. (2009).

Capítulo 4

Teoría de la Decisión Markoviana

Nota para el lector. Las fuentes primarias para desarrollar este capítulo y el Capítulo 5 fueron Kochenderfer (2015) y Sutton and Barto (2017), que pertenecen a la literatura propia de las ciencias de la computación. En las mismas, no se hace una clara distinción en la notación empleada para referirse a algunos términos matemáticos precisos. Por ejemplo, se usa indistintamente P y p para denotar a una función de densidad. Tratando de poner más rigor matemático, en general emplearemos la notación propia de la estadística bayesiana, a menos de que el contexto haga más complicado usar una notación rigurosa. La siguiente tabla es una base de la notación que se emplea en esta tesis:

Notación	**Significado**
$\Pr(\cdot)$	Función de probabilidad.
$p(\cdot)$	Función de distribución de alguna v.a.
$A, B, \ldots, Z$	Variables aleartorias.
$a, b, \ldots, z$	Posibles valores que pueden tomar las v.a.'s $A, B, \ldots, Z$.
$p(a)$	Función de distribución de la v.a. A.
$p(B)$	Distribución inicial de la familia de redes bayesianas $\mathcal{B}_n$.

En general se evita utilizar letras mayúsculas dentro de la función de densidad, por ejemplo: $p(A)$; esto porque se puede entender como la transformación de la v.a A bajo alguna función de distribución no especificada. Sin embargo se decidió dejar como excepción el caso de $p(B)$ para ser congruentes con la literatura empleada en las ciencias de la computación y porque $B \in \mathcal{B}_n$ es la notación empleada para referirse a una red bayesiana dentro de la familia de redes con n variables (véase la Sección 4.3). Con el fin de hacer más didácticos los ejemplos, se ha respetado la notación

en mayúsculas de cualquier letra para denotar una variable aleatoria (en lugar de reservar X, Y, Z, W para v.a. y A, B, C, D para eventos en el espacio muestral). Así mismo, se ha mantenido otra notación que creemos no representa mayor confusión, la cual se explica a lo largo del texto. La Sección 4.3 se basa principalmente en Carvalho (2009).

4.1. Representación de la red bayesiana

Una red bayesiana es una representación compacta de una distribución conjunta. La estructura de la red es representada por una gráfica constituida por nodos y aristas dirigidas. Cada nodo corresponde a una variable aleatoria. Las aristas dirigidas conectan nodos, y en ocasiones se les llama flechas. Los ciclos en las gráficas quedan prohibidos. Las flechas indican una relación probabilística directa; de este modo, cada nodo X_i tiene asociada una distribución de probabilidad condicional $p(x_i|pa_{x_i})$. Aquí Pa_{X_i} representa a los padres de X_i en la gráfica y pa_{x_i} al evento en que los padres de X_i realizan específicamente la tarea x_i; es decir si X_j es el único padre de X_i, entonces pa_{x_i} es el evento $X_j = x_j$.

Para clarificar esta idea, comencemos con un ejemplo. Supongamos que se está monitoreando un satélite que se encuentra a miles de kilómetros de la tierra. El satélite manda reportes varias veces al día, pero de manera súbita deja de hacerlo. Hay varias razones por las que esto puede ocurrir, como una falla del sistema eléctrico o una falla del sistema de comunicación. Más aún, una falla en los motores propulsores podría generar un desvío considerable de la trayectoria original. Sin embargo, con la información que se tiene es imposible hacer un diagnóstico completo.

En la Figura 4.1 se muestra un ejemplo de una red bayesiana con cinco variables aleatorias binarias. Los padres son las v.a. B y S que modelan las probabilidades de los eventos falla de la batería y falla de los paneles solares, respectivamente. Por fortuna es muy raro que ambos fallen al mismo tiempo; de hecho es más probable tener una falla en la batería que en los paneles solares. Las fallas en cualquiera de las dos pueden derivar en una falla del sistema eléctrico (representada por la v.a. E); por otro lado, hay más variables que pueden llevar a una falla del sistema eléctrico que no están estrechamente relacionadas con las anteriores (por ejemplo, un problema con la unidad de poder), pero por razones de simplicidad no aparece esa opción en esta red. Una falla en el sistema eléctrico puede llevar a una desviación de la trayectoria (que por fortuna puede ser verificada por el telescopio desde la tierra), o también en

una falla del sistema de comunicación. Estas últimas v.a. están representadas por las letras D y C respectivamente.

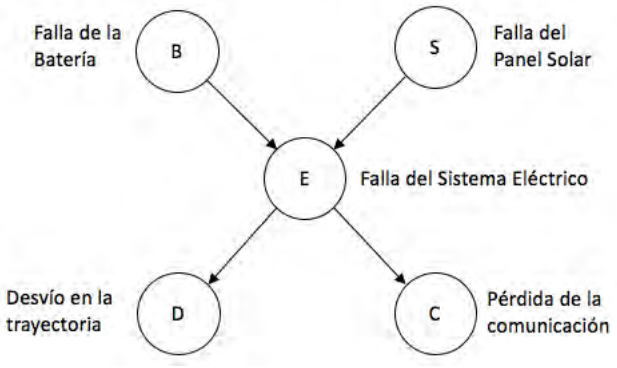


Figura 4.1: Ejemplo de la estructura de una red bayesiana.

Para cada uno de los cinco nodos de la Figura 4.1, se encuentra asociada una distribución condicional de probabilidad tal como se muestra en la Figura 4.2. Como los nodos B y S no tienen padres, entonces su distribución asociada es $p(b)$ y $p(s)$, respectivamente. Obsérvese que, dependiendo del problema que se esté representando, puede existir alguna dependencia entre la variable aleatoria B y la v.a. S. En el ejemplo anterior existe una fuerte dependencia, pues si se confirma que hay una falla de la batería, es muy poco probable que también exista una falla en el panel solar y viceversa. El nodo central E tiene dos padres por lo que su distribución de probabilidad condicional es $p(e|b,s)$. Las otras dos distribuciones son $p(d|e)$ y $p(c|e)$.

Decimos que un conjunto de nodos $\mathcal{C}$ $d-$separa a los nodos A y B si A y B son *condicionalmente independientes* dado $\mathcal{C}$ (la letra "d" proviene de la palabra "direccional"), en símbolos:

$$p(a,b|\mathcal{C}) = p(a|\mathcal{C})p(b|\mathcal{C}).$$

Obsérvese que lo anterior implica que $p(a|\mathcal{C},b) = p(a|\mathcal{C})$.

La estructura de la red bayesiana especifica también una relación de *independencia condicional* entre los nodos. Diremos que el camino entre A y B es $d-$separado por $\mathcal{C}$ si se cumple alguna de las siguientes situaciones:

1. El camino contiene una cadena de nodos (*chain*) $A \longrightarrow C \longrightarrow B$, en donde C pertenece a $\mathcal{C}$.

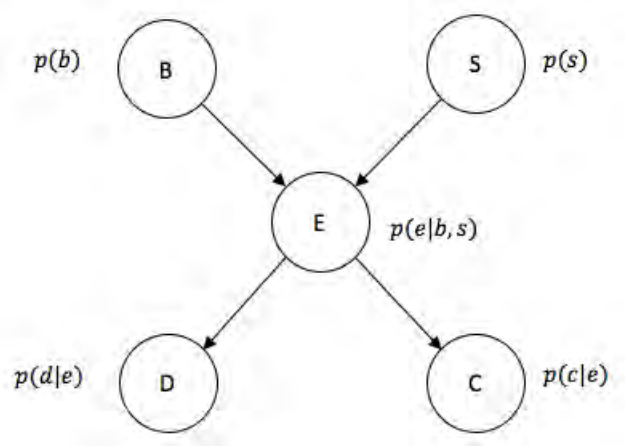


Figura 4.2: Probabilidades condicionales en una red bayesiana.

2. El camino contiene una bifurcación (*fork*) $A \longleftarrow C \longrightarrow B$, en donde C pertenece a $\mathcal{C}$. Este es el caso de la Figura 4.1.

3. El camino contiene una estructura en "V"(*inverted fork*) $A \longrightarrow C \longleftarrow B$, en donde C pertenece a $\mathcal{C}$.

En algunas ocasiones se utiliza el término *Markov blanket* para referirse al conjunto mínimo de nodos en una gráfica que $d-$separan a un nodo de todos los demás. Por ejemplo, en la Figura 4.2 el *Markov blanket* es el conjunto que únicamente contiene al nodo E.

Introduzcamos ahora la notación usual en el contexto de las redes bayesianas y que se empleará a lo largo de este capítulo: se escribe $\Pr(a^j)$ para referirse a la probabilidad de que una v.a. discreta A tome el valor j, en símbolos $\Pr(A = j)$. En otras palabras, a^j hace referencia al evento $\{A = j\}$. Otra notación usual para referirse a una colección de v.a.'s $X_1, \ldots, X_n$ es $X_{1:n}$ y se emplea $x_{i:n}$ para referirse a la colección de sus resultados respectivos $x_1, \ldots, x_n$.

Bajo la suposición de *independencia condicional*, la *regla de la cadena* aplicada a la red bayesiana especifica cómo obtener la densidad conjunta a partir de las densidades asociadas a cada nodo; en efecto, la regla de la cadena en la red queda de la siguiente forma:

$$\Pr(x_{1:n}) = \prod_{i=1}^{n} p(x_i | pa_{x_i}). \tag{4.1}$$

El poder de las redes bayesianas recae en su habilidad para reducir el número de parámetros requeridos para especificar una distribución conjunta de probabilidad.

Las redes bayesianas pueden usar variables discretas o continuas; cuando una red mezcla ambas distribuciones se le llama híbrida (*hybrid Bayesian network*).

4.1.1. Modelos temporales

Un *modelo temporal* representa cómo un conjunto de variables evolucionan con el tiempo. Un modelo temporal simple es una *cadena de Markov* en donde el estado al tiempo t se representa por S_t. La Figura 4.3 muestra la estructura de una red bayesiana, que a su vez representa una cadena de Markov. En la figura sólo se muestran los tres primeros estados pero la cadena puede tener un número infinito de estados. La distribución inicial está dada por $p(s_0)$. A la distribución condicional $p(s_t|s_{t-1})$ se le conoce como *modelo de transición de estados* (*state transition model*). Si la distribución de transición de estados no varía al cambiar el tiempo t, entonces diremos que el modelo es *estacionario*.

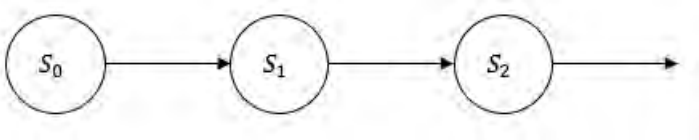


Figura 4.3: Cadena de Markov.

Las cadenas de Markov pueden extenderse añadiendo nodos de observación. En este caso la cadena recibe el nombre de *modelo oculto de Markov* y se muestra en la Figura 4.4. La observación al tiempo t se denota por O_t. Las observaciones se sombrean para indicar que los valores de esos nodos son conocidos. Por ejemplo, si los estados corresponden a la posición y velocidad de un avión, las observaciones podrían ser los datos que arroja un radar sobre el rango y azimut del avión.

Los modelos temporales *estacionarios* pueden representarse de manera abreviada utilizando dos redes bayesianas. La primera representa la distribución inicial y la otra representa las distribuciones de transición. La red de la distribución de transición tiene dos capas; en la primera se representan las variables al tiempo t y en la segunda al tiempo $t + 1$. A esta representación se le conoce como *red dinámica bayesiana*.

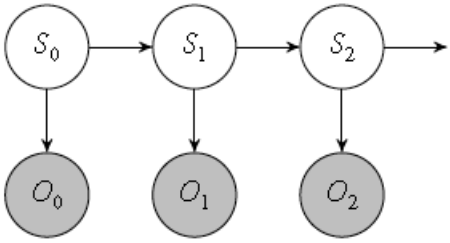


Figura 4.4: Nodos de observación en una CM. (*Modelo oculto de Markov*).

4.1.2. Inferencia para clasificación

Tal como se explicó en la Sección 3.4, se puede hacer inferencia para realizar tareas de clasificación. Dentro de los métodos supervisados se tiene el objetivo de inferir la clase a la que pertenece cierto objeto dado un conjunto de observaciones. Un modelo simple que se utiliza con frecuencia para estas tareas es el modelo ingenuo de Bayes (*naïve Bayes model*); para ahondar en este modelo véase la Sección 3.4.1.2. Su estructura puede representarse en una red bayesiana (Figura 4.5).

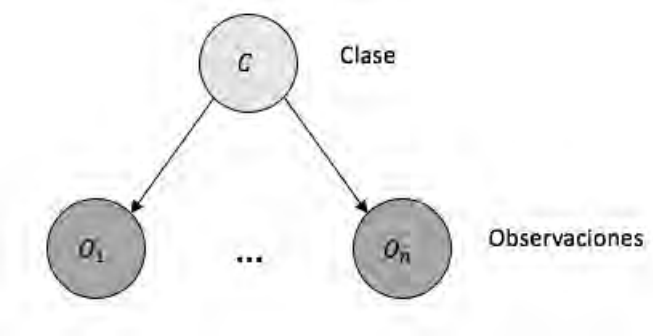


Figura 4.5: Inferencia para clasificación utilizando un modelo *naïve Bayes*.

En el modelo *naïve Bayes*, la clase C es una variable incierta, y las observaciones $O_1, \ldots, O_n$ son variables de evidencia. Bajo el supuesto de que existe independencia condicional entre las variables de evidencia dada la clase C, se tiene que: $p(o_i, o_j|c) = p(o_i|c)p(o_j|c)$. En este modelo, tenemos que especificar una distribución inicial (*prior*) y las distribuciones condicionales $p(o_i|c)$.

Utilizando las reglas de la probabilidad, las propiedades de independencia con-

dicional de la red bayesiana y finalmente la regla de la cadena (ecuación (4.1)), se obtiene que la distribución de la clasificación condicionada por las observaciones registradas es

$$\begin{aligned} p(c|o_1, \ldots, o_n) &\propto p(c, o_1, \ldots, o_n) \\ &= p(c) \prod_{i=1}^{n} p(o_i|c). \end{aligned}$$

4.1.3. Inferencia en modelos temporales

Para ilustrar la inferencia en un modelo temporal, vamos a enfocarnos en un modelo oculto de Markov con estados discretos. Utilizando la regla de Bayes, se tiene que:

$$p(s_t|o_1, \ldots, o_t) \propto p(o_t|s_t, o_1, \ldots, o_{t-1}) p(s_t|o_1, \ldots, o_{t-1}).$$

La suposición de independencia condicional de las observaciones O_t y $O_1, \ldots, O_{t-1}$ dado el estado S_t, implica que $p(o_t|s_t, o_1, \ldots, o_{t-1}) = p(o_t|s_t)$. Reescribiendo la ecuación anterior:

$$p(s_t|o_1, \ldots, o_t) \propto p(o_t|s_t) \sum_{s_{t-1}} p(s_t, s_{t-1}|o_1, \ldots, o_{t-1}).$$

Por la definición de probabilidad condicional aplicada a $p(s_t, s_{t-1}|o_1, \ldots, o_{t-1})$, se obtiene:

$$p(s_t|o_1, \ldots, o_t) \propto p(o_t|s_t) \sum_{s_{t-1}} p(s_t|s_{t-1}, o_1, \ldots, o_{t-1}) p(s_{t-1}|o_1, \ldots, o_{t-1}).$$

La estructura del modelo implica que S_t es condicionalmente independiente de $O_1, \ldots, O_{t-1}$ dado S_{t-1}, por lo que la ecuación anterior puede simplificarse de la siguiente manera:

$$p(s_t|o_1, \ldots, o_t) \propto p(o_t|s_t) \sum_{s_{t-1}} p(s_t|s_{t-1}) p(s_{t-1}|o_1, \ldots, o_{t-1}).$$

En el modelo descrito por la red bayesiana suponemos que conocemos las distribuciones $p(o_t|s_t)$ y $p(s_t|s_{t-1})$. La forma de la distribución $p(s_{t-1}|o_1, \ldots, o_{t-1})$ sugiere que una manera de calcularla es por recursión utilizando los métodos bayesianos. A este proceso se le denota *estimación bayesiana recursiva*.

4.1.4. Inferencia exacta y aproximada

Empecemos con un ejemplo de cómo calcular inferencias exactas en una red bayesiana; para esto, volvamos a la red de la Figura 4.2. Si b_0, c_0, d_0 son posibles valores de las v.a.'s B, C y D respectivamente; para inferir de manera exacta el valor de $p(b_0|c_0, d_0)$ el procedimiento es el siguiente:

$$\begin{aligned} p(b_0|c_0, d_0) &\propto p(b_0, c_0, d_0) \\ &= \sum_s \sum_e p(b_0, s, e, c_0, d_0) \\ &= \sum_s \sum_e p(b_0)p(s)p(e|b_0, s)p(d_0|e)p(c_0|e), \end{aligned}$$

donde las sumas sobre s y e representan las sumas sobre todos los valores de S y E respectivamente.

Como se aprecia en el ejemplo anterior, el problema al que nos enfrentamos al hacer inferencias exactas en redes bayesianas es el de sumar sobre todas las variables "ocultas". Cuando la red es grande, el número de términos sobre los que hay que sumar puede crecer exponencialmente con el número de variables ocultas. De hecho, se puede demostrar que la inferencia en redes bayesianas es un problema *NP-hard*[1].

Saber esto es de gran ayuda, pues así se pueden ahorrar esfuerzos evitando buscar un algoritmo de inferencia exacta que sea eficiente y que funcione para todas las redes bayesianas. Por esta razón, las investigaciones hechas desde hace un par de décadas a la fecha, se centran en métodos de inferencia aproximados.

Existen varios métodos de inferencia aproximada:

- *Ordenamiento topológico*: Se acomodan en lista todos los nodos de la red bayesiana, de modo que si hay una arista $A \longrightarrow B$ entonces A va en la lista antes que B. Obsérvese que este ordenamiento topológico no es único. Una vez que se tiene la lista ordenada, se hace un muestreo de las probabilidades de distribución condicionales empezando por la primera variable en la lista hasta llegar a la última. Es decir, si después del ordenamiento topológico se obtiene la lista $X_1, \ldots, X_n$, entonces el muestreo se realiza de la siguiente manera:

  1) y_1 es un resultado aleatorio de simular $p(x_1)$

[1]Una breve descripción sobre los problemas de tipo *P, NP, NP-completos* y *NP-hard* se encuentra en el *Apéndice*.

i) y_i es un resultado aleatorio de simular $p(x_i|pa_{x_i})$, para $i = 2, \ldots, n$ (calculadas en orden.)

así obtenemos la muestra $y_1, \ldots, y_n$.

El problema con este método es que se puede perder mucho tiempo generando muestras que son inconsistentes con las observaciones que proporciona la red, en especial si las observaciones son únicas.

- *Ponderación basada en la verosimilitud.* Genera muestras ponderadas que son consistentes con las observaciones. Se empieza igual que en el algoritmo anterior, a partir de un ordenamiento topológico. La diferencia es que, en lugar de muestrear sus valores a partir de la distribución condicional, a las variables aleatorias les asignamos el valor de las observaciones presentes en la red y ajustamos el peso de la muestra de manera apropiada. El peso de la muestra es simplemente el producto de las probabilidades condicionales en los nodos observados.

  El modo de proceder para una red bayesiana B con observaciones $O_{1:n}$ es el siguiente:

  1. Se hace un acomodo topológico de los nodos en B.
  2. A la variable del peso w se le asigna el valor de 1.
  3. Para $i = 1, \ldots, n$ se hace lo siguiente:
     3.1. Si o_i no es observada, entonces y_i toma un resultado aleatorio de simular $p(x_i|pa_{x_i})$, como en el caso anterior.
     3.2. Si o_i es conocida, entonces $y_i = o_i$ y el peso w se actualiza de la siguiente forma para ponderar la probabilidad de obtener la observación o_i dado pa_{x_i}
     $$w \longleftarrow w\, p(o_i|pa_{x_i})$$
  4. Se obtiene $(y_{1:n}, w)$.

- *Muestreo de Gibbs.* Es un tipo de técnica de *Monte Carlo* con cadenas de Markov. A diferencia de los otros dos métodos, las muestras que se producen con este método no son independientes. La siguiente muestra depende probabilísticamente de la actual ya que la secuencia de muestras forman una Cadena de Markov.

4.2. Aprendizaje paramétrico bayesiano

Hasta el momento, habíamos supuesto que los parámetros y la estructura del modelo probabilístico eran conocidos. Ahora, abordaremos el problema de hacer inferencias sobre los parámetros del modelo a partir de datos disponibles, empleando el método bayesiano.

Supongamos que las observaciones de la red están modeladas por una variable aleatoria O con parámetros desconocidos θ. De acuerdo con la Sección 3.2, $p(o|\theta)$ modela el fenómeno a estudiar, en donde conocemos la forma de $p(\cdot|\theta)$ pero el parámetro θ es desconocido. Suponemos que las variables observadas O_i $(i = 1, \ldots, n)$ son condicionalmente independientes unas de otras y que se puede especificar la distribución inicial (*prior*) $p(\theta)$ y también la distribución condicional $p(o_i|\theta)$. Utilizando los métodos bayesianos podemos llegar a la distribución final $p(\theta|o_{1:n})$.

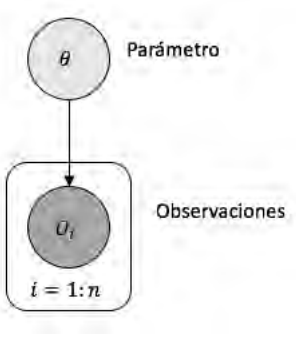


Figura 4.6: Notación abreviada del aprendizaje paramétrico en una red bayesiana.

Existen otras formas de inferir el valor del parámetro desconocido θ, tales como el *estimador máximo verosímil* o los métodos no paramétricos de aprendizaje. La descripción de ambos puede encontrarse en Kochenderfer (2015).

4.3. Inferencia de la estructura de la red bayesiana

En la sección previa teníamos como supuesto que la estructura de la red bayesiana era conocida. En esta sección se discute un método para inferir la estructura a partir de los datos disponibles. Primero explicaremos cómo calcular un puntaje de una red bayesiana, y luego se discutirá cómo hacer búsquedas en el espacio de las redes con

el fin de obtener el puntaje más alto. Aunque hay varios métodos, en este segundo punto nos centraremos únicamente en los métodos que involucran a los algoritmos genéticos. La primera parte de esta sección se basa, casi en su totalidad de Carvalho (2009), mientras que la última parte fue tomada Kochenderfer (2015).

4.3.1. Puntaje de la estructura bayesiana

Para los fines de esta sección vamos a formalizar algunos conceptos que, si bien ya fueron explicados con anterioridad, resulta conveniente tener presentes definiciones más precisas. En particular, la de *red bayesiana* que formaliza la que ya conocemos.

Definición 4.3.1 (Red bayesiana)**.** *Una red bayesiana de dimensión n es una tripleta $B = (\boldsymbol{X}, G, \Theta)$ en donde:*

- $\boldsymbol{X}$ es un vector aleatorio finito de dimensión n, en donde cada variable X_i toma valores sobre el conjunto finito $D_i \subset \mathbb{R}$. En lo sucesivo, se le denota por $\mathbf{D} = \prod_{i=1}^{n} D_i$ al conjunto en el que $\boldsymbol{X}$ toma valores conjuntos.
- $G = (N, E)$ es una gráfica dirigida acíclica con vértices $N = \{X_1, \ldots, X_n\}$ y aristas E que indican una dependencia directa entre las variables.
- Se denota por Θ al conjunto de todos los parámetros $\{\theta_{ijk}\}_{i,j,k}$ en donde $i \in \{1, \ldots, n\},\ j \in \mathbf{D}_{Pa_{X_i}},\ k \in D_i$,

  $$\theta_{ijk} = \Pr(X_i = x_{ik} | Pa_{X_i} = w_{ij}),$$

  $\mathbf{D}_{Pa_{X_i}}$ es la denotación del dominio conjunto donde toman valores las variables en Pa_{X_i}, x_{ik} es el k-ésimo valor de X_i y w_{ij} es la j-ésima configuración de Pa_{X_i}

Otra notación importante que introducimos es la que se refiere al conjunto de todas las redes bayesianas con n variables, al que se denota por $\mathcal{B}_n$.

En la estructura de redes bayesianas resulta indispensable especificar los parámetros que cuantifican la red, y es aquí donde toman sentido las tres componentes de los índices en θ_{ijk}. Dado un conjunto de datos $\boldsymbol{y}$, el problema de inferir una red bayesiana consiste en encontrar la red bayesiana que mejor se ajuste a las datos $\boldsymbol{y}$. Para cuantificar este ajuste se define una *función de puntaje* $\phi : \mathcal{B}_n \times \mathbf{D}^N \to \mathbb{R}$, donde $N = |\boldsymbol{y}|$. La función de puntaje debe ser asintóticamente correcta, esto es, la distribución de inferencia con el puntaje máximo debe tener alta probabilidad de converger a la probabilidad fundamental en la medida en que los datos incrementan. En este contexto, el problema de inferir una red bayesiana puede ser entendido como un problema de optimización.

Definición 4.3.2 (Inferir una red bayesiana)**.** *Dados los datos* $\boldsymbol{y} = \{y_1, \ldots, y_N\}$ *y una función de puntaje* ϕ*, el problema de inferir una red bayesiana consiste en encontrar la red bayesiana* $B \in \mathcal{B}_n$ *tal que maximice el valor de* $\phi(B, \boldsymbol{y})$*.*

Resolver el problema anterior no es una tarea fácil. Se ha demostrado que la inferencia de una red bayesiana general es un problema NP-*hard*; más aún, incluso encontrar una solución aproximada es un problema NP-*hard*.

Continuamos con más notación que es indispensable para el desarrollo de esta sección. El número de estados de una v.a. finita X_i es $r_i = |D_i|$. El número de posibles configuraciones del conjunto de padres Pa_{X_i} de X_i es $q_i = |\mathbf{D}_{Pa_{X_i}}|$; de modo que

$$q_i = \prod_{X_j \in Pa_{X_i}} r_j.$$

Si X_i no tiene padres, se toma $q_i = 1$. Una configuración de Pa_{X_i} es representada por w_{ij} $(1 \leq j \leq q_i)$. N_{ijk} es el número de veces que en el conjunto de datos $\boldsymbol{y}$, la variable X_i toma el valor x_{ik} y las variables Pa_{X_i} toman la configuración w_{ij}. N_{ij} es el número de veces en que en los datos $\boldsymbol{y}$, las variables Pa_{X_i} toman la configuación w_{ij}, esto es

$$N_{ij} = \sum_{k=1}^{r_i} N_{ijk}.$$

Finalmente, el número total de observaciones en el conjunto de datos $\boldsymbol{y}$ es N.

4.3.2. Función de puntaje bayesiana

En la literatura existen varias funciones propuestas para inferir redes bayesianas. De manera usual se dividen en dos categorías principales: las bayesianas y las referentes a la teoría de la información. En esta tesis nos centraremos en el estudio de una función de puntaje bayesiana de Dirichlet. Al lector interesado en conocer otras funciones, así como comparaciones entre ellas se le recomienda revisar Carvalho (2009).

En las funciones de puntaje bayesiano se emplea el método bayesiano para que a partir de la distribución inicial $p(B)$ y de la distribución condicional $p(\boldsymbol{y}|B)$ se llegue a la distribución final $p(B|\boldsymbol{y})$. Luego, se considera que la red que mejor se adapta a los datos $\boldsymbol{y}$ es la que maximiza a la distribución final. Para facilitar los cálculos, se hace uso de la función logaritmo natural, es por eso que el puntaje bayesiano se

expresa como $\ln(p(B)p(\boldsymbol{y}|B))$.

Bajo lo explicado en las subsecciones precedentes, se tiene que la probabilidad condicional está dada por:

$$p(\boldsymbol{y}|\boldsymbol{\theta}, B) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}}. \tag{4.2}$$

La distribución inicial de $\boldsymbol{\theta}$ sobre la red bayesiana se puede factorizar. Si denotamos por $\boldsymbol{\theta}_{ij} = (\theta_{ij1}, \ldots, \theta_{ijr_i})$, entonces

$$p(\boldsymbol{\theta}|B) = \prod_{i=1}^{n}\prod_{j=1}^{q_i} p(\boldsymbol{\theta}_{ij}).$$

Se puede probar que la distribución inicial $p(\boldsymbol{\theta}_{ij})$, bajo algunos supuestos débiles, sigue la distribución *Dirichlet*[2]. La distribución sobre X_i dada la j-ésima configuración parental está dada por $\mathrm{Dir}(\alpha_{ij1}, \ldots \alpha_{ijr_i})$.

Calculando

$$p(B|\boldsymbol{y}) \propto p(B)p(\boldsymbol{y}|B) \tag{4.3}$$

$$= p(B)\int p(\boldsymbol{y}|\boldsymbol{\theta}, B)p(\boldsymbol{\theta}|B)d\boldsymbol{\theta}. \tag{4.4}$$

Integrando (4.4) obtenemos:

$$p(B|\boldsymbol{y}) = p(B)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}, \tag{4.5}$$

donde

$$\alpha_{ij} = \sum_{k=1}^{r_i}\alpha_{ijk},$$

[2]La distribución *Dirichlet* es una generalización de la distribución beta y puede ser empleada para estimar parámetros de una distribución discreta. Supóngase que X es una v.a. discreta que toma valores enteros en $1, \ldots, n$. Definimos los parámetros de la distribución como $\boldsymbol{\theta} = \theta_1, \ldots, \theta_n$, en donde $\Pr(X = i) = \theta_i$. Entonces la distribución *Dirichlet* con parámetros $\boldsymbol{\alpha} = \alpha_1, \ldots, \alpha_n$ y $\alpha_0 = \sum_{i=1}^{n}\alpha_i$ está dada por

$$\mathrm{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{n}\Gamma(\alpha_i)}\prod_{i=1}^{n}\theta_i^{\alpha_i-1}.$$

$\alpha_{ijk} > 0$ y donde $\{\alpha_{ijk}\}_{k=1\ldots,r_i}$ son los hiperparámetros de la distribución Dirichlet. Más aún, cuando $\boldsymbol{\theta}_{ij}$ se distribuye Dirichlet, la esperanza de θ_{ijk} es

$$\mathbb{E}(\theta_{ijk}) = \frac{\alpha_{ijk}}{\alpha_{ij}}.$$

Ahora procedemos a encontrar la B que maximice a $p(B|\boldsymbol{y})$. Calculando el logaritmo natural de (4.5) obtenemos a lo que se le conoce como *puntaje bayesiano* (*Bayesian score*):

$$\ln p(B|\boldsymbol{y}) = \ln p(B) + \sum_{i=1}^{n}\sum_{j=1}^{q_i} \ln\left(\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\right) \sum_{k=1}^{r_i} \ln\left(\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}\right). \quad (4.6)$$

Existe una variedad de diferentes distribuciones iniciales para las gráficas (*graph priors*) que han sido exploradas en la literatura; sin embargo, la distribución inicial uniforme es utilizada en la práctica con mucha frecuencia. Una característica útil del puntaje bayesiano, incluso con distribución inicial uniforme, es que produce un equilibrio adecuado entre la complejidad del modelo y el número de datos disponibles.

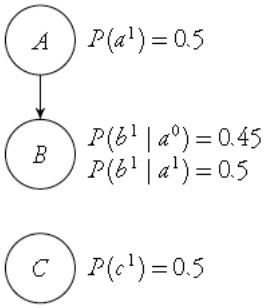


Figura 4.7: Ejemplo de una red bayesiana.

Veamos un ejemplo de cómo el puntaje bayesiano equilibra la complejidad de un modelo. Consideremos una red bayesiana simple de la Figura 4.7. El valor de A influye débilmente en el valor de B, y C es independiente de las otras variables. A este modelo llamémosle "el verdadero". Se produce una muestra del modelo "verdadero" generando los datos $\boldsymbol{y}$; luego se va intentar inferir la estructura del modelo. Hay 25 posibles estructuras de red asociadas a 3 variables; para los fines de este ejemplo sólo nos centramos en los siguientes:

- El modelo verdadero con $1 + 2 + 1 = 4$ parámetros independientes.

- El modelo totalmente conexo $A \longrightarrow B, A \longrightarrow C, B \longrightarrow C$ con $1 + 2 + 4 = 7$ parámetros independientes, y

- El modelo totalmente disconexo con $1 + 1 + 1 = 3$ parámetros independientes.

En la Figura 4.8 se muestra el puntaje bayesiano de los modelos totalmente conexos y disconexos, comparados con el modelo verdadero. En la gráfica, se resta el puntaje del modelo verdadero para que en la figura quede representado por la constante cero. De este modo, los valores arriba de 0 indican que el modelo proporciona una mejor representación que el modelo verdadero dados los datos disponibles. La gráfica muestra que el modelo totalmente disconexo es mejor que el modelo verdadero antes de las primeras 5000 muestras. El modelo totalmente conexo no llega a ser mejor que el modelo verdadero, pero empieza a ser mejor que el modelo disconexo después de alrededor de la muestra $10,000$ debido a que ya hay suficientes datos para estimar adecuadamente los 7 parámetros independientes.

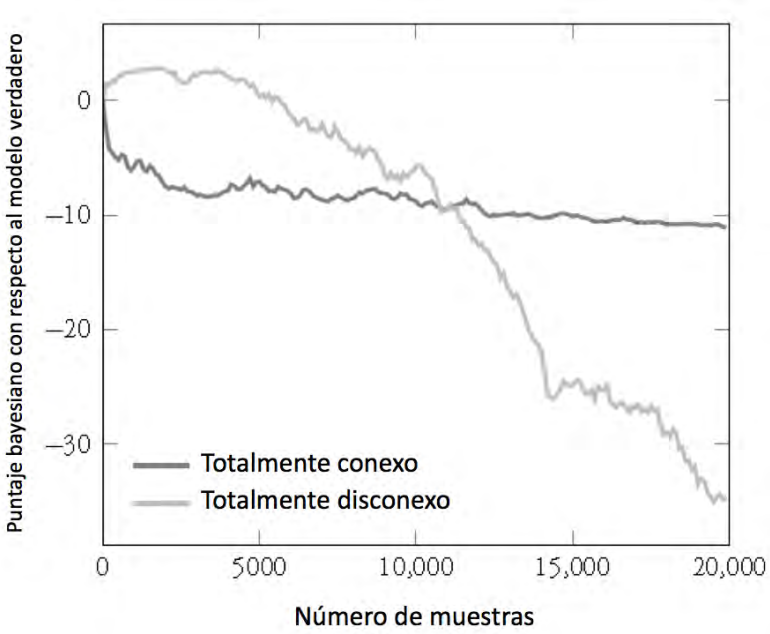


Figura 4.8: El puntaje bayesiano relativo al modelo verdadero. Imagen tomada de Kochenderfer (2015).

4.3.3. Búsqueda dirigida de la gráfica

El espacio de las posibles redes bayesianas crece de manera exponencial. Para tres variables existen 25 posibles estructuras de red, con 10 nodos hay 4.2×10^{18} posibles gráficas dirigidas acíclicas. Para 20 nodos hay 2.4×10^{72}. A excepción de las redes bayesianas con pocos nodos, no podemos enumerar el espacio de posibles estructuras para encontrar la red con el puntaje más alto. Por lo tanto, dependemos de alguna estrategia de búsqueda. Por fortuna, los problemas de búsqueda han sido ampliamente estudiados y un ejemplo de un algoritmo de búsqueda son los algoritmos genéticos que ya abordamos en el Capítulo 2. Algunos otros algoritmos de búsqueda útiles en este contexto son: K2, gradiente ascendente, reinicio aleatorio, entre otros.

En el caso de los algoritmos genéticos, el procedimiento empieza con una población inicial aleatoria de puntos en el espacio de búsqueda, representados como cadenas. Cuando se busca en el espacio de gráficas dirigidas, un *bit* en la cadena indica la presencia o ausencia de una flecha entre dos nodos. Los individuos en la población se reproducen a una tasa proporcional a su puntaje. Los individuos seleccionados para reproducirse recombinan sus genes de manera aleatoria a través de la cruza genética, se recomienda la cruza de punto único. La mutación se introduce de manera aleatoria cambiando *bits* en la cadena de manera aleatoria. El proceso de evolución continúa hasta que se encuentra un punto en el espacio de búsqueda que sea satisfactorio.

Supongamos que tenemos una red con los nodos A, B y C. Un individuo del espacio de soluciones puede ser del siguiente modo:

$$\begin{array}{ll} A & B \\ B & \\ C & \end{array}$$

donde la primera columna indica el nodo de donde sale la flecha y la segunda columna indica el nodo a donde llega la flecha; un espacio en blanco en esta posición indica que esa flecha no existe. La estuctura del individuo anterior corresponde a la de la Figura 4.7.

Algunas estrategias de búsqueda podrían funcionar mejor que otras en determinados conjuntos de datos, pero en general encontrar un óptimo global pertenece a los problemas de tipo *NP-hard*. Sin embargo, para muchas aplicaciones no es necesa-

rio encontrar el óptimo global, una estructura que alcance un óptimo local suele ser aceptable.

4.3.4. Clases de equivalencia de Markov

Ya se ha discutido que la estructura de la red bayesiana codifica un conjunto de suposiciones acerca de la independencia condicional de las variables. Una observación importante que hay que notar es que, cuando se está tratando aprender la estructura de la red bayesiana, dos gráficas diferentes pueden codificar las mismas suposiciones de independencia. Para ejemplificarlo, consideremos la red de dos variables $A \longrightarrow B$. Esta red contiene las mismas suposiciones de independencia que $A \longleftarrow B$. Basándonos únicamente en los datos observados, es difícil justificar la dirección de la arista entre A y B. Lo anterior nos lleva a la siguiente definición:

Definición 4.3.3. *Dos gráficas son Markov equivalentes si codifican el mismo conjunto de suposiciones de independencia condicional.*

Se puede probar que dos gráficas son Markov equivalentes si y sólo si ambas tienen:

(1) El mismo conjunto de aristas sin considerar su dirección, y

(2) Las mismas estructuras en forma de v (*v-estructuras*).

Se dice que un conjunto es una *clase de equivalencia de Markov* si contiene todas las gráficas dirigidas y acíclicas equivalentes entre sí.

En general, dos estructuras que pertenecen a la misma clase de equivalencia de Markov podrían tener diferentes puntajes. Un función que asigna el mismo puntaje a todas las estructuras dentro de la misma clase, se le llama *equivalentes en el puntaje* (*score equivalent*).

4.3.5. Búsqueda de gráficas parcialmente dirigidas

Una clase de Markov equivalente puede ser representada por una *gráfica parcialmente dirigida*, que a veces también recibe el nombre de *gráfica esencial* o *patrón de gráfico acíclico dirigido*. Una gráfica parcialmente dirigida puede contener tanto aristas dirigidas como aristas no dirigidas.

Una gráfica G dirigida y acíclica, es un miembro de una clase de equivalenca de Markov y a su vez es codificada por la gráfica parcialmente dirigida G^l si y sólo si:

(1) G tiene las mismas aristas que G^l sin considerar su dirección, y

(2) G tiene las mismas v-estructuras que G^l.

En la Figura 4.9 se muestra un ejemplo de una clase de equivalencia de Markov. En la Figura 4.9a se encuentra la gráfica parcialmente dirigida G^l que es el representante de la clase de equivalencia. Las gráficas en 4.9b y 4.9c son de miembros de esa clase de equivalencia, mientras que la Figura 4.9d no es miembro de la clase pues no tiene las mismas estructuras en v que 4.9a.

La importancia de esta representación radica en que abre la posibilidad de realizar la búsqueda en el espacio de clases de equivalencia de Markov representado por gráficas parcialmente dirigidas en lugar de buscar en todo el espacio de gráficas dirigidas y acíclicas. El primer espacio es más pequeño que el segundo, por lo que la búsqueda puede resultar más eficiente.

El puntaje bayesiano está definido para gráficas acíclicas dirigidas por lo que, para obtener el puntaje de una gráfica parcialmente dirigida, necesitamos generar un miembro de la clase de equivalencia de Markov y computar su puntaje.

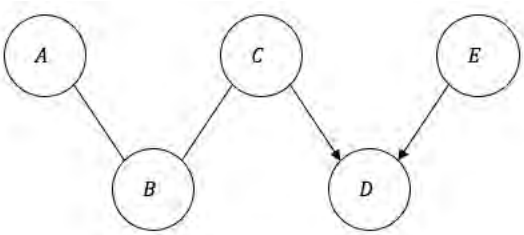
4.4. Problemas de decisión

Este capítulo retoma cómo hacer decisiones racionales basándonos en un modelo probabilístico y en una función de utilidad, tal como se estudió en la teoría de decisión de la Sección 3.3.

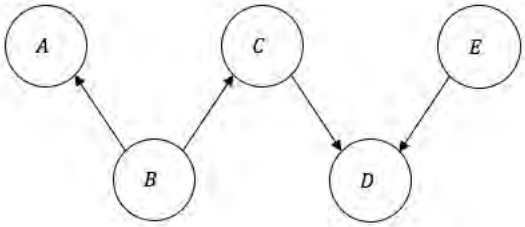
En esta sección nos enfocamos en las decisiones de un solo paso, reservando la discusión de los problemas de decisiones secuenciales para la siguiente sección.

4.4.1. Construcción racional de las preferencias

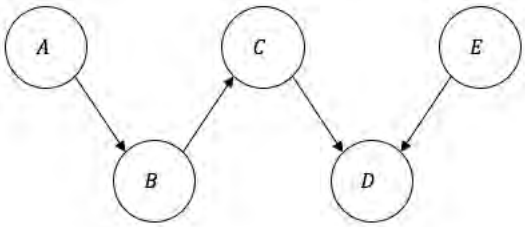
Para comparar el grado de preferencia entre dos diferentes resultados, utilizaremos el símbolo $\preceq$ que se emplea de forma análoga que en la teoría de la decisión bayesiana. Así pues, para dos resultados A y B, $A \preceq B$ denota nuestra preferencia de B por encima de la de A.



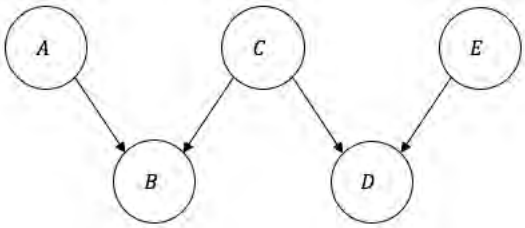
(a) Clase de equivalencia de Markov.



(b) Miembro.



(c) Miembro.



(d) No es miembro.

Figura 4.9: Ejemplo de clases de Markov equivalentes.

Antes de continuar, es pertinente introducir la siguiente notación: sea $S_{1:n}$ un conjunto de resultados y $p_{1:n}$ sus probabilidades asociadas, entonces lo anterior se denota de la siguiente forma

$$[S_1 : p_1; \ldots; S_n : p_n] \tag{4.7}$$

Para tomar decisiones de manera racional, se parte de los axiomas de coherencia de los que ya se ha hablado en la Sección 3.3.2. Como consecuencia de esos axiomas, se deriva lo siguiente:

i. Existe una medida de probabilidad con la que se compara la plausibilidad de los diferentes eventos.

ii. Existe una *función de utilidad* U con valores reales que, para dos resultados A y B dados, cumple lo siguiente:

   - $\mathrm{U}(A) > \mathrm{U}(B)$ si y sólo si $A \succ B$.
   - $\mathrm{U}(A) = \mathrm{U}(B)$ si y sólo si $A \sim B$.

iii. Debe tomarse la decisión que maximiza la utilidad esperada.

Por ejemplo, para $[S_1 : p_1; \ldots; S_n : p_n]$ introducido en la ecuación (4.7), su función de utilidad está dada por

$$\mathrm{U}([S_1 : p_1; \ldots; S_n : p_n]) = \sum_{i=1}^{n} p_i \mathrm{U}(S_i)$$

Siguiendo con el ejemplo, supongamos que estamos construyendo un sistema anticolisiones. El *resultado* de una colisión de una aeronave queda definido a través de si se emite o no una alerta en el sistema (A) y de si ocurre o no la colisión (B). Así, A y B son dos v.a. binarias, por lo que podemos escribir a la función de utilidad a través de los cuatro parámetros siguientes: $\mathrm{U}(a^0, b^0), \mathrm{U}(a^0, b^1), \mathrm{U}(a^1, b^0)$ y $\mathrm{U}(a^1, b^1)$, así que

$$\begin{aligned}\mathrm{U}([a^0, b^0 : p_1, a^0, b^1 : p_2, a^1, b^0 : p_3, a^1, b^1 : p_4]) = \\ p_1 \mathrm{U}(a^0, b^0) + p_2 \mathrm{U}(a^0, b^1) + p_3 \mathrm{U}(a^1, b^0) + p_4 \mathrm{U}(a^1, b^1).\end{aligned}$$

Ahora analicemos lo que nos dice el *principio de la máxima utilidad esperada*. Supongamos que tenemos un modelo de probabilidad $p(s'|o, a)$, que representa la probabilidad de que el estado de la naturaleza sea s' dado que se observa o y se

toma la acción a. La función de utilidad $U(s')$ codifica nuestras preferencias sobre el espacio de resultados. Entonces, la utilidad esperada de tomar la acción a dada la observación o es

$$\mathbb{E}\mathrm{U}(a|o) = \sum_{s'} p(s'|a,o)\mathrm{U}(s'). \tag{4.8}$$

Así pues, la forma racional de proceder, es elegir la acción que maximice la utilidad esperada

$$a^* = \arg\max_a \mathbb{E}\mathrm{U}(a|o). \tag{4.9}$$

4.4.2. Redes de decisión

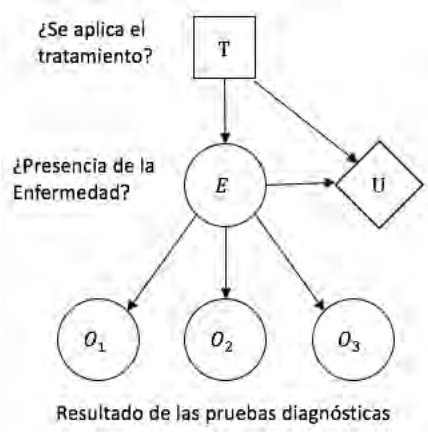
Podemos extender la noción de red bayesiana introducida en la sección anterior a red de decisión, incorporando acciones y utilidades. Las redes de decisión se componen por tres tipos de nodos:

- Un *nodo aleatorio* (*chance node*) corresponde a una variable aleatoria y es indicado por un círculo.
- Un *nodo de decisión* (*decision node*) corresponde a cada decisión que hay que hacer y es indicado por un cuadrado.
- Un *nodo de utilidad* (*utility node*) corresponde a una componente de utilidad aditiva y es indicada por un diamante.

Hay tres tipos de aristas dirigidas:

- Una *arista condicional* termina en un nodo aleatorio e indica que la incertidumbre en ese nodo aleatorio es condicionada por los valores de todos sus padres.
- Una *arista informacional* termina en un nodo de decisión e indica que la decisión asociada a ese nodo se hace con el conocimiento de los valores de sus padres. (Estas aristas a veces se dibujan con lineas punteadas y otras veces se omiten de los diagramas para simplificarlos).
- Una *arista funcional* termina en un nodo de utilidad e indica que el nodo de utilidad está determinado por los resultados de sus padres.

A las redes de decisión a veces se les llama *diagramas de influencia*. Al igual que las redes bayesianas, las redes de decisión no admiten ciclos. Representar un



T	E	$U(t,e)$
0	0	0
0	1	−10
1	0	−1
1	1	−1

Figura 4.10: Ejemplo de un tests diagnóstico en una red de decisión y su función de utilidad.

problema de decisión como una red de decisión, nos da la ventaja de aprovechar la estructura del problema al momento de computar la decisión óptima con respecto a su función de utilidad.

En esta sección nos enfocaremos en los problemas de decisión de *un paso* (*single-shot*), en los que las decisiones se toman de manera simultánea. En la siguiente sección, nos concentraremos en los problemas en los que las decisiones son tomadas de manera secuencial.

En la Figura 4.10 se muestra un ejemplo de una red de decisión. En esta red se tienen un conjunto de resultados obtenidos al realizar pruebas diagnósticas que indican la presencia de una enfermedad en particular. Nuestra decisión radica en si debemos o no aplicar un tratamiento, con base en los resultados que arrojan las pruebas diagnósticas. La función utilidad depende de si el tratamiento es aplicado y de si la enfermedad está presente en realidad.

4.4.3. Evaluación de una red de decisión

De acuerdo con (4.8) la utilidad esperada de a dada la observación o es

$$\mathbb{E}\mathrm{U}(a|o) = \sum_{s'} p(s'|a,o)\mathrm{U}(s'), \tag{4.10}$$

donde s' en la ecuación anterior representa una instancia de los nodos en la red de decisión. A manera de ejemplo, utilizaremos la ecuación anterior para calcular la utilidad esperada de tratar la enfermedad en la red de decisión de la Figura 4.10. Por ahora, supongamos que sólo tenemos el resultado del primer diagnóstico y que éste es positivo. Si queremos hacer explícito en el diagrama que sabemos el resultado del primer examen diagnóstico, entonces deberíamos dibujar una arista informacional del nodo O_1 al nodo T. Ahora bien, tenemos que:

$$\mathbb{E}\mathrm{U}(t^1|o_1^1) = \sum_{o_3}\sum_{o_2}\sum_{e} p(e, o_2, o_3|t^1, o_1^1)\mathrm{U}(t^1, e, o_1^1, o_2, o_3). \tag{4.11}$$

Podemos emplear la regla de la cadena y los supuestos de independencia condicional propios de la red bayesiana para calcular $p(e, o_2, o_3|t^1, o_1^1)$. Por otro lado, debido a que el nodo de utilidad del ejemplo únicamente depende si la enfermedad está o no presente, y de si decidimos o no tratarla, podemos simplificar $\mathrm{U}(t^1, e, o_1^1, o_2, o_3)$ por $\mathrm{U}(t^1, e)$; de este modo

$$\mathbb{E}\mathrm{U}(t^1|o_1^1) = \sum_{e} p(e|t^1, o_1^1)\mathrm{U}(t^1, e). \tag{4.12}$$

Para calcular la distribución final $p(e|t^1, o_1^1)$ se puede emplear el método exacto o los métodos aproximados analizados en la Sección 4.1.4. Y para decidir aplicar el tratamiento o no, tenemos que calcular $\mathbb{E}\mathrm{U}(t^1|o_1^1)$ y $\mathbb{E}\mathrm{U}(t^0|o_1^1)$ y elegir la opción que maximice la utilidad esperada.

En resumen, para evaluar redes de decisión de un paso, empezamos por cubrir todas las instancias de los nodos de acción y de las observaciones de los nodos aleatorios. Entonces aplicamos algún algoritmo de inferencia para calcular la distribución final sobre los padres de los nodos de utilidad. En lugar de sumar sobre todas las instancias expresadas en la ecuación (4.10), únicamente tenemos que sumar sobre las instancias de los padres de los nodos de utilidad. La decisión óptima es la que maximiza la utilidad esperada.

4.4.4. Creación de una red de decisión

En esta sección, enumeraremos de manera rápida el procedimiento para crear una red de decisión.

1. El primer paso es identificar el espacio de posibles acciones.

2. Identificamos las variables observadas y las no observadas relevantes para el problema.

3. Identificamos las relaciones entre los distintos nodos de decisión y aleatorios. A menudo, se intenta elegir la dirección de las flechas para indicar una relación causal de un nodo a otro.

4. Elegimos los modelos para representar las probabilidades de distribución condicionales.

5. Introducimos los nodos de utilidad

6. Insertamos aristas funcionales desde los nodos aleatorios relevantes y desde los nodos de decisión hacia los nodos de utilidad.

4.5. Problemas secuenciales

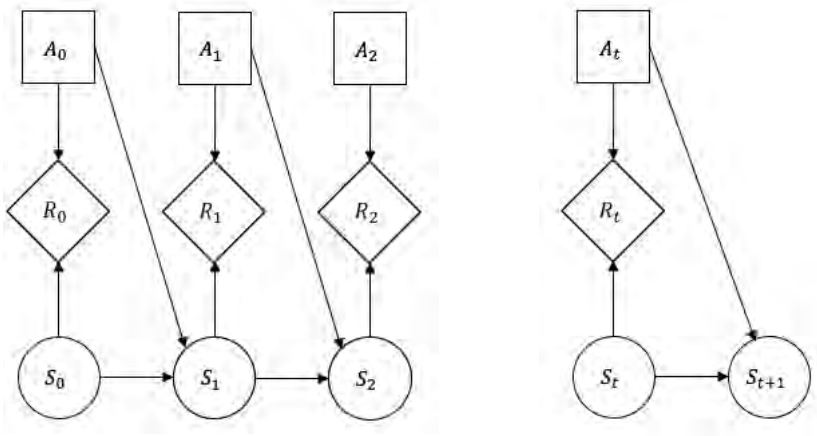
En la sección anterior se discutió el problema en el que se hacía una decisión única, pero para muchos problemas importantes se requiere tomar decisiones de manera secuencial. El mismo principio de maximizar la utilidad esperada sigue funcionando, pero para tomar una decisión óptima es necesario hacer un razonamiento acerca del futuro de la secuencia de acciones y observaciones. En esta sección discutiremos los problemas de decisión secuenciales dentro de ambientes estocásticos.

4.5.1. Formulación

En esta subsección, asumiremos que el modelo es conocido y que el ambiente es totalmente observable. Estos dos supuestos se irán relajando en las siguientes dos secciones.

4.5.1.1. Procesos de Decisión de Markov

En los *Procesos de Decisión de Markov* (PDM), un agente elige la acción a_t al tiempo t basado en lo que observa en el estado s_t. Después de esto, el agente recibe una recompensa r_t. El estado evoluciona probabilísticamente basado en el estado al tiempo presente y en la acción tomada por el agente. La suposición de que el próximo estado sólo depende del estado y la acción al tiempo presente y no de ningún estado o acción anterior se le llama *suposición de Markov*. Una *estrategia* es la toma sucesiva



(a) Representación general. (b) Representación estacionaria.

Figura 4.11: Diagrama de un proceso de decisión de Markov.

de decisiones.

Un PDM se puede representar utilizando una red de decisión como se muestra en la Figura 4.11a. En ésta, hay aristas informativas de $A_{0:t-1}$ y $S_{0:t}$ hacia A_t, las cuales no están dibujadas. La función de utilidad está distribuida en las recompensas $R_{0:t}$.

Nos centraremos en los PDM *estacionarios*, en los que $p(S_{t+1}|S_t, A_t)$ y $p(R_t|A_t, S_t)$ no varían con el tiempo. Este proceso puede representarse de forma compacta a través del *diagrama de decisión dinámica* que se muestra en la Figura 4.11b. La *Función de transición de estados* $\mathrm{T}(s'|s, a)$ representa la probabilidad de transición del estado s al estado s' después de realizar la acción a. La *función de recompensa* $\mathrm{R}(s, a)$ representa la recompensa que se espera recibir al realizar la acción a desde el estado s. Estamos suponiendo que la función de recompensa es una función determinista de s y a; pero en general este no es el caso.

4.5.1.2. Utilidad y recompensa

Las recompensas en un PDM son tratadas como sumandos de una *función de utilidad aditiva*; esto es, si tenemos n variables $X_{1:n}$, la utilidad conjunta queda

determinada por la suma de las utilidades de cada variable, es decir

$$\mathrm{U}(x_1, \ldots, x_n) = \sum_{i=1}^{n} \mathrm{U}(x_i).$$

En este contexto, en un problema de *horizonte finito* con n decisiones, la utilidad asociada con la sucesión de recompensas $r_{0:n-1}$ es simplemente

$$\sum_{t=0}^{n-1} r_t. \tag{4.13}$$

En los problemas de *horizonte infinito* el número de decisiones es no acotado, por lo que la suma de recompensas puede llegar a ser infinito. Y con esto, la utilidad esperada de una buena estrategia podría ser igual a la utilidad esperada de una mala estrategia. Una forma de definir la utilidad en términos de recompensas individuales en problemas de horizonte infinito es agregando un *factor de descuento* $\gamma \in (0, 1)$. De este modo, la utilidad está dada por

$$\sum_{t=0}^{\infty} \gamma^t r_t. \tag{4.14}$$

El factor de descuento hace que las recompensas sean más valiosas en el tiempo presente que en el futuro, un concepto que es relevante, por ejemplo, en la economía.

Otra forma de definir la utilidad en problemas de horizonte infinito es usar la *recompensa ponderada* dada por

$$\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} r_t. \tag{4.15}$$

4.5.2. Programación dinámica

La estrategia óptima se puede encontrar usando una técnica computacional llamada *programación dinámica.*

4.5.2.1. Políticas y utilidades

Una política en un PDM determina qué acción elegir dada la historia de los estados y las acciones. La acción a elegir al tiempo t, dada la historia $h_t = (s_{0:t}, a_{0:t-1})$ se

escribe como $\pi_t(h_t)$. Como estamos en un PDM, podemos centrar nuestra atención en las políticas que dependen sólo del estado actual.

En los PDM de horizonte finito con transiciones y recompensas estacionarias, podemos restringirnos aún más y centrarnos sólo en las políticas estacionarias. En este caso, escribiremos $\pi(s)$ (sin el subíndice temporal) para referirnos a la acción asociada a la política estacionaria π, en el estado s.

En los problemas de horizonte finito, puede ser beneficioso seleccionar diferentes acciones dependiendo del número de pasos restantes. Por ejemplo, en un juego de baloncesto una buena estrategia es lanzar un tiro de larga distancia cuanto quedan pocos segundos en el cronómetro, pero podría ser una mala decisión bajo otras circunstancias.

La utilidad esperada de ejecutar π desde el estado s se denota $\mathrm{U}^{\pi}(s)$. En el contexto de los PDM, a veces nos referimos a U^{π} como *la función de valor*. Una *política óptima* π^* es la política que maximiza la utilidad esperada:

$$\pi^*(s) = \arg\max_{\pi} \mathrm{U}^{\pi}(s) \tag{4.16}$$

para todos los estados s. Dependiendo del modelo, puenden existir múltiples políticas que sean óptimas.

4.5.2.2. Evaluación de políticas

La *evaluación de una política* es el cálculo de la utilidad esperada obtenida de ejecutar una política. Utilizamos programación dinámica para evaluar la utilidad de una política π por t pasos, así $\mathrm{U}_0^{\pi} = 0$ es la utilidad de no ejecutar ninguna política; y $\mathrm{U}_1^{\pi}(s) = \mathrm{R}(s, \pi(s))$ es la utilidad de realizar la política durante un paso.

Supongamos que conocemos la utilidad de realizar la política π durante $t-1$ pasos, entonces la utilidad de realizar π durante t pasos está dada por

$$\mathrm{U}_t^{\pi}(s) = \mathrm{R}(s, \pi(s)) + \gamma \sum_{s'} \mathrm{T}(s'|s, \pi(s)) \mathrm{U}_{t-1}^{\pi}(s'). \tag{4.17}$$

En el caso de problemas con horizonte infinito, se puede calcular la ecuación (4.17) para el caso en el que se cuenta con un número finito de n estados. Lo que

procede es resolver el sistema de n ecuaciones lineales representado por el sistema en forma matricial

$$U^{\pi} = \mathrm{R}^{\pi} + \gamma \mathrm{T}^{\pi} \mathrm{U}^{\pi}, \tag{4.18}$$

en donde U^{π} y R^{π} son las funciones de utilidad y recompensa representadas en la forma de un vector n-dimensional. Si las matrices son las adecuadas, llegamos a que $\mathrm{U}^{\pi} = (I - \gamma \mathrm{T}^{\pi})^{-1} \mathrm{R}^{\pi}$, sin embargo encontrar la solución para U^{π} es un camino que toma un tiempo del orden de $O(n^3)$.

4.5.2.3. Iteración de la política

Este método se emplea para calcular una política óptima π^*. La iteración de la política da inicio con cualquier política π_0 y se iteran los siguientes dos pasos:

- *Evaluar la política.* Dada la política actual, evaluar U^{π_k}.
- *Mejorar la política.* Utilizando U^{π_k}, calcular una nueva política utilizando la siguiente ecuación:

$$\pi_{k+1}(s) = \arg\max_{a} \left(\mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a) \mathrm{U}^{\pi_k}(s') \right) \tag{4.19}$$

  para todos los estados s.

4.5.2.4. Iteración del valor

Para esta política, primero calculamos la función de valor óptimo U_n asociada con un horizonte n y sin descuento. Si $n = 0$ entonces $\mathrm{U}_0(s) = 0$ para toda s. Se calcula U_n de forma recursiva con la siguiente fórmula.

$$\mathrm{U}_n(s) = \max_{a} \left(\mathrm{R}(s,a) + \sum_{s'} \mathrm{T}(s'|s,a) \mathrm{U}_{n-1}(s') \right). \tag{4.20}$$

Para los problemas de horizonte infinito con descuento γ, se puede demostrar que el valor de la política óptima satisface las ecuaciones de Bellman:

$$\mathrm{U}^*(s) = \max_{a} \left(\mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a) \mathrm{U}^*(s') \right). \tag{4.21}$$

La función de valor óptimo U^* aparece en ambos lados de la ecuación. La iteración de valor aproxima a U^* a través de actualizar iterativamente la estimación de U^*

usando 4.21. Una vez que conozcamos U^*, podemos extraer una política óptima utilizando

$$\pi(s) \longleftarrow \arg\max_a \left(\mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a)\mathrm{U}^*(s') \right). \tag{4.22}$$

4.5.3. Programación dinámica aproximada

El campo de la *programación dinámica aproximada* se encarga de encontrar aproximaciones a las políticas óptimas para problemas con espacios muy grandes o continuos. Esta área de estudio comparte ideas con el campo del *aprendizaje reforzado*. El aprendizaje reforzado consiste en acumular rápidamente tanta recompensa como sea posible sin tener un conocimiento del modelo. Muchos de los algoritmos de aprendizaje reforzado (veremos uno en la próxima sección), pueden ser empleados directamente en la programación dinámica aproximada. En este apartado sólo daremos una breve idea de lo que consisten la mayoría de las estrategias de aproximación local y global, para encontrar de manera eficiente funciones de valor y políticas cuando el modelo es conocido.

- *Aproximación local.* Se basa en la suposición de que los estados cercanos unos a otros tienen valores similares. Si conocemos el valor asociado a un número finito de estados $s_{1:n}$, entonces podemos aproximar el valor de un número arbitrario de estados utilizando la ecuación:

  $$\mathrm{U}(s) = \sum_i^n \lambda_i \beta_i(s) = \boldsymbol{\lambda}^T \beta(s), \tag{4.23}$$

  donde $\beta_{1:n}$ son funciones de ponderación, tales que $\sum_{i=1}^n \beta_i(s) = 1$. El valor λ_i es el valor de el estado s_i. En general, $\beta_i(s)$ podría ser asignado en relación a qué tan lejos o cerca se encuentra s de s_i.

  Una vez que la aproximación a la función de valor es conocida, podemos aproximar la política óptima con la ecuación (4.22):

  $$\pi(s) \longleftarrow \arg\max_a \left(\mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a)\boldsymbol{\lambda}^T \beta(s) \right). \tag{4.24}$$

- *Aproximación global.* Utiliza un conjunto fijo de parámetros $\lambda_{1:m}$ para aproximar la función de valor sobre todo el espacio $\mathcal{S}$. Una de las aproximaciones

globales más usadas está basada en la *regresión lineal*. Definimos la *función base* $\beta_{1:m}$, donde $\beta_i : \mathcal{S} \longrightarrow \mathbb{R}$. La aproximación de $\mathrm{U}(s)$ es una combinación lineal de los parámetros y de los resultados de la función base:

$$\mathrm{U}(s) = \sum_{i}^{m} \lambda_i \beta_i(s) = \boldsymbol{\lambda}^T \beta(s). \tag{4.25}$$

4.5.4. Búsqueda directa de la política

En la sección anterior presentamos métodos que involucran calcular de manera aproximada la función de valor. Una alternativa es buscar directamente dentro del espacio de políticas. A pesar de que el espacio de estados pueda tener una dimensión grande y hacer una aproximación de la función de valor pueda ser difícil, el espacio de posibles políticas puede tener una dimensión relativamente baja y puede ser relativamente fácil buscar directamente una política.

Aunque existen varios métodos de búsqueda directa de políticas, nos centraremos en los métodos evolutivos. Aquí juegan un papel importante los algoritmos genéticos (Capítulo 2).

Un enfoque relacionado es la *programación genética* que involucra estructuras de árbol evolutivas las cuales representan políticas. Los árboles están formados por símbolos elegidos de un conjunto predefinido de terminales y no-terminales, permitiendo más flexibilidad en la representación de las políticas que si se utilizara una cadena de tamaño fijo. La cruza consiste en intercambiar sub-árboles y en la mutación se modifican sub-árboles de manera aleatoria.

Los algoritmos genéticos pueden ser combinados con otros métodos, por ejemplo, con la búsqueda local. Así, podríamos emplear un algoritmo genético para obtener una política satisfactoria; y luego usar *búsqueda local* para mejorar aún más la política. Esta aproximación se llama *búsqueda genética local*.

4.6. Incertidumbre en el modelo

En la sección anterior discutimos los problemas de decisión secuencial, en donde el modelo de transiciones y recompensas son conocidos. En muchos problemas, las transiciones y las recompensas no se conocen del todo bien, y es el agente quien tiene que aprenderlas a través de la experiencia. Por medio de la observación de los

resultados de sus acciones (traducidos en la forma que toman las transiciones entre estados y las recompensas), es que el agente escoge la acción que maximiza la acumulación de recompensas al largo plazo. Resolver el tipo de problemas en los que el modelo es incierto, es el campo de estudio del *aprendizaje reforzado* (*reinforcement learning*) y es también, el foco de esta sección.

Hay varios desafíos que afrontar cuando se aborda el estudio de modelos inciertos. Primero, el agente debe balancear cuidadosamente la exploración del ambiente con el aprovechamiento del conocimiento adquirido a través de la experiencia. En segundo lugar, las recompensas pueden recibirse mucho después de que las decisiones importantes se han tomado, por lo que el crédito de recompensas posteriores debe ser asignado a decisiones anteriores. En tercer lugar, el agente debe hacer generalizaciones a partir de una experiencia limitada. Este capítulo da una revisión a la teoría y se centrará en los algoritmos bayesianos que, entre otros, son clave para afrontar estos desafíos.

4.6.1. Exploración y explotación

En los problemas de *aprendizaje reforzado*, es esencial balancear cuidadosamente la *exploración* del ambiente y el *aprovechamiento* del conocimiento adquirido a través de la experiencia. Si sólo nos dedicamos a explorar el ambiente, vamos a lograr un conocimiento muy basto del mismo, pero perderemos la oportunidad de acumular más recompensa. Si, por el contrario, nos centramos en sólo tomar las decisiones que incrementen la posibilidad de recibir mayor recompensa, entonces podemos quedar atrapados en un máximo local, perdiendo la oportunidad de mejorar nuestra estrategia para acumular una mayor recompensa.

4.6.2. Modelo de estimación basado en métodos bayesianos

Los métodos bayesianos nos permiten balancear de manera óptima la exploración con el aprovechamiento sin tener que depender de otros métodos heurísticos.

4.6.2.1. Estructura del problema

En el aprendizaje reforzado bayesiano, se especifica una distribución inicial sobre todo el modelo de parámetros, es decir, sobre los parámetros que rigen a las distribuciones de las recompensas inmediatas y los parámetros que rigen las probabilidades de transición entre estados. En esta sección nos centramos en los parámetros que

rigen las probabilidades de transición entre estados. Si $\mathcal{S}$ representa el espacio de estados y $\mathcal{A}$ el espacio de acciones, entonces el vector de parámetros $\boldsymbol{\theta}$ está constituido por $|\mathcal{S}|^2|\mathcal{A}|$ componentes que representan cada posible probabilidad de transición. La componente de $\boldsymbol{\theta}$ que rige la probabilidad de transición $T(s'|s,a)$ se denota por $\theta_{(s,a,s')}$.

La estructura del problema puede representarse a través de la red de decisión que se muestra en la Figura 4.12, la cual es una extensión de la red en la Figura 4.11b haciendo explícito el modelo de parámetros. Como se indica por los nodos sombreados, los estados son observados, mientras que el modelo de parámetros no lo es. Por lo general asumiremos que el modelo para los parámetros es invariante en el tiempo, así que $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$. Sin embargo, nuestras creencias sobre $\boldsymbol{\theta}$ evolucionan con el tiempo en la medida en que transitamos hacia un nuevo estado.

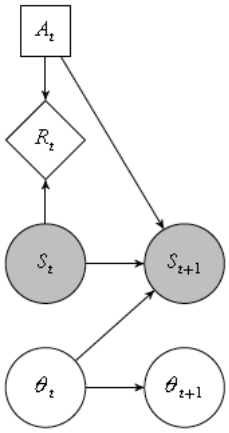


Figura 4.12: Proceso de decisión de Markov con incertidumbre en el modelo.

4.6.2.2. Creencia acerca del modelo parametral

Lo que queremos es representar una creencia inicial sobre $\boldsymbol{\theta}$, y una manera natural de hacerlo para espacios de estados discretos es con el producto de distribuciones Dirichlet. Cada factor Dirichlet podría representar una distribución sobre el próximo estado dado el estado actual s y la acción a. Si $\boldsymbol{\theta}_{(s,a)} = (\theta_{(s,a,s_1)}, \ldots, \theta_{(s,a,s_{|\mathcal{S}|})})$ es el vector de $|\mathcal{S}|$ elementos que representa la distribución sobre el próximo estado, la distribución inicial está dada por

$$\text{Dir}(\boldsymbol{\theta}_{(s,a)}|\boldsymbol{\alpha}_{(s,a)}). \tag{4.26}$$

Esta distribución, se rige por los $|\mathcal{S}|$ parámetros en $\boldsymbol{\alpha}_{(s,a)}$. Es común emplear una distribución inicial uniforme, con todos los componentes de $\boldsymbol{\alpha}_{(s,a)}$ iguales a 1, pero si tenemos conocimiento inicial acerca de la dinámica, entonces podemos establecer otros parámetros que se ajusten más a la información con la que contamos.

La distribución inicial sobre $\boldsymbol{\theta}$ está dada por el producto

$$b_0(\boldsymbol{\theta}) = \prod_s \prod_a \text{Dir}(\boldsymbol{\theta}_{(s,a)} | \boldsymbol{\alpha}_{(s,a)}). \tag{4.27}$$

Esta distribución se utiliza con frecuencia para espacios discretos pequeños, pero otras representaciones paramétricas de dimensiones pequeñas pueden ser deseables.

La distribución final sobre $\boldsymbol{\theta}$ después de t pasos se denota b_t. Supongamos que durante los t primeros pasos, observamos $m_{(s,a,s')}$ transiciones desde s a s' realizando la acción a. Si $\mathbf{m}_{(s,a)}$ representa un vector de conteo de transiciones, entonces la distribución final es

$$b_t(\boldsymbol{\theta}) = \prod_s \prod_a \text{Dir}(\boldsymbol{\theta}_{(s,a)} | \boldsymbol{\alpha}_{(s,a)} + \mathbf{m}_{(s,a)}). \tag{4.28}$$

4.6.2.3. Proceso de decisión de Markov Bayes-adaptado

Podemos formular el problema de actuar de forma óptima en un PDM con un modelo *desconocido* como un PDM de dimensión superior con modelo *conocido*. A este PDM de dimensión superior se le llama *proceso de decisión de Markov Bayes-adaptado* (*Bayes-adaptative Markov decision process*), que esta relacionado con los PDM parcialmente observados (no se estudian en este trabajo, pero puede consultarse en Kochenderfer (2015)).

El espacio de estados en un PDM Bayes-adaptado es el producto cartesiano $\mathcal{S} \times \mathcal{B}$ en donde $\mathcal{B}$ es el espacio de todas las posibles creencias sobre el modelo de parámetros $\boldsymbol{\theta}$. Aunque $\mathcal{S}$ es discreto, $\mathcal{B}$ con frecuencia es un espacio continuo de dimensión superior. Un estado en un PDM Bayes-adaptado se escribe como el par (s, b), que consiste en el estado s del PDM base y el estado de creencia b. El espacio de acción y la función de recompensas son exactamente las mismas que para el PDM base.

La función de transición en un PDM Bayes-adaptado es $\text{T}(s', b' | s, b, a)$, que representa la probabilidad de transitar a algún estado nuevo s' con un nuevo estado de creencia b', dado que se parte de un estado s con creencia b y se realiza la acción

a. El nuevo estado de creencia b' es una función determinista de s, b, a y s' que se calcula empleando la regla de Bayes. Denotemos con τ a esta función determinista, entonces $b' = \tau(s, b, a, s')$. La función de transición de un PDM Bayes-adaptado se puede descomponer de la siguiente manera:

$$\mathrm{T}(s', b'|s, b, a) = \delta_{\tau(s,b,a,s')}(b')p(s'|s, b, a), \tag{4.29}$$

donde $\delta_x(y)$ es la *delta de Kronecker.*

Para calcular $p(s'|s, b, a)$ es necesario integrar

$$p(s'|s, b, a) = \int_\theta b(\theta)p(s'|s, \theta, a)d\theta = \int_\theta b(\theta)\theta_{(s,a,s')}d\theta. \tag{4.30}$$

4.6.2.4. Métodos de solución

Podemos emplear la ecuación de Bellman (ecuación (4.21)) que utilizamos para los PDM con modelo conocido y generalizarla al caso en que el modelo es desconocido:

$$\mathrm{U}^*(s, b) = \max_a \left(\mathrm{R}(s, a) + \gamma \sum_{s'} p(s'|s, b, a)\mathrm{U}^*(s', \tau(s, b, a, s')) \right). \tag{4.31}$$

Desafortunadamente, no podemos simplemente utilizar los algoritmos de iteración de la política ni los de iteración del valor, debido a que b es continua. Sin embargo, podemos emplear otros métodos, como la aproximación de la Subsección 4.5.3. Hay otros métodos que se adaptan de mejor manera a la estructura del PDM Bayes-adaptado y pueden encontrarse en Kochenderfer (2015).

Una alternativa para resolver el valor óptimo de la función de valor sobre el espacio de creencias es usar una técnica conocida como *muestreo de Thompson.* La idea aquí es representar una muestra $\boldsymbol{\theta}$ a partir de la creencia actual b_t y luego suponer que $\boldsymbol{\theta}$ es el modelo verdadero. Utilizamos programación dinámica para resolver la mejor acción. Al siguiente paso, adaptamos nuestras creencias, representamos una nueva muestra, y resolvemos el PDM. La ventaja de este enfoque es que no tenemos que decidir a partir de una exploración heurística de los parámetros. Sin embargo, se ha demostrado que el muestreo de Thompson explora en exceso, y resolver el PDM a cada paso puede ser costoso.

4.6.3. Métodos libres de modelos

En contraste con los métodos basados en modelos (que son los métodos vistos anteriormente en esta sección), los métodos libres de modelos (*model-free*) para el aprendizaje reforzado no requieren una construcción explícita de las transiciones y las recompensas del modelo. Evitar estas representaciones resulta atractivo, sobre todo cuando se atacan problemas de dimensiones altas. A continuación se explica uno de los más famosos métodos libres de modelos.

4.6.3.1. Estimación incremental

Muchos de los métodos libres de modelos involucran el uso de la estimación incremental del retorno de descuento esperado (*expected discounted return*) de los diversos estados en el problema. Supóngase que se tiene una variable aleatoria X y que se quiere estimar la media de un conjunto de muestras $x_{1:n}$. Entonces se tiene el estimador:

$$\hat{x}_n = \frac{1}{n}\sum_{1=1}^{n} x_i. \tag{4.32}$$

Se puede demostrar que

$$\hat{x}_n = \hat{x}_{n-1} + \frac{1}{n}(x_n - \hat{x}_{n-1}) \tag{4.33}$$

$$= \hat{x}_{n-1} + \alpha(n)(x_n - \hat{x}_{n-1}). \tag{4.34}$$

La función $\alpha(n)$ es conocida como la *tasa de aprendizaje* (*learning rate*). La tasa de aprendizaje puede ser una función distinta a $1/n$; existen condiciones bastante holgadas, que se aplican sobre la tasa de aprendizaje para asegurar una convergencia a la media. Si la tasa de aprendizaje es constante, lo que es común en las aplicaciones de aprendizaje reforzado, entonces los pesos de las muestras antiguas decaen exponencialmente a una tasa de $(1 - \alpha)$. Con las tasas de aprendizaje constante, se puede adaptar un estimador después de observar x empleando la siguiente regla:

$$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x}). \tag{4.35}$$

La regla de adaptación anterior aparece en muchos métodos de aprendizaje reforzado y está relacionado con el gradiente descendente estocástico. La magnitud de la adaptación es proporcional a la diferencia en la muestra previamente estimada. La diferencia entre la muestra y el estimador anterior es llamado *error de diferencia temporal* (*temporal difference error*).

4.6.3.2. Q-Learning

Uno de los algoritmos más populares dentro de los métodos libres de modelos es *Q-learning*. La idea es aplicar estimaciones crecientes a la ecuación de Bellman

$$\mathrm{Q}(s,a) = \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a)\mathrm{U}(s') \tag{4.36}$$

$$= \mathrm{R}(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a) \max_{a'} \mathrm{Q}(s',a'). \tag{4.37}$$

En lugar de usar T y R, se emplea el siguiente estado observado s' y la recompensa r para obtener la siguiente regla de actualización de los incrementos:

$$\mathrm{Q}(s,a) \longleftarrow \mathrm{Q}(s,a) + \alpha(r + \gamma \max_{a'} \mathrm{Q}(s',a') - \mathrm{Q}(s,a)). \tag{4.38}$$

Así como otros métodos libres de modelos, en éste se requiere alguna estrategia de exploración para asegurar que Q converge a la función de valor del estado-acción óptimo.

4.7. Búsqueda aleatoria simple

La presente sección se basa en el artículo Mania et al. (2018). En la sección anterior se explican de manera breve los métodos libres de modelo (*model-free methods*); en estos, la búsqueda de soluciones puede realizarse dentro del espacio de políticas o dentro del espacio de acciones. La creencia generalizada apunta a que es más eficiente resolver problemas de aprendizaje reforzado a través de explorar el espacio de acciones que a través de buscar en el espacio de posibles políticas. Esta sección disipa hasta cierto punto esa creencia, por medio del uso de un método de búsqueda aleatorio que utiliza políticas lineales para los problemas de control continuo.

Los métodos de aprendizaje reforzado libres de modelos, son aquellos que tienen el objetivo de ofrecer una solución para controlar el sistema dinámico, sin necesidad de modelar el sistema dinámico; es decir, no requieren de una construcción explícita de las transiciones y las recompensas del modelo. En términos informales, los métodos basados en modelos son aquellos en los que un "agente" se entrena para tomar la mejor decisión a través de probabilidades de transición y recompensas; esto le da la facilidad de predecir sus propios comportamientos ante situaciones nuevas. Por su parte, los métodos libres de modelos se centran en realizar búsquedas dentro del espacio de posibles soluciones; por lo que no hay un "agente" que aprenda y pueda

predecir situaciones nuevas.

Los métodos libres de modelos han mostrado ser muy eficientes al realizar tareas en ambientes controlados, por ejemplo en videojuegos o en juegos de estrategia como el Go. Sin embargo, no se han podido implementar de forma eficiente en tareas dentro de ambientes físicos, donde no hay un control del entorno. Las condiciones por lo que esto ocurre es que la mayoría de los métodos libres de modelos necesitan alimentarse de una gran cantidad de datos para ser eficientes. Por su parte, los métodos de *muestra eficiente*, es decir, aquellos diseñados para ocupar pocos datos, en general son muy robustos y complicados, lo que incrementa la complejidad del problema.

En la actualidad, un punto de referencia importante para probar estos algoritmos son las *tareas de locomoción MuJoCo* (*MuJoCo locomotion task*), un gimnasio virtual en donde diversos robots (desde un gusano hasta un humanoide), tienen que aprender tareas locomotoras como avanzar, caminar y mantener el equilibrio. El ambiente simula la gravedad y el robot virtual puede moverse en dos o tres dimensiones, articulando cada una de sus extremidades de manera independiente. En particular, el modelo del humanoide es considerado como uno de los problemas de control continuo más desafiantes en la actualidad, donde sólo las técnicas de aprendizaje reforzado que pertenecen al estado del arte son capaces de resolverlo (véase la Sección 4.7.4).

En esta sección se desarrolla un método simple de aprendizaje reforzado libre de modelos, el cual es un algoritmo de optimización sin derivadas (*derivative-free*) para entrenar políticas lineales. Se demuestra que un método de búsqueda aleatoria simple puede igualar o exceder el estado del arte en eficiencia muestral para las tareas locomotoras MuJoCo. Más aún, este método es al menos 15 veces más eficiente computacionalmente que el método competitivo más veloz. Esta evidencia contradice la creencia de que las técnicas basadas en políticas de gradiente tienen más eficiencia muestral que los métodos basados en diferencias finitas.

4.7.1. Características del problema de decisión

En los problemas de aprendizaje reforzado se requiere encontrar una política dentro del sistema dinámico, que maximice la recompensa esperada para cierta tarea dada. Estos problemas pueden formularse de manera abstracta del siguiente modo:

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^n} \mathbb{E}_{\xi}[\mathrm{R}(\pi_\theta, \xi)], \tag{4.39}$$

donde $\boldsymbol{\theta} \in \mathbb{R}^n$ parametriza a la política $\pi_\theta : \mathbb{R}^n \to \mathbb{R}^p$. La variable aleatoria ξ representa la aleatoriedad del ambiente, es decir, los estados iniciales aleatorios y las probabilidades de transición. La función $\mathrm{R}(\pi_\theta, \xi)$ es la recompensa adquirida por la política π_θ en la trayectoria generada por el sistema.

4.7.2. Búsqueda aleatoria básica

Observe que el problema planteado en (4.39) tiene como objetivo optimizar la recompensas a través de optimizar directamente sobre el espacio de políticas parametrizadas por $\boldsymbol{\theta}$. Recuerde que otra forma de optimizar es explorar sobre el espacio de posibles acciones, pero en el método propuesto en esta sección nos centraremos en el primer caso. Esta opción permite que el entrenamiento en aprendizaje reforzado sea equivalente a una optimización sin derivadas, donde se añade ruido a las evaluaciones de las funciones.

Uno de los métodos más simples y antiguos para una optimización sin derivadas es la *búsqueda aleatoria*. En ésta, se escoge una dirección aleatoria uniforme sobre la esfera del espacio de parámetros, para luego optimizar la función a lo largo de esa dirección. Una forma primitiva de la búsqueda aleatoria consiste en calcular una aproximación de diferencias finitas a lo largo de la dirección aleatoria, y entonces dar un paso en esa dirección sin usar una búsqueda lineal. El método *Augmented Random Search* ARS, descrito en la Sección 4.7.6, se basa en esta estrategia.

Para actualizar los parámetros $\boldsymbol{\theta}$ de la política π_θ, el método ARS utiliza actualizaciones de la dirección de la forma:

$$\frac{\mathrm{R}(\pi_{\theta+v\delta}, \xi_1) - \mathrm{R}(\pi_{\theta+v\delta}, \xi_2)}{v}, \tag{4.40}$$

para dos v.a. i.i.d. ξ_1 y ξ_2, y donde v es un número real positivo, y δ un vector gaussiano con media cero. Se sabe que este tipo de incremento de actualización es un estimador insesgado del gradiente con respecto a $\boldsymbol{\theta}$ de $\mathbb{E}_\delta \mathbb{E}_\xi[\mathrm{R}(\pi_{\theta+v\delta}, \xi)]$; que a su vez es una versión suave de (4.39) y muy cercana a ésta cuando v es pequeña. Cuando hay ruido en las evaluaciones de la función, se pueden usar grupos pequeños de datos (*minibatches*) para reducir la varianza en la estimación del gradiente. El algoritmo de búsqueda aleatoria básica (BRS por sus siglas en inglés) se bosqueja a continuación. El BRS también es conocido con el nombre de *bandit gradient descent*.

Algoritmo 1. Muestra aleatoria básica (BRS).

1. **Hiperparámetros:** tamaño del paso $\boldsymbol{\alpha}$, número de direcciones muestreadas por cada iteración N, desviación estándar del ruido de la exploración v.

2. **Inicialización:** $\boldsymbol{\theta}_0 = 0$, y $j = 0$.

3. **Ciclo *while*** que continua mientras que la condición final no se cumpla

   3.1. Muestra de $\delta_1, \delta_2, \ldots, \delta_N$ del mismo tamaño que $\boldsymbol{\theta}_j$, con entradas i.i.d. normal estándar.

   3.2. Recolectar $2N$ consultas del horizonte H (véase la sección 4.7.5) y sus recompensas correspondientes al utilizar la política

   $$\pi_{j,k,+}(x) = \pi_{\theta_j + v\delta_k}(x) \quad \text{y} \quad \pi_{j,k,-}(x) = \pi_{\theta_j - v\delta_k}(x),$$

   con $K \in \{1, 2, \ldots, N\}$.

   3.3. Etapa de actualización:

   $$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j + \frac{\alpha}{N} \sum_{k=1}^{N} [\mathrm{R}(\pi_{j,k,+}) - \mathrm{R}(\pi_{j,k,-})]\delta_k.$$

   3.4. $j \longleftarrow j + 1$

4. *Fin del ciclo while*

4.7.3. Otros trabajos relacionados

Con la reciente adopción de entornos virtuales como puntos de referencia para la evaluación de políticas, se han desarrollado una gran cantidad de investigaciones en los métodos de aprendizaje reforzado de control continuo dentro de estos ambientes. De este modo, las plataformas de simulación han sido empleadas para una gran variedad de investigaciones, que en diferentes contextos, comparan las técnicas de aprendizaje reforzado. A continuación se enlistan algunas de estas implementaciones.

Uno de los algoritmos más populares es el *Trust Region Policy Optimization* (TRPO), este método está relacionado con el método de gradiente natural. TRPO maximiza en cada iteración un promedio aproximado de la recompensa, regularizada por una penalización basada en la *divergencia de Kullback-Leibler.* A partir de este

método, se han propuesto otros con algunas mejoras como el *Proximal Policy Optimization* (PPO), que es más fácil de implementar que el TRPO y tiene mejoras en la complejidad muestral.

En otra dirección, se han explorado métodos en los que no se utilizan políticas (*off-policy*), como el *Q-learning*, en donde se emplean todos los datos recolectados por el sistema independientemente de las políticas empleadas para la recolección de los mismos. Esta idea, combinada con los métodos de entrenamiento de políticas deterministas, en los que se toma en cuenta políticas exploratorias; junto con los avances en *deep Q-learning*, se obtiene el método *Deep Deterministic Policy Gradient* (DDPG).

Con frecuencia los problemas de optimización en aprendizaje reforzado son no convexos, lo que ocasiona que muchos métodos encuentren sólo un óptimo local. Para evitar este problema, se ha propuesto el algoritmo *Soft Q-learning*, para inferir políticas estocásticas multimodales a través de maximizar la entropía, lo que permite una mejor exploración en ambientes que presentan recompensas multimodales. El algoritmo *Soft Actor-Critic* (SAC) combina estas técnicas junto con un marco actor-crtico logrando un método en el que el actor tiene como objetivo maximizar la recompensa esperada y la entropía de la política estocástica.

Por otro lado, se encuentra el trabajo en el que se emplean políticas lineales como un método de simplificación del espacio de búsqueda. Para esto se han utilizado gradientes naturales, que son políticas gradiente adaptadas a la métrica de los parámetros dentro del espacio de búsqueda.

Mientras que todos estos métodos dependen de la exploración dentro del espacio de posibles acciones, existen otros métodos de aprendizaje reforzado libres de modelos que se desarrollan para explorar los parámetros del espacio de políticas. Uno de éstos, es el método tradicional de diferencias finitas para aproximar el gradiente en el aprendizaje reforzado libre de modelos, en donde se emplean perturbaciones de los pesos de la política para cada coordenada y regresión lineal para la agregación de medidas. El método desarrollado en esta sección está basado en el de diferencias finitas a lo largo de direcciones distribuidas uniformemente, un método de optimización libre de derivadas (*derivative-free*). Aunque la eficiencia de los métodos de búsqueda aleatoria de diferencias finitas libres de derivadas en optimización convexa han sido probados de manera teórica, se perciben como ineficientes cuando son aplicados a problemas de aprendizaje reforzado no convexos. En esta sección se ofrece evidencia

de lo contrario.

4.7.4. MuJoCo

La información de esta subsección fue obtenida de la página oficial de MuJoCo `http://www.mujoco.org`; y las imágenes fueron obtenidas de la página oficial de *OpenAi Gym*, disponible en `https://gym.openai.com`.

Las siglas MuJoCo provienen de la frase en ingles *Multi-Joint dynamics with Contact*. Es un ambiente virtual que simula un entorno físico y posibilita la aplicación de técnicas de cómputo intensivas, tales como la aplicación a sistemas dinámicos complejos que son ricos en sus características. De manera tradicional, las aplicaciones y algoritmos se prueban primero en este ambiente antes de que sean implementados en el desarrollo de robots físicos, visualizaciones científicas interactivas, ambientes virtuales, animaciones y juegos.

En la Figura 4.13 pueden verse algunos de los ambientes virtuales con los que se puede trabajar para desarrollar tareas locomotoras en MuJoCo.

4.7.5. Un modelo de oráculo

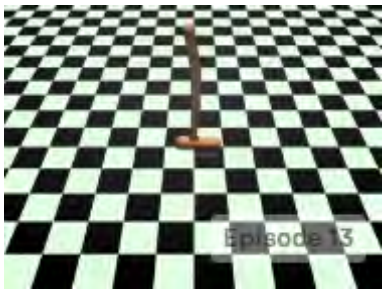
El modelo de oráculo cuantifica la información que el sistema emplea para muchos de los métodos en aprendizaje reforzado. Un algoritmo de aprendizaje reforzado puede consultar al oráculo enviándole una propuesta de política π_θ. Entonces, el oráculo muestrea una variable aleatoria ξ, independiente del pasado, y genera una trayectoria desde el sistema de acuerdo a la política π_θ y la variable aleatoria ξ. El oráculo le regresa al algoritmo una sucesión de estados, acciones y recompensas $\{(s_t, a_t, \mathrm{R}_t)\}_{t=0}^{H-1}$ que representan una trayectoria generada desde el sistema de acuerdo a la política π_θ. A las consultas se les llama episodios; y el número de consultas que hace cada algoritmo al oráculo es una forma de medir y comparar su eficiencia. En un caso ideal, se espera que un algoritmo no haga ninguna consulta al oráculo.

4.7.6. El método ARS

El presente algoritmo se describe en el contexto de los ambientes virtuales del paquete MuJoCo. A partir de aquí y hasta finalizar la sección, se emplea la letra $\boldsymbol{M}$ para denotar los parámetros de las políticas. Ya que este método usa políticas

(a) *Ant-v2.* Hacer caminar a un robot 3D de cuatro patas.

(b) *HalfCheetah-v2.* Hacer correr a un robot leopardo 2D.



(c) *HalfCheetah-v2.* Hacer que salte un robot 2D.

(d) *Swimmer-v2.* Hacer que nade un robot 2D.

(e) *Humanoid-v2.* Hacer caminar a un robot 3D de dos piernas.

(f) *Walker2d-v2.* Hacer caminar a un robot 2D.

Figura 4.13: Diversas tareas locomotoras que pueden realizarse en MuJoCo.

lineales, $\boldsymbol{M}$ es una matriz de $p \times n$.

La primera versión del método es ARS **V1**, se obtiene a partir de BRS (Sección 4.7.2), pero en el inciso 3.3 del Algoritmo 1 los pasos de actualización se ponderan con σ_{R} la desviación estándar de la recompensa recolectada en cada iteración (véase la Sección 4.7.7 en donde se explica la motivación para usar esta escala). Este algoritmo es capaz de entrenar políticas lineales que alcanzan la recompensa máxima expuesta en la literatura, para los entornos *Swimmer-v1, Hopper-v1, HalfCheetah-v1, Walker2d-v1* y *Ant-v1*.

Sin embargo, ARS **V1** emplea un gran número de episodios para entrenar satisfactoriamente las políticas en estos entornos, y no es capaz de entrenar políticas para el entorno *Humanoid-v1*. Para corregir este problema, en el Algoritmo 2 se propone a ARS **V2**. Para este algoritmo las políticas son mapeos lineales de los estados, normalizados por la media y la desviación estándar calculada en línea (*online*[3]). Para una explicación más detallada de este procedimiento véase la Sección 4.7.8.

Para mejorar los algoritmos ARS **V1** y ARS **V2**, se implementaron algunas modificaciones y se desarrollaron los algoritmos ARS **V1-t** y ARS **V2-t**. En estas versiones, los ARS evitan algunas direcciones de perturbación, lo que produce una mejora en la recompensa. En la Sección 4.7.9 se muestra la motivación y explicación de esta implementación.

Algoritmo 2. Muestra aleatoria aumentada (ARS): cuatro versiones **V1**, **V1-t**, **V2** y **V2-t**.

1. **Hiperparámetros:** tamaño del paso $\boldsymbol{\alpha}$, número de direcciones muestreadas por cada iteración N, desviación estándar del ruido de la exploración v, numero

[3]Dentro del aprendizaje reforzado, los algoritmos pueden clasificarse en dos tipos de métodos:

- Métodos *Offline.* En el aprendizaje reforzado, los métodos *offline* son los que se desarrollan a partir de calcular la política de todo el espacio de acciones previo a su ejecución en el entorno (*offline*).
- Métodos *Online.* En el aprendizaje reforzado, los métodos *online* son aquellos que restringen el cálculo de la política sólo hacia aquellos estados que se alcanzan a partir del estado actual. Debido a que el espacio de estados "alcanzables" puede tener una magnitud mucho menor que el espacio completo, los métodos *online* pueden reducir de manera significativa la cantidad de almacenamiento y los requerimientos computacionales necesarios para seleccionar (o aproximar) el óptimo.

de direcciones de alto rendimiento b ($b \leq N$ es permitido sólo para **V1-t** y **V2-t**).

2. **Inicialización:** $\boldsymbol{M}_0 \in \mathbb{R}^{p\times n}, \mu_0 = 0 \in \mathbb{R}^n, \Sigma_0 = \boldsymbol{I}_n \in \mathbb{R}^{n\times n}$ y $j = 0$.

3. **Ciclo *while*** que continua mientras que la condición final no se cumpla

   3.1. Muestra de $\delta_1, \delta_2, \ldots, \delta_N \in \mathbb{R}^{p\times n}$ con entradas i.i.d. normal estándar.

   3.2. Recolectar $2N$ consultas del horizonte H y sus recompensas correspondientes al utilizar las $2N$ políticas

$$\boldsymbol{V1}: \begin{cases} \pi_{j,k,+}(x) = (\boldsymbol{M}_j + v\delta_k)x \\ \pi_{j,k,-}(x) = (\boldsymbol{M}_j - v\delta_k)x \end{cases}$$
$$\boldsymbol{V2}: \begin{cases} \pi_{j,k,+}(x) = (\boldsymbol{M}_j + v\delta_k)\text{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \\ \pi_{j,k,-}(x) = (\boldsymbol{M}_j - v\delta_k)\text{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

   para $K \in \{1, 2, \ldots, N\}$.

   3.3. Ordenar las direcciones δ_k con máx$\{\text{R}(\pi_{j,k,+}, -\text{R}(\pi_{j,k,-})\}$, denotando por $\delta_{(k)}$ la k-ésima dirección más grande, y $\pi_{j,(k),+}$ y $\pi_{j,(k),-}$ su política correspondiente.

   3.4. Etapa de actualización:

$$\boldsymbol{M}_{j+1} = \boldsymbol{M}_j + \frac{\alpha}{b\sigma_{\text{R}}} \sum_{k=1}^{b} [\text{R}(\pi_{j,(k),+}) - \text{R}(\pi_{j,(k),-})]\delta_{(k)},$$

   donde σ_{R} es la desviación estándar de las $2b$ recompensas empleadas en el paso de actualización.

   3.5. **V2**: Se establece a μ_{j+1}, Σ_{j+1} como la media y la covarianza de los $2NH(j+1)$ estados que se encuentran desde el inicio del entrenamiento.

   3.6. $j \longleftarrow j + 1$

4. *Fin del ciclo while*

4.7.7. Desviación estándar como escala

En el progreso del entrenamiento de las políticas, la búsqueda aleatoria dentro del espacio de parámetros de las políticas puede llevar a grandes variaciones en las

recompensas en la medida en que aumentan las iteraciones. Como resultado, es difícil seleccionar un número fijo de pasos α que no permita cambios perjudiciales entre pasos grandes y pequeños.

Para corregir las grandes variaciones entre las diferencias $\mathrm{R}(\pi_{M+v\delta}) - \mathrm{R}(\pi_{M-v\delta})$, el paso de actualización se escala mediante la desviación estándar σ_R de las $2N$ recompensas recolectadas en cada iteración. En la tarea locomotora *Humanoid-v1*, σ_R tiene una tendencia a aumentar a lo largo del progreso del entrenamiento; por lo que si no se realiza este escalamiento, al rededor de la iteración número 300 se podrían tomar pasos que son miles de veces más grandes que al principio del entrenamiento.

4.7.8. Normalización de los estados

La normalización de los estados que se emplea en **V2** es una transformación lineal de los datos de la familia *whitening*. Estas transformaciones se utilizan en tareas de regresión y consiste en transformar linealmente un vector de variables aleatorias con matriz de covarianzas conocida, en un conjunto de nuevas v.a. cuya covarianza es la matriz identidad. Esto implica que las nuevas variables son no correlacionadas y tienen varianza 1.

Supóngase que X es un vector aleatorio con media 0 y cuya matriz de covarianzas M es no singular. Si W es una matriz *whitening*, satisface la condición $W^T W = M^{-1}$. Entonces, la transformación $Y = WX$ proporciona un vector aleatorio "blanqueado" Y con matriz de covarianzas igual a la identidad. Hay una cantidad infinita de posibles matrices W que satisfacen la condición anterior. Sin embargo es común utilizar la matriz $W = M^{1/2}$ (tal como se hace en el Algoritmo ARS **V2**).

Intuitivamente, en el algoritmo **V2** esta transformación asegura que las políticas mantengan el mismo peso para las diferentes componentes de los estados. Por ejemplo, supóngase que una coordenada de estado sólo toma valores en el rango $[90, 100]$, mientras que otra toma valores en el rango $[-1, 1]$. De esta forma, un pequeño cambio en el control del estado en su primer coordenada podría generar un gran cambio en las acciones, debido a que dar el mismo salto en las coordenadas de la segunda componente del estado significaría un cambio muy grande. De esta forma, *whitening* permite una exploración isotrópica de la búsqueda aleatoria, dando igual influencia sobre los diversas componentes de los estados. Por esto, el algoritmo ARS puede verse como una forma no isotrópica de exploración en el espacio de parámetros de políticas lineales.

Para implementar de manera eficiente la técnica de *whitening* en **V2**, no se requiere almacenar la información de todos los estados; así que sólo se mantiene un seguimiento de la diagonal de la matriz Σ_{j+1}.

La motivación para crear la versión 2 del algoritmo está fundada en que las tareas de entrenamiento del *Humanoid-v1* son ineficientes sin la normalización de los estados. Sin embargo, la medida de complejidad de ARS **V2** también resulta mejor para las otras tareas locomotoras de MuJoCo (véase la Sección 4.7.10).

4.7.9. Uso de direcciones de alto rendimiento

Aunque el ARS **V2** alcanza o excede el estado del arte en el desempeño de varias de las tareas locomotoras en el entorno MuJoCo; sin embargo, para entrenar al modelo *Walker2d-v1* y al *Ant-v1* se requiere de dos a tres veces más consultas que en otros métodos.

Para mejorar este rendimiento, se proponen las versiones ARS **V1-t** y ARS **V2-t**. En los pasos de actualización de los algoritmos ARS **V1** y **V2** cada dirección de perturbación δ es ponderada por la diferencia de las recompensas $\mathrm{R}(\pi_{j,k,+})$ y $\mathrm{R}(\pi_{j,k,-})$. Estas dos recompensas se obtienen de hacer dos consultas al oráculo descrito en la Sección 4.7.5, usando las políticas

$$\pi_{j,k,+}(x) = (\boldsymbol{M}_j + v\delta_k)\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j), \quad \text{y}$$
$$\pi_{j,k,-}(x) = (\boldsymbol{M}_j - v\delta_k)\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j).$$

Si $\mathrm{R}(\pi_{j,k,+}) > \mathrm{R}(\pi_{j,k,-})$, entonces los pasos de actualización de ARS **V1** y **V2** empujan los pesos de la política $\boldsymbol{M}_j$ hacia la dirección δ_k. Por el contrario, si $\mathrm{R}(\pi_{j,k,+}) < \mathrm{R}(\pi_{j,k,-})$, los pasos de actualización de ARS **V1** y **V2** empujan los pesos de la política $\boldsymbol{M}_j$ hacia la dirección $-\delta_k$. Sin embargo, como $\mathrm{R}(\pi_{j,k,+})$ y $\mathrm{R}(\pi_{j,k,-})$ son evaluaciones contaminadas por ruido del rendimiento de las políticas parametrizadas por $\boldsymbol{M}_j + v\delta_k$ y $\boldsymbol{M}_j - v\delta_k$, podría ocurrir que ARS **V1** y **V2** empujaran los pesos de la política $\boldsymbol{M}_j$ hacia δ_k, aun cuando $-\delta_k$ fuera mejor; o viceversa. Más aún, puede haber direcciones de perturbación δ_k tal que la actualización de los pesos de la política $\boldsymbol{M}_j$ en cualquiera de las direcciones δ_k o $-\delta_k$, conduciría a un rendimiento subóptimo.

Por ejemplo, supongamos que las políticas $\mathrm{R}(\pi_{j,k,+})$ y $\mathrm{R}(\pi_{j,k,-})$ son ambas pequeñas comparadas con el resto de las recompensas observadas; esto podría sugerir que mover a $\boldsymbol{M}_j$ hacia cualquiera de las direcciones δ_k o $-\delta_k$ decrecería el promedio

de las recompensas obtenidas. Para corregir este problema, en ARS **V1-t** y **V2-t** se ordenan de manera decreciente las direcciones de perturbación δ_k de acuerdo con máx$\{\mathrm{R}(\pi_{j,k,+}), \mathrm{R}(\pi_{j,k,-})\}$, después sólo se emplean las b direcciones más grandes para actualizar los pesos de la política (véase el Algoritmo 2 inciso 3d).

Esta optimización en el algoritmo, intuitivamente hace una mejora en el paso de actualización del ARS, ya que con esto las actualizaciones son resultado de promediar sólo las direcciones hacia las que se obtienen altas recompensas. En la Sección 4.7.10 se muestra que ARS **V2-t** excede o alcanza el estado del arte en el desempeño de todas las tareas locomotoras MuJoCo incluidas en el paquete *OpenAI Gym*.

4.7.10. Resultados

En Mania et al. (2018), los algoritmos ARS fueron evaluados en las tareas locomotoras MuJoCo incluidas en el ambiente *OpenAI Gym-v0.9.3*. El ambiente *OpenAI Gym* proporciona funciones de recompensas que actuan como punto de referencia para comparar la efectividad de cada algoritmo. Los resultados mostrados a continuación son los que se reportan en Mania et al. (2018) empleando estas recompensas de referencia. Las recompensas que se reportan son el promedio sobre 100 consultas independientes.

En la Tabla 4.14 se muestran los resultados para alcanzar un umbral de recompensa de 6000, tras evaluar el desarrollo del algoritmo ARS en el ambiente *Humanoid-v1*; así mismo, se muestran otros umbrales de las demás tareas locomotoras que son referencia en la literatura. Por último, se comparan con los resultados obtenidos con otros algoritmos: NG-lin, NG-rbf y TRPO-nn. El número de episodios necesarias para alcanzar el umbral es mucho menor con el algoritmo ARS.

La Tabla 4.15 muestra las máximas recompensas obtenidas al ejecutar los algoritmos ARS; se toma el máximo sobre todos los conjuntos de hiperparámetros considerados, así como el máximo de tres semillas aleatorias fijas. En particular, para *Humanoid-v1* y *Walker2d-v1*, el algoritmo ARS encuentra políticas que alcanzan recompensas significativamente más altas que cualquier otra registrada en la literatura. Estos resultados muestran que las políticas lineales son perfectamente adecuadas para ejecutar las tareas locomotoras MuJoCo, evitando así el uso de políticas que son más complejas y caras en términos computacionales.

En general, es difícil decir que un algoritmo es mejor que otro cuando la com-

Promedio sobre los episodios necesarios para alcanzar el umbral de recompensa.

Task	Threshold	ARS				NG-lin	NG-rbf	TRPO-nn
		V1	V1-t	V2	V2-t			
Swimmer-v1	325	100	100	427	427	1450	1550	N/A
Hopper-v1	3120	89493	51840	3013	1973	13920	8640	10000
HalfCheetah-v1	3430	10240	8106	2720	1707	11250	6000	4250
Walker2d-v1	4390	392000	166133	89600	24000	36840	25680	14250
Ant-v1	3580	101066	58133	60533	20800	39240	30000	73500
Humanoid-v1	6000	N/A	N/A	142600	142600	≈130000	≈130000	UNK

Tabla 4.14: Comparación de los algoritmos ARS, NG y TRPO en las tareas locomotoras MuJoCo. Tabla tomada de Mania et al. (2018).

Task	ARS
Swimmer-v1	365
Hopper-v1	3909
HalfCheetah-v1	6722
Walker	11389
Ant	5146
Humanoid	11600

Tabla 4.15: Máxima recompensa alcanzada. Tabla tomada de Mania et al. (2018).

paración entre ellos se hace únicamente en ambientes simulados, ya que alguno de los algoritmos puede estar explotando particularidades del simulador utilizado. A pesar de esto, parece preferible desarrollar algoritmos que se centren en encontrar buenas soluciones para problemas específicos, que centrarse en desarrollar algoritmos generales que sean aplicables a diferentes clases de problemas.

Capítulo 5

Aplicaciones

Este capítulo se divide en dos secciones. En la primera se tiene como objetivo la optimización, mediante el uso de algoritmos genéticos, del algoritmo de árboles de clasificación CART. En la segunda sección se plantea un problema de toma de decisiones secuenciales con base en la teoría de decisión markoviana, se resuelve mediante dos técnicas distintas y se comparan los resultados obtenidos. La primera de las técnicas empleadas es la programación dinámica y la segunda la búsqueda directa en el espacio de posibles políticas; cabe resaltar que para llevar a cabo esta búsqueda se utiliza un algoritmo genético.

5.1. Algoritmos genéticos en árboles de clasificación

5.1.1. Algoritmo evolutivo para la inducción global de árboles de decisión

Esta sección se basa en Krętowski and Czajkowski (2010). Aquí se explica un algoritmo evolutivo de árboles de regresión, que en particular puede ser aplicado a CART (véase la Sección 3.5). En contraste con CART y otros métodos de estructuras de árboles que se construyen *de arriba hacia abajo* (*top-down methods*), empezando del nodo raíz, estos algoritmos buscan la división óptima local de acuerdo con alguna medida de optimización previamente establecida (a ésta se les conoce como *prueba* o *test*) para reagrupar la muestra de entrenamiento en nuevos nodos. Este procedimiento se sigue de manera recursiva hasta satisfacer algún criterio de convergencia. Finalmente, se aplica algún tipo de poda al árbol, para mejorar la efectividad del modelo en la generalización de predicciones.

El método aquí propuesto realiza una búsqueda global de la mejor estructura de árbol. La población inicial se crea a partir de un método de arriba hacia abajo empleando sub-muestras aleatorias de los datos de entrenamiento. Se diseñan operadores genéticos especializados que permiten al algoritmo desarrollar eficientemente árboles de regresión. El término de complejidad empleado en la función de aptitud ayuda a reducir el problema de sobre-ajuste. La validación experimental preliminar es prometedora, ya que los árboles obtenidos pueden ser significativamente menos complejos, y con al menos un rendimiento comparable al de su contraparte clásica de arriba-abajo.

5.1.1.1. Introducción

En la Sección 3.5 se abordó uno de los métodos más populares basados en árboles: CART. La popularidad de estos métodos puede ser explicada por su sencilla aplicación, rapidez operacional y lo que podría ser lo más importante, su efectividad. Además, la estructura jerárquica del árbol, en la que se aplican de manera consecutiva *pruebas* apropiadas para separar los datos, se acerca mucho a la forma en que los humanos toman decisiones; por esto, los métodos basados en árboles son fáciles de entender incluso para los analistas con poca experiencia.

Existen diversos métodos basados en árboles; la mayoría de éstos, incluido el CART, inducen los árboles de regresión con el método de arriba hacia abajo. Hay que tener presente que este método se caracteriza por ser una técnica *voraz* (*greedy*), de tal modo que a pesar de resultar eficiente para resolver muchos problemas prácticos, no existe ninguna garantía de alcanzar la solución óptima global.

De acuerdo con Khosrow-Pour (2017), la *inducción global* es un método para generar árboles de decisión en donde se busca, al mismo tiempo, tanto la estructura del árbol como todas las pruebas (*tests*); se basan usualmente en aproximaciones evolutivas, en contraste con los métodos de inducción de arriba hacia abajo.

Es de esperarse que para ciertas situaciones será más adecuada la inducción global que el método de arriba hacia abajo. Por esto, a lo largo de la sección se propone una aproximación a la inducción global de árboles de regresión basada en un algoritmo evolutivo especializado.

5.1.1.2. Inducción evolutiva para árboles de regresión

La estructura general del sistema sigue el típico marco de los algoritmos evolutivos. Éstos son una implementación de los algoritmos genéticos a un espacio de posibles soluciones conformado por árboles. Las operaciones de selección, cruza, mutación y aptitud, se adaptan a la estructura propia de los árboles.

- ***Representación, inicialización y condiciones de finalización***

  - **Representación.** Cada *prueba* en un nodo interno (*non-terminal node*) permite sólo un atributo (valuación nominal o continua). Adicionalmente, en cada uno de los nodos se almacena la información sobre los *vectores de aprendizaje*[1] asociados con dicho nodo. Esto permite al algoritmo desarrollar estructuras locales y pruebas de modificación más eficientes durante la aplicación de los operadores genéticos. En cada *hoja* (nodo terminal) se calcula la media de las variables dependientes de los objetos de entrenamiento, dicha media es el valor predictivo estimado y ese valor se asocia a la hoja dada.

    En el caso de atributos nominales al menos un valor es asociado con cada rama, y para los atributos de variable continua típicamente se emplean pruebas de desigualdad. Como divisiones potenciales sólo se consideran candidatos previamente calculados. Un candidato a umbral para cierto atributo se define como un punto medio entre pares sucesivos de ejemplos en una serie ordena de manera creciente de acuerdo al valor del atributo; en donde los ejemplos se caracterizan por diferentes valores predictivos. Esta solución limita significativamente el número de posibles divisiones y enfoca el proceso de búsqueda.
  - **Inicialización.** En el enfoque propuesto, se crea a los individuos iniciales aplicando el algoritmo clásico de arriba hacia abajo a través de seleccionar sub-muestras aleatorias de la muestra de entrenamiento original (valor arbitrario predeterminado: 10 % de los datos por cada sub-muestra, pero no más de 500 datos). Para cada árbol inicial se emplea alguna estrategia de búsqueda de pruebas, como las técnicas CART. Se detiene el proceso cuando todos los objetos de entrenamiento en un nodo son caracterizados por el mismo valor predictor, el número de objetos de un nodo es menor que el valor predefinido (valor predeterminado: 5) o se alcanza la profundidad máxima del árbol (valor predeterminado: 10).

[1] En estos vectores está la información de las pruebas aplicadas a cada nodo.

- **Condición de finalización.** Clásicamente, la evolución finaliza cuando la aptitud del mejor individuo de la población no mejora durante un número fijo de generaciones (el valor predeterminado es 1000). Además, se especifica un número máximo de generaciones, con lo que se limita el tiempo de cómputo en caso de convergencias lentas (valor predeterminado: 5000).

- ***Operadores genéticos.***

Los dos tipos de operadores genéticos son mutación y cruza. La aplicación de cualquiera de éstos puede generar cambios en la estructura del árbol y en las pruebas de los nodos internos. Después de aplicar cualquier operador, generalmente es necesario reubicar los vectores de aprendizaje entre las partes del árbol enraizadas en el nodo alterado. Esto puede ocasionar que algunas partes del árbol no contengan ningún vector de aprendizaje y tengan que ser podadas.

- **Operador mutación.** Se proponen 5 distintos operadores de mutación. Cada tipo de operador mutación es aplicado con cierta probabilidad dada (valor predeterminado 0.8), logrando con esto una probabilidad muy alta de que al menos un nodo del individuo pueda mutar. Para empezar, se escoge de manera aleatoria y con igual probabilidad el tipo de nodo (hoja o nodo interno). Si el tipo de mutación resulta imposible para este nodo, entonces se escoge el otro tipo de nodo. Se crea una lista jerarquizada de nodos del género seleccionado para decidir cuál de los nodos será afectado. En lo referente a los nodos internos, se considera la posición (nivel) del nodo en el árbol y la calidad del sub-árbol que empieza en el nodo seleccionado. Es evidente que hacer modificaciones en el nodo raíz tiene un gran impacto ya que afecta a todo el árbol, mientras que las mutaciones en los nodos internos que se encuentran en las partes más bajas del árbol tienen sólo un impacto local. En el método propuesto, los nodos en los niveles más altos del árbol tienen muy poca probabilidad de ser seleccionados; y entre los nodos del mismo nivel se utiliza el error absoluto calculado en el vector de aprendizaje localizado en el sub-árbol para ordenarlos. En el caso de las hojas, sólo el error absoluto se emplea para ordenarlas, pero no se incluyen las hojas homogéneas, es decir, aquellas en las que todos sus objetos pertenecen a una misma clase. Como resultado, aquellas hojas que tienen peor precisión son las que se mutan con mayor probabilidad.

Las modificaciones diseñadas por el operador mutación dependen del tipo de nodo (es decir, si es una hoja o un nodo interno). Para nodos internos se proponen las siguientes posibilidades:

- Alterar la prueba existente cambiando los umbrales (cuando se trata de atributos continuos), o reagrupando los valores de los atributos (atributos nominales).
- Se remplaza una prueba por otra, o las pruebas pueden ser intercambiadas.
- Se remplaza un sub-árbol por otro sub-árbol que surja del mismo nodo.
- Se poda el nodo para convertirlo en una hoja.

Modificar una hoja tiene sentido sólo si tiene objetos con diferentes valores en la variable dependiente. Se transforma la hoja en un nodo interno y se escoge una nueva prueba de la forma antes mencionada.

- **Operador cruza.** Hay variantes para hacer la combinación de genes. Todas ellas empiezan con la selección de las posiciones de cruza de los dos individuos afectados. Se escoge un nodo de manera aleatoria en cada uno de los dos árboles. En la variante más sencilla, se intercambian los sub-árboles que empieza en los nodo seleccionados. Esta es la clásica cruza presente en los algoritmos de programación genética. La segunda variante es intercambiar las pruebas asociadas con cada uno de los nodos seleccionados. La tercer variante consiste en intercambiar las ramas que surgen del nodo seleccionado en orden aleatorio.

En la implementación realizada en esta tesis (Sección 5.1.2) se utiliza la técnica de cruza que consiste en elegir un nodo aleatorio en cada uno de los dos árboles para luego intercambiar los sub-árboles que empiezan en dichos nodos. Observe que cada uno de estos sub-árboles representa una sub-región en la partición del espacio que hace el árbol (véase la Figura 3.2b). Las sub-regiones están divididas a menos que representen a una hoja, por lo que al intercambiar distintas sub-regiones (no hojas) también se intercambian los valores de las divisiones internas, siempre y cuando tengan cabida en la otra sub-región, de no ser este el caso, se omite la nueva división y sería equivalente a hacer un podado. Como cada una de las divisiones se realiza con un algoritmo voraz, se tiene que cada división es óptima para algún paso en particular y para cierto conjunto de datos. Por esta razón, es que la cruza de sub-árboles puede mejorar la aptitud de los cromosomas en la población.

- ***Selección.*** Como mecanismo de selección se utiliza la *selección por jerarquía lineal*[2]. Además, el individuo con mayor valor de aptitud en la iteración pasa a la próxima generación (estrategia de elitismo).

- ***Función de aptitud.*** En las tareas de predicción es bien sabido que la minimización de la medida del error de predicción sobre el conjunto de aprendizaje, lleva a un problema de sobre-ajuste. En la inducción típica de arriba hacia abajo para árboles de decisión, el problema de sobre especialización se reduce parcialmente cuando se emplea una condición de paro y se realiza un podado. En este enfoque, la búsqueda de una estructura óptima se integra en el algoritmo evolutivo a través de incorporar un término de complejidad dentro de la aptitud. Este término funciona como una penalización para el incremento del tamaño del árbol.

  La función de aptitud se minimiza de la siguiente forma:

$$\text{Aptitud}(T) = \frac{\text{MAE}(T)}{\text{MAE}_{Max}} + \alpha \cdot (S(T) - 1.0), \tag{5.1}$$

  donde $S(T)$ es el tamaño del árbol, es decir, el número de nodos; y $\text{MAE}(T)$ es el error absoluto medio del árbol T medido sobre el conjunto de entrenamiento. Al sustraer 1.0 se elimina la penalización de cuando el árbol se compone sólo de una hoja. MAE_{Max} es el máximo $\text{MAE}(T)$ para el conjunto de entrenamiento. Se puede calcular fácilmente, como se observa cuando el árbol se compone sólo por una hoja. Para cualquier árbol T más complejo, se tiene que $\text{MAE}(T) \leq \text{MAE}_{Max}$. El factor α es de relativa importancia para el término de complejidad (el valor predeterminado es 0.001).

  Debe observarse que no existe un valor óptimo de α para todos los posibles conjunto de datos, por lo que ajustarlo permite mejorar los resultados de un problema en específico.

[2]De acuerdo con Coello (2016), esta selección consiste en ordenar de manera decreciente a los individuos de 1 a N con base en su aptitud. Se escoge algún real Max tal que $1 \leq Max \leq 2$ (el valor predeterminado es $Max = 1.1$), entonces $Min = 2 - Max$; el *valor esperado* para el individuo a se define como:

$$\text{Valesp}(a) = Min + (Max - Min)\frac{\text{jerarquía}(a) - 1}{N - 1}.$$

Por último se emplea algún método de selección proporcional con base a los valores esperados; tal como la *rueda de ruleta* (Sección 2.3.5.1).

5.1.1.3. Validación experimental

En Kręłowski and Czajkowski (2010) se reporta la realización de dos grupos de experimentos para validar la aproximación global a la inducción de árboles de regresión (en las tablas se denota por *GRT* de acuerdo a sus siglas en inglés). Con el propósito de comparar, los resultados fueron obtenidos con el clásico inductor de arriba hacia abajo *REPTree*, que está disponible para el público en el sistema *Weka* (Frank et al. (2016)). Ambos sistemas se corren con los valores predeterminados de los parámetros. Todos los resultados presentados en la tabla corresponden al promedio de 10 ejecuciones y fueron obtenidos utilizando conjuntos de pruebas (cuando estaban disponibles) o por validación cruzada de 10 iteraciones. El número promedio de nodos se da como una medida de complejidad de los árboles de regresión.

Se hicieron experimentos en conjuntos de datos tanto simulados como reales, en este resumen sólo se presentan las conclusiones obtenidas con los datos reales.

Los conjuntos de datos reales fueron tomados de *UCI Machine Learning Repository* (Blake et al. (1998)). La Tabla 5.1 presenta las características de los conjuntos de datos investigados y los resultados obtenidos.

Se observa que la predicción de precisión de ambos sistemas es comparable. Sin embargo, hay que resaltar que los árboles globalmente inducidos presentan una complejidad significativamente menor. Esto se vislumbra especialmente en los árboles más grandes generados por el *REPTree*, pues el *GRT* alcanza soluciones más pequeñas (por ejemplo, 67.2 nodos en comparación con 819, o 53 comparados con 553).

Cabe mencionar que los resultados presentados en la Tabla 5.1 se obtuvieron con el valor predeterminado de α. Con el fin de verificar el impacto de este parámetro en los resultados, se realizó una serie de experimentos variando el valor de α en dos conjuntos de datos. Se observó que en la medida en que α decrece, aumenta la complejidad del árbol.

5.1.2. Implementación

En esta sección se realiza un algoritmo de inducción global basado en la Sección 5.1.1, pero procurando una implementación sencilla; es decir, se omiten muchos de los pasos complejos propuestos en Kręłowski and Czajkowski (2010) y se sustituyen por otros que son más fáciles de programar.

Dataset	Properties	*GRT*		*REPTree*	
		RMSE	Tree size	RMSE	Tree size
Abalone	4177/7/1	2.30	49.3	2.36	201
Ailerons	13750/40/0	0.00022	53.3	0.00020	553
Auto-Mpg	392/4/3	3.59	145.6	3.65	94
Auto-Price	159/17/10	2505	91.2	2760	32
Delta Ailerons	7129/6/0	0.000182	29.8	0.000175	291
Delta Elevators	9517/6/0	0.00156	25.0	0.0015	319
Elevators	16559/40/0	0.0044	65.9	0.0040	503
Housing	506/14/0	4.29	88.6	4.84	41
Kinematics	8192/8/0	0.193	67.2	0.191	819
Machine CPU	209/7/0	60.4	75.8	92.3	15
Pole	15000/48/0	10.23	47.8	8.26	223
Pyrimidines	74/28/0	0.101	81.5	0.136	1
Stock	950/10/0	1.317	59.2	1.186	137
Triazines	186/61/0	0.149	159.0	0.152	7

Tabla 5.1: Características del conjunto de datos (número de objetos/número de atributos numéricos/número de atributos nominales) y resultados obtenidos. El error de la raíz cuadra de la media (RMSE) se presenta como la medida de error y el número de nodos como el tamaño del árbol. Tabla tomada de Krętowski and Czajkowski (2010), página 163.

5.1.2.1. Pseudo-código

A continuación se enlistan los pasos que se siguieron para programar el algoritmo. El código completo se encuentra en `https://github.com/BrunoRGutierrez/Global_induction_algoritm_`.

1. Se selecciona una base de datos.
2. **Población inicial**
   - 2.1. La base de datos se divide en el conjunto de entrenamiento y el conjunto de prueba (valores prederminados: 70 % y 30 % respectivamente).
   - 2.2. El conjunto de entrenamiento se divide en N subconjuntos. Cada uno se selecciona aleatoriamente sin reemplazo y la selección es independiente de la de los otros subconjuntos (valor predeterminado $N = 50$), donde N representa el número de árboles de la población inicial. Cada subconjunto tiene un tamaño predeterminado de 10 % de los datos (pero no más de 500 datos por subconjunto).
   - 2.3. En cada subconjunto se crea un árbol utilizando el método CART. Dicho algoritmo fue tomado de Browniee (2016). Se detiene el proceso cuando:
     - todos los objetos de entrenamiento en un nodo son caracterizados por el mismo valor predictor;
     - el número de objetos en cada nodo entra al umbral del número mínimo permitido, tras lo cual la división queda prohibida (valor predeterminado: 5);
     - se alcanza la profundidad máxima del árbol (valor predeterminado: 10).
3. **Función de aptitud.**
   - 3.1. Se diseña una función de aptitud empleando la tasa de clasificación errónea (*Misclasification error*) sobre todas las regiones R_m en que el árbol divide el espacio. Si el número de regiones es M, entonces la función de aptitud del árbol a está dada por:

$$f(a) = \sum_{m=1}^{M} \frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)),$$

donde $k(m)$ es la etiqueta de clase clase de la región R_m.

4. **Selección.** Se utiliza la *selección por jerarquía lineal*:

   4.1. Se crea una lista ordenada de los árboles de la población de acuerdo a su valor de aptitud (1 para el individuo peor valuado, N para el mejor valuado).

   4.2. Se escoge un número real Max tal que $1 \leq Max \leq 2$ (el valor predeterminado es $Max = 1.7$). Mientras más pequeño es el valor de Max, menos distancia habrá entre los pesos de las probabilidades de la lista; mientras que en la medida que Max se acerque a 2, los pesos de las probabilidades se distanciarán más.

   4.3. Se define $Min = 2 - Max$.

   4.4. El *valor esperado* para el individuo a se define como:

$$\text{Valesp}(a) = Min + (Max - Min)\frac{\text{jerarquía}(a) - 1}{N - 1},$$

   donde la *jerarquía* es igual al lugar que ocupa el árbol a en la lista ordenada.

   4.5. Se emplea el método de *rueda de ruleta* a partir de la función anterior.

5. **Cruza.**

   5.1. Se selecciona un nodo al azar en cada uno de los árboles. El nodo raíz no puede ser seleccionado.

   5.2. Se intercambian los sub-árboles que empiezan en los nodos seleccionados.

6. **Mutación.** Todos los nuevos individuos son mutados.

   6.1. Se selecciona de manera uniforme un nodo en cada uno de los árboles hijos.

   6.2. Si el nodo es hoja se selecciona el nodo interno más cercano a la hoja.

   6.3. Se pregunta el atributo de división del nodo seleccionado.

   6.4. Se cambia el umbral del atributo a partir de una lista ordenada de posibles puntos de división seleccionados a partir de una muestra aleatoria de datos (tamaño de la muestra: 100 elementos).

   - Se obtiene un número aleatorio entre 0 y 1.
   - Si cae 0, se utiliza el umbral que está abajo del nodo seleccionado en la lista.

- Si cae 1, se utiliza el umbral que está arriba del nodo seleccionado en la lista.

7. **Reemplazo.** Los mejores N individuos (Población inicial: N) con mayor valor de aptitud en la iteración pasa a la próxima generación.
8. Se repiten los pasos del 4 al 7 en un ciclo *while*.
9. **Condición de finalización.** Finaliza cuando ocurre alguno de los siguientes casos:
   - La aptitud del mejor individuo de la población es cero.
   - La aptitud del mejor individuo de la población no cambia después de un número fijo de generaciones (valor predeterminado 1000).
   - Se alcanza el número máximo de generaciones permitidas (valor predeterminado: 5000).
10. **Evaluación.** Para terminar, se construye un árbol con el método CART sobre todos los datos de entrenamiento y se evalúa su tasa de clasificación errónea empleando los datos de prueba. Se compara este error con el obtenido en el algoritmo de inducción global.

5.1.2.2. Resultados

Se utilizó la base de datos *Data Banknote Authentication* disponible en *UCI Machine Learning Repository* (Blake et al. (1998)). Los parámetros empleados fueron los predeterminados salvo para los siguientes rubros: profundidad máxima: 5 y mínimo de datos permitidos en cada hoja: 10.

En la Tabla 5.2 se reportan 4 de los resultados obtenidos tras correr el algoritmo y en la Figuras 5.3 se grafican algunas las convergencias del algoritmo evolutivo que se describen en el índice 2 y 4 de la tabla.

Al observar la Figura 5.2 se puede apreciar la competitividad del algoritmo de inducción global frente al método CART tradicional. Se debe resaltar que la implementación utilizada aquí dista de la sugerida en Krętowski and Czajkowski (2010); ya que hace falta implementar algunas medidas de optimización. La más notoria es la de diseñar un método para podar el árbol. Con esta deficiencia, la implementación aquí propuesta arroja árboles más grandes que el método CART tradicional, y es posible que muchos de los nodos se encuentren vacíos, no aporten ninguna ventaja

Índice	Entrenamiento AG	Prueba AG	Prueba CART
1	1.664	2.919	3.649
2	1.977	3.406	3.163
3	1.873	1.946	2.433
4	1.560	1.703	3.163
Promedio	1.803	2.351	2.919

Tabla 5.2: En la primer columna se muestra el índice, la segunda corresponde a la tasa de clasificación errónea del mejor árbol del algoritmo de inducción global evaluado con los datos de entrenamiento (Entrenamiento AG), en la tercera columna se muestra el error obtenido al evaluar el árbol con los datos de prueba (Prueba AG); finalmente la cuarta columna expresa la evaluación que se obtiene con los datos de prueba sobre el árbol generado con el método CART (Prueba CART).

al árbol de inducción global o, más aún, sobre-ajusten los datos. A pesar de esto, los resultados obtenidos en el método genético son competitivos y en muchas ocasiones mejores a los obtenidos con el método CART tradicional; por esta razón se puede concluir que la implementación aquí expuesta cumple el objetivo de explorar este método y probar su efectividad.

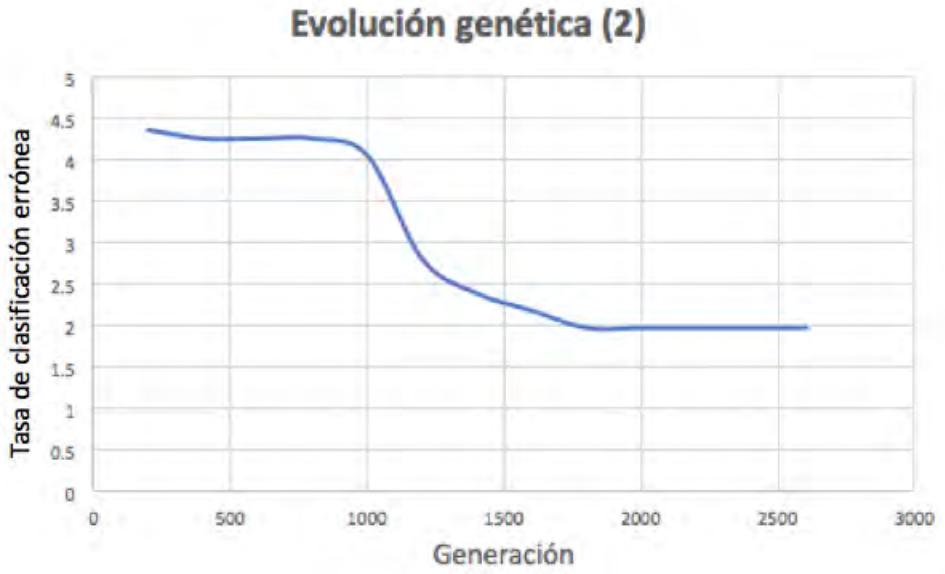
El metodo que se expone en Krętowski and Czajkowski (2010) es más riguroso en muchos detalles, además de que es comparado con un algoritmo CART en donde, a diferencia de la presente implementación, sí se incluye una optimización por podado. Por estas razones es natural suponer que dicho algoritmo es mejor que la implementación aquí expuesta. Como trabajo futuro se propone implementar una versión más cercana a la propuesta en Krętowski and Czajkowski (2010).

5.2. Un problema de decisión secuencial

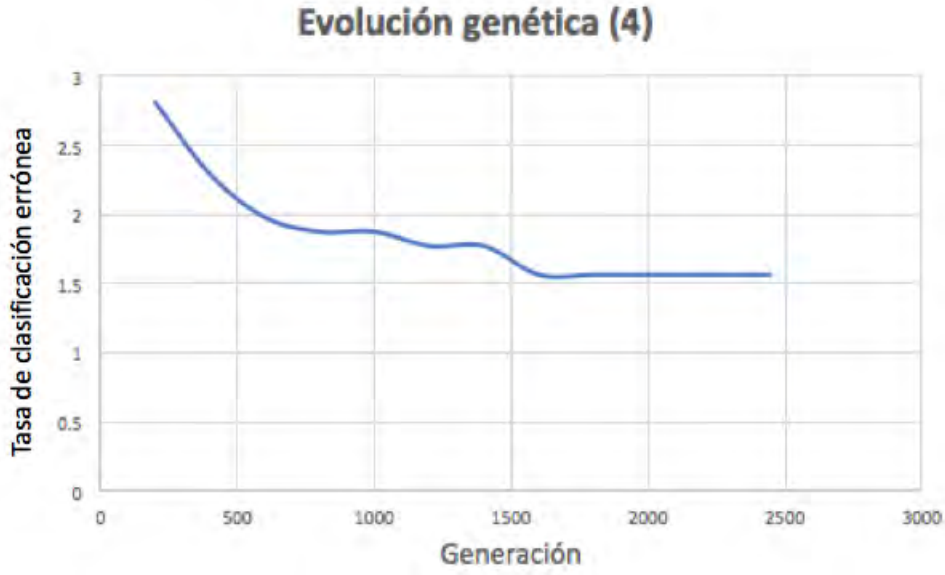
5.2.1. El problema del apostador

En Sutton and Barto (2017) se describe siguiente problema:

Un apostador tiene la oportunidad de hacer apuestas sobre los resultados de una



(a) Algoritmo evolutivo 2. Véase la Tabla 5.2, índice 2.



(b) Algoritmo evolutivo 4. Véase la Tabla 5.2, índice 4.

Figura 5.3: Convergencia del mejor individuo a través de las generaciones en las implementaciones 2 y 4 del algoritmo genético de inducción global (véase la Tabla 5.2).

secuencia de volados. Si en la moneda sale águila, gana la cantidad de pesos que haya apostado; y si sale sol, pierde el dinero apostado. El juego termina cuando el apostador haya ganado el premio máximo de $5 pesos, o se quede en ruina tras perder todo su dinero. En cada volado, el apostador debe tomar la decisión de qué porción de su capital apostar, en un número entero de pesos. El problema de encontrar una estrategia óptima puede formularse como un PDM sin descuento, en donde:

- Los estados son el capital del apostador, $s \in \mathcal{S} = \{1, 2, 3, 4\}$.
- Las acciones son el dinero que se pone en juego, $a \in \mathcal{A} = \{0, 1, \ldots, \text{mín}(s, 5 - s)\}$.
- La recompensa es cero en todas las transiciones, excepto en las que el apostador se lleva el premio máximo, donde la recompensa es 1, es decir

$$\mathrm{R}(s, a) = \begin{cases} 1 & \text{si } s + a = 5 \\ 0 & \text{en cualquier otro caso.} \end{cases} \tag{5.2}$$

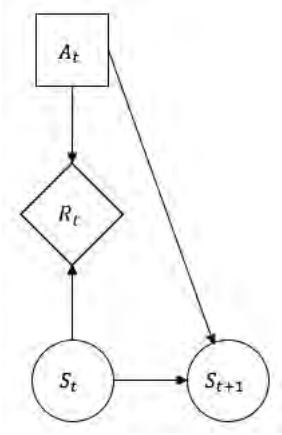


Figura 5.4: Problema del apostador.

El PDM se puede ilustrar con la gráfica de la Figura 5.4. Se observa que se trata de un PDM estacionario con espacio de estados finito. Si suponemos que sale águila con probabilidad p_a entonces, las funciones de transición están dadas por

$$\mathrm{T}(s'|s, a) = \begin{cases} p_a & \text{si } s' = s + a \\ 1 - p_a & \text{si } s' = s - a \\ 0 & \text{en cualquier otro caso.} \end{cases} \tag{5.3}$$

Para encontrar la política óptima, utilizaremos la técnica de *iteración de valor*, por lo que calcularemos la función de valor óptimo U_n; de acuerdo con (4.20) se tiene que $\mathrm{U}_0(s) = 0$ para cualquier estado s y que

$$\mathrm{U}_n(s) = \max_a \left(\mathrm{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathrm{T}(s'|s,a)\mathrm{U}_{n-1}(s') \right). \tag{5.4}$$

Calculando

$$\begin{aligned}
\mathrm{U}_1(1) &= \max_a \mathrm{R}(1,a) = 0 \\
\mathrm{U}_1(2) &= \max_a \mathrm{R}(2,a) = 0 \\
\mathrm{U}_1(3) &= \max_a \mathrm{R}(3,a) = \mathrm{R}(3,2) = 1 \\
\mathrm{U}_1(4) &= \max_a \mathrm{R}(4,a) = \mathrm{R}(4,1) = 1
\end{aligned}$$

Para determinar una política a través de una función de valor U_n se utiliza la siguiente ecuación:

$$\pi_n(s) = \arg\max_a \left(\mathrm{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathrm{T}(s'|s,a)\mathrm{U}_{n-1}(s') \right) \tag{5.5}$$

Por lo tanto, la política (que no es única) determinada por U_1 podría ser $\pi_1(1) = 0, \pi_1(2) = 1, \pi_1(3) = 2, \pi_1(4) = 1$. Obsérvese que también se pudo elegir $\pi_1(1) = 1, \pi_1(2) = 2$ para la misma función de valor. Para calcular U_2, observemos que de (5.3) y de (5.4), basta con calcular

$$\mathrm{U}_2(s) = \max_a \left(\mathrm{R}(s,a) + \mathrm{T}(s+a|s,a)\mathrm{U}_1(s+a) + \mathrm{T}(s-a|s,a)\mathrm{U}_1(s-a) \right), \tag{5.6}$$

siempre que $s + a, s - a \in \mathcal{S}$. Por lo que

$$\mathrm{U}_2(1) = \max \left\{ \begin{array}{ll} \mathrm{R}(1,0) + p_a\mathrm{U}_1(1) + (1-p_a)\mathrm{U}_1(1) & = 0 \\ \mathrm{R}(1,1) + p_a\mathrm{U}_1(2) & = 0 \end{array} \right\} = 0$$

$$\mathrm{U}_2(2) = \max \left\{ \begin{array}{ll} \mathrm{R}(2,0) + p_a\mathrm{U}_1(2) + (1-p_a)\mathrm{U}_1(2) & = 0 \\ \mathrm{R}(2,1) + p_a\mathrm{U}_1(3) + (1-p_a)\mathrm{U}_1(1) & = p_a \\ \mathrm{R}(2,2) + p_a\mathrm{U}_1(4) & = p_a \end{array} \right\} = p_a$$

$$\mathrm{U}_2(3) = \max \left\{ \begin{array}{ll} \mathrm{R}(3,0) + p_a\mathrm{U}_1(3) + (1-p_a)\mathrm{U}_1(3) & = p_a + (1-p_a) = 1 \\ \mathrm{R}(3,1) + p_a\mathrm{U}_1(4) + (1-p_a)\mathrm{U}_1(2) & = p_a \\ \mathrm{R}(3,2) + (1-p_a)\mathrm{U}_1(1) & = 1 \end{array} \right\} = 1$$

$$\mathrm{U}_2(4) = \max \left\{ \begin{array}{ll} \mathrm{R}(4,0) + p_a\mathrm{U}_1(4) + (1-p_a)\mathrm{U}_1(4) & = p_a + (1-p_a) = 1 \\ \mathrm{R}(4,1) + (1-p_a)\mathrm{U}_1(3) & = 1 + (1-p_a) \end{array} \right\} = 2 - p_a$$

Una política π_2 obtenida a partir de la función de valor óptimo U_2 puede ser $\pi_2(1) = 0, \pi_2(2) = 1, \pi_2(3) = 0, \pi_2(4) = 1$. Obsérvese que esta política tampoco es única. Calculemos la siguiente iteración teniendo en cuenta la ecuación (5.6).

$$
\begin{aligned}
\mathrm{U}_3(1) &= \text{máx} \left\{ \begin{array}{ll} \mathrm{R}(1,0) + p_a \mathrm{U}_2(1) + (1-p_a)\mathrm{U}_2(1) & = 0, \\ \mathrm{R}(1,1) + p_a \mathrm{U}_2(2) & = p_a^2 \end{array} \right\} \quad = p_a^2 \\
\mathrm{U}_3(2) &= \text{máx} \left\{ \begin{array}{ll} \mathrm{R}(2,0) + p_a \mathrm{U}_2(2) + (1-p_a)\mathrm{U}_2(2) & = p_a^2 + (1-p_a)p_a = p_a, \\ \mathrm{R}(2,1) + p_a \mathrm{U}_2(3) + (1-p_a)\mathrm{U}_2(1) & = p_a, \\ \mathrm{R}(2,2) + p_a \mathrm{U}_2(4) & = p_a(2-p_a) = 2p_a - p_a^2 \end{array} \right\} \\
&= 2p_a - p_a^2 \\
\mathrm{U}_3(3) &= \text{máx} \left\{ \begin{array}{ll} \mathrm{R}(3,0) + p_a \mathrm{U}_2(3) + (1-p_a)\mathrm{U}_2(3) & = p_a + (1-p_a) = 1, \\ \mathrm{R}(3,1) + p_a \mathrm{U}_2(4) + (1-p_a)\mathrm{U}_2(2) & = p_a(2-p_a) + (1-p_a)p_a \\ & = 3p_a - 2p_a^2, \\ \mathrm{R}(3,2) + (1-p_a)\mathrm{U}_2(1) & = 1 \end{array} \right\} \\
&= 1 \\
\mathrm{U}_3(4) &= \text{máx} \left\{ \begin{array}{ll} \mathrm{R}(4,0) + p_a \mathrm{U}_2(4) + (1-p_a)\mathrm{U}_2(4) & = p_a(2-p_a) + (1-p_a)(2-p_a) \\ & = 2 - p_a, \\ \mathrm{R}(4,1) + (1-p_a)\mathrm{U}_2(3) & = 1 + (1-p_a) = 2 - p_a \end{array} \right\} \\
&= 2 - p_a
\end{aligned}
\tag{5.7}
$$

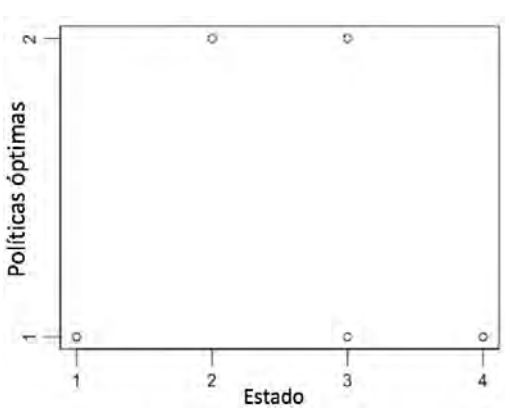
Una política π_3 puede ser $\pi_3(1) = 1, \pi_3(2) = 2, \pi_3(3) = 0, \pi_3(4) = 0$. Obsérvese que esta política no es única (en general la política óptima no es única). A partir de la siguiente iteración, conocer el máximo depende del valor que tome p_a. En efecto,

$$
\begin{aligned}
\mathrm{U}_4(1) &= \text{máx}\left\{\begin{array}{ll} \mathrm{R}(1,0) + p_a\mathrm{U}_3(1) + (1-p_a)\mathrm{U}_3(1) & = p_a^2, \\ \mathrm{R}(1,1) + p_a\mathrm{U}_3(2) & = p_a(2p_a - p_a^2) \end{array}\right\} \\
&= 2p_a^2 - p_a^3 \\
\mathrm{U}_4(2) &= \text{máx}\left\{\begin{array}{ll} \mathrm{R}(2,0) + p_a\mathrm{U}_3(2) + (1-p_a)\mathrm{U}_3(2) & = 2p_a - p_a^2, \\ \mathrm{R}(2,1) + p_a\mathrm{U}_3(3) + (1-p_a)\mathrm{U}_3(1) & = p_a + (1-p_a)p_a^2 = -p_a^3 + p_a^2 + p_a, \\ \mathrm{R}(2,2) + p_a\mathrm{U}_3(4) & = p_a(2-p_a) = 2p_a - p_a^2 \end{array}\right\} \\
\mathrm{U}_4(3) &= \text{máx}\left\{\begin{array}{ll} \mathrm{R}(3,0) + p_a\mathrm{U}_3(3) + (1-p_a)\mathrm{U}_3(3) & = 1, \\ \mathrm{R}(3,1) + p_a\mathrm{U}_3(4) + (1-p_a)\mathrm{U}_3(2) & = p_a(2-p_a) + (1-p_a)(2p_a - p_a^2) \\ & = 4p_a - 4p_a^2 + p_a^3, \\ \mathrm{R}(3,2) + (1-p_a)\mathrm{U}_3(1) & = 1 + (1-p_a)p_a^2 \end{array}\right\} \\
\mathrm{U}_4(4) &= \text{máx}\left\{\begin{array}{ll} \mathrm{R}(4,0) + p_a\mathrm{U}_3(4) + (1-p_a)\mathrm{U}_3(4) & = 2 - p_a \\ \mathrm{R}(4,1) + (1-p_a)\mathrm{U}_3(3) & = 1 + (1-p_a) = 2 - p_a \end{array}\right\} \\
&= 2 - p_a.
\end{aligned}
\tag{5.8}
$$

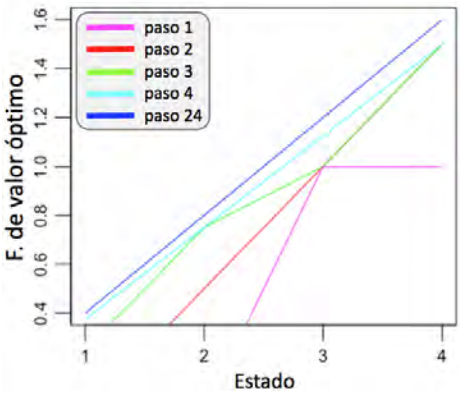
Para conocer $\mathrm{U}_4(2)$ y $\mathrm{U}_4(3)$ es necesario determinar el valor de p_a. Para dar continuidad a este ejemplo, se realizó un programa que calcula el valor de U_n con $p_a = 0.5$. El programa íntegro se encuentra disponible en la dirección `https://github.com/BrunoRGutierrez/Problema_del_apostador/blob/master/codigo` y un resumen del código puede revisarse en la Sección 5.2.1.1. Los resultados de las políticas óptimas para U_4 con $p_a = 0.5$ se encuentran en la Figura 5.5a. La gráfica muestra dos posibles políticas π_4, las cuales sólo se diferencian por el valor que toma $\pi_4(3) \in \{1, 2\}$, ya que $\pi_4(1) = 1, \pi_4(2) = 1$ y $\pi_4(4) = 1$ en cualquiera de las dos políticas.

En la Figura 5.5b se aprecian las gráficas de las funciones de valor de $\mathrm{U}_1, \mathrm{U}_2, \mathrm{U}_3, \mathrm{U}_4$ y U_{24}. Las iteraciones mayores a U_{24} no se grafican porque se asemejan bastante a ésta. En ese sentido podemos decir que U_{24} se aproxima mucho a su valor de convergencia. Dado que las gráficas de las funciones de valor de U_4 y U_{24} son hasta cierto punto parecidas, no es de sorprenderse que sus respectivas gráficas de políticas óptimas sean la misma. En efecto, para $p_a = 0.5$ la gráfica de las políticas óptimas de U_i para $i \geq 4$ es igual a la Figura 5.5a.

Las siguientes imágenes (Figura 5.6) representan las políticas óptimas y las funciones de valor óptimo cuando p_a toma los valores $0.4, 0.3, 0.2$ y 0.1. Es curioso notar que en este caso la política óptima es única y es la misma para cualquiera de los valores de p_a mencionados. La diferencia radica en la velocidad de convergencia de



(a) Políticas óptimas a partir de U_4.



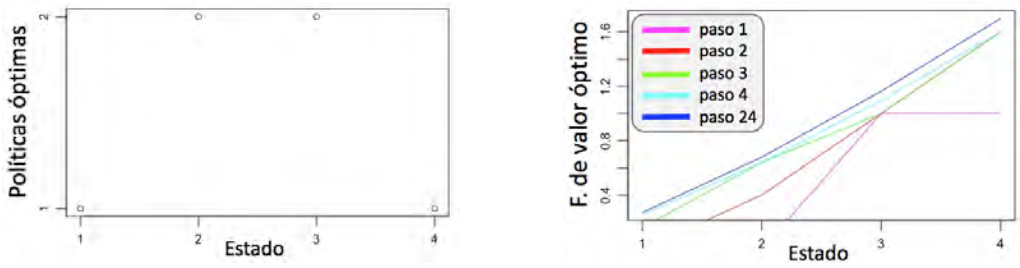
(b) Funciones de valor óptimo y convergencia en el paso U_{24}.

Figura 5.5: Para $p_a = 0.5$ las políticas óptimas U_4 y U_{24} son iguales aunque sus funciones de valor cambian.

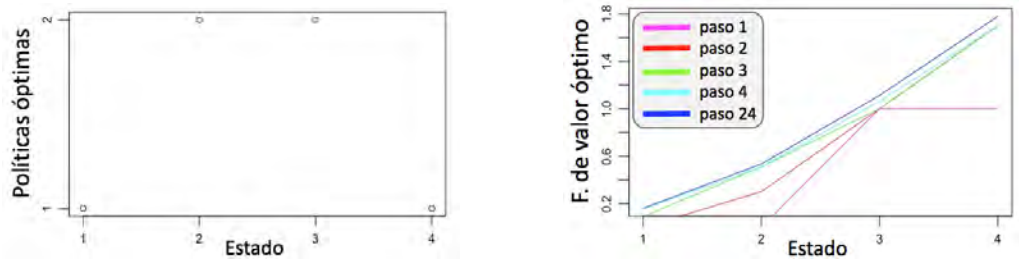
las funciones de valor.

Si observamos el comportamiento cuando p_a se acerca a cero, la política óptima sigue sin cambiar; y la velocidad de convergencia de las funciones de valor es muy rápida, tanto que a partir de la segunda iteración casi no se percibe un cambio en dichas funciones (véase la Figura 5.7). En los casos en que la probabilidad es mayor a cero pero muy pequeña, (Figuras 5.7a, 5.7b y 5.7c) la función de valor para los estado 1 y 2 es muy cercana a cero pero nunca es cero. Por ejemplo, para $p_a = 10^{-10}$ se tiene que la utilidad óptima $U^*(1) = 2 \times 10^{-20}$ y $U^*(2) = 2 \times 10^{-10}$. Por otro lado, para los estados 2 y 3 se tiene que $U^*(3)$ se aproxima mucho a 1 por arriba y $U^*(4)$ se aproxima mucho a 2 por abajo. Por ejemplo, para el caso $p_a = 0.01$ se tienen $U^*(3) = 1.001$ y $U^*(4) = 1.99$, pero para el caso $p_a = 10^{-10}$ se tienen $U^*(3) = 1$ y $U^*(4) = 2$. Con estos resultados en las funciones de valor y retomando los cálculos de las ecuaciones (5.8) se puede hacer una idea de por qué el programa concluye que las políticas óptimas son $\pi^*(1) = 1, \pi^*(2) = 2, \pi^*(3) = 2$ y $\pi^*(4) = 1$, aún cuando la situación es muy desfavorable para el apostador.

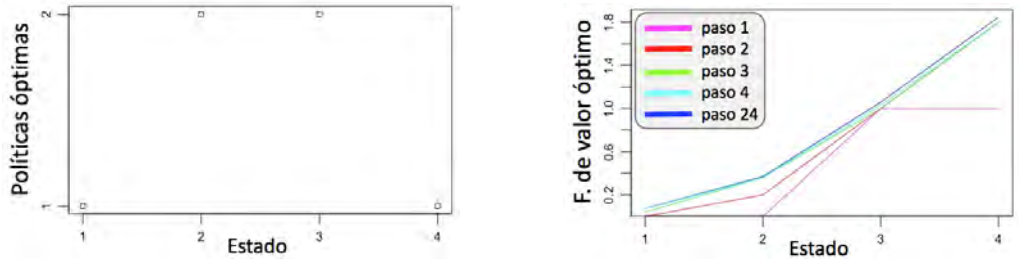
En el caso extremo con $p_a = 0$ (Figura 5.7d) la política óptima cambia con respecto a las anteriores ya que la función de valor alcanza las cantidades $U^*(1) = 0, U^*(2) = 0, U^*(3) = 1$ y $U^*(4) = 2$, lo que abre una gama nueva de opciones para $\pi^*(1)$ y $\pi^*(2)$. Para leer los valores de π^* cuando hay pocas posibilidades de ganar,



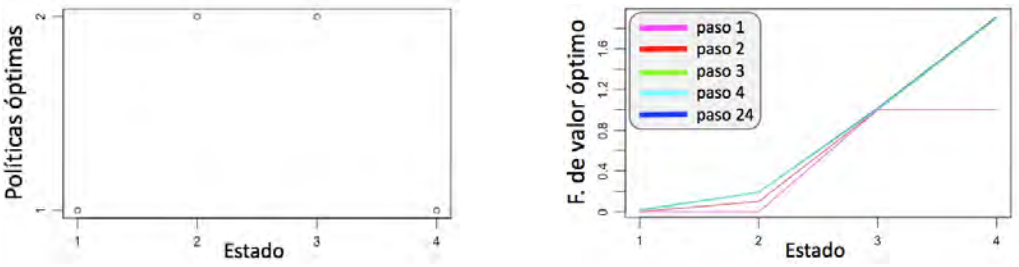
(a) $p_a = 0.4$.



(b) $p_a = 0.3$.

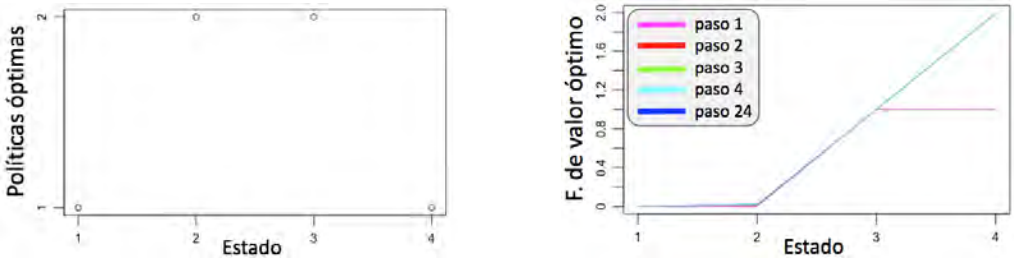


(c) $p_a = 0.2$.

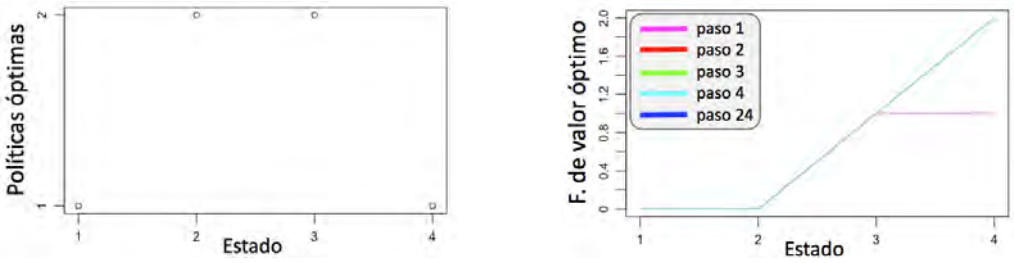


(d) $p_a = 0.1$.

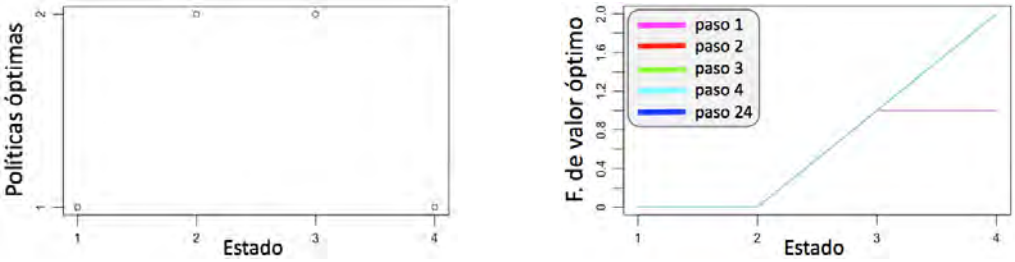
Figura 5.6: Funciones de valor y políticas óptimas cuando el premio es de $5 pesos.



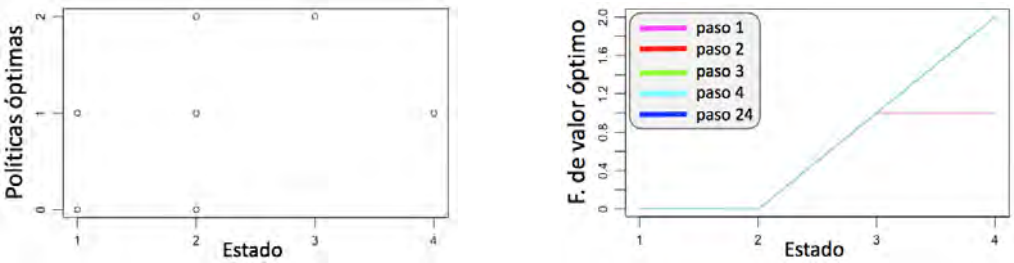
(a) $p_a = 0.01$.



(b) $p_a = 10^{-5}$.



(c) $p_a = 10^{-10}$.



(d) $p_a = 0$.

Figura 5.7: Funciones de valor y políticas óptimas cuando el premio es de $5 pesos.

expresados en general en las laminas de las Figuras 5.6 y 5.7, hay que tener en cuenta que el algoritmo alcanza una recompensa de 1 sólo en el caso llegar al premio máximo de \$5 pesos, y una recompensa de cero en todos los demás casos; es decir, para el algoritmo es equivalente irse a la ruina que abandonar el juego apostando \$0 pesos en algún turno.

Ahora, se analiza lo que ocurre cuando se tiene una probabilidad más alta de obtener la recompensa máxima; en efecto para los casos $p_a = 0.6, p_a = 7, p_a = 8$ y $p_a = 0.9$ véase la Figura 5.8 y para $p_a = 0.99999$ y $p_a = 1$ véase la Figura 5.9. Es sencillo entender la gráfica de las políticas óptimas cuando $p_a = 1$ si se observar la ecuación (5.7), ya que la función de valor óptimo converge en U_3.

Siguiendo con el ejemplo, a continuación se muestran las gráficas (Figura 5.10) del valor óptimo y la política óptima del problema del apostador cuando el premio máximo es de \$25 pesos y $p_a = 0.4$. Finalmente, se muestran (Figura 5.11) las gráficas de las estrategias optimas para ganar en el problema del apostador, cuando el premio máximo es de \$100 pesos y $p_a = 0.4$.

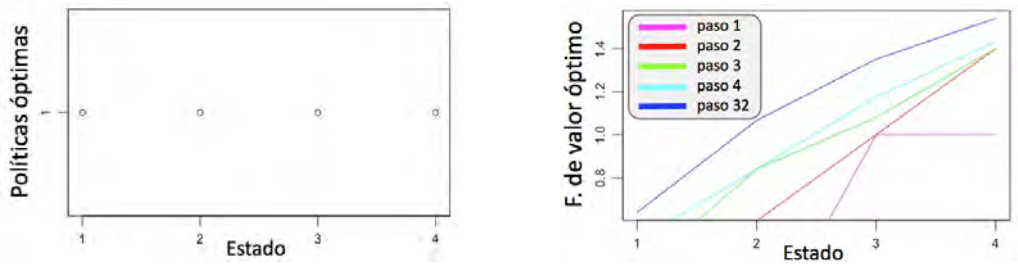
5.2.1.1. Código

La siguiente es una versión resumida del código que resuelve el problema del apostador. Está programado en el lenguaje R y puede encontrarse completo en la siguiente dirección: `https://github.com/BrunoRGutierrez/Problema_del_apostador/blob/master/codigo`.

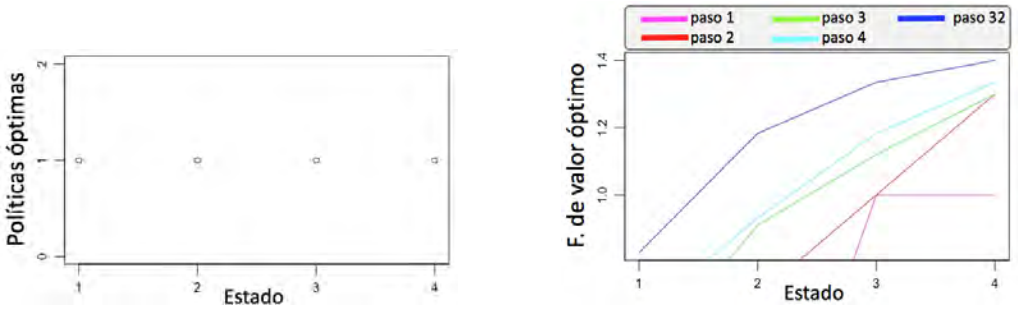
En la primera parte del código, se designa el premio, la probabilidad de que salga águila (p_a) y el número de iteraciones que se van a calcular de la función de valor.

```
# Se llenan los  siguientes  datos:
premio <- 100
# Probabilidad de obtener águila.
p <- 0.4
# Número de iteraciones de funciones de valor.
N <- 32
```

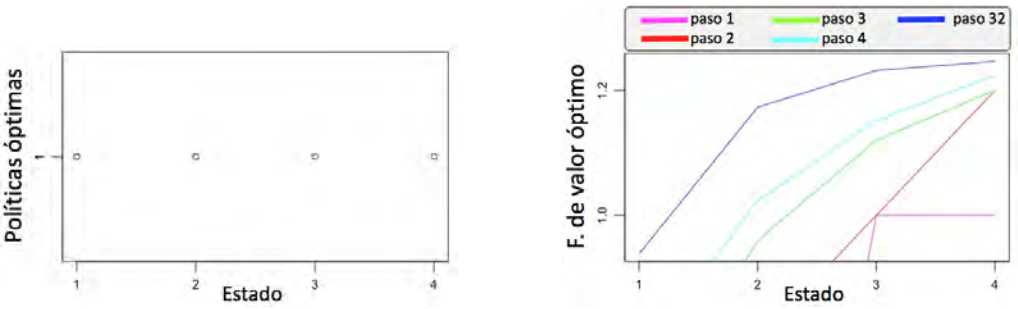
En la segunda parte, se calcula una o varias políticas óptimas para el problema del apostador empleando programación dinámica; específicamente el algoritmo de *iteración de valor* (Sección 4.5.2.4).



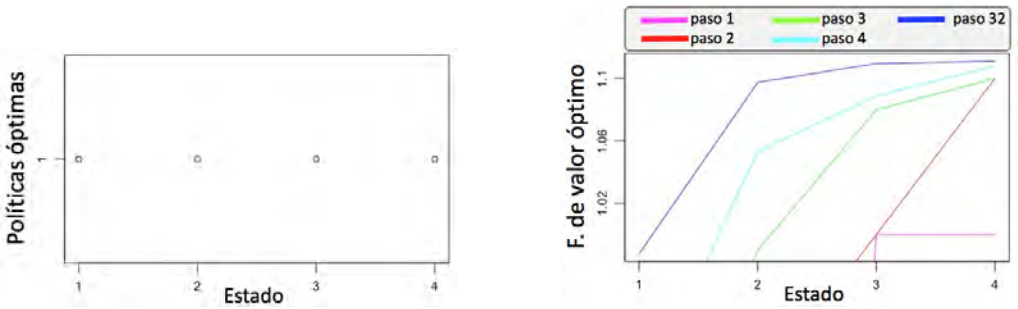
(a) $p_a = 0.6$.



(b) $p_a = 0.7$.

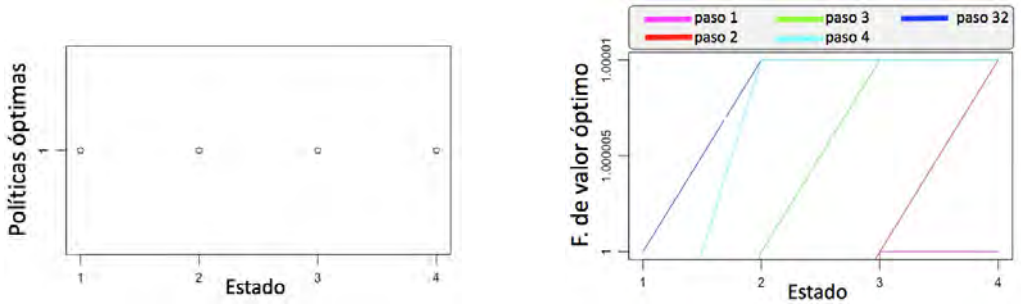


(c) $p_a = 0.8$.

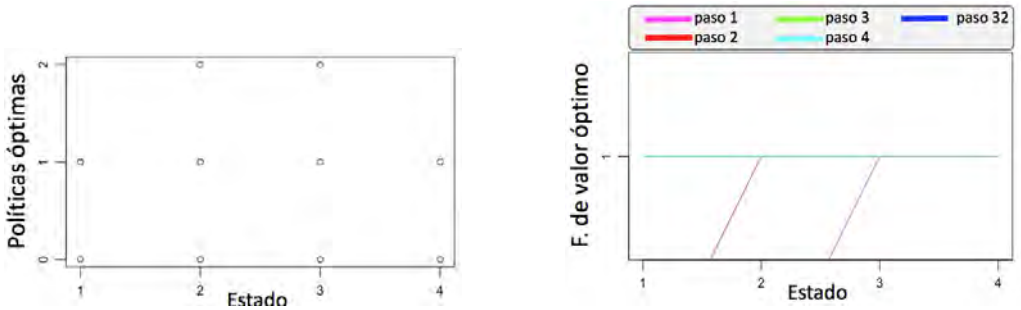


(d) $p_a = 0.9$.

Figura 5.8: Funciones de valor y políticas óptimas cuando el premio es de \$5 pesos.

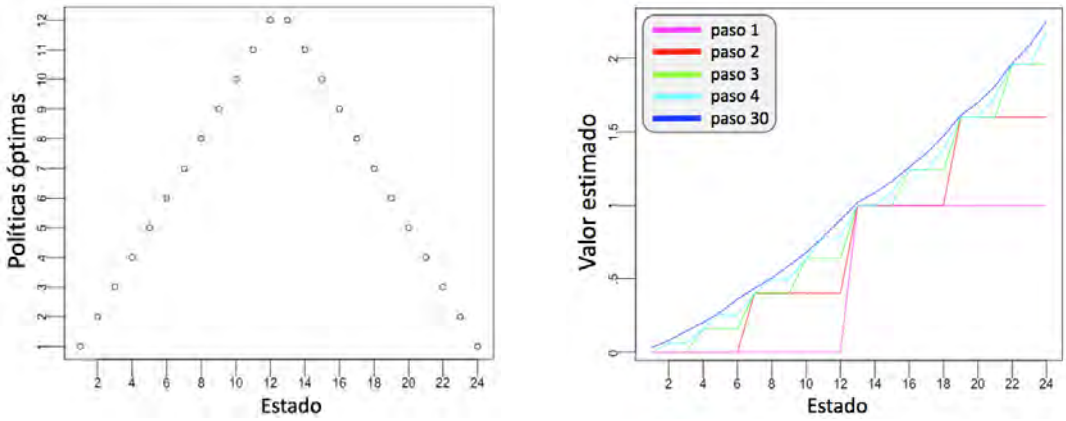


(a) $p_a = 0.99999$.



(b) $p_a = 1$.

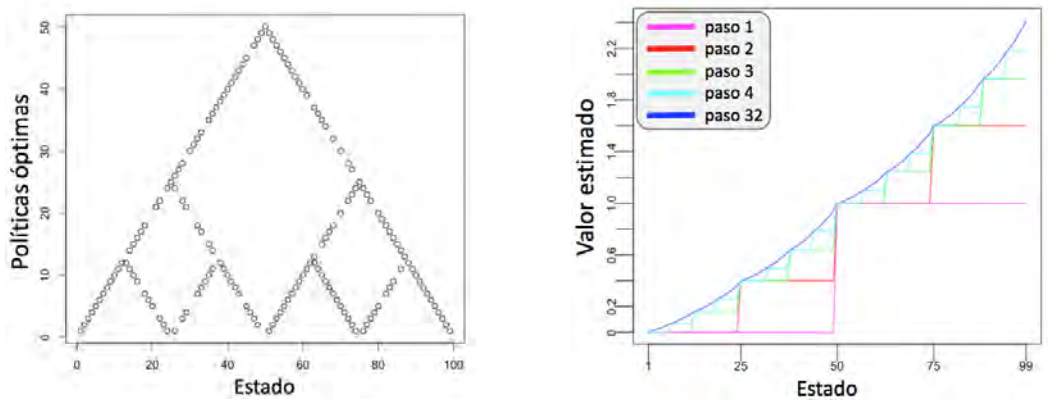
Figura 5.9: Funciones de valor y políticas óptimas cuando el premio es de \$5 pesos.



(a) Valor óptimo.

(b) Política óptima.

Figura 5.10: Premio máximo de \$25 pesos con $p_a = 0.4$.



(a) Valor óptimo. (b) Política óptima.

Figura 5.11: Premio máximo de $100 pesos con $p_a = 0.4$.

```
# número máximo de estados
maximo_st <- premio-1

# Función recompensa
R <- function(s,a){
  if(s+a == premio ){
    recompensa <- 1
  } else {
    recompensa <- 0
  }
  return(recompensa)
}

# Probabilidad de transición del estado 'w' dado el estado 's' y la acción 'a'.
transicion <- function(w,s,a){
  if(w == s+a){
    proba <- p
  } else {
    if(w == s-a){
      proba <- 1-p
    } else {
      proba <- 0
    }
```

```
  }
  return(proba)
}

# Cálculo de las N funciones de valor.
for(n in 1:N){
  # Se toma la función de valor de la iteración anterior
  U <- U2
  # Para cada estado 's' se calcula la nueva función de valor.
  for(s in 1:maximo_st){
    # Con 's' fijo, delimitamos lo máximo posible a apostar.
    apuesta_maxima <- min(s,premio-s)

    # Se calculan las utilidades esperadas para cada acción 'a'.
    for(a in 0:apuesta_maxima){
      utilidades[a] <- R(s,a) + suma sobre w (transicion(w,s,a) * U[w])
    }

    # Se selecciona la máxima utilidad esperada. Esa será la utilidad del estado
        's'
    U2[s] <- max(utilidades)

    # Se encuentra la política óptima, es decir, la acción o acciones 'a' que
        alcanzan la máxima utilidad esperada
    politicas_optimas[[s]] <- which(utilidades == max(utilidades))

  }
}
```

5.2.2. Búsqueda directa de la política óptima empleando AG

En la sección anterior se resolvió el problema del apostador empleando programación dinámica; sin embargo, la Sección 4.5.4 propone que este problema de decisión secuencial también puede solucionarse a través de buscar directamente en el espacio de posibles políticas, en particular, a través de los algoritmos genéticos (AG); véase el Capítulo 2. Esta sección se encarga de encontrar una solución al problema del apostador mediante una búsqueda directa en el espacio de posibles políticas con la

técnica de AG.

El planteamiento del problema es el mismo que se explica en la Sección 5.2.1, y queda resumido en el diagrama de decisión dinámica de la Figura 5.12.

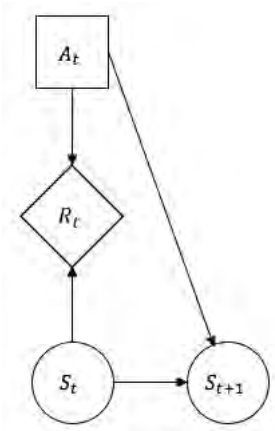


Figura 5.12: Problema del apostador.

Supongamos que se quiere resolver el problema del apostador para el caso en que el premio máximo es de $\$N$ pesos. Entonces se tienen los estados $\mathcal{S} = \{1, 2, \ldots, N-1\}$ y para cada estado $s \in \mathcal{S}$ las posibles acciones son $\mathcal{A}_s = \{0, 1, \ldots, \text{mín}(s, N-s)\}$. Así, el número de posibles políticas cuando N es impar son $\lfloor \frac{N}{2} + 1 \rfloor ! \lfloor \frac{N}{2} + 1 \rfloor !$, y si N es par hay $(\frac{N}{2} + 1)![\frac{N}{2}!]$ políticas. Por ejemplo, para $N = 100$ existen 4.717587×10^{130} posibles políticas; evaluar cada una hasta encontrar el máximo sería una tarea muy costosa. Por esta razón se emplea un método de búsqueda basado en algoritmos genéticos (AG).

El reto de emplear AG radica en ajustar los parámetros del algoritmo hasta obtener resultados satisfactorios. En términos generales, se obtuvieron buenos resultados con las siguientes especificaciones:

1. *Cromosoma.* Se define al cromosoma como una cadena de tamaño $N-1$ en donde el caracter en la s-ésima posición es un número dentro del conjunto $\{0, 1, \ldots, \text{mín}(s, N-s)\}$.

2. *Población inicial.* Dado que el tamaño del espacio de políticas es demasiado grande, se genera una población inicial aleatoria de tamaño $5N$.

3. *Aptitud.* Para medir la aptitud de cada cromosoma, se genera una función aleatoria a través de la simulación de 100 caminatas aleatorias sobre cada estado de la política dada. El resultado de la función será el número de veces que se obtiene la recompensa de 1 (momento en que se alcanza el premio máximo), entre el total de iteraciones ($100(N-1)$). Aunque el resultado de la función de aptitud es aleatorio, la Ley Débil de los Grandes Números asegura que el resultado es aproximadamente igual para todas las veces que se calcule.

4. *Selección.* Se escogen dos padres de manera aleatoria. Para favorecer valores pequeños se generan dos números aleatorios entre 0 y 1 y se multiplican entre sí. Este número aleatorio se multiplica por $5N$, el tamaño de la población. La parte entera de la cantidad resultante indica la posición del padre elegido dentro de una lista ordenada. Obsérvese que dicho número tiene mayor probabilidad de ser un entero pequeño, es decir, de seleccionar a los padres dentro de los individuos mejor valuados.

5. *Cruza.* Se combinan las tiras genéticas de ambos padres mediante la cruza de punto único. El punto de cruce es la mitad de la cadena.

6. *Mutación.* La mutación se da con probabilidad 2/3. Se escoge un gen al azar del nuevo individuo, con probabilidad de 1/3 muta a la siguiente acción dentro del conjunto $\mathcal{A}$, y con 1/3 muta a la acción anterior.

7. *Inserción.* Si el nuevo individuo es mejor valuado que el peor elemento de la población, entonces el primero remplaza al segundo. De otro modo, el nuevo elemento se desecha.

8. *Fin.* La búsqueda finaliza cuando el cromosoma mejor valuado de la población obtiene una calificación de aptitud igual a 1, o hasta trascurrir mil generaciones sin obtener ninguna mejora.

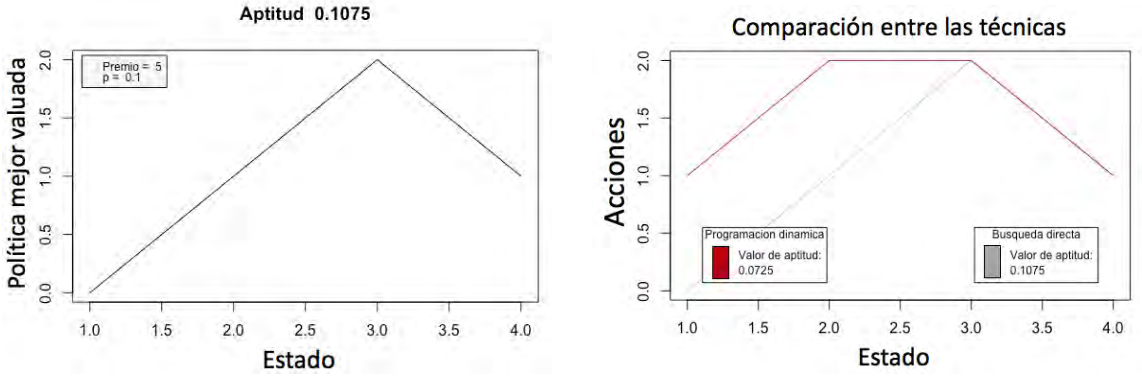
Un resumen del algoritmo se encuentra en la Sección 5.2.2.1 y puede consultarse completo en la dirección `https://github.com/BrunoRGutierrez/Problema_del_apostador/blob/master/codigo_ga`.

Como punto de referencia para comparar la efectividad de este algoritmo, para p_a y N fijos, se mide la aptitud de una de las políticas óptimas obtenidas con el algoritmo de programación dinámica (Sección 5.2.1.1) y se compara con los resultados del algoritmo que busca directamente en el espacio de políticas (Sección 5.2.2.1). En las

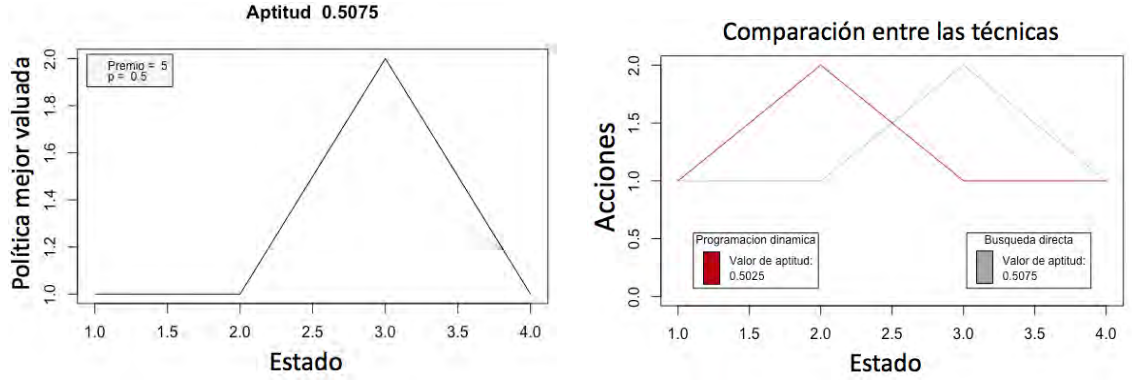
Figuras 5.13, 5.14 y 5.15 se presentan las soluciones de hacer una búsqueda directa dentro del espacio de políticas con la técnica de AG (en la columna izquierda) y su respectiva comparación con lo que arroja el algoritmo basado en programación dinámica (columna derecha). En particular se analizan los casos en que el premio máximo es de $N = 5$ y p_a toma valores en $\{0.1, 0.5, 0.6, 1\}$, $N = 25$ con $p_a \in \{0.4, 0.5\}$; y cuando $N = 100$ con $p_a \in \{0.4, 0.5, 0.6\}$.

Al hacer la comparación entre los resultados de ambos algoritmos (Figuras 5.13, 5.14 y 5.15 columna derecha), se encuentran en la mayoría de los casos valores muy parecidos, y si se tiene presente que la función de aptitud es aleatoria, se puede concluir que en esos casos ambas políticas son aceptables. Sin embargo, llaman la atención tres situaciones en las que las aptitudes difieren bastante. La primera corresponde a la Figura 5.13c, en la que el algoritmo genético arroja una política con valor de aptitud de 0.64 mientras que la política generada con programación dinámica llega a 0.97. La otra es el caso extremo de la Figura 5.14a en donde $p_a = 1$; si se regresa a la la Figura 5.9b de la sección anterior, puede verse que una política óptima que arroja el algoritmo de programación dinámica es $\pi^*(j) = 0$ para todos los estado $j = 1, \ldots, 4$, cuya aptitud es igual a cero. Por otro lado, la política obtenida con AG tiene aptitud de 1. Sin embargo, podríamos justificar la falta de precisión del primer algoritmo ya que se trata de un caso extremo; y más aún, las otras políticas óptimas que se exponen en la Figura 5.9b también alcanzan una aptitud igual a 1. Por último, hay que resaltar el caso de la Figura 5.15c; en la que se tiene una puntuación de 0.95 para la programación dinámica y de 0.80 para los AG; aunque la diferencia entre aptitudes no es tan grande, el tiempo para llegar a ambos resultados fue significativamente distinto: el AG tardó unas 100 veces más que el tiempo empleado por el algoritmo basado en programación dinámica.

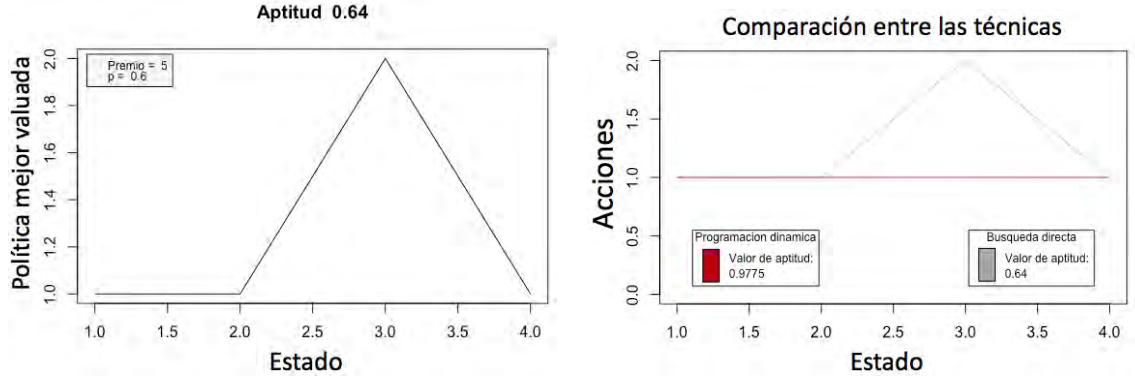
Tras analizar estos ejemplos, se puede concluir que en términos generales las técnicas de programación dinámica superan a las de AG en las instancias analizadas del problema del apostador, ya que su eficiencia es mejor en cuanto a resultados y costo. Esta conclusión alimenta la creciente popularidad por utilizar programación dinámica para resolver problemas de decisión secuenciales. Sin embargo, en problemas más complejos o de mayores dimensiones, es posible que las técnicas de programación dinámica no sean viables, y es donde los AG pueden tomar ventaja.



(a) Premio= \$5, $p_a = 0.1$.

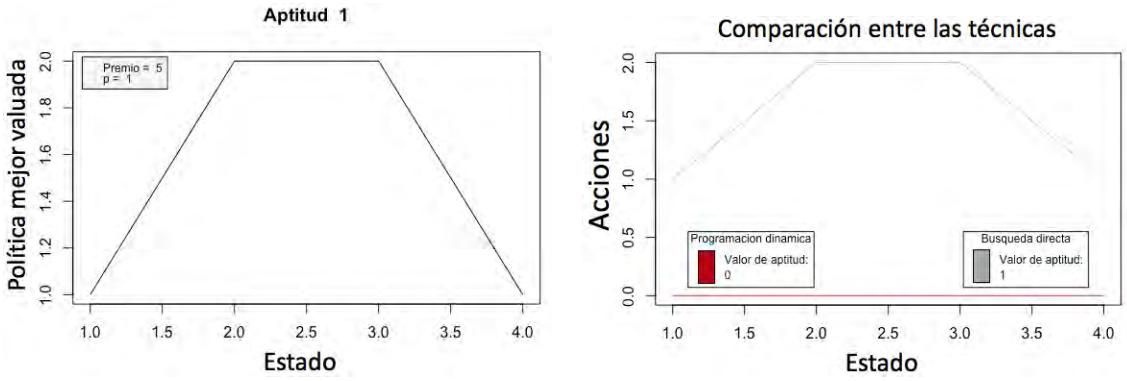


(b) Premio= \$5, $p_a = 0.5$.

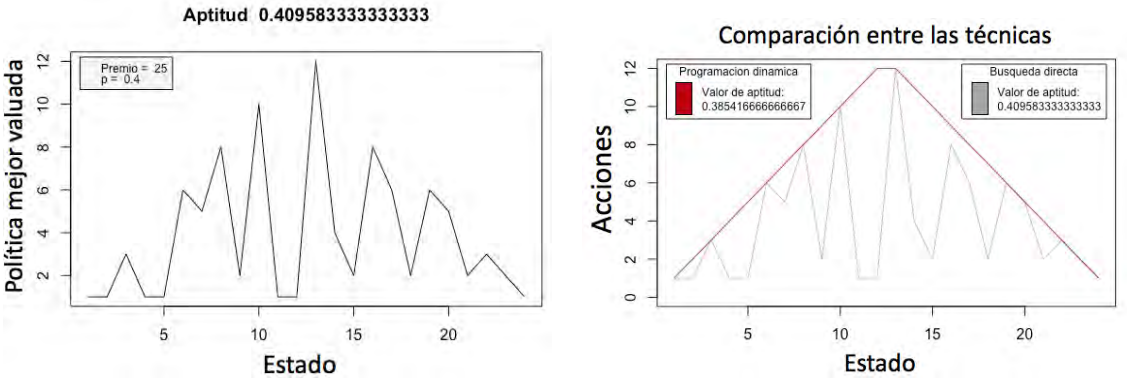


(c) Premio= \$5, $p_a = 0.6$.

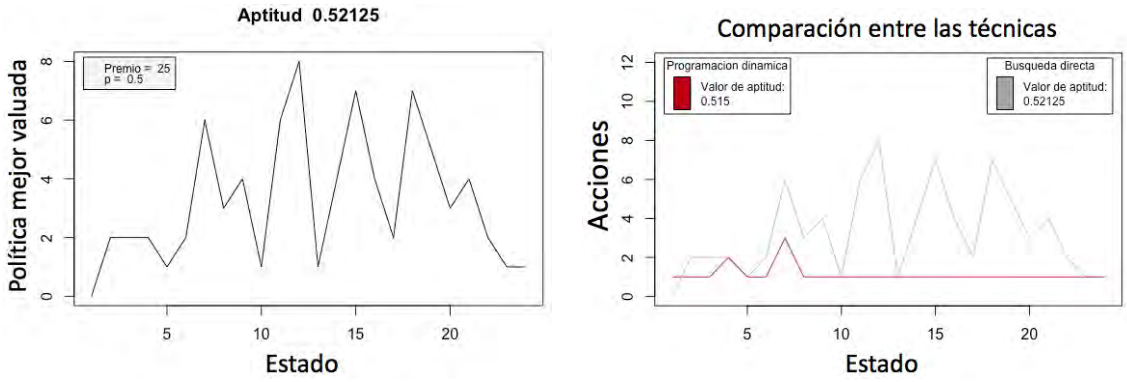
Figura 5.13: Política mejor valuada empleando GA (izq.) y comparación entre técnicas (der.).



(a) Premio= \$5, $p_a = 1$.

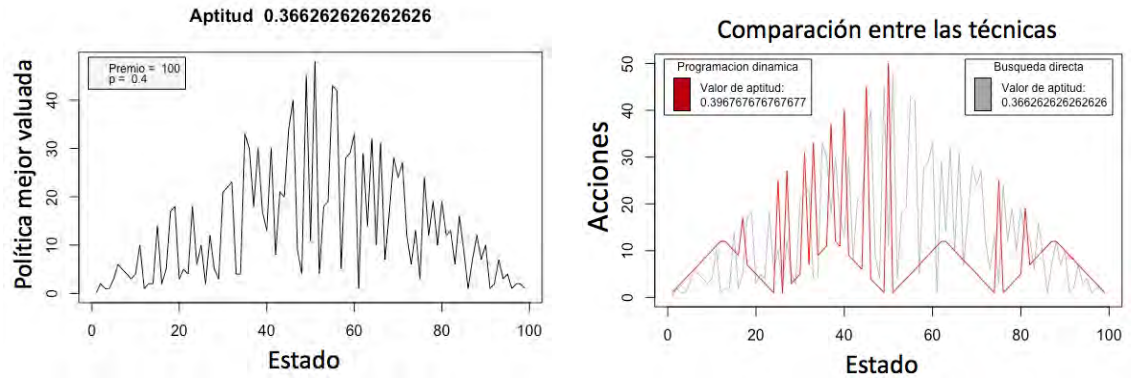


(b) Premio= \$25, $p_a = 0.4$.

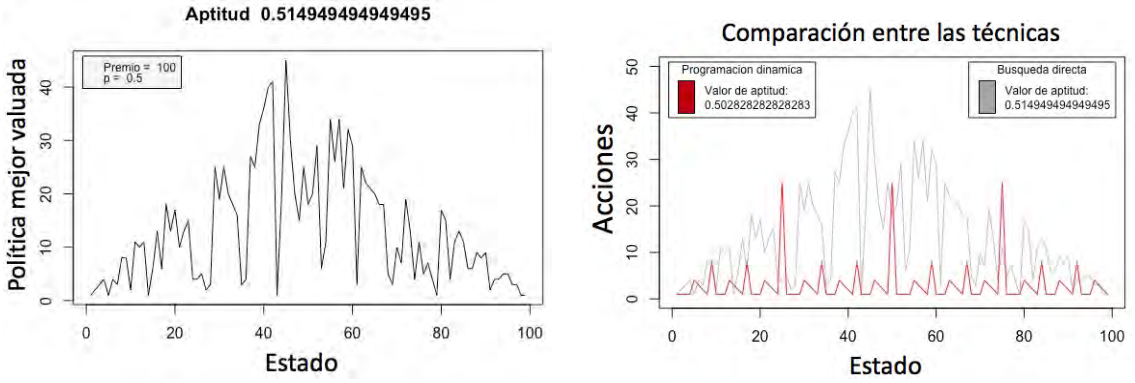


(c) Premio= \$25, $p_a = 0.5$.

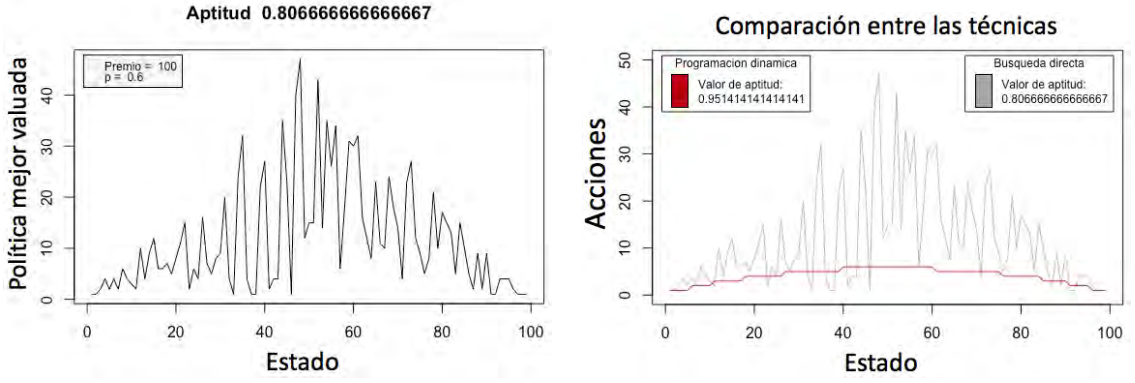
Figura 5.14: Política mejor valuada empleando GA (izq.) y comparación entre técnicas (der.).



(a) Premio= \$100, $p_a = 0.4$.



(b) Premio= \$100, $p_a = 0.5$.



(c) Premio= \$100, $p_a = 0.6$.

Figura 5.15: Política mejor valuada empleando GA (izq.) y comparación entre técnicas (der.).

5.2.2.1. Código de una búsqueda directa en la política

El siguiente es un resumen del código empleado para resolver el problema del apostador haciendo una búsqueda directa dentro del espacio de posibles políticas empleando el método de algoritmos genéticos. El código completo se encuentra en la siguiente dirección de Internet: `https://github.com/BrunoRGutierrez/Problema_del_apostador/blob/master/codigo_ga`.

En la primera parte se designa el premio y la probabilidad de obtener águila.

```
premio <- 100
p <- 0.4
```

Se calcula el tamaño de la población inicial y la recompensa

```
# Se determina el número de la población inicial
poblacion <- 5 * premio

# Se define el número maximo de estados
maximo_st <- premio-1

# Se define la funcion recompensa
Rew <- function(s){
   1 si se alcanza la recompensa
   0 en otro caso
}
```

Se crea la función de aptitud (*fitness*) a través de simular 100(N-1) caminatas aleatorias sobre la política dada.

```
# Fitness function

fitness_function <- function(cromosoma){
  # Se irá sumando la recompensa acumulada
  recompensa <- 0

  # Se inicia una c.a. en el estado 's' apostando lo que indica la política 'a'.
  for(s in 1:maximo_st){
    # Se repite 100 veces la c.a
    for(k in 1:100){
```

```
      ss <- s
      while(ss < premio){
        # La c.a. termina cuando se alcanza el premio, llega a 0, o la política
            es apostar 0.
        if (ss == 0) break
        if (cromosoma[ss] == 0) break

        moneda <- rbinom(1,1,p)
        if(moneda == 1){
          ss <- ss + cromosoma[ss] # Si sale águila (==1), avanzamos al
              estado s+a
        } else {
          ss <- ss - cromosoma[ss] # Si sale sol (==0), retrocedemos al
              estado s-a
        }
        recompensa <- recompensa + Rew(ss)
      }
    }
  }
  # La aptitud es la recompensa obtenida entre las iteraciones realizadas
  fitness_value <- recompensa / (100 * maximo_st)
  return(fitness_value)
}
```

Ahora se crea la población inicial de individuos en el espacio de posibles políticas. Se califica a cada una y ambos valores se guardan en el *genpool*.

```
genpool_ga <- vector("list", poblacion)
for(i in 1:poblacion){
  # Creamos cada cromosoma dentro del espacio de políticas
  cromosoma <- vector("numeric", maximo_st)

  for(s in 1:maximo_st){
    # Para cada estado s, delimitamos lo maximo posible a apostar.
    apuesta_maxima <- min(s,premio-s)
    # Se escoge la acción 'a' aleatoriamente de entre las posibles
    a <- sample(0:apuesta_maxima, 1, replace=TRUE)
    cromosoma[s] <- a
  }
```

```
  # Se califica cada cromosoma con la función fitness
  valor <- fitness_function(cromosoma)
  # Se agregan al 'genpool_ga'.
  genpool_ga[[i]] <- list(cromosoma, valor)
}
```

En la siguiente parte se escogen 2 padres de manera aleatoria dentro de los mejor valuados.

```
random_padre <- function(genpool_ga){
  # Se obtiene un numero aleatorio.
  aleatorio = runif(1) * runif(1) * (poblacion - 1)
  aleatorio = 1 + floor(aleatorio)
  return(genpool_ga[[aleatorio ]][[1]])
}
```

La función 'cruza' combina la mitad de los genes de cada padre para formar un nuevo individuo.

```
cruza <- function(padre1, padre2){
  hijo <- vector("numeric", maximo_st)
  # en la primera mitad se copiarán los genes del padre1
  for(j in 1: division){
    hijo[j] <- padre1[j]
  }
  # en la segunda mitad los del padre2
  for(k in (division +1): maximo_st){
    hijo[k] <- padre2[k]
  }
  return(hijo)
}
```

Se define la función 'mutación'.

```
# Se escoge un gen (un estado 's') y dada la acción 'a' se muta con
    probabilidad 1/3 a la acción 'a+1' y con 1/3 a la acción 'a-1'.
mutacion <- function(cromosoma){
  # Elegimos de manera aleatoria la posición del gen.
  gen <- sample(1:maximo_st, 1)
```

```
  # Se muta el gen con el número aleatorio
  alelo <- cromosoma[gen] + sample(c(-1,0,1),1, replace = TRUE)
  cromosoma[gen] <- alelo
  return(cromosoma)}
```

Por último se hace un ciclo *while* en donde se lleva a cabo todo el proceso genético.

```
l <- 1
while(TRUE){
  mejor_aptitud_anterior <- genpool_ga[[1]][[2]]
  #1. Ordenamos la población de mayor a menor, con respecto a su fitness.
  genpool_ga <- genpool_ga[order(sapply(genpool_ga,'[[',2), decreasing =
      TRUE)]
  l <- l +1

  # 2. Si el mejor elemento (primer individuo de la lista) tiene fitness == 1 o
      si transcurren mil generaciones sin mejora se termina el ciclo.
  if(genpool_ga [[1]][[2]] == 1){break}
  if(cont_repeticiones == 1000){break}

  #3. Selección de los padres
  padre1 <- random_padre(genpool_ga)
  padre2 <- random_padre(genpool_ga)

  # 4. Cruza
  hijo <- cruza(padre1, padre2)

  # 5. Mutación
  hijo <- mutacion(hijo)

  # 6. Valuación del nuevo individuo
  hijo_evaluado <- list(hijo, fitness_function(hijo))

  # 7. Se inserta al hijo en la 'genpool_ga' sólo si es mejor que alguien de la
      población.
  if(hijo_evaluado[[2]] > genpool_ga[[poblacion]][[2]]) {
    genpool_ga[[poblacion]] <- hijo_evaluado } }
```

Capítulo 6

Conclusiones

Los algoritmos aquí presentados son, en general, muy robustos pero los códigos de sus implementaciones carecen de la eficiencia computacional de un buen algoritmo (llegar a un buen resultado con el mínimo de pasos requeridos). Esto se debe, en parte, a la nula formación que recibí en los lenguajes de programación como *Python* y *R* durante mis estudios como matemático. Así mismo, realizar el primer código que se expone en esta tesis me tomó un tiempo muy superior al promedio. Considero que tener fluidez en estos lenguajes de programación es una necesidad en la mayoría de los estadísticos actuales, por lo que es importante promover más y nuevos cursos enfocados a desarrollar estas habilidades.

Hablando de los algoritmos genéticos, se estudió (Capítulo 2) que una de sus desventajas es que el ajuste de los parámetros se hace de manera heurística e intuitiva, por lo que no hay un indicador que señale las condiciones bajo las cuales se obtiene el ajuste ideal. Aunque se sospecha que esto pudiera llegar a ser un problema en la práctica, en las tres implementaciones aquí hechas este factor no representó mayores complicaciones.

En el Capítulo 4 se explicó qué es el aprendizaje reforzado; una teoría que hoy en día tiene un crecimiento muy prometedor, en gran medida por los resultados alcanzados al ofrecer soluciones a problemas de control que antes carecían de una respuesta eficiente. Aunque esta teoría no pertenece en su totalidad al área de la estadística, en esta tesis se abordó su estudio desde sus fundamentos en la teoría de la decisión, dándole especial peso a todo lo relacionado con los métodos bayesianos. De esta forma se puede apreciar la relación tan estrecha que existe entre la estadística bayesiana y el aprendizaje reforzado. Un ejemplo de los resultados que ofrece esta teoría se muestra

en la Sección 4.7, donde se controla con éxito un humanoide dentro del ambiente virtual MuJoCo.

Por otro lado, la implementación en el Capítulo 5 de un algoritmo genético para el análisis de árboles de decisión resultó ser una tarea fascinante, pero a la vez muy ardua, dado los pocos conocimientos que el autor de este trabajo tenía sobre lenguajes de programación *Python* y *R*. Aunque hay un fundamento teórico que sustenta la eficiencia de los algoritmos genéticos, resulta sorprendente, también para el que escribe esta tesis, ver cómo converge una determinada generación de árboles de decisión a un óptimo mejor que el propuesto en un principio por el método CART. Se debe confesar que existían dudas de que esto fuera a ocurrir, ya que la implementación aquí realizada fue más sencilla que la propuesta originalmente en el artículo Krętowski and Czajkowski (2010). Sin embargo, esto nos deja concluir que la eficiencia de los algoritmos genéticos puede ser tan grande que incluso una implementación simple llega a arrojar buenos resultados.

La mayoría de los temas aquí expuestos fueron nuevos para mí, exceptuando las primeras tres secciones del Capítulo 3. Eso incrementó la dificultad del presente trabajo, pues requirió de mucha investigación y de una asimilación rápida de nuevos conceptos. La propuesta natural en este punto, es la creación y apertura de cursos en licenciatura o maestría que aborden los tópicos aquí estudiados; los cursos deberían estar disponibles para completar la formación de matemáticos y actuarios, lo cual es totalmente justificable dada la estrecha relación de estos temas con la estadística y la alta demanda actual de personas con una sólida formación en estas áreas.

Para el estadístico convencional no debe ser difícil abordar estos temas, ya que en el fondo tiene la formación para entenderlos y desarrollarlos; sin embargo, sería de gran ayuda una guía que a través de la cual pueda darse cuenta que se tratan de dos áreas muy relacionadas entre sí. La propuesta es realizar un curso o un diplomado de *Machine Learning* enfocado a dar estas herramientas a los estadísticos.

Por último, como trabajo futuro derivado de la investigación en esta tesis, se pretende estudiar los problemas de aprendizaje reforzado en los que el espacio de posibles acciones es muy grande. Aunque ya se han obtenido algunos resultados importantes en este problema, ninguno se ha podido implementar con éxito en ambientes físicos reales (sólo en ambientes simulados). Un posible enfoque para resolver este problema es el expuesto en Mania et al. (2018), ya que los métodos libres de modelo pueden ser atractivos para abordar problemas con espacios demasiado grandes. Proponer un

algoritmo eficiente que resuelva este problema, significaría la solución u optimización de muchos procesos presentes en la industria o en el área de la inteligencia artificial.

Apéndice A

Ideas básicas de complejidad computacional

En el siguiente apéndice se explica de manera breve qué son los problemas P, NP, NP-completos y NP-*hard*. La información aquí recopilada fue obtenida de Kleinberg (2006); al lector interesado en ahondar en el estudio de estos problemas se le recomienda remitirse a la fuente original.

A.1. Definición de NP y Certificación Eficiente

En las ciencias de la computación es importante caracterizar los problemas con los que estamos trabajando. Los datos (entradas) con los que trabajamos en los problemas computacionales están codificados en una cadena finita binaria s que recibe el nombre de *string* (o simplemente *cadena*) y su tamaño se denota por $|s|$. Sea X un problema de decisión, vamos identificar a X con el conjunto de *strings* para los que su respuesta es "sí". Sea A un algoritmo que recibe a s como entrada y cuya salida $A(s)$ es "sí" o "no". Decimos que A soluciona el problema X si para todo *string* s, se tiene que

$$A(s) = \text{“sí”} \quad \textit{si y sólo si} \quad s \in X.$$

Decimos que el algoritmo A corre en tiempo polinomial si existe un polinomio $p(\cdot)$ tal que para toda entrada s, $A(s)$ puede ser calculado en a lo más $\mathcal{O}(p(|s|))$ pasos. Ahora bien, denotamos por P al conjunto de todos los problemas X para los cuales existe un algoritmo A que corre en tiempo polinomial y que soluciona a X. En ese sentido, decimos que P es el conjunto de problemas de decisión que pueden

resolverse en tiempo polinomial.

A.1.1. Certificado eficiente

Vamos a formalizar la idea de que la solución a un problema puede ser comprobada eficientemente, sin tomar en cuenta si éste se pudo o no resolver de manera eficiente. Un "algoritmo de comprobación" para un problema X, tiene una estructura diferente que un algoritmo que busca resolver el problema. Para un algoritmo de comprobación se requiere la cadena de entrada s, así como una cadena t que es un "certificado" con la evidencia de que s es una instancia "sí" de X.

Decimos que B es un *certificado eficiente* para un problema X si se cumplen las siguientes propiedades:

- B es un algoritmo que corre en tiempo polinomial con dos entradas: s y t.

- Existe una función polinomial $g(\cdot)$ tal que, para toda cadena s, se tiene que $s \in X$ si y sólo si existe una cadena t tal que $|t| \leq g(|s|)$ y $B(s,t) =$ "sí".

Lo que hace un certificado eficiente, es evaluar de manera eficiente si $s \in X$ a través de "pruebas" (siempre y cuando no sean demasiado largas).

A.1.2. Problemas del tipo NP

Al conjunto de todos los problemas para los cuales existe un certificado eficiente lo denotamos por NP. De manera inmediata se observa que

$$\mathrm{P} \subseteq \mathrm{NP}$$

Demostración. Sea $X \in$ P, entonces existe un algoritmo A que resuelve a X en tiempo polinomial. Para que $X \in$ NP, necesitamos mostrar que existe un certificado eficiente B para X.

Para diseñar B lo único que hacemos es darle el valor de A, es decir, $B(s,t) = A(s)$. De este modo, B simplemente ignora el valor de la prueba propuesta t y resuelve el problema. Veamos que B es un certificado eficiente; i) corre en tiempo polinomial y, ii) si $s \in X$, entonces para todo t de tamaño a lo más $p(|s|)$ se tiene que $B(s,t) =$

"sí"; y por otro lado, para $s \notin X$ y para toda t de tamaño a lo más $p(|s|)$ se tiene que $B(s,t) =$ "no". □

Hasta ahora no ha sido posible demostrar que existe un problema NP que requiera más tiempo que el polinomial para resolverse. Es decir, no se ha podido demostrar que exista un problema X en NP tal que X no pertenezca a P. Por lo que queda la pregunta abierta

$$\text{¿Es P} = \text{NP?}$$

La creencia general apunta a que P $\neq$ NP, pero no existe suficiente evidencia técnica de que esto sea así. Esta conjetura se basa más en la idea de que sería realmente sorprendente que P = NP. ¿Cómo podría haber una transformación general que nos lleve de la tarea de *comprobar* una solución, a la mucho más difícil tarea de *encontrar* la solución? ¿Cómo podría existir un método general para diseñar algoritmos eficientes, suficientemente poderosos para manejar todos estos problemas difíciles sin que nadie los haya descubierto aún? Añadiendo a lo anterior, una gran cantidad de esfuerzos han fracasado en el intento de diseñar algoritmos para problemas NP que corran en tiempo polinomial; y la explicación más natural de estas persistentes fallas es que simplemente no pueden ser resueltos en tiempo polinomial.

A.2. Problemas NP-completos

Como no se ha encontrado una respuesta a la pregunta de si P = NP, se ha recurrido a una pregunta relacionada pero más accesible: ¿Cuáles son los problemas más difíciles en NP? Responder esta pregunta ha permitido conocer ciertas propiedades importantes de la estructura de los problemas NP que salen de los alcances de este apéndice.

Para comparar la dificultad de un problema con respecto a otro, se utiliza la siguiente analogía: supongamos que X y Y son problemas y que existe una "caja negra" capaz de resolver X. Decimos que X es al menos tan difícil como Y si con la misma "caja negra" somos capaces de resolver Y. Para expresar esto de manera un poco más precisa, agregamos la hipóteis de que X puede ser resuelto en tiempo polinomial utilizando directamente un modelo computacional y que existe una "caja negra" tal que para cualquier instancia de X que ingresemos a ella, ésta arrojará la respuesta correcta en un solo paso. Nos preguntamos:

¿Es posible resolver instancias arbitrarias del problema Y utilizando un número polinomial de pasos estándares, más un número polinomial de llamadas a la "caja

negra" que soluciona a X?

Si la respuesta es sí, entonces se escribe $Y \leq_p X$ y se lee "Y es *reducible* a X en tiempo polinomial," o "X es al menos tan difícil como Y (con respecto al tiempo polinomial)". Observemos que si $Y \leq_p X$ y si X puede ser resuelto en tiempo polinomial, entonces Y puede ser resuelto en tiempo polinomial.

Ahora bien, la manera más natural de definir que un problema en NP es de los más "difíciles", es empleando las siguientes propiedades:

- $X \in$ NP
- Para todo $Y \in$ NP, $Y \leq_p X$.

Es decir, se requiere que todo problema en NP pueda ser reducido a X. A cada una de estas X la llamaremos problema *NP-Completo.* Para este tipo de problemas, se cree que no existe una solución eficiente.

A.3. Problemas NP-*hard*

Utilizamos el término *NP-hard* para referirnos a los problemas que son al menos tan difíciles como los NP-completos. Generalmente se evita llamar NP-completos a los problemas de optimización, pues este término técnicamente aplica sólo para problemas de decisión.

La siguiente figura ilustra a los problemas P, NP, NP-completos y NP-*hard*, tanto en el caso en que P $\neq$ NP (izq.) como en el caso en que P = NP (der).

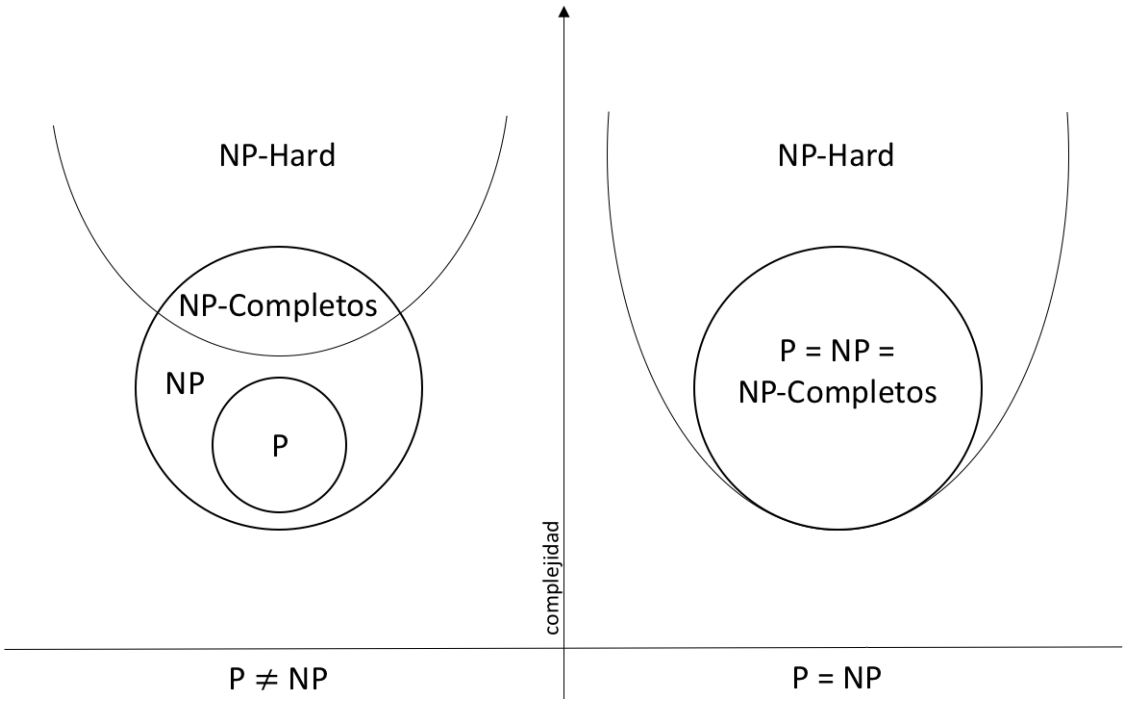


Figura A.1: Problemas P, NP, NP-completos y NP-*hard*.

Bibliografía

Affenzeller, M., Wagner, S., Winkler, S., and Beham, A. (2009). *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Numerical Insights. CRC Press.

Bernardo, J. (1981). *Bioestadística: Una Perspectiva Bayesiana*. VICENS universidad. VCENS-VIVES, S.A.

Bernardo, J. and Smith, A. (2000). *Bayesian Theory*. Wiley Series in Probability and Statistics. Wiley.

Blake, C., Keogh, and E., Merz, C. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Browniee, J. (2016). *How to Implement the Decision Tree Algorithm from Scratch in Python. Blog*. Machine Learning Mastery. `https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/`.

Carvalho, A. M. (2009). Scoring functions for learning bayesian networks. Technical Report 54-2009, INESC-ID. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.157.3919`.

Coello, C. (2016). Introducción a la computación evolutiva, notas de curso. Cinvestav-IPN.

Frank, E., Hall, M. A., and Witten, I. H. (2016). *The WEKA Workbench. Online appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, Fourth edition.

Gutiérrez-Peña, E. (2004). Bayesian classification methods. *Psychology Science*, (46):52–64.

Gutiérrez-Peña, E. (2016). *Estadística Bayesiana: Teoría y Conceptos Básicos*. [Diapositivas de PowerPoint]. XXXI Foro Internacional de Estadística.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York.

Khosrow-Pour, D. (2017). *Encyclopedia of Information Science and Technology, Fourth Edition*. IGI Global.

Kleinberg, J. (2006). *Algorithm Design*. Pearson/Addison-Wesley, Boston.

Kochenderfer, M. (2015). *Decision Making Under Uncertainty: Theory and Application*. MIT Lincoln Laboratory Series. MIT Press.

Krętowski, M. and Czajkowski, M. (2010). An evolutionary algorithm for global induction of regression trees. In Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L. A., and Zurada, J. M., editors, *Artifical Intelligence and Soft Computing*, pages 157–164, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *CoRR*, abs/1803.07055. `https://dblp.org/rec/bib/journals/corr/abs-1803-07055`.

Reeves, C. (1995). *Modern Heuristic Techniques for Combinatorial Problems*. Advanced topics in computer science series. McGraw-Hill.

Sivanandam, S. and Deepa, S. (2007). *Introduction to Genetic Algorithms*. Springer Berlin Heidelberg.

Sutton, R. and Barto, A. (2017). *Reinforcement Learning: An Introduction*. Second edition, in progress. Complete Draft. MIT Press.