Introducción a la Inteligencia Artificial para Profesionales de la seguridad

por el Equipo de Ciencia de Datos de Cylance

Introducción a la Inteligencia Artificial para Profesionales de la seguridad

por el Equipo de Ciencia de Datos de Cylance.

THE CYLANCE PRESS
IRVINE, CA

© 2017 Equipo de Cylance Data Science

Todos los derechos reservados. Queda prohibida la reproducción, el almacenamiento en un sistema de recuperación o la transmisión en cualquier forma o por cualquier medio electrónico, mecánico, fotocopia, grabación o de otro modo, de cualquier parte de esta publicación sin el permiso previo por escrito del editor.

Publicado por
Equipo de Cylance Data Science.

Introducción a la inteligencia artificial para profesionales de la seguridad / Equipo de Cylance Data Science. – Irvine, CA : The Cylance Press, 2017.

p. ; cm.

Resumen: Presentamos el mundo de la Inteligencia Artificial y el Aprendizaje Automático a profesionales de seguridad informática a través de explicaciones y ejemplos.

ISBN13: 978-0-9980169-0-0

1. Inteligencia Artificial. 2. Seguridad internacional. I. Título.

TA347.A78 C95 2017
006.3—dc23 2017943790

PRIMERA EDICIÓN

Coordinación del proyecto por Jenkins Group, Inc.
www.BookPublishing.com

Diseño interior por Brooke Camfield

Impreso en los Estados Unidos de América
21 20 19 18 17 • 5 4 3 2 1

Contenido

Prólogo

por Stuart McClure

Mi primera exposición a la aplicación de una ciencia a las computadoras fue en la Universidad de Colorado, Boulder, donde, de 1987 a 1991, estudié Psicología, Filosofía y Aplicaciones de Ciencias Informáticas. Como parte del programa de Ciencias Informáticas, estudiamos Estadística y cómo programar una computadora para hacer lo que nosotros, como humanos, queríamos que hicieran. Recuerdo la euforia pura de controlar la máquina con lenguajes de programación: estaba enamorado.

En esas clases de ciencias informáticas conocimos a Alan Turing y al "Test de Turing" por excelencia. La prueba es simple: formule a dos "personas" (una de ellas es una computadora) un conjunto de preguntas escritas y use las respuestas para tomar una determinación. Si la computadora no puede distinguirse del humano, entonces ha "pasado" la prueba. Este concepto me intrigó. ¿Podría una computadora ser tan natural como un humano en sus respuestas, acciones y pensamientos? Siempre pensé, *¿por qué no?*

Avancemos rápido hasta 2010, dos años después de volver a unirme a una compañía de antivirus de primer nivel. Mi trabajo consistía en ayudar a explicar nuestro mapa de ruta y nuestra visión para el futuro. Desafortunadamente, cada conversación fue la misma que había tenido durante más de veinte años: necesitamos acelerar la detección de malware

y ciberataques. Más rápido, seguimos diciendo. Entonces, en lugar de actualizaciones de firmas mensuales, nos esforzaríamos por tener actualizaciones semanales. Y entonces, pasaríamos a fantasear con actualizaciones de firmas diarias. Pero a pesar de los millones de dólares destinados a acelerar la detección, nos dimos cuenta de que esto no es posible. Los malos siempre serán más rápidos. ¿Y si pudiéramos saltarlos? ¿Qué pasaría si realmente pudiéramos predecir qué harían antes de que lo hicieran?

Desde 2004, me han preguntado con frecuencia: "Stuart, ¿qué utilizas en tu computadora para protegerte?" Debido a que pasé gran parte de la década del 2000 como ejecutivo sénior dentro de una compañía mundial de antivirus, la gente siempre esperaba que dijera: "Bueno, por supuesto, uso los productos de la empresa para la que trabajo". Sin embargo, no pude mentir. No usé ninguno de sus productos. ¿Por qué? Porque no confiaba en ellos. Yo era de la vieja escuela. Solo confiaba en mis propias decisiones sobre lo que era malo y bueno. Entonces, cuando finalmente dejé esa compañía, me pregunté a mí mismo: "¿Por qué no podría entrenar una computadora para que piense como yo, como un profesional de seguridad que sabe qué es malo y bueno? En lugar de confiar en los humanos para crear firmas del pasado, ¿no podríamos aprender de nuestras experiencias pasadas lo suficientemente bien como para eliminar la necesidad de firmas, pudiendo eventualmente predecir ataques y prevenirlos en tiempo real?"

Y así nació Cylance.

Mi Director Científico, Ryan Permeh, y yo emprendimos este loco y formidable viaje para usurpar por completo los poderes fácticos y sacudir los cimientos de lo establecido: aplicar las matemáticas y la ciencia en un campo que en gran medida no había logrado adoptarlas de manera significativa. Entonces,

con el destacado y brillante equipo de Cylance Data Science, logramos nuestro objetivo: proteger cada computadora, usuario y cosa bajo el sol con inteligencia artificial capaz de predecir y prevenir ataques cibernéticos.

Por ello, si bien se han escrito numerosos libros sobre inteligencia artificial y aprendizaje automático a lo largo de los años, muy pocos han ofrecido una guía práctica y realista desde una perspectiva puramente de ciberseguridad. Lo que el equipo de Cylance Data Science ofrece en estas páginas es la instrucción práctica, accesible y muy realista de cómo cualquier persona en ciberseguridad puede aplicar el aprendizaje automático a los desafíos a los que se enfrentan a diario: los piratas informáticos o hackers.

Así comienza su viaje y recuerde siempre: confíe en usted mismo y pruebe usted mismo.

FM/EM

Introducción
Inteligencia Artificial: El camino a seguir en la seguridad informática

Las tecnologías de Inteligencia Artificial (IA) se están moviendo rápidamente más allá de los ámbitos de la academia y la ficción especulativa para pasar a ser parte de los estándares comerciales. Productos innovadores como el asistente digital Siri® de Apple y el motor de búsqueda de Google, entre otros, utilizan IA para transformar la manera en que accedemos y aplicamos la información en Internet. Según un informe de diciembre de 2016 de la Oficina del Presidente:

> *Los avances en la tecnología de Inteligencia Artificial (IA) y campos relacionados han abierto nuevos mercados y nuevas oportunidades para el progreso en áreas fundamentales como la salud, la educación, la energía, la inclusión económica, el bienestar social y el medioambiente.*[1]

La inteligencia artificial también se ha vuelto estratégicamente importante para la defensa nacional y para asegurar nuestras infraestructuras críticas de finanzas, energía, inteligencia y comunicaciones contra los ciberataques patrocinados por

1. Oficina Ejecutiva del Presidente, *Inteligencia Artificial, Automatización y Economía*, 20 de diciembre de 2016. Disponible para descargar en https://www.whitehouse.gov/sites/whitehouse.gov/files/images/EMBARGOED%20AI%20Economy%20Report.pdf

el estado. Según un informe de octubre de 2016[2] publicado por el Comité de Tecnología del Consejo Nacional de Ciencia y Tecnología (National Science and Technology Council Committee on Technology, NSTCC) del gobierno federal:

> *La IA tiene aplicaciones importantes en seguridad cibernética, y se espera que desempeñe un papel cada vez más relevante para las acciones cibernéticas defensivas y ofensivas. . . . El uso de la IA puede ayudar a mantener la respuesta rápida requerida para detectar y reaccionar ante amenazas en desarrollo.*

Conforme a estas proyecciones, el NSTCC ha publicado un Plan Estratégico Nacional de Investigación y Desarrollo de Inteligencia Artificial[3] para orientar la investigación y el desarrollo financiados por el gobierno federal.

Como todas las nuevas tecnologías importantes, AI ha ocasionado tanto entusiasmo como recelo entre los expertos de la industria y los medios populares. Leímos sobre las computadoras que superaron a los maestros de ajedrez y Go, sobre la inminente superioridad de los autos sin conductor y sobre las inquietudes de algunos expertos en ética de que las máquinas algún día podrían tomar el control y volver obsoletos a los humanos. Creemos que algunos de estos temores son exagerados y que la Inteligencia Artificial jugará un papel positivo en nuestras vidas, siempre que la investigación y el desarrollo de la IA se rijan por principios éticos sólidos que garanticen que

2. Subcomité del Consejo Nacional de Ciencia y Tecnología sobre Aprendizaje Automático e Inteligencia Artificial, *Preparing for the Future of Artificial Intelligence*, octubre de 2016. Disponible para descargar en https://obamawhitehouse.archives.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf

3. Subcomité del Consejo Nacional de Ciencia y Tecnología sobre Aprendizaje Automático e Inteligencia Artificial, *Plan Estratégico Nacional de Investigación y Desarrollo de la Inteligencia Artificial,* octubre de 2016. Disponible para descargar en https://www.nitrd.gov/PUBS/national_ai_rd_strategic_plan.pdf

los sistemas que creamos ahora y en el futuro sean totalmente transparentes y respondan a los humanos.

Sin embargo, en el corto plazo, creemos que es importante para los profesionales de seguridad obtener una comprensión práctica sobre qué es la IA, qué puede hacer y por qué es cada vez más significativa para nuestras carreras y la forma en que enfrentamos los problemas de seguridad del mundo real. Es esta convicción la que nos motivó a escribir *Introducción a la inteligencia artificial para profesionales de la seguridad*.

Puede aprender más acerca de los enfoques de agrupamiento, clasificación y modelos probabilísticos descritos en este libro de numerosos sitios web, así como de otros métodos, como los modelos generativos y el aprendizaje por refuerzo. Los lectores que tienen inclinaciones técnicas quizás también quieran informarse sobre los principios matemáticos y las operaciones en las que se basan estos métodos. Intencionalmente excluimos dicho material para hacer de este libro un punto de partida adecuado para los lectores que son nuevos en el campo de la IA. Para obtener una lista de materiales complementarios recomendados, visite https://www.cylance.com/intro-to-ai.

Esperamos sinceramente que este libro lo inspire a comenzar un programa continuo de autoaprendizaje que enriquezca sus habilidades, mejore sus perspectivas profesionales y aumente su eficacia en sus funciones actuales y futuras como profesional de la seguridad.

IA: Percepción frente a realidad

El campo de la IA en realidad abarca tres áreas distintas de investigación:

- **Superinteligencia artificial (SIA)** es el tipo popularizado en la ficción especulativa y en películas como *The Matrix*.

El objetivo de la investigación de la SIA es producir computadoras que sean superiores a los humanos en prácticamente todos los sentidos, que posean lo que el autor y analista William Bryk denominó "memoria perfecta y poder analítico ilimitado".[4]

- **Inteligencia artificial general (IAG)** se refiere a una máquina que es tan inteligente como un ser humano e igualmente capaz de resolver la amplia gama de problemas que requieren aprendizaje y razonamiento. Una de las pruebas clásicas de IAG es la capacidad de pasar lo que se conoce como "El Test de Turing"[5] en la cual un evaluador humano lee una conversación basada en texto que ocurre de forma remota entre dos entidades invisibles, una humana y otra, una máquina. Para pasar la prueba, el evaluador no debe poder distinguir entre el sistema de IAG y el humano.

  La mayoría de los expertos están de acuerdo en que estamos a décadas de lograr IAG y algunos sostienen que la SIA puede resultar inalcanzable. De acuerdo con el informe del NSTC de octubre de 2016,[6] "es muy poco probable que las máquinas exhiban inteligencia de aplicación amplia comparable o superior a la de los humanos en los próximos 20 años".

4. William Bryk, Artificial Intelligence: The Coming Revolution, *Harvard Science Review,* otoño de 2015. Disponible para descargar en https://harvardsciencereview.files.wordpress.com/2015/12/hsrfall15invadersanddefenders.pdf

5. A.M. Turing (1950), *Computing Machinery and Intelligence*, Mind, 59, 433-460. Disponible para descargar en http://www.loebner.net/Prizef/TuringArticle.html

6. Subcomité del Consejo Nacional de Ciencia y Tecnología sobre Aprendizaje Automático e Inteligencia Artificial, *Preparing for the Future of Artificial Intelligence*, octubre de 2016. Disponible para descargar en https://obamawhitehouse.archives.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf

- **Inteligencia artificial débil (IAD)** explota la capacidad superior de una computadora para procesar grandes cantidades de datos y detectar patrones y relaciones que de otro modo serían difíciles o imposibles de detectar para un ser humano. Dichos sistemas centrados en datos son capaces de superar a los humanos solo en tareas específicas, como jugar al ajedrez o detectar anomalías en el tráfico de la red que podrían merecer un análisis más exhaustivo por parte de un cazador de amenazas o un equipo forense. Estos son los tipos de enfoques en los que nos centraremos exclusivamente en las páginas a continuación.

El campo de la Inteligencia Artificial abarca una amplia gama de tecnologías destinadas a dotar a las computadoras de capacidades similares a las humanas para el aprendizaje, el razonamiento y la obtención de información útil. En los últimos años, la mayor parte de la investigación y los avances fructíferos provienen de la subdisciplina de IA denominada *Aprendizaje Automático* (*AA*), que se centra en enseñar a las máquinas a aprender aplicando algoritmos a los datos. A menudo, los términos IA y AA se usan indistintamente. En este libro, sin embargo, nos centraremos exclusivamente en los métodos que se clasifican dentro del espacio del Aprendizaje Automático.

No todos los problemas en IA son candidatos para una solución de Aprendizaje Automático. El problema debe ser uno que se pueda resolver con datos; debe existir una cantidad suficiente de datos, y ser adquirible; y los sistemas con potencia de cálculo suficiente deben estar disponibles para realizar el procesamiento necesario dentro de un marco de tiempo razonable. Como veremos, muchos problemas de seguridad interesantes se ajustan muy bien a este perfil.

Aprendizaje Automático en el dominio de seguridad

Para perseguir metas bien definidas que maximizan la productividad, las organizaciones invierten en sus activos de sistemas, información, redes y humanos. En consecuencia, no es práctico ni deseable simplemente cerrar todos los vectores de ataque posibles. Tampoco podemos evitar incursiones centrándonos exclusivamente en el valor o las propiedades de los activos que buscamos proteger. En cambio, debemos considerar el *contexto* en el que se acceden y utilizan estos activos. Con respecto de un ataque a un sitio web, por ejemplo, es el contexto de las conexiones lo que importa, no el hecho de que el atacante esté apuntando a un activo de sitio web particular o un tipo de funcionalidad.

El contexto es crítico en el dominio de seguridad. Afortunadamente, el dominio de seguridad genera grandes cantidades de datos de registros, sensores de redes y agentes de punto final, así como de sistemas de recursos humanos y directorios distribuidos que indican qué actividades del usuario son aceptables y cuáles no. Colectivamente, esta masa de datos puede proporcionar los indicios contextuales que necesitamos para identificar y mejorar las amenazas, pero solo si tenemos herramientas capaces de desentrañarlas. Este es precisamente el tipo de procesamiento en el que el AA sobresale.

Al adquirir una comprensión amplia de la actividad que rodea a los activos bajo su control, los sistemas del AA permiten a los analistas discernir cómo se relacionan los eventos en el tiempo y entre hosts, usuarios y redes dispares. Si se aplica correctamente, el AA puede proporcionar el contexto que necesitamos para reducir los riesgos de una infracción al tiempo que aumenta significativamente el "costo del ataque".

El futuro del aprendizaje Automático

A medida que el AA prolifera en el ámbito de la seguridad, aumentan las exigencias para los atacantes. Cada vez es más difícil penetrar los sistemas en comparación con unos años atrás. En respuesta, es probable que los atacantes adopten técnicas del AA para encontrar nuevos caminos. A su vez, los profesionales de seguridad tendrán que utilizar el AA de manera defensiva para proteger los activos de redes y de información.

El partido que tuvo lugar en marzo de 2016 entre el jugador profesional Lee Sedol, campeón mundial de Go 18 veces, y AlphaGo, un programa de computadora desarrollado en DeepMind, un laboratorio de IA con sede en Londres que ha sido adquirido por Google, nos brinda un indicio de lo que vendrá. En el segundo juego, AlphaGo hizo un movimiento que nadie había visto antes. Los comentaristas y expertos que observaban el partido estaban desconcertados. El propio Sedol estaba tan anonadado que tardó casi quince minutos en responder. AlphaGo resultó ganador de la serie de 5 juegos .

En muchos sentidos, las posturas de seguridad del ataque y la defensa son similares a la competencia en juegos complejos como Go y ajedrez. Si se agrega AA a la fórmula, seguramente surgirán amenazas completamente nuevas e inesperadas. En aproximadamente una década, nos encontraremos con una situación en la cual los "bots de batalla" atacan y defienden las redes casi en tiempo real. Se necesitará contar con AA para la defensa simplemente para mantener la igualdad de condiciones.

Por supuesto, cualquier tecnología puede ser vencida en ocasiones con suficiente esfuerzo y recursos. No obstante, las defensas basadas en el AA son mucho más difíciles de vencer porque abordan una región mucho más amplia del espacio de amenazas que cualquier otra cosa que hayamos visto

anteriormente y porque poseen capacidades similares a las humanas para aprender de sus errores.

Qué significa la IA para usted

Los sistemas empresariales se actualizan, modifican y amplían constantemente para servir a nuevos usuarios y nuevas funciones comerciales. En un entorno tan fluido, es útil tener "agentes" habilitados para que el AA pueda destacarse y señalar anomalías u otros indicadores que proporcionen valor forense. El AA servirá como un multiplicador de productividad que permite a los profesionales de seguridad enfocarse en la estrategia y la ejecución en lugar de invertir incontables horas analizando los datos de registro y eventos de aplicaciones, controles de punto final y defensas perimetrales. El AA nos permitirá hacer nuestro trabajo de manera más eficaz que nunca.

La tendencia a incorporar capacidades del AA en productos de seguridad nuevos y existentes continuará creciendo. Según un informe de Gartner de abril de 2016[7]:

- Para el año 2018, el 25 % de los productos de seguridad utilizados para la detección habrán incorporado algún tipo de Aprendizaje Automático.
- Para el año 2018, el análisis prescriptivo se implementará en al menos un 10 % de los productos del Análisis de comportamiento de entidad y usuario (UEBA, por sus siglas en inglés) para automatizar la respuesta a los incidentes, desde cero hoy en día.

A fin de implementar y administrar correctamente estos productos, deberá comprender la función de los componentes

7. Gartner Core Security, *The Fast-Evolving State of Security Analytics, abril de 2016*, ID de informe: G00298030 accessed at https://hs.coresecurity.com/gartner-reprint-2017

del AA para poder utilizarlos de manera efectiva y en todo su potencial. Los sistemas de AA no son omniscientes ni siempre producen resultados perfectos. Las mejores soluciones incorporarán sistemas de Aprendizaje Automático *y* operadores humanos. Por lo tanto, dentro de los próximos tres a cuatro años, comprender profundamente el AA y sus capacidades será un requisito profesional.

Sobre este libro

Este libro está organizado en cuatro capítulos:

1. **Capítulo uno: Agrupamiento:** El agrupamiento (del inglés clustering) abarca una variedad de técnicas para subdividir las muestras en distintos subgrupos o *clústeres* en función de las similitudes entre sus características y atributos clave. El agrupamiento es particularmente útil en la exploración de datos y el análisis forense gracias a su capacidad de escrutar grandes cantidades de datos para identificar irregularidades y anomalías que requieren una investigación más exhaustiva. En este capítulo, examinamos:
    - Los cálculos detallados realizados por los algoritmos de agrupamiento *k*-means y DBSCAN.
    - Cómo los analistas avanzan a través de las etapas típicas de un procedimiento de agrupamiento. Estas incluyen selección y muestreo de datos, extracción de características, codificación y vectorización de características, cómputo y gráficos de modelos, y validación y pruebas de modelos.
    - Conceptos fundamentales como normalización, hiperparámetros y espacio de características.

- Cómo incorporar tipos de datos continuos y categóricos.
- Concluimos con una sección de aprendizaje práctico que muestra cómo los modelos *k*-means y DBSCAN se pueden aplicar para identificar vulnerabilidades de seguridad similares a los asociadas con la filtración de los Panama Papers, que, en 2015, resultó en la exfiltración de 11,5 millones de documentos confidenciales y 2,6 terabytes de datos de clientes de la firma de abogados panameña Mossack Fonseca.

2. **Capítulo dos: Clasificación:** La clasificación abarca un conjunto de métodos computacionales para predecir la probabilidad de que una muestra determinada pertenezca a una clase predefinida, por ejemplo, si un determinado correo electrónico es correo no deseado o no. En este capítulo, examinamos:
   - Los cálculos detallados realizados por los algoritmos de regresión logística y árbol de decisión ACR para asignar muestras a las clases
   - Las diferencias entre los enfoques de aprendizaje supervisado y no supervisado
   - La diferencia entre clasificadores lineales y no lineales
   - Las cuatro fases de un procedimiento típico de aprendizaje supervisado, que incluyen entrenamiento, validación, prueba e implementación modelo
   - Para la regresión logística: conceptos fundamentales como ponderaciones de regresión, parámetros de regularización y penalización, límites de decisión, ajuste de datos, etc.

- Para árboles de decisión: conceptos fundamentales sobre tipos de nodos, variables divididas, puntuaciones de beneficios y criterios de detención.
- Cómo se pueden utilizar las matrices de confusión y las métricas, como la precisión y el recuerdo, para evaluar y validar la precisión de los modelos producidos por ambos algoritmos.
- Concluimos con una sección práctica de aprendizaje que muestra cómo se pueden aplicar los modelos de regresión logística y árbol de decisión para detectar sistemas de mando y control de botnets que aún están circulando libremente.

3. **Capítulo tres: Probabilidad:** En este capítulo, consideramos la probabilidad como una técnica de modelo predictivo para clasificar y agrupar muestras. Los temas incluyen:
   - Los cálculos detallados realizados por el clasificador bayesiano ingenuo (BI) y el algoritmo de agrupamiento del modelo de mezclas gaussianas (MMG).
   - Conceptos fundamentales, como prueba, resultado y evento, junto con las diferencias entre los tipos de probabilidad conjunta y condicional.
   - Para el BI: la función de la probabilidad a posteriori, la probabilidad a priori de clase, la probabilidad a priori de predictorar y la probabilidad de resolver un problema de clasificación.
   - Para el MMG: las características de una distribución normal y cómo cada distribución puede ser identificada de manera única por sus parámetros de media y varianza. También consideramos cómo el MMG utiliza la técnica de optimización de

maximización de expectativas en dos pasos para asignar muestras a las clases.

- Concluimos con una sección de aprendizaje práctico que muestra cómo se pueden aplicar los modelos BI y MMG para detectar mensajes spam enviados por SMS.

4. **Capítulo cuatro: Aprendizaje profundo:** Este término abarca una amplia gama de métodos de aprendizaje basados principalmente en el uso de redes neuronales, una clase de algoritmos llamados así porque simulan la forma en que las redes de neuronas densamente interconectadas interactúan en el cerebro. En este capítulo, consideramos cómo se pueden aplicar dos tipos de redes neuronales para resolver un problema de clasificación. Esto incluye:
   - Los cálculos detallados realizados por los tipos de redes neuronales de memoria a largo y corto plazo (MLCP) y convolucional (RNC).
   - Conceptos fundamentales, como nodos, capas ocultas, estados ocultos, funciones de activación, contexto, índices de aprendizaje, regularización por exclusión (dropout) y niveles crecientes de abstracción.
   - Las diferencias entre las arquitecturas de redes neuronales prealimentadas (feedforward) y recurrentes y la importancia de incorporar capas completamente conectadas frente a capas parcialmente conectadas.
   - Concluimos con una sección de aprendizaje práctico que muestra cómo se pueden aplicar los modelos de MLCP y RNC para determinar la longitud de la clave XOR utilizada para ocultar una muestra de texto.

Creemos firmemente que no hay sustituto para la experiencia práctica. Por eso, podrá encontrartodos los scripts y conjuntos de datos que demostramos en las secciones de aprendizaje práctico disponibles para su descarga en:

https://www.cylance.com/intro-to-ai

Para simplificar, todos estos scripts cuentan con codificación fija de configuraciones que sabemos que son útiles. Sin embargo, le sugerimos que experimente modificándolos, y también creando otros nuevos, para que pueda apreciar plenamente cuán flexibles y versátiles son realmente estos métodos.

Lo que es más importante, le recomendamos enfáticamente que considere cómo se puede emplear el Aprendizaje Automático para abordar los tipos de problemas de seguridad que suele encontrar en su propio lugar de trabajo.

Clasificación utilizando los algoritmos de regresión logística y árbol de decisión

Agrupamiento: Uso de los algoritmos *K*-Means y DBSCAN

El objetivo del análisis de grupos es segregar los datos en un conjunto de grupos discretos o *clústeres* en función de las similitudes entre sus características o atributos clave. Dentro de un grupo dado, los elementos de datos serán más similares entre sí que lo que son para los elementos de datos dentro de un grupo diferente. Se pueden usar diversas técnicas estadísticas, de Inteligencia Artificial y de Aprendizaje Automático para crear estos grupos, con el algoritmo específico aplicado determinado por la naturaleza de los datos y los objetivos del analista.

Si bien el análisis de grupos surgió por primera vez aproximadamente hace ochenta y cinco años en el campo de las ciencias sociales, se ha demostrado que es un método sólido y ampliamente aplicable para explorar datos y extraer ideas significativas. Las empresas minoristas de todo tipo, por ejemplo, han utilizado con éxito el análisis de grupos para segmentar a sus clientes en grupos con hábitos de compra similares mediante el

análisis de terabytes de registros de transacciones almacenados en grandes almacenes de datos. Los minoristas pueden utilizar los modelos de segmentación de clientes resultantes para realizar ofertas personalizadas de ventas cruzadas y ventas adicionales que tienen una probabilidad mucho mayor de ser aceptadas. El agrupamiento también se utiliza con frecuencia en combinación con otras técnicas analíticas en tareas tan diversas como el reconocimiento de patrones, el análisis de datos de investigación, la clasificación de documentos y, aquí en Cylance, en la detección y el bloqueo de malware antes de que pueda ejecutarse.

En el dominio de seguridad la red, el análisis de grupos generalmente se lleva a cabo a través de una serie bien definida de operaciones de preparación y análisis de datos. Al final de este capítulo, encontrará enlaces a un sitio web de Cylance con datos e instrucciones para seguir este mismo procedimiento por su cuenta.

Paso 1: Selección de datos y muestreo

Antes de comenzar con cualquier enfoque de Aprendizaje Automático, debemos analizar algunos datos. Idealmente, analizaríamos *todas* las operaciones de nuestras redes y los datos de nuestro sistema para garantizar que nuestros resultados reflejen con precisión nuestro entorno informático y de redes. A menudo, sin embargo, esto no es posible ni práctico debido al enorme volumen de los datos y la dificultad de recopilar y consolidar los datos distribuidos a por fuentes de datos y sistemas heterogéneos. En consecuencia, normalmente aplicamos técnicas de muestreo estadístico que nos permiten crear un subconjunto de datos más manejable para nuestro análisis. La muestra debe reflejar las características del conjunto de datos total con tanta certeza como sea posible, o la

exactitud de nuestros resultados puede verse comprometida. Por ejemplo, si decidimos analizar la actividad de Internet para diez computadoras diferentes, nuestra muestra debe incluir entradas de registro representativas de los diez sistemas.

Paso 2: Extracción de características

En esta etapa, decidimos qué elementos de datos de nuestras muestras deberían extraerse y someterse a análisis. En el Aprendizaje Automático, nos referimos a estos elementos de datos como "características", es decir, atributos o propiedades de los datos que se pueden analizar para generar información útil.

En el análisis de reconocimiento facial, por ejemplo, las características relevantes probablemente incluirían la forma, el tamaño y la configuración de los ojos, la nariz y la boca. En el dominio de seguridad, las características relevantes pueden incluir el porcentaje de puertos que están abiertos, cerrados o filtrados, la aplicación que se ejecuta en cada uno de estos puertos y los números de versión de la aplicación. Si estamos investigando la posibilidad de exfiltración de datos, deberíamos incluir las características referidas a la utilización del ancho de banda y los tiempos de inicio de sesión.

Por lo general, tenemos miles de características para elegir. Sin embargo, cada característica que agregamos aumenta la carga en el procesador y el tiempo que lleva completar nuestro análisis. Por lo tanto, es una buena práctica incluir tantas características como sean necesarias, y excluir aquellas que sabemos que son irrelevantes en función de nuestra experiencia previa en la interpretación de dichos datos y nuestra experiencia general en la materia. También se pueden usar medidas estadísticas para eliminar automáticamente características inútiles o sin importancia.

Paso 3: Codificación y vectorización de características

La mayoría de los algoritmos de Aprendizaje Automático requieren que los datos se codifiquen o representen de alguna manera matemática. Una forma muy común de codificar datos es mapeando cada muestra y su conjunto de características a una grilla de filas y columnas. Una vez estructurado de esta manera, cada muestra se denomina "vector". El conjunto completo de filas y columnas se conoce como "matriz". El proceso de codificación que utilizamos depende de si los datos que representan cada característica son *continuos, categóricos* o de algún otro tipo.

Los datos que son continuos pueden ocupar cualquiera de un número infinito de valores dentro de un rango de valores. Por ejemplo, la utilización de la CPU puede variar de 0 a 100 por ciento. Por lo tanto, podríamos representar el uso promedio de la CPU para un servidor durante más de una hora como un conjunto de vectores simples, como se muestra a continuación.

Muestra (Hora)	% de utilización de la CPU
2 a. m.	12
9 a. m.	76
1 p. m.	82
6 p. m.	20

A diferencia de los datos continuos, los datos categóricos se representan generalmente mediante un pequeño conjunto de valores permitidos dentro de un rango mucho más limitado. El nombre del software y el número de versión son dos buenos ejemplos. Los datos categóricos son intrínsecamente útiles para definir grupos. Por ejemplo, podemos usar características categóricas como el sistema operativo y el número de versión para identificar un grupo de sistemas con características similares.

Las categorías como estas deben codificarse como números antes de que puedan someterse al análisis matemático. Una forma de hacerlo es crear un espacio dentro de cada vector para acomodar cada valor de datos permitido que se asigna a una categoría junto con un indicador dentro de cada espacio para indicar si ese valor está presente o no. Por ejemplo, si tenemos tres servidores que ejecutan una de las tres versiones diferentes de Linux, podríamos codificar la característica del sistema operativo de la siguiente manera:

Host	Ubuntu	Red Hat Enterprise Linux	SUSE Linux Enterprise Server
A	1	0	0
B	0	1	0
C	0	0	1

Como podemos ver, el Host A está ejecutando Ubuntu, mientras que los Hosts B y C ejecutan versiones de Red Hat y SUSE de Linux, respectivamente.

Alternativamente, podemos asignar un valor a cada sistema operativo y vectorizar nuestros hosts en consecuencia:

Sistema operativo	Valor asignado	Host	Vector
Ubuntu	1	A	1
Red Hat Enterprise Linux	2	B	2
SUSE Linux Enterprise Server	3	C	3

Sin embargo, debemos tener cuidado de evitar mapeos arbitrarios que puedan causar que una operación de Aprendizaje Automático, como un algoritmo de agrupamiento, asigne erróneamente un significado a estos valores cuando ninguno existe realmente. Por ejemplo, al utilizar los mapeos anteriores, un algoritmo podría asumir que Ubuntu es "menor que" Red Hat porque 1 es menor que 2 o llegar a la conclusión opuesta si

los valores se invierten. En la práctica, los analistas utilizan un método de codificación algo más complicado que a menudo se denomina "codificación única o one-hot".

En muchos casos, los datos continuos y categóricos se usan en combinación. Por ejemplo, podríamos incluir un conjunto de características continuas (por ejemplo, el porcentaje de puertos abiertos, cerrados y filtrados) en combinación con un conjunto de características categóricas (por ejemplo, el sistema operativo y los servicios que se ejecutan en cada puerto) para identificar un grupo de nodos con perfiles de riesgo similares. En situaciones como estas, a menudo es necesario comprimir el rango de valores en los vectores continuos a través de un proceso de "normalización" para garantizar que las características dentro de cada vector tengan igual importancia. El algoritmo *k*-means, por ejemplo, usa la distancia promedio desde un punto central a los vectores de un grupo por similitud. Sin la normalización, *k*-means puede dar mayor importancia a los efectos de los datos categóricos y, en consecuencia, distorsionar los resultados.

Consideremos el siguiente ejemplo:

Muestra (Servidor)	Solicitudes por segundo	% de utilización de la CPU
Alfa	200	67
Bravo	160	69
Charlie	120	60
Delta	240	72

Aquí, los valores de la característica Solicitudes por segundo tienen un rango diez veces mayor que los de la característica % de utilización de la CPU. Si estos valores no se normalizaran, el cálculo de la distancia probablemente se distorsionaría para destacar los efectos de esta disparidad de rango.

En el gráfico a continuación, por ejemplo, podemos ver que la diferencia entre el servidor Alfa y el servidor Bravo con respecto a Solicitudes por segundo es de 40, mientras que la diferencia entre los servidores con respecto al % de utilización de la CPU es de solo 2. En este caso, Solicitudes por segundo representa el 95 % de la diferencia entre los servidores, una disparidad que puede distorsionar fuertemente los cálculos de distancia subsiguientes.

Abordaremos este problema de distorsión normalizando ambas características en el rango 0-1 usando la fórmula: x-xmin / xmax-xmin.

Muestra (Nombre)	Solicitudes por segundo	% de utilización de la CPU
Alfa	0,66	0,58
Bravo	0,33	0,75
Charlie	0	0
Delta	1	1

Después de la normalización, la diferencia en Solicitudes por segundo entre los servidores Alfa y Bravo es de 0,33, mientras que la diferencia en el % de utilización de la CPU se ha reducido a 17. Las Solicitudes por segundo ahora representan solo el 66 % de la diferencia.

Paso 4: Cómputo y gráficos

Una vez que terminemos de convertir las características en vectores, estamos listos para importar los resultados a un análisis estadístico adecuado o a una aplicación de minería de datos como IBM SPSS Modeler y SAS Data Mining Solution. Alternativamente, podemos utilizar una de las cientos de bibliotecas de software disponibles para realizar dicho análisis. En los ejemplos que siguen, utilizaremos scikit-learn, una

biblioteca de minería de datos de código abierto y funciones estadísticas integradas en el lenguaje de programación Python.

Una vez que se cargan los datos, podemos elegir qué algoritmo de agrupamiento aplicar primero. En scikit-learn, por ejemplo, nuestras opciones incluyen *k*-means, propagación por afinidad, Mean-Shift, agrupamiento espectral, agrupamiento jerárquico de Ward, agrupamiento aglomerativo, DBSCAN, mezclas gaussianas y Birch. Consideremos dos de los algoritmos de agrupamiento más populares: *k*-means y DBSCAN.

Agrupamiento con K-Means

Como humanos, experimentamos el mundo como si constara de tres dimensiones espaciales, lo que nos permite determinar la distancia entre dos objetos al medir la longitud de la línea recta más corta que los conecta. Esta "distancia euclidiana" es lo que calculamos cuando utilizamos el Teorema de Pitágoras.

El análisis de agrupamiento introduce el concepto de "espacio de características" que puede contener miles de dimensiones, una para cada característica de nuestro conjunto de muestras. Los algoritmos de agrupamiento asignan vectores a coordenadas particulares en este espacio de características y luego miden la distancia entre dos vectores para determinar si son lo suficientemente similares como para formar parte del mismo grupo. Como veremos, los algoritmos de agrupamiento pueden emplear una variedad de métricas de distancia para hacerlo. Sin embargo, *k*-means utiliza solo la distancia euclidiana. En *k*-means, y en la mayoría de los demás algoritmos de agrupamiento, cuanto menor es la distancia euclidiana entre dos vectores, más probable es que se asignen al mismo grupo.

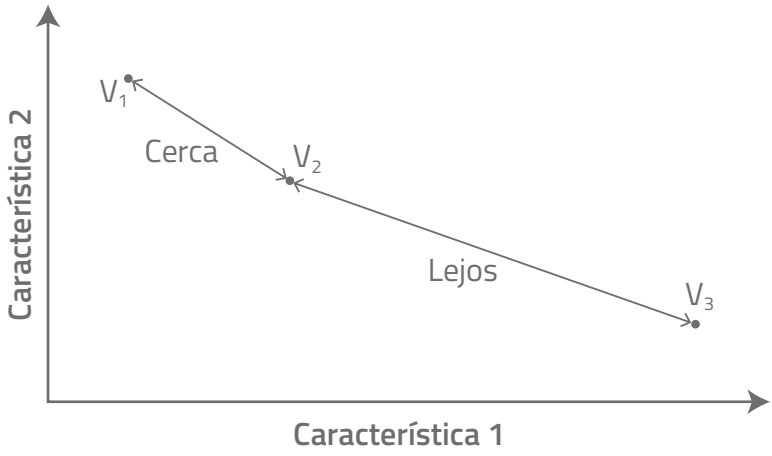


FIGURA 1.1: Vectores en el espacio de características

K-Means es computacionalmente eficiente y aplicable a una amplia gama de operaciones de análisis de datos, aunque con algunas salvedades:

- La versión de *k*-means que discutiremos funcionan solo con datos continuos. (Las versiones más sofisticadas también funcionan con datos categóricos).
- Los patrones subyacentes dentro de los datos deben permitir que los grupos se definan dividiendo el espacio de características en regiones que usan líneas rectas y planos.
- Los datos se pueden agrupar de manera significativa en una serie de grupos de tamaño similar.

Si se cumplen estas condiciones, la sesión de agrupamiento procede de la siguiente manera:

1. Se muestrea, vectoriza y normaliza un conjunto de datos, y luego se importa a scikit-learn.

2. El analista de datos aplica el algoritmo *k*-means y especifica "*k*", una variable de entrada o "hiperparámetro" que indica a *k*-means cuántos grupos crear. (Nota: Casi todos los algoritmos incluyen uno o más hiperparámetros para "ajustar" el comportamiento del algoritmo). En este ejemplo, *k* se establecerá en tres para que, como máximo, se creen tres grupos.
3. *K*-Means selecciona aleatoriamente tres vectores del conjunto de datos y los asigna a las coordenadas en el espacio de características, uno para cada uno de los tres grupos que se crearán. Estos puntos se conocen como "centroides".
4. *K*-Means comienza a procesar el primer vector en el conjunto de datos calculando la distancia euclidiana entre sus coordenadas y las coordenadas de cada uno de los tres centroides. Luego, asigna la muestra al grupo con el centroide más cercano. Este proceso continúa hasta que todos los vectores se hayan asignado de esta manera.
5. *K*-Means examina los miembros de cada grupo y calcula su distancia promedio desde su centroide correspondiente. Si la ubicación actual del centroide coincide con este promedio calculado, permanece fijo. De lo contrario, el centroide se mueve a una nueva coordenada que coincide con el promedio calculado.
6. *K*-Means repite el paso cuatro para todos los vectores y los reasigna a grupos basados en las nuevas ubicaciones del centroide.
7. *K*-Means repite los pasos 5 a 6 hasta que ocurra una de las siguientes situaciones:
    - el centroide deja de moverse y pertenencia permanece fija, un estado conocido como "convergencia";

- el algoritmo completa el número máximo de iteraciones especificado previamente por el analista.

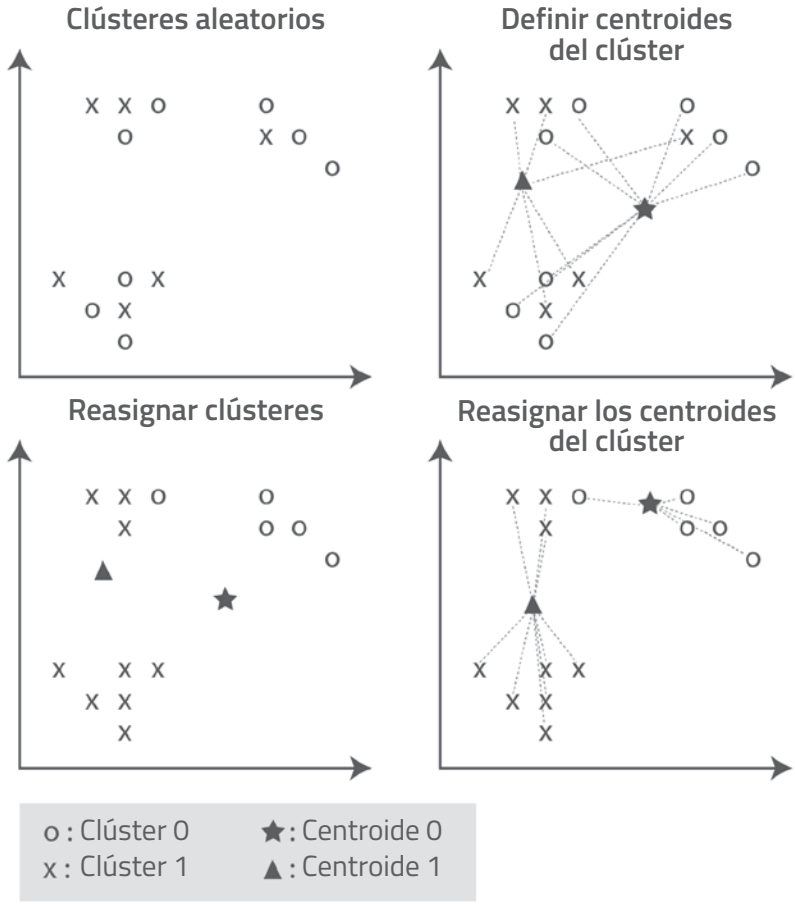


FIGURA 1.2: Proceso de agrupamiento de K-Means

Una vez que se completa el agrupamiento, el analista de datos puede:

- evaluar la exactitud de los resultados usando una variedad de técnicas de validación;
- convertir los resultados en un modelo matemático para evaluar la pertenencia al grupo de nuevas muestras;

- analizar los resultados del grupo en mayor detalle utilizando técnicas estadísticas y de Aprendizaje Automático adicionales.

Este mismo proceso también se aplica a los espacios de características dimensionales más altos, que contienen cientos o incluso miles de dimensiones. Sin embargo, el tiempo de cómputo para cada iteración aumentará en proporción a la cantidad de dimensiones que se analicen.

ESCOLLOS Y LIMITACIONES DE *K*-MEANS

Si bien es fácil de usar y puede producir excelentes resultados, la versión de *k*-means que hemos estado discutiendo es vulnerable a una serie de errores y distorsiones:

el analista debe hacer una suposición informada desde el principio sobre cuántos grupos se deben crear. Esto requiere una experiencia considerable sobre dominios. En la práctica, a menudo es necesario repetir la operación de agrupamiento varias veces hasta que se haya identificado la cantidad óptima de grupos.

Los resultados del agrupamiento pueden variar dramáticamente dependiendo de dónde se colocan inicialmente los centroides. El analista no tiene control sobre esto ya que esta versión de *k*-means asigna estos lugares aleatoriamente. De nuevo, el analista tal vez deba ejecutar el procedimiento de agrupamiento varias veces y luego seleccionar los resultados de dicho agrupamiento que sean más útiles y coherentes con los datos.

La distancia euclidiana se descompone como una medida de similitud en espacios de características dimensionales muy elevados. Este es uno de los problemas a los que los expertos en Aprendizaje Automático se refieren con el término general

"la maldición de la dimensionalidad". En estas situaciones, se deben emplear diferentes algoritmos y métodos para medir la similitud.

Agrupamiento con DBSCAN

Otro algoritmo de agrupamiento comúnmente utilizado es DBSCAN o "agrupamiento espacial basado en densidad de aplicaciones con ruido". DBSCAN fue presentado por primera vez en 1996 por Hans-Peter Kriegel.

Como su nombre lo indica, DBSCAN identifica grupos mediante la evaluación de la densidad de puntos dentro de una región determinada del espacio de características. DBSCAN crea grupos en las regiones con más vectores y considera que los puntos en las regiones más dispersas son ruido.

En contraste con *k*-means, DBSCAN:

- descubre por sí mismo cuántos grupos crear en lugar de requerir que el analista especifique esto de antemano con el hiperparámetro *k*;
- es capaz de crear grupos de prácticamente cualquier forma y tamaño.

DBSCAN presenta al analista dos hiperparámetros que determinan cómo se desarrolla el proceso de agrupamiento:

- Epsilon (Eps) especifica el radio de la región circular que rodea cada punto que se utilizará para evaluar su pertenencia al grupo. Esta región circular se conoce como el "vecindario Epsilon". El radio se puede especificar usando diversas métricas de distancia.
- Puntos mínimos (MinPts): especifica el número mínimo de puntos que debe aparecer dentro de un vecindario Epsilon para que los puntos dentro se incluyan en un grupo.

DBSCAN realiza el agrupamiento examinando cada punto en el conjunto de datos y luego lo asigna a una de tres categorías:

- un *punto central* es un punto que tiene más de la cantidad especificada de MinPts dentro de su vecindario Epsilon;
- un *punto de frontera* se encuentra dentro del vecindario de un punto central, pero no tiene vecinos propios suficientes para calificar como punto central;
- un *punto de ruido* es uno que no es ni un punto central ni un punto de frontera.

A continuación, se muestran ejemplos de puntos centrales, de frontera y de ruido.

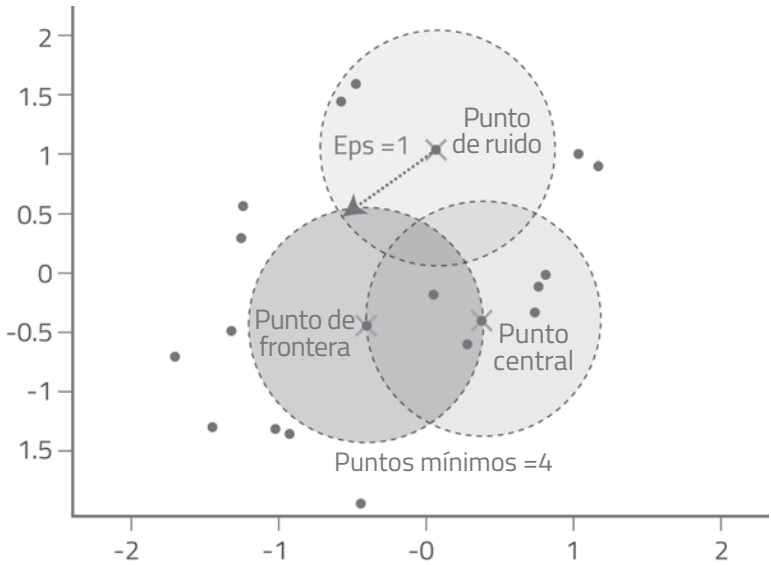


FIGURA 1.3: Proceso de agrupamiento de DBSCAN

Una sesión de agrupamiento de DBSCAN en scikit-learn normalmente procede de la siguiente manera:

1. Se muestrea, vectoriza y normaliza un conjunto de datos, y luego se importa a scikit-learn.

2. El analista crea un objeto DBSCAN y especifica los valores iniciales Eps y MinPts.
3. DBSCAN selecciona aleatoriamente uno de los puntos en el espacio de características, por ejemplo, el Punto A, y luego cuenta el número de puntos, incluido el Punto A, que se encuentran dentro del vecindario Eps del Punto A. Si este número es igual o mayor que los MinPts, entonces el punto se clasifica como un punto central y DBSCAN agrega el Punto A y sus vecinos a un nuevo grupo. Para distinguirlo de los grupos existentes, al nuevo grupo se le asigna una identificación o ID de grupo.
4. DBSCAN se mueve desde el Punto A a uno de sus vecinos, por ejemplo, el Punto B, y luego lo clasifica como punto central o de frontera. Si el Punto B califica como un punto central, entonces el Punto B y sus vecinos se agregan al grupo y se les asigna la misma ID de grupo. Este proceso continúa hasta que DBSCAN haya visitado a todos los vecinos y haya detectado todos los puntos centrales y de frontera de ese grupo.
5. DBSCAN avanza hasta un punto que no ha visitado antes y repite los pasos 3 y 4 hasta que se hayan categorizado todos los puntos vecinos y de ruido. Cuando concluye este proceso, se han identificado todos los grupos les han asignado sus ID de grupo.

Si los resultados de este análisis son satisfactorios, la sesión de agrupamiento finaliza. De lo contrario, el analista tiene una serie de opciones. Pueden ajustar los hiperparámetros Eps y MinPts y ejecutar DBSCAN nuevamente hasta que los resultados cumplan con sus expectativas. Alternativamente, pueden redefinir cómo funciona el hiperparámetro Eps al definir vecindarios Eps aplicando una métrica de distancia diferente. DBSCAN admite varias métricas diferentes, que incluyen:

- **Distancia euclidiana:** Este es el método de "la línea recta más corta entre puntos" que describimos anteriormente.
- **Distancia Manhattan o distancia de ciudad:** Como su nombre lo indica, este método es similar a uno que podríamos usar para medir la distancia entre dos ubicaciones en una gran ciudad dispuestas en una cuadrícula bidimensional de calles y avenidas. Aquí, estamos limitados a movernos a lo largo de una dimensión a la vez, navegando a través de una serie de giros en una y otra dirección hasta llegar a nuestro destino. Por ejemplo, si estamos caminando en Manhattan desde la 3.ª Avenida y la calle 51 hasta la 2.ª Avenida y la calle 59, debemos ir una cuadra al este y luego ocho cuadras al norte para llegar a nuestro destino, lo que representan una distancia total de nueve cuadras. De la misma manera, DBSCAN puede calcular el tamaño del vecindario de Eps y la distancia entre puntos tratando el espacio de características como una grilla multidimensional que solo puede atravesarse de a una dimensión a la vez. Aquí, la distancia entre los puntos se calcula sumando el número de unidades a lo largo de cada eje que debe atravesarse para pasar del Punto A al Punto B.
- **Similitud de coseno:** En el análisis de grupos, la similitud en las características se representa por la distancia relativa en el espacio de características. Mientras más cerca estén los dos vectores, es más probable que se ubiquen dentro del mismo vecindario de Eps y compartan la misma pertenencia de grupo. Sin embargo, la distancia entre dos vectores también puede definirse tratando cada vector como el vértice de un triángulo con el tercer vértice ubicado en el punto de origen del gráfico. En esta situación, la distancia se estima calculando el coseno

para el ángulo formado por las líneas que conectan los dos vectores al punto de origen. Cuanto menor sea el ángulo, más probable es que los dos puntos tengan características similares y vivan en el mismo vecindario de Eps. Del mismo modo, cuanto mayor es el ángulo, es más probable que tengan características diferentes y pertenezcan a diferentes grupos.

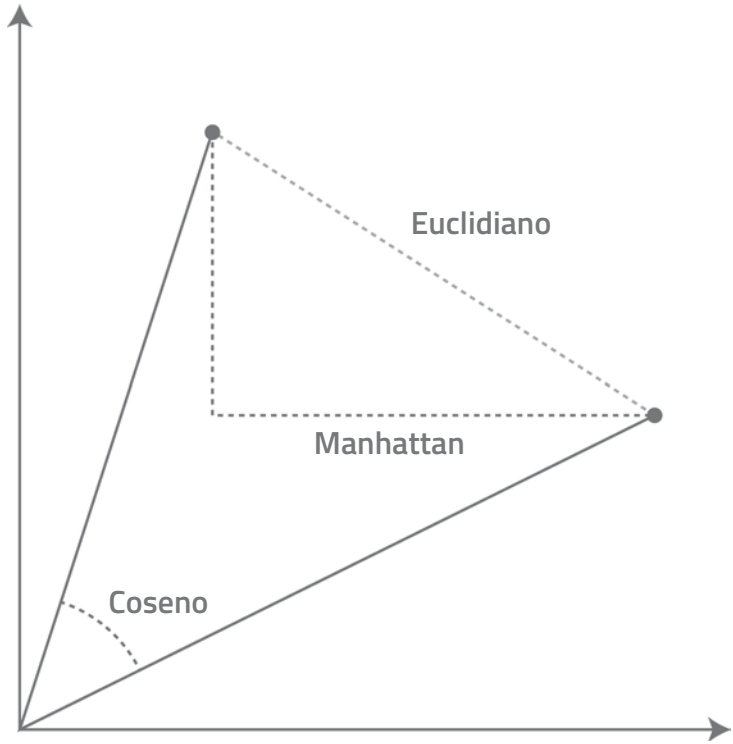


FIGURA 1.4: Distancias euclidiana, Manhattan y de Coseno

ESCOLLOS Y LIMITACIONES DE DBSCAN

Si bien puede descubrir una variedad más amplia de formas y tamaños de grupo que *k*-means, DBSCAN:

- es sumamente sensible incluso a pequeños cambios en la configuración de MinPts y Eps, lo que hace que fragmente

grupos bien definidos en conjuntos fragmentos de grupos más pequeños;

- se vuelve menos eficiente desde el punto de vista computacional a medida que se agregan más dimensiones, lo que resulta en un rendimiento inaceptable en espacios de características dimensionales sumamente altos;
- no funciona bien con conjuntos de datos que producen regiones de densidades variables debido a los valores fijos que se deben asignar a MinPts y Eps.

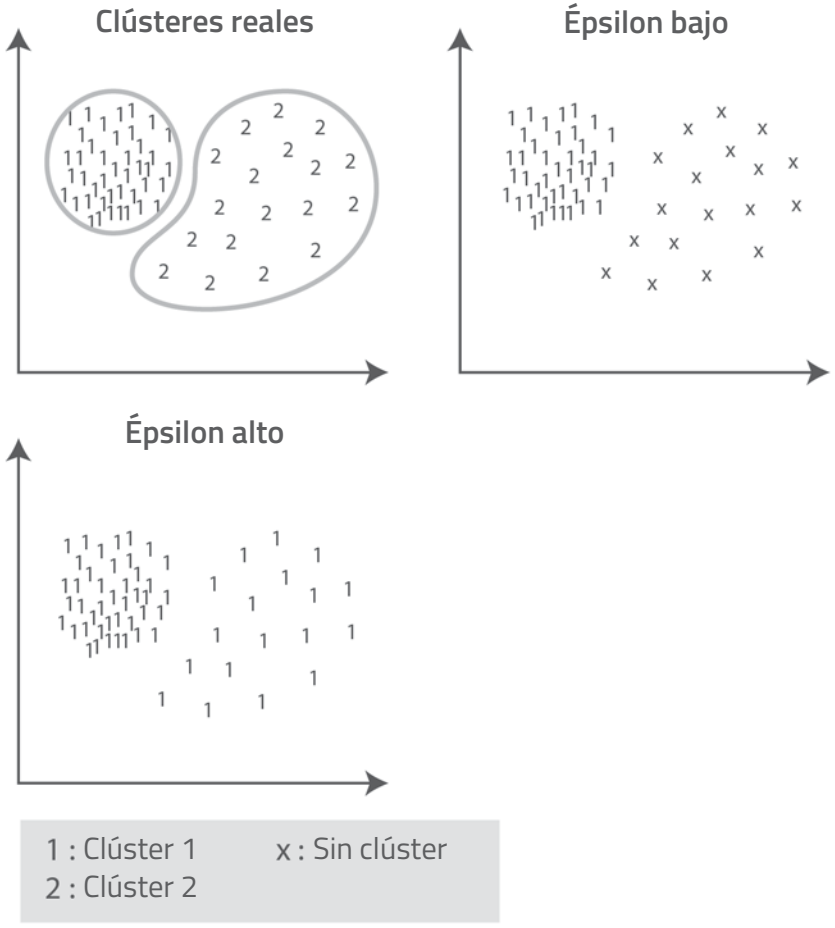


FIGURA 1.5: Escollo de la densidad del grupo de DBSCAN

Evaluación de la validez del grupo

Al final de cada procedimiento de agrupamiento, se nos presenta una solución que consiste en un conjunto de grupos k. Pero, ¿cómo vamos a evaluar si estos grupos son representaciones precisas de los datos subyacentes? El problema se agrava cuando ejecutamos una operación de grupos varias veces con diferentes algoritmos o el mismo algoritmo varias veces con diferentes configuraciones de hiperparámetros.

Afortunadamente, existen numerosas maneras de validar la integridad de nuestros grupos. Estos se conocen como "índices" o "criterios de validación". Por ejemplo, podemos:

- ejecutar nuestro conjunto de muestras en un modelo externo y comprobar si las asignaciones de grupos resultantes coinciden con las nuestras;
- probar nuestros resultados con "datos retenidos", es decir, vectores de nuestro conjunto de datos que no usamos para nuestro análisis de grupos. Si nuestros resultados de grupo son correctos, las nuevas muestras deberían asignarse a los mismos grupos que nuestros datos originales;
- utilizar métodos estadísticos. Con k-means, por ejemplo, podemos calcular un Coeficiente de Silueta, que compara la distancia promedio entre los puntos que se encuentran dentro de un grupo dado a la distancia promedio entre los puntos asignados a diferentes grupos. Cuanto menor sea el coeficiente, más seguros podemos estar de que los resultados de nuestro agrupamiento sean precisos.
- Compare los resultados de agrupamiento producidos por diferentes algoritmos o por el mismo algoritmo usando diferentes configuraciones de hiperparámetros. Por ejemplo, podemos calcular los Coeficientes de Silueta

para *k*-means y DBSCAN para ver qué algoritmo ha producido los mejores resultados, o comparar resultados de ejecuciones DBSCAN que utilizan diferentes valores para Eps.

Análisis de grupos aplicado a situaciones de amenazas del mundo real

Como hemos visto, el análisis de grupos nos permite examinar grandes cantidades de operaciones de la red y datos del sistema para detectar relaciones ocultas entre los miembros del grupo en función de las similitudes y diferencias de las características que los definen. Pero, ¿cómo ponemos estas capacidades analíticas en práctica en la detección y prevención de ataques a la red del mundo real? Consideremos cómo el análisis de grupos podría haber sido útil con respecto a la filtración de los Panamá Papers, que resultó en la exfiltración de aproximadamente 11,5 millones de documentos confidenciales y 2,6 terabytes de datos de clientes de la firma de abogados panameña Mossack Fonseca (MF).

Comenzamos con tres salvedades:

- Si bien varios medios y organizaciones de seguridad han presentado pruebas contundentes sobre los vectores de ataque más probables, nadie puede decir con certeza cómo el pirata informático logró penetrar el servidor web, el servidor de correo electrónico y las bases de datos de clientes de MF en el transcurso de un año o más. Tendríamos que someter la red y el sistema de datos de MF a un análisis forense profundo para confirmar la naturaleza y el alcance de estas vulnerabilidades de seguridad.
- Esta información debería ser de suficiente alcance y calidad para admitir la variedad de métodos de uso

intensivo de datos que comúnmente empleamos para detectar y prevenir ataques.

- Nuestro análisis no se limitará al agrupamiento únicamente. Idealmente, emplearíamos una combinación de Aprendizaje Automático, Inteligencia Artificial y métodos estadísticos junto con el agrupamiento.

Por ahora, sin embargo, procederemos con una situación en la que se contemple solamente el agrupamiento basada en la evidencia presentada por medios creíbles y fuentes de la industria.

Según la empresa de ingeniería de software Wordfence[8], por ejemplo, el pirata informático podría haber comenzado apuntando a las vulnerabilidades conocidas en el complemento WordPress Revolution Slider documentadas en el sitio web Exploit Database en noviembre de 2014. El pirata podría haber aprovechado esta vulnerabilidad para cargar un script PHP en el servidor web de WordPress. Esto le habría proporcionado acceso al shell y la capacidad de ver archivos del servidor como wp-config.php, que almacena las credenciales de la base de datos de WordPress en texto sin cifrar. Al contar con acceso a la base de datos, también podría haber capturado todas las credenciales de la cuenta de correo electrónico almacenadas allí en texto sin cifrar por el complemento ALO EasyMail Newsletter, que MF utilizó para sus capacidades de administración de listas de correo electrónico. Colectivamente, estos y otrosaccesos ilegales al servidor de correo electrónico habrían permitido al pirata acceder y filtrar grandes cantidades de correos electrónicos de MF.

8. Mark Maunder, "Panama Papers: Email Hackable via WordPress, Docs Hackable via Drupal" (8 de abril de 2016), consultado el 15 de mayo de 2016 desde https://www.wordfence.com/blog/2016/04/panama-papers-wordpress-email-connection/. También consulte: Mark Maunder, "Mossack Fonseca Breach—WordPress Revolution Slider Plugin Possible Cause" (7 de abril de 2016), consultado el 15 de mayo de 2016 desde https://www.wordfence.com/blog/2016/04/mossack-fonseca-breach-vulnerable-slider-revolution/.

La revista Forbes[9] también informó que, en el momento del ataque, MF utilizaba la versión 7.23 de Drupal para administrar el portal "seguro" que los clientes usaban para acceder a sus documentos privados. Se sabía que esta versión era vulnerable a diversos ataques, incluido un ataque de inyección SQL que, por sí solo, habría sido suficiente para abrir las compuertas para una exfiltración masiva de documentos.

Basándose en esta y otra información, creemos que es probable que el análisis de grupos, realizado como parte de un programa de detección en curso, podría haber detectado anomalías en la actividad de la red de MF y proporcionado indicios importantes sobre la naturaleza y el alcance de los ataques del pirata. Normalmente, los miembros del equipo de detección analizarían los registros del servidor web y del correo por separado. Luego, si se detecta un ataque a uno de los servidores, el equipo de detección podría analizar los datos del otro servidor para ver si los individuos maliciosos podrían estar involucrados en ambos conjuntos de ataques y lo que esto podría indicar sobre la magnitud del daño.

Del lado del servidor de correo, las características relevantes que se extraerán pueden incluir la hora y la fecha de inicio de sesión del usuario, la dirección IP, la ubicación geográfica, el cliente de correo electrónico, los privilegios administrativos y la actividad del servidor SMTP. Del lado del servidor web, las características relevantes pueden incluir la dirección IP y la ubicación del usuario, la versión del navegador, la ruta de las páginas a las que se accede, los códigos de estado del servidor web y la utilización del ancho de banda asociado.

9. Jason Bloomberg, "Cybersecurity Lessons Learned from 'Panama Papers' Breach," Forbes.com (Abril de 2016), http://www.forbes.com/sites/jasonbloomberg/2016/04/21/cybersecurity-lessons-learned-from-panama-papers-breach/#47c9045d4f7a

Después de completar este análisis de grupos, la mayor parte de los vectores web y de correo electrónico resultantes deberían agruparse en un conjunto de grupos bien definidos que reflejen patrones operativos normales y un número menor de grupos muy dispersos o puntos de ruido que indican usuarios y actividad de red anómalos. Luego, podríamos investigar estas anomalías mediante la compilación de nuestros datos de registro para relacionar esta actividad sospechosa con los individuos maliciosos a través de sus direcciones IP.

Este análisis podría revelar:

- **Patrones de autenticación anómalos.** Podríamos preguntarnos por qué un grupo de ejecutivos de MF con sede en nuestra oficina de Londres comenzó repentinamente a acceder a sus cuentas de correo electrónico con un cliente de correo electrónico que nunca antes habían utilizado. De otro modo, podríamos observar que un grupo de empleados ubicados en nuestra oficina de Londres acceden periódicamente a sus cuentas de correo electrónico desde ubicaciones en las que no tenemos oficinas, clientes o socios comerciales.
- **Comportamiento anómalo del usuario.** Es posible que identifiquemos grupos de clientes que inician sesión y luego pasan largas horas descargando grandes cantidades de documentos sin subir ninguno. O bien, podríamos encontrar grupos de usuarios de correo electrónico que pasan largas horas leyendo correos electrónicos pero nunca envían ninguno.
- **Patrones de tráfico de red anómalos.** Podríamos observar un fuerte aumento en el volumen de tráfico dirigido a la página del portal del cliente y otras URL que incluyen a Drupal en sus declaraciones de ruta.

Por supuesto, estos ejemplos son solo hipotéticos. El grado en que el análisis de agrupamiento podría indicar un ataque como la filtración de los Panama Papers estaría determinado por el contenido real de la red y los datos del sistema y la experiencia de los analistas de datos del equipo de detección. Sin embargo, está claro que el análisis de grupos puede proporcionar indicios importantes sobre una brecha de seguridad difícil de desentrañar entre las miles de entradas de registro que se generan típicamente cada semana en una red de tamaño medio. Además, estos conocimientos podrían deducirse de los datos en sí mismos sin depender de las firmas de la vulnerabilidad de seguridad o las alertas de un sistema IDS/IPS.

Sesión de agrupamiento que utiliza datos de registro HTTP

Aplicaremos lo aprendido para ver cómo se puede usar el agrupamiento en una situación real para revelar un ataque y rastrear su progreso. En este caso, analizaremos los datos de registro del servidor HTTP de secrepo.com que revelarán varias vulnerabilidades de seguridad similares a las que precedieron a la exfiltración de los Panama Papers. Si desea probar este ejercicio por su cuenta, visite https://www.cylance.com/intro-to-ai, donde podrá descargar todas las instrucciones y los archivos de datos pertinentes.

Los registros del servidor HTTP capturan diversos datos forenses útiles sobre los usuarios finales y sus patrones de acceso a Internet. Esto incluye direcciones IP, sellos de fecha/hora, lo que se solicitó, cómo respondió el servidor, etc. En este ejemplo, agruparemos las direcciones IP en función de los verbos HTTP (por ejemplo, GET, POST, etc.) y los códigos de respuesta HTTP (por ejemplo, 200, 404, etc.). Buscaremos evidencia de una posible violación luego de recibir información de un WAF o de un

servicio de monitoreo de amenazas que indique que la dirección IP 70.32.104.50 se ha asociado con ataques dirigidos a servidores de WordPress. Es posible que nos preocupe especialmente si se ha informado recientemente de una vulnerabilidad grave de WordPress, como Revolution Slider. Por lo tanto, agruparemos las direcciones IP para detectar patrones de comportamiento similares a los informados en 70.32.104.50 que podrían indicar que nuestros servidores se han visto comprometidos.

Los códigos de respuesta HTTP utilizados para este conjunto de datos específico son los siguientes:

200, 404, 304, 301, 206, 418, 416, 403, 405, 503, 500

Los verbos HTTP para este conjunto de datos específico son los siguientes:

GET, POST, HEAD, OPTIONS, PUT, TRACE

Ejecutaremos nuestro procedimiento de agrupamiento dos veces, una vez con *k*-means y luego una segunda vez con DBSCAN. Concluiremos cada procedimiento volviendo a nuestros archivos de registro y examinando de cerca el comportamiento de las direcciones IP que aparecen como valores atípicos o miembros de un grupo sospechoso.

ANÁLISIS DE GRUPOS CON *K*-MEANS

Paso 1: Vectorización y normalización

Comenzamos preparando nuestras muestras de registro para el análisis. Aquí tomaremos un atajo y aplicaremos un script escrito expresamente para vectorizar y normalizar este conjunto de datos en particular.

Para cada dirección IP, contaremos el número de verbos y códigos de respuesta HTTP. Sin embargo, en lugar de simplemente sumar la cantidad de incidentes, representaremos

estas características como valores continuos normalizándolos. Si no hiciéramos esto, dos IP con patrones de comportamiento casi idénticos podrían agruparse de manera diferente simplemente porque uno hizo más solicitudes que el otro.

Con el tiempo y la potencia de CPU suficientes, podríamos examinar las 16.407 direcciones IP en nuestro archivo de registro de más de 181.332 entradas. Sin embargo, comenzaremos con las primeras 10.000 direcciones IP y veremos si esta muestra es suficiente para que podamos determinar si se ha producido un ataque. También limitaremos nuestra muestra a direcciones IP asociadas con al menos cinco entradas de registro cada una. Es poco probable que aquellas con actividad escasa presenten una amenaza seria para nuestros servidores web y de WordPress.

El siguiente script de Python invocará el proceso de vectorización:

`python vectorize_secrepo.py`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python vectorize_secrepo.py
Finished prebuilding samples
```

Esto produce "secrepo.h5", un archivo de formato de datos jerárquicos (HDF5) que contiene nuestros vectores junto con un conjunto de ID de grupo y "notas" que indican qué dirección IP está asociada a cada vector. Usaremos estas direcciones más adelante cuando regresemos a nuestros registros para investigar actividades potencialmente maliciosas.

Paso 2: Graficar nuestros vectores

Ahora estamos listos para visualizar nuestros vectores en el espacio de características.

Los seres humanos no pueden visualizar entornos espaciales que exceden tres dimensiones. Esto hace que sea difícil para el analista interpretar los resultados del agrupamiento

obtenidos en espacios de características dimensionales altos. Afortunadamente, podemos aplicar técnicas de reducción de características que nos permitan ver nuestros grupos en un formato gráfico tridimensional. El siguiente script aplica una de estas técnicas, el Análisis de componentes principales. Ahora, podremos explorar los grupos rotando el gráfico a lo largo de cualquiera de sus tres ejes. Sin embargo, la rotación es un proceso computacionalmente intensivo que puede ocasionar que la pantalla se actualice lentamente. A menudo, es más rápido y más conveniente preparar varios ángulos de visión por adelantado durante el proceso de representación gráfica. Posteriormente, podemos alternar rápidamente entre cada una de las vistas preparadas para ver nuestros grupos desde diferentes ángulos.

Utilizaremos el siguiente script para visualizar nuestros vectores:

```
`python visualize_vectors.py -i secrepo.h5`
```

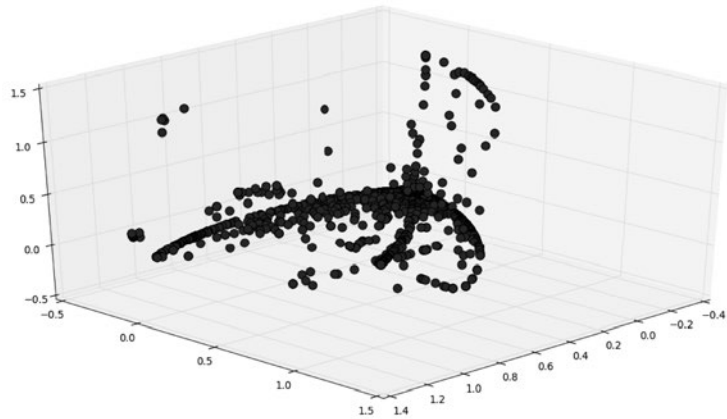


FIGURA 1.6: Visualización proyectada de nuestros vectores

Paso 3: Primera pasada de agrupamiento con K-Means

Como se señaló anteriormente, *k*-means solo nos exige establecer el hiperparámetro *k*, que especifica cuántos grupos crear. No sabremos inicialmente cuál debería ser el valor correcto de *k*. Por lo tanto, procederemos iterativamente a través del proceso de agrupamiento, estableciendo diferentes valores *k* e inspeccionando los resultados hasta conseguir un modelo de datos preciso y satisfactorio. Comenzaremos por establecer *k* en "2". También le indicaremos a *k*-means que use las ID de grupo que hemos especificado durante la vectorización para nombrar cada grupo:

```
`python cluster_vectors.py -c kmeans -n 2 -i secrepo.h5 -o
secrepo.h5`
```

Como se muestra a continuación, *k*-means ha analizado nuestras muestras, aplicado nuestras etiquetas y calculado la cantidad de vectores que se colocarán en cada uno de los dos grupos.

```
mldemo@mldemo-virtual-machine:~/mlbook$ python cluster_vectors.py -c kmeans -n 2 -i secrepo.h5 -o secrepo.h5
Clustered samples into 2 clusters
Label 0 has 662 samples
Label 1 has 3192 samples
```

Paso 4: Validar los grupos estadísticamente

Ahora que tenemos las ID de grupo, podemos determinar qué tan bien se han agrupado nuestras muestras aplicando la Puntuación de Silueta. Las puntuaciones variarán de -1 a +1. Cuanto más cerca estén las puntuaciones de +1, más seguros podemos estar de que nuestro agrupamiento sea preciso.

Produciremos las Puntuaciones de Silueta con el siguiente script:

stats_vectors.py

```
mldemo@mldemo-virtual-machine:~/mlbook$ python stats_vectors.py secrepo.h5
Vectors shape: (3854, 17)
Minimum feature value: 0.0
Mean feature value: 0.128143090123
Maximum feature value: 1.0
Percentage of null values: 83.2259836991%

Minimum distance between vectors: 0.0
Mean distance between vectors: 0.592780335395
Maximum distance between vectors: 2.0

Number of labels: 2
Number of items in label 0: 662  (17.1769590036%) (avg dist: 0.9581536657) (avg silhouette: 0.179235978039)
Number of items in label 1: 3192  (82.8230409964%) (avg dist: 0.32877885612) (avg silhouette: 0.727458032666)

Minimum label centroid distance: 0.899422434302
Mean label centroid distance: 0.899422434302
Maximum label centroid distance: 0.899422434302
Overall Silhouette Score: 0.633290155094
```

Como podemos ver, el Grupo 1 está bien agrupado, mientras que el Grupo 0 no lo está. También notamos que el Grupo 1 contiene muchas más muestras que el Grupo 0. Podemos interpretar que esto significa que el Grupo 1 refleja la actividad normal de la red, mientras que el Grupo 0 contiene un comportamiento del usuario menos típico y posiblemente malicioso.

Paso 5: Inspeccionar nuestros grupos

Ahora podemos interrogar a nuestros grupos para ver cuál contiene la dirección IP del individuo malicioso. Usaremos el siguiente script para imprimir las etiquetas y notas para cada uno de nuestros vectores.

`python label_notes.py -i secrepo.h5 | grep 70.32.104.50`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python label_notes.py -i secrepo.h5 | grep 70.32.104.50
0 70.32.104.50
```

Ahora podemos ver que la IP 70.32.104.50 es miembro del Grupo 0, nuestro grupo sospechoso, y el que tiene la Puntuación de Silueta promedio más baja. Dado este resultado, podríamos considerar someter a todos los miembros del Grupo 0 al análisis forense. Sin embargo, el capital humano es costoso e investigar todas estas IP sería ineficiente. Por lo tanto, tiene más sentido para nosotros centrarnos en mejorar los resultados de nuestro agrupamiento primero, por lo que tenemos menos muestras para investigar.

Paso 6: Modificar K para optimizar los resultados del grupo

En términos generales, tiene sentido comenzar una sesión de agrupamiento con *k*-means con el hiperparámetro *k* establecido de forma tal que cree al menos dos grupos. Después de eso, puede iterar valores más altos de *k* hasta que sus grupos estén bien formados y validados para reflejar con precisión la distribución de las muestras. En este caso, realizamos los pasos tres y cuatro varias veces hasta que finalmente determinamos que 12 era el número óptimo para este conjunto de datos.

Vamos a seguir adelante y generar estos 12 grupos con el siguiente script:

`python cluster_vectors.py -c kmeans -n 12 -i secrepo.h5 -o secrepo.h5`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python cluster_vectors.py -c kmeans -n 12 -i secrepo.h5 -o secrepo.h5
Clustered samples into 12 clusters
Label 0 has 16 samples
Label 1 has 1994 samples
Label 2 has 223 samples
Label 3 has 73 samples
Label 4 has 123 samples
Label 5 has 79 samples
Label 6 has 53 samples
Label 7 has 225 samples
Label 8 has 60 samples
Label 9 has 319 samples
Label 10 has 42 samples
Label 11 has 647 samples
```

Paso 7: Repetir los procedimientos de inspección y validación

Una vez más, ejecutaremos un script para extraer la ID del grupo que ahora contiene la IP maliciosa:

`python label_notes.py -i secrepo.h5 | grep 70.32.104.50`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python label_notes.py -i secrepo.h5 | grep 70.32.104.50
6 70.32.104.50
```

Como podemos ver, la IP maliciosa es miembro del Grupo 6. Validemos este grupo con la Puntuación de Silueta:

`python stats_vectors.py secrepo.h5`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python stats_vectors.py secrepo.h5
Vectors shape: (3854, 17)
Minimum feature value: 0.0
Mean feature value: 0.128143090123
Maximum feature value: 1.0
Percentage of null values: 83.2259836991%

Minimum distance between vectors: 0.0
Mean distance between vectors: 0.592780335395
Maximum distance between vectors: 2.0

Number of labels: 12
Number of items in label 0: 16  (0.415153087701%) (avg dist: 0.233847802898) (avg silhouette: 0.797796650703)
Number of items in label 1: 1994  (51.7384535547%) (avg dist: 0.100538559594) (avg silhouette: 0.645049288869)
Number of items in label 2: 223  (5.78619615983%) (avg dist: 0.212557574151) (avg silhouette: 0.635324939352)
Number of items in label 3: 73  (1.89413596264%) (avg dist: 0.424114268165) (avg silhouette: 0.398054166997)
Number of items in label 4: 123  (3.1914893617%) (avg dist: 0.443824030386) (avg silhouette: 0.603191718161)
Number of items in label 5: 79  (2.04961037052%) (avg dist: 0.520204140369) (avg silhouette: 0.614395574605)
Number of items in label 6: 53  (1.37519460301%) (avg dist: 0.038767542786) (avg silhouette: 0.953514629418)
Number of items in label 7: 225  (5.8380902958%) (avg dist: 0.265812329702) (avg silhouette: 0.455568312l2)
Number of items in label 8: 60  (1.55682407888%) (avg dist: 0.461322425847) (avg silhouette: 0.382470554132)
Number of items in label 9: 319  (8.27711468604%) (avg dist: 0.184890513566) (avg silhouette: 0.495367654549)
Number of items in label 10: 42  (1.08977685522%) (avg dist: 0.398037851835) (avg silhouette: 0.483237183886)
Number of items in label 11: 647  (16.7877529839%) (avg dist: 0.14448543024) (avg silhouette: 0.40990207014l)

Minimum label centroid distance: 0.228328581566
Mean label centroid distance: 1.20139629193
Maximum label centroid distance: 1.87341191894
Overall Silhouette Score: 0.572507223029
```

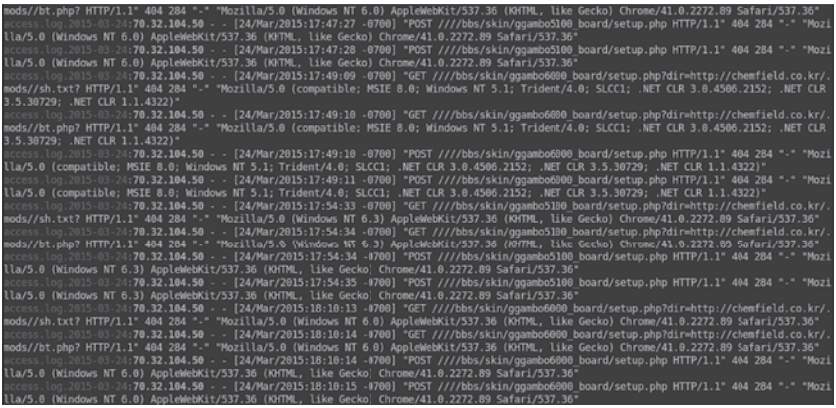
Como podemos ver, el Grupo 6 tiene una Puntuación de Silueta alta, lo que indica que todos los miembros son muy similares entre sí y a la IP que sabíamos desde un principio que era maliciosa. Nuestro siguiente paso debería ser ver qué han estado haciendo estas direcciones IP al rastrear su actividad en los registros de nuestro servidor web. Comenzaremos imprimiendo todas las muestras en el Grupo 6 utilizando el siguiente comando:

`python label_notes.py -i secrepo.h5 -l <label>`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python label_notes.py -i secrepo.h5 -l 6
6 95.168.199.227
6 74.208.16.118
6 213.165.70.246
6 181.65.186.34
6 181.65.186.35
6 212.227.18.39
6 192.99.18.191
6 87.106.187.164
6 114.202.2.31
6 184.107.137.250
6 112.218.68.155
6 37.9.169.17
6 180.179.212.185
6 82.98.160.235
6 222.122.56.211
6 180.179.212.214
6 46.252.18.54
6 77.232.91.201
6 211.47.181.38
6 203.58.0.155
6 71.6.150.241
6 41.193.5.54
6 112.175.50.226
6 202.154.23.202
6 70.32.104.50
6 91.142.215.248
6 201.175.20.65
6 185.27.238.194
6 198.144.188.26
6 178.33.226.103
6 212.34.151.164
```

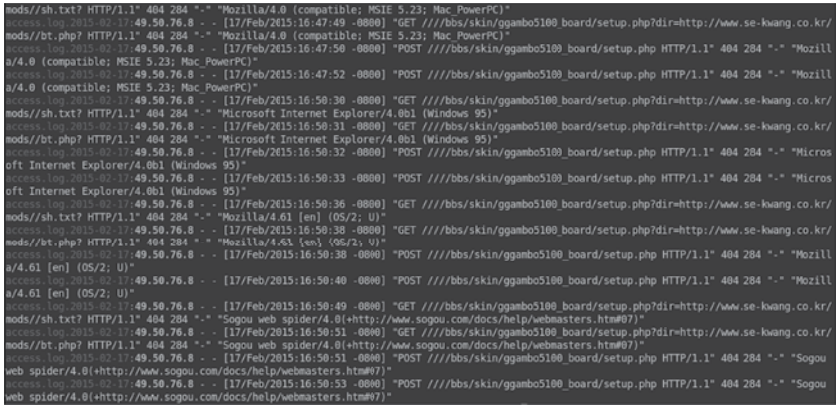
Ahora, podemos utilizar el comando grep para buscar en nuestros registros y mostrar las entradas en las que aparecen estas direcciones IP. Comenzaremos con nuestra conocida IP incorrecta:

`grep -ar 70.32.104.50 datasets/http/secrepo/www.secrepo.com/self.logs/`



Como podemos ver, esta IP ha intentado aprovechar las vulnerabilidades de inclusión de archivos remotos para instalar una carga útil de script PHP. Ahora intentemos con otro miembro del grupo sospechoso:

`grep -ar 49.50.76.8 datasets/http/secrepo/www.secrepo.com/self.logs/`



Esta IP también ha estado actuando con intenciones maliciosas. Debemos repetir este paso hasta que hayamos examinado a todos los miembros del Grupo 6 y hayamos completado nuestro análisis forense.

ANÁLISIS DE GRUPOS CON DBSCAN

Como ya hemos creado el archivo secretopo.h5 para nuestro ejemplo con *k*-means, saltaremos al Paso 3 y comenzaremos nuestra primera pasada de sesión de agrupamiento con DBSCAN. Comenzaremos configurando los hiperparámetros Eps y MinPts a 0,5 y 5 respectivamente. Como se señaló anteriormente, DBSCAN no requiere que pronostiquemos el número correcto de grupos por adelantado. Calculará la cantidad de grupos por sí solo en función de la densidad de vectores en el espacio de características y el tamaño de grupo mínimo.

Para generar estos grupos, ejecutaremos el siguiente script:

`python cluster_vectors.py -c dbscan -e 0.5 -m 2 -i secrepo.h5 -o secrepo.h5`

```
Label -1 has 854 samples
Label 0 has 1782 samples
Label 1 has 117 samples
Label 2 has 5 samples
Label 3 has 15 samples
Label 4 has 38 samples
Label 5 has 57 samples
Label 6 has 38 samples
Label 7 has 29 samples
Label 8 has 40 samples
Label 9 has 30 samples
Label 10 has 25 samples
Label 11 has 26 samples
Label 12 has 16 samples
Label 13 has 12 samples
Label 14 has 6 samples
Label 15 has 15 samples
Label 16 has 16 samples
Label 17 has 10 samples
Label 18 has 25 samples
Label 19 has 54 samples
Label 20 has 25 samples
Label 21 has 8 samples
Label 22 has 34 samples
Label 23 has 42 samples
Label 24 has 21 samples
Label 25 has 9 samples
Label 26 has 17 samples
Label 27 has 11 samples
Label 28 has 44 samples
Label 29 has 8 samples
Label 30 has 7 samples
```

Como podemos ver, DBSCAN ha creado 62 grupos y dejó 854 muestras sin asignar. Cada uno de estos 854 puntos de ruido puede estar asociado a actividad maliciosa. Podríamos volver a nuestros registros ahora e investigarlos a todos, pero esto llevaría mucho tiempo y sería ineficaz. En cambio, aumentaremos nuestra configuración de Eps de 5 a 6. Esto debería producir menos grupos y también una cantidad menor de muestras potencialmente sospechosas.

Aplicaremos la nueva configuración de hiperparámetro con el siguiente comando:

`python cluster_vectors.py -c dbscan -e 6 -m 5 -i secrepo.h5 -o secrepo.h5`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python cluster_vectors.py -c dbscan -e 6 -m 5 -i secrepo.h5 -o secrepo.h5
Clustered samples into 12 clusters
Label -1 has 25 samples
Label 0 has 3570 samples
Label 1 has 66 samples
Label 2 has 42 samples
Label 3 has 13 samples
Label 4 has 15 samples
Label 5 has 7 samples
Label 6 has 40 samples
Label 7 has 53 samples
Label 8 has 7 samples
Label 9 has 11 samples
Label 10 has 5 samples
```

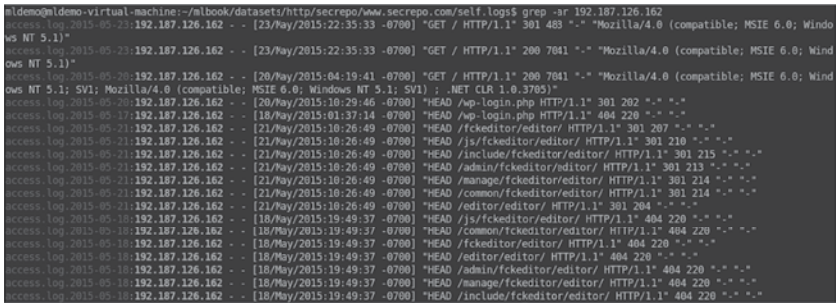
Esta vez, DBSCAN generó 11 grupos y solo 25 puntos de ruido, un número mucho más manejable. Pasaremos por alto los pasos de validación e inspección del grupo que describimos anteriormente para *k*-means y saltaremos para comenzar a investigar el comportamiento de estas 25 muestras sospechosas. Comenzaremos enumerando estas muestras con el siguiente comando:

`python label_notes.py -i secrepo.h5 -l -1`

```
mldemo@mldemo-virtual-machine:~/mlbook$ python label_notes.py -i secrepo.h5 -l -1
-1 66.117.9.219
-1 52.5.121.103
-1 52.8.91.105
-1 122.237.72.179
-1 107.20.245.168
-1 185.92.73.125
-1 74.6.254.140
-1 136.243.36.82
-1 180.149.143.26
-1 74.6.254.95
-1 37.110.10.31
-1 188.143.232.11
-1 212.124.109.166
-1 52.6.2.64
-1 23.253.252.202
-1 136.243.48.85
-1 108.45.93.78
-1 52.0.175.28
-1 68.180.228.155
-1 217.91.57.119
-1 54.151.42.39
-1 164.85.131.130
-1 192.187.126.162
-1 111.13.102.4
-1 188.143.232.10
```

Ahora, podemos usar el siguiente script para hacer grep a través de nuestros archivos de registro y descubrir qué han estado haciendo estas IP:

`grep -ar 192.187.126.162 datasets/http/secrepo/www.secrepo.com/self.logs/`



Como resultado, la gran mayoría de estos miembros del grupo se agruparon debido a un comportamiento extraño o fallas temporales en lugar de intentos de vulnerabilidad de seguridad. Sin embargo, descubrimos una serie de bots sondeando en busca de servidores vulnerables. Por ejemplo, 192.187.126.162 estaba sondeando nuestros servidores para determinar si se instalaron WordPress o fckeditor. El pirata podría haber usado bots muy similares a estos para identificar a Mossack Fonseca como un objetivo vulnerable. Si MF hubiera utilizado el agrupamiento para detectar estos sondeos y luego hubiera corregido estas vulnerabilidades, es posible que el incidente de filtración de los Panama Papers no se hubiera producido.

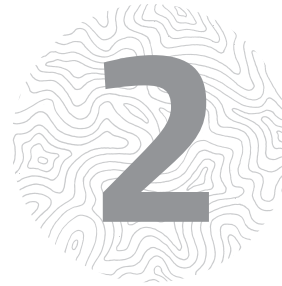
Conclusiones del agrupamiento

Como hemos visto, el agrupamiento proporciona un enfoque matemáticamente riguroso para detectar patrones y relaciones entre redes, aplicaciones, archivos y datos de usuarios que podrían ser difíciles o imposibles de salvaguardar de otra manera.

Sin embargo, nuestra historia analítica tan solo comienza con el agrupamiento. En los siguientes capítulos, describiremos algunas de las otras técnicas estadísticas, de Inteligencia Artificial y de Aprendizaje Automático que comúnmente empleamos para desarrollar y desplegar nuestras soluciones de seguridad de redes. Por ahora, sin embargo, estas son las conclusiones clave sobre el agrupamiento:

- El análisis de grupos se puede aplicar a prácticamente todo tipo de datos una vez que se han extraído y normalizado las características relevantes.
- En el análisis de grupos, la similitud entre las muestras y su pertenencia al grupo resultante se determina midiendo la distancia entre vectores en función de su ubicación en el espacio de características. Se pueden aplicar diversas métricas de distancia, entre las que se incluyen la euclidiana, Manhattan, de coseno y más.
- *K*-means y DBSCAN son fáciles de usar, computacionalmente eficientes y ampliamente aplicables a una variedad de situaciones de agrupamiento. Sin embargo, ambos métodos son vulnerables a la "maldición de la dimensionalidad" y pueden no ser adecuados al analizar espacios de características dimensionales sumamadamente altos.
- Los resultados del agrupamiento deben validarse estadísticamente y evaluarse con cuidado con respecto a las amenazas de seguridad del mundo real. Esto requiere una amplia experiencia sobre dominios y una comprensión profunda de las capacidades, los pros y los contras de cada método de agrupamiento.

- El agrupamiento es particularmente útil en la exploración de datos y el análisis forense, ya que nos permite examinar grandes cantidades de datos para identificar valores atípicos y anomalías. También podemos someter nuestras muestras y resultados de agrupamiento a otras formas de análisis como las que exploraremos en los capítulos siguientes.



Clasificación utilizando los algoritmos de regresión logística y árbol de decisión

Los humanos empleamos una amplia variedad de estrategias cognitivas para dar sentido al mundo que nos rodea. Una de las más útiles es nuestra capacidad de asignar objetos e ideas a categorías discretas basadas en relaciones abstractas entre sus rasgos y características. En muchos casos, las categorías que utilizamos son binarias. Ciertos alimentos son buenos para comer, otros no. Ciertas acciones son moralmente correctas, mientras que otras son moralmente incorrectas. Categorías como estas nos permiten hacer generalizaciones sobre objetos y acciones que ya conocemos para predecir las propiedades de objetos y acciones que son completamente nuevos para nosotros.

Al presentarse un objeto ovalado con cáscara amarilla, un interior suave y un olor intenso y dulce, podríamos recurrir a nuestro conocimiento previo para predecir que pertenece a la categoría "fruta". Podríamos probar la precisión de nuestra predicción llevando el objeto a una frutería. Si encontramos un

recipiente lleno de objetos similares etiquetados como "mangos" podríamos concluir que nuestra predicción fue correcta. Si es así, podríamos generalizar, a partir de nuestro conocimiento de la fruta, para predecir que el mango tiene un sabor agradable y ofrece beneficios nutricionales. Podríamos aplicar este conocimiento categórico para decidir si consumimos el mango. Este proceso de asignar un objeto desconocido a una categoría conocida para tomar decisiones informadas se denomina *clasificación*.

En el Aprendizaje Automático, la clasificación se refiere a un conjunto de métodos computacionales para predecir la probabilidad de que una muestra determinada pertenezca a una clase predefinida, como si un correo electrónico pertenece a la clase "spam" (correo no deseado) o una conexión de red es benigna o está asociada a una botnet. Estos son ejemplos de un problema de clasificación binaria, por ejemplo, uno con solo dos clases de resultado: "spam" y "no spam", "botnet" o "benigna". Por convención, las muestras que poseen el atributo que estamos investigando (por ejemplo, que un correo electrónico es spam) se etiquetan como pertenecientes a la clase "1" mientras que las muestras que no poseen este atributo (por ejemplo, correo que *no* es spam) como perteneciente a la clase "0". Estas etiquetas de clase 1 y 0 a menudo se denominan casos positivo y negativo respectivamente.

La clasificación también se puede usar con problemas en los que:

- una muestra puede pertenecer a múltiples clases al mismo tiempo. Por ejemplo, al mango que identificamos anteriormente se le pueden asignar etiquetas correspondientes a las clases de fruta, amarillo, tropical, etc.;

- estamos realizando una clasificación *multinomial*, en lugar de binaria. En este caso, se asigna una muestra a una clase entre un conjunto de tres o más. Por ejemplo, podríamos querer clasificar una muestra de correo electrónico como perteneciente a la Clase 1 (benigna), Clase 2 (spam) o Clase 3 (una vulnerabilidad de seguridad de tipo phishing).

A efectos de este capítulo, sin embargo, solo consideraremos los problemas de clasificación binaria.

Los algoritmos utilizados para realizar la clasificación se conocen como "clasificadores". Existen numerosos clasificadores disponibles para resolver problemas de clasificación binarios, cada uno con sus propias fortalezas y debilidades. En este capítulo, examinaremos los métodos y principios asociados con dos de los clasificadores más comunes: regresión logística y árboles de decisión, tal como se proporciona en el kit de herramientas de scikit-learn.

Aprendizaje supervisado frente a aprendizaje no supervisado

La clasificación es un ejemplo de *aprendizaje supervisado*, en el cual un analista crea un modelo con muestras que ya han sido identificadas, o *etiquetadas*, con respecto a la propiedad que se investiga. En el caso del spam, por ejemplo, el analista crea un modelo de clasificación utilizando un conjunto de datos de muestras que ya han sido etiquetadas como spam (el caso positivo) o como *no* spam (el caso negativo). Aquí, el trabajo del clasificador es determinar cómo se pueden usar los atributos de las características de cada clase para predecir la clase de muestras nuevas sin etiquetar. En contraposición, el agrupamiento es un ejemplo de aprendizaje no supervisado,

en el que deben descubrirse las propiedades que distinguen un grupo de muestras de otro.

No es raro usar métodos supervisados y no supervisados en combinación. Por ejemplo, el agrupamiento se puede usar para segregar vectores de tráfico de red en grupos distintos. Luego, los miembros del equipo forense pueden investigar a los miembros de los grupos sospechosos para ver si han realizado algún tipo de actividad indeseable en la red. Si es así, los vectores asociados con esta actividad pueden etiquetarse como pertenecientes a la Clase 1 (por ejemplo, como tráfico de botnet) mientras que todos los demás vectores pueden etiquetarse como Clase 0 (por ejemplo, como actividad benigna). Una vez etiquetados, los vectores pueden ser utilizados por un clasificador para crear un modelo que prediga si un nuevo vector no etiquetado es benigno o un miembro de una red de robots informáticos.

Para producir un modelo preciso, los analistas necesitan salvaguardar una cantidad suficiente de datos que hayan sido correctamente muestreados, vectorizados y etiquetados. Estos datos suelen dividirse en dos o tres conjuntos distintos para el entrenamiento, la validación y las pruebas. Es preferible dividir en tres conjuntos cuando hay suficientes datos disponibles. En cualquier caso, el conjunto de entrenamiento es típicamente el subconjunto más grande, que comprende entre el 70 al 90 % del total de las muestras. Como regla general, cuanto mayor sea el conjunto de entrenamiento, más probable es que el clasificador produzca un modelo preciso. Sin embargo, siempre se deben retener suficientes datos de prueba para realizar una evaluación confiable de la precisión del modelo.

Una sesión de clasificación normalmente se desarrolla a través de cuatro fases:

1. Una fase de *entrenamiento* o "aprendizaje" en la cual el analista crea un modelo aplicando un clasificador

a un conjunto de datos de entrenamiento. El conjunto de entrenamiento consta de dos archivos: una matriz de datos de muestras y un vector de etiquetas (una etiqueta para cada fila en la matriz). Tanto la regresión logística como los algoritmos del árbol de decisión proporcionan un conjunto de hiperparámetros que el analista puede ajustar para controlar cómo se crean los modelos resultantes.

2. Una fase de *validación* en la que el analista aplica los datos de validación al modelo para evaluar su precisión, y utiliza diversos procedimientos para optimizar la configuración de hiperparámetros del algoritmo. Para obtener más información sobre estos métodos de optimización, consulte los enlaces provistos en la sección de recursos al final de este capítulo.
3. Una fase de *prueba* para evaluar la precisión del modelo con datos de prueba que fueron retenidos de los procesos de entrenamiento y validación. El analista ejecuta los vectores de prueba en el modelo y luego compara la pertenencia a una clase prevista de cada muestra de prueba con su pertenencia real. Si los resultados cumplen con los umbrales de precisión y rendimiento requeridos, el analista puede pasar a la fase de implementación. De lo contrario, pueden regresar a la fase de entrenamiento para refinar y recrear el modelo.
4. Una fase de *implementación* en la que se aplica el modelo para predecir la pertenencia a una clase de datos nuevos sin etiquetar.

En la práctica, un analista puede entrenar y probar múltiples modelos usando diferentes algoritmos y configuraciones de hiperparámetros. Luego, pueden comparar los modelos y elegir

uno que ofrezca la combinación óptima de precisión y el uso más eficiente de los recursos informáticos.

Tenga en cuenta: Los métodos utilizados para evaluar la precisión del modelo durante la fase de prueba son sustancialmente similares a los aplicados típicamente durante la fase de validación. Como se señaló anteriormente, la realización de una etapa de validación por separado, aunque deseable, solo es factible cuando hay suficientes datos disponibles para crear un entrenamiento, una validación y pruebas de subconjuntos de la magnitud adecuados. Por simplicidad, por lo tanto, omitiremos referencias adicionales a la fase de validación y reservaremos nuestra discusión de la evaluación del modelo a la sección del capítulo sobre pruebas. Además, dado que los conjuntos de datos en nuestros propios ejemplos prácticos son relativamente pequeños, solo mostraremos los procedimientos de entrenamiento y prueba.

Desafíos de la clasificación

Los clasificadores a menudo pueden producir excelentes resultados en condiciones favorables. Sin embargo, este no siempre es el caso. Por ejemplo:

- Puede ser sumamente difícil para los analistas obtener un conjunto suficientemente grande y clasificado con precisión de datos etiquetados.
- La precisión se verá comprometida si las muestras seleccionadas no reflejan con precisión la prevalencia real de los casos positivos y negativos. Además, la proporción de casos positivos a negativos debe estar dentro de las tolerancias aceptables para que la clasificación funcione bien. En términos generales, sin embargo, los analistas normalmente pueden crear modelos precisos si cuentan con una cantidad suficiente de datos de cada clase.

Clasificación por regresión logística (RL)

En el capítulo anterior, presentamos el concepto de espacio de características y describimos cómo se puede evaluar la similitud de los grupos de vectores en función de la distancia entre ellos y sus vecinos más cercanos. El espacio de características también juega un papel en la regresión logística, sin embargo, los mecanismos utilizados para evaluar la similitud y asignar vectores a las clases operan de forma algo diferente.

Matemáticamente, la RL es un clasificador lineal, lo que significa que utiliza líneas rectas y planos para distinguir vectores pertenecientes a una clase de otra. En la clasificación binaria, el objetivo del analista es crear un modelo que registre el espacio de características en dos regiones, donde cada región incluya vectores que pertenecen a una sola clase. Este proceso

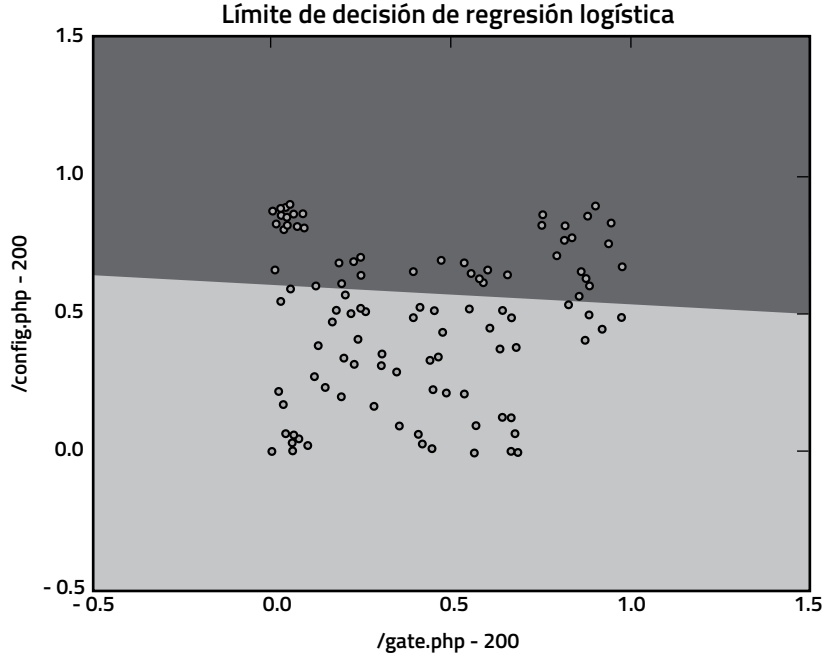


FIGURA 2.1: Límite de decisión en la regresión logística

se conoce como *ajustar* los datos. En la RL, la línea o el plano que separa una región de otra se denomina *límite de decisión*. A continuación se muestra un ejemplo de un límite de decisión que separa vectores que pertenecen a dos clases diferentes:

La RL incluye varias funciones de *resolución* diferentes para determinar la ubicación del límite de decisión y asignar vectores a las clases. En el análisis a continuación, describiremos el solucionador *liblinear* y cómo aplica el método de *descenso coordenado* para lograr esto.

LA FUNCIÓN DE LA PONDERACIÓN EN LA REGRESIÓN

La *ponderación en la regresión* desempeña una función central en la determinación de cuánto contribuye cada característica y valor de característica para que un vector determinado pertenezca a una clase. Esto se logra multiplicando cada valor de característica por su ponderación correspondiente como se muestra a continuación.

	Característica 1	**Característica 2**	**Característica 3**
Valor de característica	250	14	42
Ponderación de regresión	0,05	75	-6
Producto	12,5	1050	-252

Los valores de ponderación positivos y negativos contribuyen a las clasificaciones de Clase 1 y Clase 0, respectivamente. Como podemos ver, el gran valor positivo de la Característica 1 podría indicar que tiene una gran influencia en la predicción de que esta muestra pertenece a la clase positiva. Sin embargo, el impacto de la Característica 1 se ve significativamente disminuido por su baja ponderación de regresión. En consecuencia, se considera entonces que la Característica 1 contribuye mínimamente al

potencial de esta muestra para etiquetarse como de Clase 1. Por el contrario, la contribución de la Característica 2 se ha incrementado significativamente debido a una mayor ponderación de regresión. A su vez, la influencia de la Característica 3 se ha multiplicado por seis, pero en la dirección de predecir la pertenencia a una clase negativa.

En la práctica, el producto de una única combinación de valor de característica/ponderación de regresión probablemente tenga un efecto insignificante en la predicción de la pertenencia a una clase de muestras, ya que cada vector puede contener valores de cientos o incluso miles de características. En cambio, es el total de estos cálculos lo que resulta significativo.

Para predecir una clase, la RL suma todos los productos junto con un valor de *tendencia* calculado. Si la suma total es mayor o igual que cero, la RL predecirá la muestra como perteneciente a la Clase 1. Si la suma es menor que cero, la RL predecirá la muestra como un miembro de Clase 0. En nuestro ejemplo hipotético, la RL agrega nuestro valor de tendencia (+5) a la suma de nuestros productos vectoriales (12,5) + (1050) + (-252) para un total de 815,5. Como la suma es mayor que cero, nuestra muestra se predeciría como perteneciente a la Clase 1. Ahora podríamos comparar la pertenencia a una clase prevista de la muestra con la pertenencia a una clase real para comprobar si nuestras ponderaciones de regresión eran correctas.

La mayor parte de la fase de entrenamiento de una sesión de RL está dedicada a optimizar estas ponderaciones. Primero, sin embargo, se debe aplicar un conjunto inicial de ponderaciones. Existen numerosos métodos para hacerlo. Por ejemplo, los valores iniciales de pondración se pueden establecer arbitrariamente usando un generador de números aleatorios. En última instancia, el clasificador casi siempre calculará los valores óptimos si cuenta con suficiente tiempo de cómputo.

LA FUNCIÓN DE LA REGULARIZACIÓN Y LOS PARÁMETROS DE PENALIZACIÓN

Como se señaló en el capítulo de agrupamiento, las características con valores grandes pueden distorsionar los resultados del modelo, un problema que los analistas normalmente abordan a través de la normalización. Las ponderaciones de regresión con valores muy grandes pueden ocasionar distorsiones similares. En consecuencia, el algoritmo de RL proporciona una serie de parámetros de *penalización* que los analistas pueden utilizar para mitigar estos efectos.

Por ejemplo, los analistas pueden usar el parámetro de penalización *C* para comprimir el rango de ponderaciones de regresión de la misma manera que utilizan la normalización para comprimir los valores de las características. Este proceso se conoce como *regularización*. C controla la magnitud de los valores de la ponderación. Los modelos con rangos de ponderación sumamente altos pueden ser muy útiles en la predicción de la clase de vectores de entrenamiento, pero producen resultados inferiores cuando se aplican a los datos de prueba. Se dice que modelos como estos *sobreajustan* los datos. Esta disparidad en la precisión de un modelo puede ser un indicio importante para el analista de que se necesita una regularización más agresiva.

La regularización también puede ser útil cuando el analista sospecha que un solucionador se está enfocando excesivamente en un conjunto pequeño de características altamente influyentes. Al regularizar, el analista puede obligar al solucionador a incorporar características adicionales de una manera controlada y medida.

La regularización también se puede usar para controlar qué características pueden influir en el clasificador al calcular las ponderaciones de regresión. Esto se logra utilizando los parámetros de penalización *L1* y *L2*. Los dos parámetros

funcionan de manera diferente. L1 establece un umbral que determina qué tan agresiva debe ser la RL en la eliminación de características con un poder de predicción comparativamente bajo. Cuanto mayor sea la ponderación asignada a L1, más características serán excluidas. En contraste, L2 minimiza el impacto de un grupo de características altamente correlacionadas de modo que su influencia colectiva no distorsione los resultados.

Echemos un vistazo ahora a la secuencia de pasos y metodología que comprende la fase de entrenamiento de una típica sesión de regresión logística con scikit-learn.

FASE DE ENTRENAMIENTO DE REGRESIÓN LOGÍSTICA

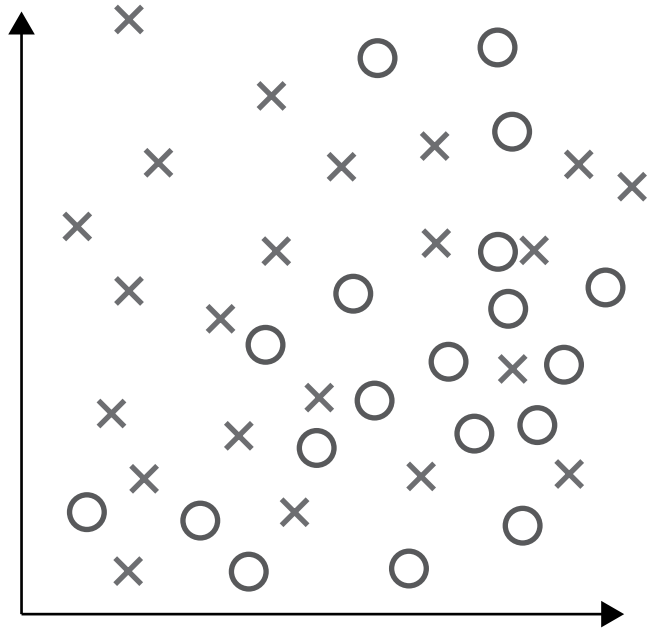
Durante esta fase, el objetivo principal del analista es ajustar los datos produciendo un conjunto optimizado de ponderaciones de regresión. Durante la siguiente fase de prueba, las ponderaciones se aplicarán para predecir a qué clase pertenece cada vector de prueba. A continuación, los resultados estarán sujetos a funciones de validación que determinan con qué precisión las asignaciones de clase previstas coinciden con las etiquetas de clase conocidas.

Paso 1: Importación de datos y configuración de sesión

El analista comienza por importar dos archivos:

1. una matriz de muestras de entrenamiento normalizadas;
2. un vector de etiquetas que define a qué clase pertenece cada muestra.

Como se detalla a continuación, cada vector se puede representar gráficamente como un punto en el espacio de características de manera que su pertenencia a la clase se indique visualmente.



Vectores de dos clases diferentes en el espacio característico
FIGURA 2.2: Vectores etiquetados en el espacio de características

Paso 2: Regularización y optimización de la ponderación de regresión
Se asigna un conjunto inicial de ponderaciones de regresión, y el analista aplica una función de *probabilidad*. Esto compara el número real de casos positivos y negativos con la cantidad total prevista utilizando las ponderaciones iniciales. La puntuación resultante se usa para calcular un ajuste positivo o negativo al valor de cada ponderación. El analista puede controlar la magnitud de este ajuste para cada característica utilizando un *parámetro de velocidad de aprendizaje*.

Tras realizar ciclos de cálculo repetidos, las ponderacione de regresión se acercarán de manera gradual e incremental a sus valores óptimos. Después de cada optimización, el analista puede experimentar con diferentes configuraciones de parámetros de penalización y luego evaluar el modelo resultante.

Sin embargo, este enfoque de fuerza bruta para la optimización requiere de tiempo y cálculos. En algún momento, las mejoras incrementales en la precisión del modelo quizás ya no justifiquen refinamientos adicionales. Cuando eso ocurra, el analista puede elegir finalizar el proceso de entrenamiento y pasar a la fase de prueba.

Paso 3: Asignación de puntuaciones de probabilidad a las predicciones de clase

Recuerde que la clasificación de un vector se calcula sumando todos sus productos de valor/ponderación de características junto con el valor de tendencia; el vector se asigna a la Clase 1 si la suma es igual o mayor que cero, y a la Clase 0 si no lo es. Sin embargo, la RL es intrínsecamente un método para predecir la probabilidad de que un vector dado pertenezca a una clase particular. Por lo tanto, la RL incluye una *función logit* que convierte el resultado de la clasificación en un punto a lo largo de una curva de probabilidad que va de cero a uno, como se muestra a continuación.

Cuanto más cerca esté el punto de una puntuación de probabilidad que se aproxima a y=1, más sólida será la predicción de que la muestra pertenece a la Clase 1 (el caso positivo). Del mismo modo, cuanto más cerca esté el punto de p=0, más firmemente se predecirá que pertenece a la Clase 0 (el caso negativo). Los resultados que se aproximan a los p=0,5 puntos desde cualquier dirección se vuelven cada vez más ambiguos con respecto a las predicciones de clase. Como se muestra arriba, el límite de decisión está representado por la ubicación p=0,5 a lo largo de la curva de probabilidad. Si una muestra se ubicara en esta coordenada, entonces sería igualmente probable que pertenezca a cualquiera de las clases, haciendo imposible una predicción segura.

Paso 4: Exportación del modelo de regresión logística

El modelo de clasificación resultante ahora se puede exportar y someter a prueba. Matemáticamente, el modelo consiste en el valor de tendencia junto con un vector de ponderaciones de regresión. Una vez que se han calculado, se pueden calcular las coordenadas del límite de decisión. En el caso más simple de un problema de clasificación con solo dos características, la ecuación toma la forma x2 = -(m1/m2) x1 +b en la que *x1* y *x2* son los valores de las características, *m1* y *m2* son sus respectivas ponderaciones de regresión y *b* es el valor de tendencia.

En la práctica, sin embargo, esta ecuación se expande para sumar los productos de cada combinación de ponderación de regresión/valor de característica. La forma del límite de decisión resultante está determinada por el número de características que se utilizan para la clasificación. Si hay dos características, el límite de decisión comprenderá una línea. Si hay tres dimensiones, comprenderá un plano. En espacios dimensionales superiores a estos, el límite de decisión comprenderá un *hiperplano,* y añadirá una dimensión para cada característica adicional.

Si se ha calculado un valor de tendencia distinto de cero, el punto de origen para este límite de decisión se desplazará por el número de unidades especificadas. En el siguiente ejemplo, el punto de origen para el límite de decisión se ha desplazado cinco unidades hacia arriba a lo largo del eje x2.

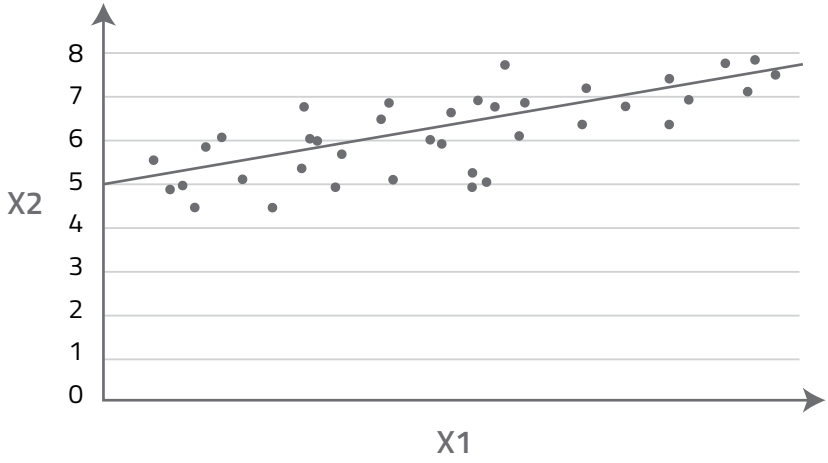


FIGURA 2.3: Regresión logística con una tendencia de 5

FASE DE PRUEBA DE REGRESIÓN LOGÍSTICA

Durante esta fase, el analista evalúa el modelo exponiéndolo a datos que no había visto antes (es decir, los vectores de prueba) y luego mide la precisión de sus predicciones. Uno de los métodos de validación más comunes emplea una función de *matriz de confusión* que examina cada muestra sucesivamente y luego compara su pertenencia de clase prevista con su etiqueta de clase real. A continuación, asigna la predicción para esa muestra a una de cuatro categorías:

- Un *verdadero positivo* (VP) es una muestra que se ha previsto correctamente como perteneciente a la Clase 1. En el siguiente ejemplo, hay 150 verdaderos positivos.
- Un verdadero negativo (VN) es una muestra que se ha previsto correctamente como perteneciente a la Clase 0. En el siguiente ejemplo, hay 900 verdaderos negativos.

- Un *falso positivo* (FP) es una muestra que se ha previsto que pertenece a la Clase 1, pero que en realidad pertenece a la Clase 0. En el siguiente ejemplo, hay 50 falsos positivos .
- Un *falso negativo* (FN) es una muestra que se ha previsto que pertenece a la Clase 0, pero que en realidad pertenece a la Clase 1. En el siguiente ejemplo, hay 35 falsos negativos.

Ejemplo de matriz de confusiones	**Previsto por modelo**		
Pertenencia de clase	**Real**	**Clase 0**	**Clase 1**
Muestras que pertenecen a la Clase 0	950	900	50
Muestras que pertenecen a la Clase 1	**185**	**35**	**150**
Total de muestras	1135	935	200
Total de verdaderos positivos	150		
Total de verdaderos negativos	900		
Total de falsos positivos	50		
Total de falsos negativos	35		

Una vez que se ha creado la matriz, los analistas pueden usarla para calcular una variedad de métricas de validación. Dos de las más comunes son *precisión* y *recuperación*.

- La **precisión**, que mide la velocidad a la que la predicción positiva de un modelo es correcta, por ejemplo, la proporción de mensajes que se predice que son spam y que en realidad lo son. Esto se calcula dividiendo el número de verdaderos positivos por la suma de los verdaderos positivos más los falsos positivos. En este ejemplo, la precisión es 0,75.
- La **recuperación** mide la velocidad a la que el modelo clasifica correctamente un caso positivo, por ejemplo, la proporción de mensajes de spam reales que se reconocen correctamente como spam. La recuperación, a veces

denominada "tasa de verdadero positivo", se calcula al dividir el número de verdaderos positivos por la suma de los verdaderos positivos más los falsos negativos. En este ejemplo, la puntuación de recuperación es 0,81.

Los analistas también pueden medir la precisión general de un modelo y los errores que comete al clasificar los casos negativos. Las métricas incluyen:

- **La precisión media**, que se calcula al sumar el número de verdaderos positivos y verdaderos negativos y luego dividir el resultado por el número total de muestras. En este ejemplo, la tasa de precisión media es de aproximadamente 0,93.
- **La tasa de clasificación errónea** (también denominada "tasa de error"), que se calcula restando la tasa de precisión media a 1. En este ejemplo, la tasa de error es de aproximadamente 0,07.
- **La tasa de falsos positivos**, que mide con qué frecuencia el modelo predice una muestra de clase negativa como positiva, por ejemplo, clasificando un mensaje benigno como spam. Esto se calcula dividiendo el número de falsos positivos por el número de negativos reales. En este ejemplo, la tasa de falsos positivos es de aproximadamente 0,05.
- La **especificidad**, que mide con qué frecuencia la predicción negativa de un modelo es correcta, por ejemplo, la proporción de mensajes que se predice son benignos y que en realidad lo son. Esto se calcula dividiendo el número de negativos verdaderos por el número de negativos reales. En este ejemplo, la tasa de especificidad es aproximadamente 0,95.

- La **prevalencia**, que mide la proporción de muestras positivas en el conjunto de muestras. Esto se calcula dividiendo el número de positivos reales por el número de muestras de prueba. En este ejemplo, la puntuación de prevalencia es 0,16. Ahora podemos ver por qué el muestreo juega un papel tan fundamental en la creación de un modelo preciso. Si la tasa de prevalencia está distorsionada con respecto a la verdad terreno, todos los demás cálculos también se distorsionarán, lo que dará como resultado un modelo con una precisión inferior.

En general, el modelo ilustrado aquí parece ser relativamente bueno, con una tasa de precisión cercana a 0,93 y una tasa de falsos positivos de aproximadamente 0,05. Sin embargo, las métricas también muestran que nuestro modelo es mejor para predecir con precisión los casos negativos (una tasa de especificidad cercana a 0,95) que los casos positivos (una tasa de recuperación de 0,81 y una tasa de precisión de 0,75). Esto puede ser aceptable en situaciones, como la detección de spam, en la que algunos falsos negativos no resulten particularmente dañinos. Sin embargo, cuando se trata de detectar una incursión grave en la red, un modelo como este probablemente requeriría una gran cantidad de reentrenamiento y reajuste para producir la precisión requerida. Como siempre, los umbrales de precisión deben estar determinados por la naturaleza del problema de clasificación y su impacto en la organización.

EVALUACIÓN DEL MODELO UTILIZANDO LAS CURVAS CARACTERÍSTICA OPERATIVA DEL RECEPTOR

Las curvas de Característica Operativa del Receptor (ROC, por sus siglas en inglés) proporcionan una forma práctica e intuitiva de evaluar la calidad de las predicciones de un modelo

y comparar la precisión de un modelo con otro para resolver un problema de clasificación binaria. Esto se logra creando un gráfico del *espacio ROC* en el que la tasa de verdaderos positivos (es decir, la Recuperación) se traza en el eje y contra la tasa de falsos positivos trazada en el eje x para cada posible umbral de clasificación.

Como se muestra a continuación, cuanto más preciso sea un modelo, más cerca estará su curva ROC del borde superior del cuadrante izquierdo del espacio ROC. Las curvas que se encuentran junto a la línea punteada "sin discriminación" representan casos en los que las predicciones del modelo no serían mejores que las producidas por una suposición aleatoria. A continuación se muestra una ilustración de las curvas ROC en el espacio ROC.

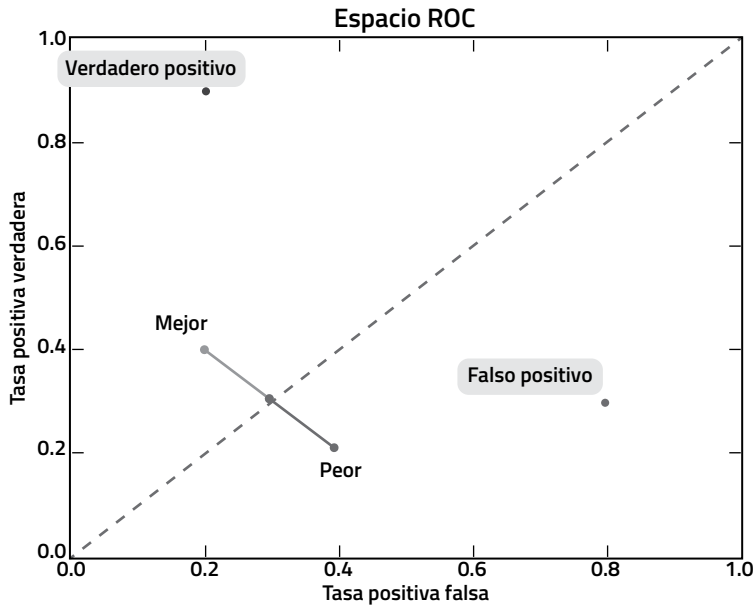


FIGURA 2.4: Descripción del espacio ROC

Para graficar nuestra curva, comenzamos contando el número de muestras que nuestro modelo calificó correcta e incorrectamente en cada umbral de probabilidad. En la siguiente figura, por ejemplo, los verdaderos negativos y verdaderos positivos se indican con las barras izquierda y derecha, respectivamente. La puntuación de probabilidad para cada una de estas predicciones se muestra en el eje x. El número de muestras correspondientes a cada puntuación de probabilidad se mide en el eje y. Podemos ver que hay muy poca superposición entre las barras izquierda y derecha, lo que indica que nuestro modelo ha hecho un buen trabajo clasificando con precisión nuestros casos positivos y negativos. Como era de esperar, la mayoría de los errores se encuentran adyacentes a la puntuación de probabilidad de 0,5, donde el modelo tuvo la mayor dificultad para diferenciar una clase de otra.

Para definir nuestra curva ROC, calculamos la tasa de verdaderos positivos (Recuperación) y la tasa de falsos positivos para un conjunto representativo de umbrales de probabilidad. La tasa de verdaderos positivos (TVP) nos proporcionará las coordenadas para el eje y. La tasa de falsos positivos (TFP) proporcionará las coordenadas para el eje x. Aquí nos limitaremos a calcular las coordenadas de la curva ROC para las muestras ubicadas en los umbrales de probabilidad de 0,4, 0,5 y 0,6. Comenzaremos con el umbral de 0,5.

TVP = N.° de verdaderos positivos ÷ la suma de los verdaderos positivos más los falsos negativos
= 0,92380952381 (la coordenada en el eje y)
TFP = N.° de falsos positivos ÷ el número de negativos reales
= 0,121428571429 (la coordenada en el eje x)

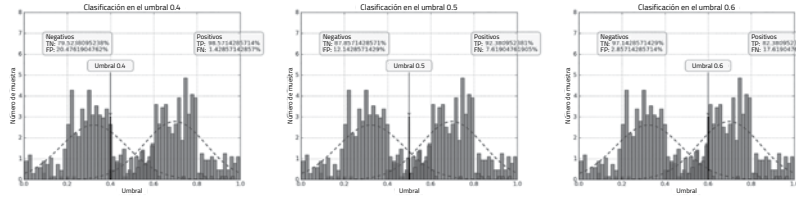


FIGURA 2.5: Resultados de clasificación en diferentes umbrales

Continuaremos de esta manera con los umbrales de probabilidad de 0,4 y 0,6 hasta que produzcamos la curva ROC que se muestra a continuación. La ubicación de la curva en el cuadrante superior izquierdo confirma que nuestro modelo es sumamente preciso en un amplio rango de umbrales de probabilidad.

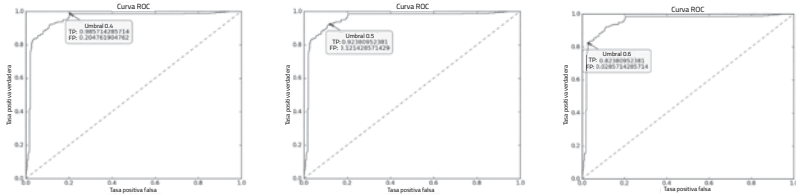


FIGURA 2.6: Curva ROC

Ahora repitamos el mismo proceso con un modelo que funciona mal. Comenzaremos por calcular las TVP y TFP para los mismos tres umbrales de probabilidad.

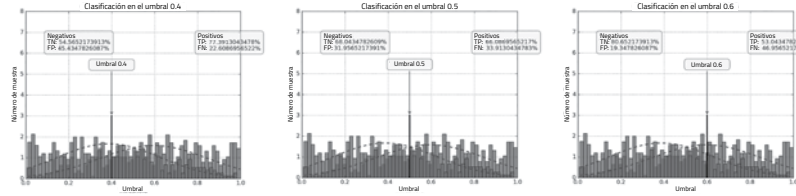


FIGURA 2.7: Resultados de clasificación en diferentes umbrales

Ahora, veamos la curva ROC resultante.

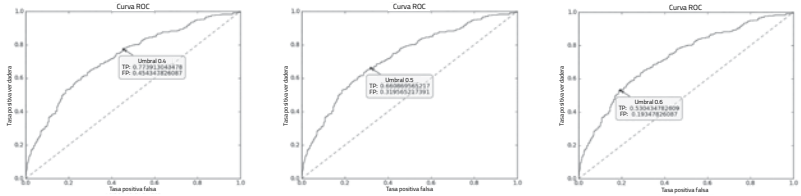


FIGURA 2.8: Curva ROC

Como podemos ver, la curva ROC se ubica mucho más cerca de la línea "sin discriminación", lo que indica que este modelo es mucho menos preciso en todo el rango de probabilidades.

ESCOLLOS Y LIMITACIONES DE LA REGRESIÓN LOGÍSTICA

El algoritmo de regresión logística en scikit-learn es eficiente y puede producir excelentes resultados dadas ciertas limitaciones:

- Los datos subyacentes deben ser intrínsecamente compatibles con la clasificación lineal. Operativamente, esto significa que debe ser posible clasificar vectores con precisión utilizando límites de decisión que dividen el espacio de características con líneas rectas y planos. Si el conjunto de datos no es separable linealmente de esta manera, se pueden emplear métodos más complejos de representación de características o el analista puede decidir utilizar un algoritmo de clasificación diferente.
- La RL es vulnerable a un ajuste insuficiente cuando los conjuntos de datos tienen muchos valores atípicos, características con valores desproporcionadamente grandes y conjuntos de características que están altamente correlacionados. La normalización y la regularización solo pueden compensar parcialmente estos efectos.

Clasificación a través de árboles de decisión (AD)

Los algoritmos del árbol de decisión determinan si un vector dado pertenece a una clase u otra al definir una secuencia de reglas de decisión "si-entonces-si no" que terminan en una predicción de clase. El tipo de algoritmo de AD seleccionado depende de si la etiqueta de clase tendrá valores categóricos o continuos.

- Si la etiqueta de la clase es categórica, por ejemplo, queremos predecir si una conexión de red determinada está asociada o no con una botnet, utilizaríamos la *clasificación de AD*.
- Si la etiqueta de la clase tiene valores continuos, por ejemplo, queremos predecir el mejor precio de venta para un nuevo producto, entonces utilizaríamos la *regresión de AD*.

El algoritmo proporcionado por scikit-learn es un *árbol de decisión ACR*, lo que significa que puede generar árboles de clasificación y de regresión. En este capítulo, nos centraremos en la clasificación de AD, ya que este método es adecuado para resolver los tipos de problemas de seguridad de redes con los que es más probable que se encuentre.

Matemáticamente, un AD es un clasificador *no lineal*. Esto significa que, a diferencia de la RL, el AD no crea límites de decisión con líneas rectas y planos. En cambio, divide el espacio de características en rectángulos que pueden contener apenas un solo vector cada uno. Como veremos, esta diferencia tiene implicaciones importantes con respecto al proceso de ajuste y cómo esto influye en la precisión del modelo resultante.

TERMINOLOGÍA DEL ÁRBOL DE DECISIÓN

Los bien llamados árboles de decisión utilizan raíces, ramas y hojas para producir predicciones de clase. Considere el ejemplo hipotético de un árbol de decisión utilizado para identificar URL maliciosas. Aquí, el caso positivo indica las URL asociadas con las vulnerabilidades de seguridad frente a las que no lo están.

Como es habitual, el árbol se creó de forma vertical comenzando con la raíz, que contiene todas las muestras de entrenamiento. En este caso, se trata de una combinación de URL maliciosas y benignas. Nuestro objetivo es dividir estas muestras en subconjuntos cada vez más "puros" basado en sus características y valores de características. Continuaremos de esta manera hasta que produzcamos uno o más subconjuntos de URL que pertenecen solo a una clase, o el proceso de creación del modelo finaliza debido a nuestra configuración de hiperparámetros.

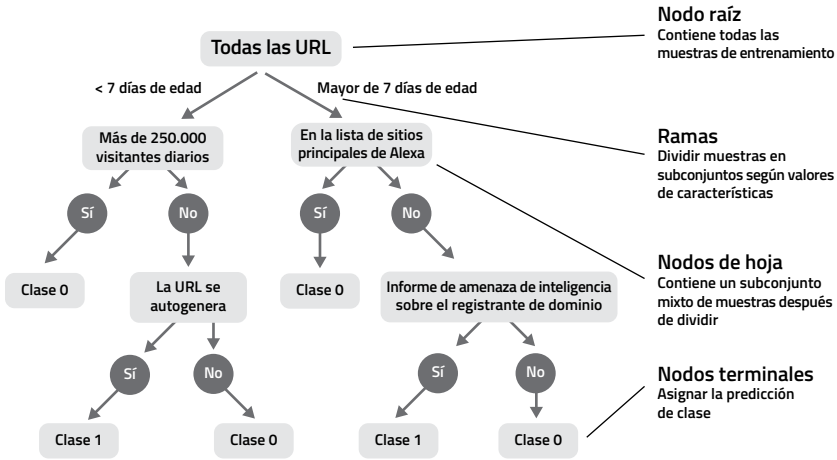


FIGURA 2.9: Ejemplo y terminología del árbol de decisión

Comenzaremos dividiendo nuestro conjunto de entrenamiento en dos ramas según la característica *antigüedad del dominio*. Tenemos dos valores de características para dividir:

"<7 días de antigüedad" y ">7 días de antigüedad". El subconjunto de muestras que coinciden con la primera división se copiará en el nodo "Más de 250.000 visitantes diarios". Del mismo modo, las muestras que coincidan con la segunda división se copiarán en el nodo "En la lista de sitios principales de Alexa". En cada nodo, encontraremos una mezcla de muestras que pertenecen a las clases 0 y 1 porque los nodos aún no son puros. Por lo tanto, se necesitarán ramas adicionales antes de que el árbol esté completo.

Continuaremos dividiendo en función de nuestros valores de características "URL generada automáticamente" e "Informe de monitoreo de amenazas sobre el titular del dominio" hasta que se complete el árbol. Al trabajar en la rama *izquierda* (menos de siete días de antigüedad), podemos predecir que una URL se clasificará como:

- **Maliciosa** (Clase 1) si menos de 250.000 visitantes por día accedían al sitio y la URL se autogeneraba.
- **Benigna** (Clase 0) si menos de 250.000 visitantes por día accedían al sitio pero la URL *no* se autogeneraba.

Al trabajar en la rama *derecha* (más de siete días de antigüedad), podemos predecir que una URL se clasificará como:

- **Maliciosa** (Clase 1) si *no* se incluyó en la lista de Sitios principales de Alexa y *se* emitió un informe de monitoreo de amenazas sobre el titular del dominio.
- **Benigna** (Clase 0) si:
  - Se *incluyó* en la lista de sitios principales de Alexa,

  O bien,
  - Si *no se incluyó* en la lista de Sitios principales de Alexa, pero no se emitió ningún informe de monitoreo de amenazas sobre titular del dominio.

PROCESO DE ENTRENAMIENTO DE UN AD

¿Cómo determina el clasificador de un AD qué característica y valores de característica se deben usar para crear cada rama? ¿Cómo entra en juego la pureza del nodo? Echemos un vistazo al proceso detallado por el cual el clasificador de un AD avanza en la evaluación de los puntos de división y la creación de árboles. Consideraremos el caso hipotético en el que tenemos 500 muestras en nuestro conjunto de entrenamiento:

1. El AD examina los 500 vectores y selecciona la primera característica disponible para evaluarla como un posible punto de división. Esto también se conoce como la *variable de división*.
2. A continuación, el AD examina el rango de valores asociados con esta característica y elige un *valor de división* inicial. En el siguiente ejemplo, los valores de las características van del 1 al 5. Por lo tanto, el AD podría comenzar eligiendo un valor de división que se encuentre entre 1 y 2. En este ejemplo, el AD establece el valor de división en 1,5.

Rango de valores de características	1	↓	2	↓	3	↓	4	↓	5
Ejemplo de variables de división		**1,5**		2,5		3,5		4,5	

3. El AD copia muestras con valores de características menores que 1,5 en el nodo hijo n.° 1 y muestras con valores de características iguales o superiores a 1,5 en el nodo hijo n.° 2. Ahora tenemos 300 muestras de clases mixtas en el nodo hijo n.° 1 y 200 muestras de clases mixtas en el nodo hijo n.° 2.

4. El AD evalúa la reducción de impurezas resultante calculando una puntuación de *impureza de Gini* para cada uno de los nodos hijos u *hojas*. A continuación, las dos puntuaciones se suman para producir una *puntuación de beneficio* general para ese candidato dividido. (Las puntuaciones de beneficios también se pueden calcular usando medidas de "entropía" en lugar de impurezas).
5. El AD elige el siguiente valor de división y repite el proceso. En este ejemplo, el nuevo valor de división es 2,5. Una vez más, las puntuaciones de Gini y de beneficios se calculan para cada uno de los nodos hijos candidatos. El AD continúa el proceso para cada valor de división hasta que se hayan calculado todas las puntuaciones de impurezas de Gini y beneficios para esa característica.

Rango de valores de características	1	↓	2	↓	3	↓	4	↓	5
Ejemplo de variables de división		1,5		**2,5**		3,5		4,5	

6. El AD selecciona la siguiente característica en el conjunto de características y luego repite los pasos 2 a 5, iterando en forma recurrente a través de cada valor de división y produciendo puntuaciones de impurezas de Gini y beneficios para cada nuevo conjunto de hojas candidatas. Este proceso continúa hasta que se hayan evaluado todas las características y los valores de división.
7. El AD elige la combinación de característica/valor de división que produjo la mejor puntuación de beneficio general y la usa para agregar nuevas ramas y hojas al árbol. En este nivel, cada una de las hojas resultantes todavía contiene una combinación de muestras de

Clase 0 y Clase 1. La proporción de muestras de cada clase se puede usar para calcular una puntuación de probabilidad. Por ejemplo, si una hoja contiene 100 muestras en las que 70 pertenecen a la Clase 0 y 30 pertenecen a la Clase 1, entonces podemos predecir que una nueva muestra producida por esta división tendrá un 30 % de probabilidad de pertenecer a la Clase 1 (30/100) y una probabilidad del 70 % de pertenecer a la Clase 0 (70/100).

8. El AD comienza el mismo proceso para cada una de las dos hojas nuevas hasta que el árbol esté *saturado* (es decir, todos los nodos son perfectamente puros) o bien encuentra un *criterio de detención* asociado con una de las configuraciones del hiperparámetro.
9. Si es necesario, el analista puede *podar* el árbol para eliminar ramas superfluas que agregan complejidad computacional al modelo sin mejorar su precisión.

A continuación, podemos ver las primeras cuatro ramas de un árbol de decisión que detecta sitios web que son sistemas de comando y control de botnets. Observe cómo cada nodo incluye un nombre de característica, recuentos de muestras, puntuación de impureza de Gini, valores de división y predicciones de clase. Más adelante en este capítulo, examinaremos el árbol completo y el proceso utilizado para generarlo con más detalle.

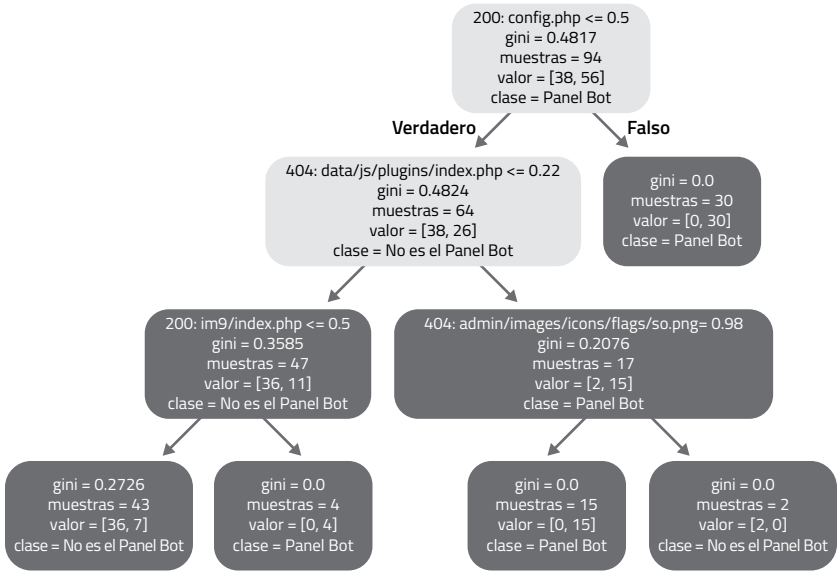


FIGURA 2.10: Detalles del árbol de decisión

CLASIFICACIÓN DEL ÁRBOLES DE DECISIÓN

Echemos un vistazo a la secuencia de pasos y metodología que comprende las fases de entrenamiento, prueba e implementación de una sesión típica de clasificación de árboles de decisión con scikit-learn.

Paso 1: Importación de datos y configuración de sesión

Una vez más, el analista comienza por importar dos archivos:

1. Una matriz de muestras de entrenamiento.
2. Un vector de etiquetas que define a qué clase pertenece cada muestra.

Paso 2: Generar y personalizar el árbol

Anteriormente notamos que la mayor parte del tiempo dedicado al desarrollo de un modelo de RL se destina a optimizar las ponderaciones de regresión aplicando diversos parámetros de

regularización. En el entrenamiento de un árbol de decisión, la mayor parte del tiempo de procesamiento del algoritmo está dedicado a optimizar los puntos de división para producir un modelo eficiente que cumpla con el umbral de precisión deseado sin sobreajustar los datos.

La configuración de hiperparámetros predeterminada que se proporciona con el clasificador de AD en scikit-learn a menudo será suficiente para lograr esto sin modificaciones. En tales casos, el analista simplemente aplica el solucionador, y el algoritmo produce el árbol de decisión y el conjunto asociado de puntuaciones de impureza de Gini y las curvas ROC.

En otros casos, especialmente cuando hay un gran número de características o numerosos valores atípicos, un analista puede necesitar experimentar con la modificación de la configuración de hiperparámetros para superar la tendencia de los clasificadores de árboles de decisión de "sobreajustar" datos, lo que genera un número excesivo de ramas. Durante el entrenamiento, el modelo resultante parecerá proporcionar un alto grado de precisión. Sin embargo, cuando se aplica a los datos de prueba, las puntuaciones de precisión serán mucho más bajas. Los analistas se refieren a esto como un *fallo al generalizar*. Es posible que los modelos que sobreajustan los datos también sean ineficientes en términos de cómputos y difíciles de implementar. Como se señaló anteriormente, los analistas pueden aplicar *criterios de detención* para limitar la producción de ramas utilizando hiperparámetros tales como:

- *max_depth*. Esto determina la cantidad máxima de ramas y hojas permitidas antes de que termine la creación del árbol.
- *min_samples_split*. Esto determina el número mínimo de muestras que un nodo debe poseer para ser elegible para una división.

- *min_samples_leaf*. Esto determina el número mínimo de muestras necesarias para crear una hoja hija.
- *max_leaf_nodes*. Esto determina cuántas hojas se pueden crear en total.

Los analistas también pueden aplicar la regularización mediante la configuración de hiperparámetros que modifican la influencia de cada característica en el proceso de división. Estas incluyen:

- *feature_importances*. Este parámetro se puede aplicar a una sola característica o a un grupo de características. Mientras más alta sea la configuración, más probable será que se utilice la característica como una variable de división.
- *n_features*. Esto determina cuántas características generales se pueden usar para crear el árbol.

Asignación de puntuaciones de probabilidad a las predicciones de clase

El algoritmo de árbol de decisión intrínsecamente genera una puntuación de probabilidad para cada predicción de clase en cada hoja en función de la proporción de muestras positivas y negativas que contiene. Esto se calcula dividiendo el número de muestras de cada clase por el número total de muestras en esa hoja. Considere, por ejemplo, una hoja que contiene 15 muestras de Clase 0 y 5 muestras de Clase 1. En esta situación, la probabilidad de que una muestra en esta hoja pertenezca a la Clase 1 es igual al número de muestras positivas (5) dividido por el número total de muestras (20) = 20 %. Del mismo modo, podemos predecir que cualquier muestra copiada a esta hoja tendrá un 80 % de probabilidad de pertenecer a la Clase 0.

Paso 3: Pruebas e implementación

Una vez que se ha creado el modelo de AD, se somete a los mismos procedimientos de prueba y validación que describimos anteriormente para la regresión logística. Una vez que el modelo ha sido suficientemente validado, puede implementarse para clasificar datos nuevos sin etiquetar.

ESCOLLOS Y LIMITACIONES DEL ÁRBOL DE DECISIÓN

Incluso con su configuración predeterminada de hiperparámetros, el algoritmo de AD en scikit-learn funciona bien y requiere relativamente poco esfuerzo previo para preparar los datos de muestras. También puede producir modelos que son altamente eficientes ya que emplean solo el subconjunto de características requeridas para clasificar en lugar de todo el conjunto de características. Por el contrario, los modelos de RL generalmente incluyen todas las características en la matriz de muestras, excepto aquellas eliminadas intencionalmente mediante la regularización.

Sin embargo, los árboles de decisión están sujetos a ciertos errores y limitaciones característicos.

- Al igual que todos los clasificadores de AD, la versión scikit-learn puede producir árboles excesivamente complejos que sobreajustan los datos y tienen un rendimiento deficiente cuando se exponen a los datos de prueba. En general, cuanto mayor sea el conjunto de características, más probable es que se produzca un sobreajuste.
- La implementación de puntos de división se determina conforme a las optimizaciones "locales" entre un nodo padre y su hoja hija en lugar de aquello que podría ser óptimo para el árbol en su totalidad. En consecuencia,

no hay forma de garantizar que un árbol determinado haya tomado la forma óptima.

- Los árboles de decisión pueden ser inestables. Pequeñas variaciones en los datos de muestras pueden causar que se produzca un árbol completamente diferente.

Los analistas pueden abordar estos problemas creando conjuntos de árboles a partir de subconjuntos aleatorios de los datos de entrenamiento. Cada árbol puede "votar" si una muestra dada pertenece a una u otra clase. La predicción que recibe la mayoría de los votos gana. En scikit-learn, esto se logra usando el algoritmo Random Forest.

Clasificación aplicada a amenazas de seguridad del mundo real

El 5 de junio de 2013, Microsoft anunció la culminación de una exitosa "operación coordinada" en conjunto con entidades de aplicación de la ley y la industria de servicios financieros para derribar 1.462 botnets Citadel cortando las conexiones entre sus sistemas de mando y control (C&C, por sus siglas en inglés) y los millones de computadoras que administran Al momento del anuncio, se creía que el malware Citadel había afectado a aproximadamente cinco millones de personas y causado más de USD 500 millones en pérdidas a individuos y empresas.

Los métodos forenses utilizados por Microsoft y sus socios para identificar los sistemas de C&C no se han ofrecido al público. Sin embargo, mostraremos cómo se puede aplicar el análisis de clasificación para detectar los sistemas C&C de botnet que actualmente están circulando libremente. Ilustraremos estos métodos dos veces, primero usando un árbol de decisión y luego nuevamente con regresión logística. En ambos casos, entrenaremos nuestros modelos con datos producidos mediante

la emisión de solicitudes HTTP para los 4.789 desplazamientos (offsets) de sitios web que se sabe están asociados con 13 paneles de botnets diferentes.

A continuación, ordenaremos los archivos devueltos por estas solicitudes en grupos según sus códigos de respuesta y luego usaremos el programa ssdeep para producir hashes difusos de cada archivo. Los hashes difusos proporcionan una forma conveniente de evaluar qué tan similar es un archivo a otro sin necesidad de examinar el contenido del archivo directamente. Los valores de comparación resultantes variarán entre el 0 y el 100 %. Normalizaremos estos valores para que oscilen entre 0 y 1 para mayor conveniencia.

Durante el entrenamiento, los algoritmos de AD y RL examinarán todas las características y sus valores hash para evaluar el grado de similitud entre los desplazamientos que se originan de C&C y los servidores benignos, respectivamente. Durante la implementación, el modelo resultante utilizará estas comparaciones para predecir si un desplazamiento en particular está asociado con un sistema de C&C de botnet o no.

Sin embargo, antes de comenzar nuestro análisis, primero debemos pasar por el proceso de extracción y vectorización de las características. Como veremos, cada uno de los vectores resultantes incluirá 357.947 características. Cada característica consistirá en un código de respuesta y un valor hash asociado. Con nuestro vector de muestra y vector de etiqueta creado, estaremos listos para comenzar el proceso de entrenamiento.

Una vez que nuestros modelos hayan sido entrenados y probados, los implementaremos contra muestras de desplazamiento del servidor web sin etiquetar para clasificarlos. A menos que se haya aplicado la regularización para eliminar características, es probable que el modelo de RL incluya el valor de tendencia calculado junto con las ponderaciones de

regresión para la totalidad de las 357.947 características en nuestro conjunto de datos de muestras. En contraste, nuestro modelo de árbol de decisión será mucho más pequeño y mucho más eficiente desde el punto de vista computacional, ya que empleará solo ocho características para detectar paneles de botnet con precisión.

Para simplificar, hemos condensado las fases de entrenamiento y prueba por separado en un solo procedimiento de entrenamiento/prueba en el cual el 70 % de nuestras muestras se utilizan para entrenamiento y el 30 % restante se reserva para pruebas.

Tenga en cuenta: Todos estos ejemplos se basan en ID Panel, una herramienta que Cylance presentó en agosto de 2016 en la conferencia Blackhat en Las Vegas. Si desea experimentar con ID Panel o realizar los procedimientos de clasificación que se describen a continuación, visite https://www.cylance.com/intro-to-ai, donde podrá descargar todas las instrucciones, aplicaciones y archivos de datos pertinentes.

RECOLECTAR Y PREPARAR NUESTROS DATOS DE MUESTRAS

Paso 1: Recopilación de datos

Antes de que se pueda entrenar a un modelo, primero debemos recopilar un conjunto de datos de muestras apropiado. Ya hemos hecho esto al emitir solicitudes HTTP a todos los servidores web de C&C y servidores web benignos, y luego consolidar los códigos de respuesta y los archivos dentro del archivo prevectors.json. Por lo tanto, no es necesario que se conecte a ningún servidor a menos que desee agregar nuevos desplazamientos a los archivos de datos.

Tras haber recopilado un conjunto de datos de muestras, comenzaremos ejecutando *python create_prevectors.py* con la expectativa de que el script no inicie ninguna solicitud HTTP.

```
bwall@highwind:~/code/IDPanel$ python create_prevectors.py
Making 0 requests
bwall@highwind:~/code/IDPanel$
```

Si *se generan* solicitudes, es probable que su copia de los datos de prevector haya sido almacenada en la ubicación incorrecta.

Si desea incorporar desplazamientos de servidores adicionales, simplemente puede agregarlos al directorio c2_labels con el siguiente comando:

```
bwall@highwind:~/code/IDPanel$ echo "https://bwall.github.io/" >> c2_labels/not_panel.txt
bwall@highwind:~/code/IDPanel$
```

Luego, debe ejecutar python create_prevectors.py nuevamente para explorar los sitios recientemente agregados.

```
bwall@highwind:~/code/IDPanel$ python create_prevectors.py
```

Esto genera las mismas 4.789 solicitudes que utilizamos para crear nuestras muestras existentes. Puede ver las primeras 109 de estas solicitudes en el siguiente ejemplo de código.

```
Requesting https://bwall.github.io/img/flags/zm.png
94 completed out of 4789
Requesting https://bwall.github.io/theme/js/tinymce/jscripts/tiny_mce/themes/advanced/skins/highcontrast/content.css
95 completed out of 4789
Requesting https://bwall.github.io/mods/mod_stats.php
96 completed out of 4789
Requesting https://bwall.github.io/rdm/img/malawi.png
97 completed out of 4789
Requesting https://bwall.github.io/aata/images/form_button.png
98 completed out of 4789
Requesting https://bwall.github.io/resources/scripts/global.php
100 completed out of 4789
Requesting https://bwall.github.io/admin/images/icons/flags/mh.png
101 completed out of 4789
Requesting https://bwall.github.io/get-functions.php
102 completed out of 4789
Requesting https://bwall.github.io/includes/Smarty-3.1.8/libs/sysplugins/smarty_internal_compile_section.php
103 completed out of 4789
Requesting https://bwall.github.io/imgs/flags/tv.png
104 completed out of 4789
105 completed out of 4789
Requesting https://bwall.github.io/admin/images/icons/flags/lu.png
Requesting https://bwall.github.io/theme/js/contextMenu/demo/disabled-callback.html
106 completed out of 4789
Requesting https://bwall.github.io/data/images/lang/32/tg.png
107 completed out of 4789
Requesting https://bwall.github.io/data/images/lang/16/bo.png
108 completed out of 4789
Requesting https://bwall.github.io/img/flags/vc.gif
109 completed out of 4789
```

Y aquí está el conjunto restante de solicitudes numeradas del 4.776 al 4.789.

```
Requesting https://bwall.github.io/includes/Smarty-3.1.0/libs/sysplugins/smarty_internal_compile_for.php
4776 completed out of 4789
4777 completed out of 4789
Requesting https://bwall.github.io/includes/design/images/modules/module_bitkinex.png
Requesting https://bwall.github.io/data/images/lang/32/sc.png
4778 completed out of 4789
Requesting https://bwall.github.io/imgs/flags/cf.png
4779 completed out of 4789
Requesting https://bwall.github.io/img/flags/aw.png
4780 completed out of 4789
4781 completed out of 4789
4782 completed out of 4789
4783 completed out of 4789
4784 completed out of 4789
4785 completed out of 4789
4786 completed out of 4789
4787 completed out of 4789
4788 completed out of 4789
4789 completed out of 4789
bwall@highwind:~/code/IDPanel$
```

Paso 2: Extracción de características

Ahora que hemos recopilado nuestras muestras, podemos extraer las características que usaremos para crear nuestros vectores con el siguiente comando.

```
bwall@highwind:~/code/IDPanel$ python extract_features_from_prevectors.py
```

Como se explica, esto produce 357.947 características asociadas con 14 etiquetas diferentes. Trece de estas se vinculan con los sitios de panel de botnet que conocemos. La decimocuarta etiqueta representa nuestro conjunto de servidores limpios.

```
bwall@highwind:~/code/IDPanel$ python extract_features_from_prevectors.py
Loaded 421432 prevectors
Extracted 357947 features
Features cover 4789 requests
Vectors cover 14 labels
bwall@highwind:~/code/IDPanel$
```

Paso 3: Vectorización

Ahora, podemos usar el siguiente código para crear nuestros vectores usando el archivo de características que acabamos de crear:

```
bwall@highwind:~/code/IDPanel$ python vectorize_with_raw_features.py
Loading prevectors
```

Esto produce una matriz de vectores que incluye los que se muestran a continuación:

```
Vector for http://o-z-o.ru/Panelinator/Pony/includes/design/adsfsdf/ completed
Vector for http://www.domedicosdelnorte.com/live/Panel/ completed
Vector for http://0rgil.com/blessedoldpony/ completed
Vector for http://nellisrealestate.com/wp-includes/images/okk/panelnew/ completed
Vector for http://93.190.68.229/ completed
Vector for http://rustywood.ca/php/ completed
Vector for http://cgpdskiteam.org/wp-admin/includes/sj/ completed
Vector for http://213.155.31.195/ completed
Vector for http://drunkensheriff.glupoty.com/wp-admin/ completed
Vector for http://fausin.honor.es/ completed
Vector for http://bilimteknoloji.info/wplog/ completed
Vector for http://datarecoveryoxfordshire.co.uk/ven/ completed
Vector for http://www.osoperfume.com/errors/sols/ completed
Vector for http://45df36.de/df/ completed
Vector for http://y99978em.bget.ru/ completed
Vector for http://j906793s.bget.ru/NeT/ completed
Vector for http://189.1.162.151/tmp/ completed
Vector for http://www.sec.parmankulov.com/ completed
Vector for http://petroyeda.com/lege/server/ completed
Vector for http://datarecoveryoxfordshire.co.uk/chap/ completed
Vector for http://195.117.230.152/Panel/ completed
Vector for http://xram.onlinewebshop.net/XRAM/ completed
Vector for http://www.pardox.site11.com/Web%20Panel/ completed
Vector for http://www.thefashionspot.com/ completed
Vector for http://xbledge.com/dendroid/ completed
bwall@highwind:~/code/IDPanel$
```

Con nuestro vector de etiquetas y matriz de etiquetas creados, ahora podemos comenzar el proceso de entrenamiento del modelo.

Flujo de trabajo de la sesión: Clasificación con árboles de decisión

Como se señaló anteriormente, el clasificador de AD scikit-learn generalmente hace un excelente trabajo creando modelos con su configuración de hiperparámetros predeterminada. A continuación, hemos utilizado el argumento *–h* para mostrar la lista de opciones de hiperparámetros para el script de AD *train_model.py*.

```
bwall@highwind:~/code/IDPanel$ python train_model.py -h
usage: train_model.py [-h] [-c {gini,entropy}] [-s {random,best}]
                      [-m {auto,sqrt,log2}] [-d MAX_DEPTH]
                      [-S MIN_SAMPLES_SPLIT] [-l MIN_SAMPLES_LEAF]
                      [-w MIN_WEIGHT_FRACTION_LEAF] [-n MAX_LEAF_NODES]

Train Decision Tree

optional arguments:
  -h, --help            show this help message and exit
  -c {gini,entropy}, --criterion {gini,entropy}
  -s {random,best}, --splitter {random,best}
  -m {auto,sqrt,log2}, --max-features {auto,sqrt,log2}
  -d MAX_DEPTH, --max-depth MAX_DEPTH
  -S MIN_SAMPLES_SPLIT, --min-samples-split MIN_SAMPLES_SPLIT
  -l MIN_SAMPLES_LEAF, --min-samples-leaf MIN_SAMPLES_LEAF
  -w MIN_WEIGHT_FRACTION_LEAF, --min-weight-fraction-leaf MIN_WEIGHT_FRACTION_LEAF
  -n MAX_LEAF_NODES, --max-leaf-nodes MAX_LEAF_NODES
bwall@highwind:~/code/IDPanel$
```

Si estamos satisfechos con los valores predeterminados, podemos comenzar a entrenar un modelo inmediatamente ejecutando *python train_model.py*.

```
bwall@highwind:~/code/IDPanel$ python train_model.py
```

Una vez que se completa el procesamiento, el algoritmo generará el archivo de modelo *bot_model.mdl* y mostrará un gráfico de umbral y una curva ROC que evaluará su precisión. Como podemos ver, nuestro modelo está haciendo un excelente trabajo clasificando con precisión los sitios de C&C en todo el rango de umbrales de probabilidad. Su gráfico puede ser diferente ya que la generación del modelo se basa al menos parcialmente en valores aleatorios.

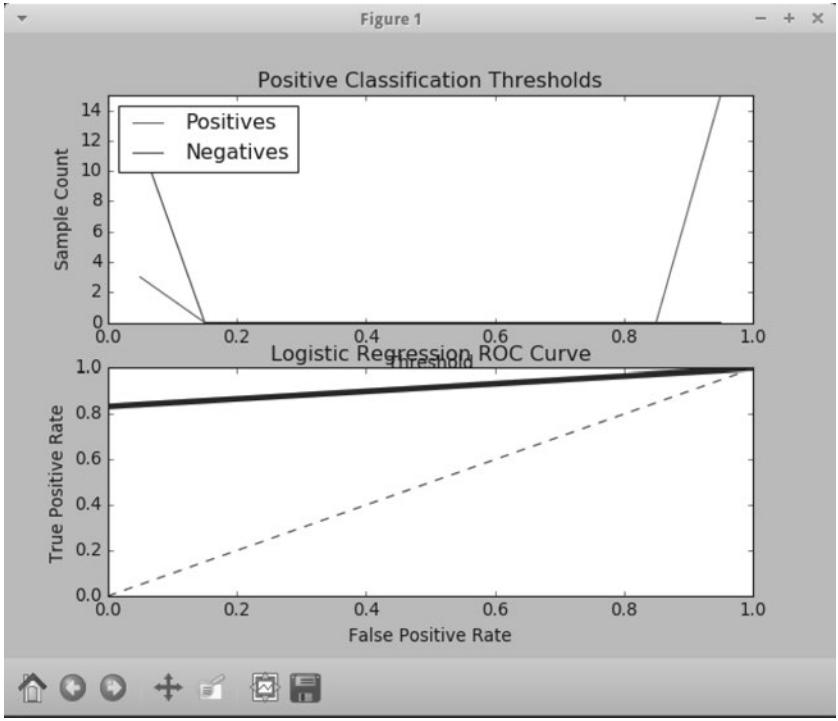


FIGURA 2.11: Umbrales de clasificación y curva ROC

Una vez que los gráficos están cerrados, podemos exportar el árbol de decisión en forma de un archivo de imagen para que podamos ver su estructura. Nuestro archivo de imagen se llama *tree.png*. Ahora podemos rastrear visualmente el proceso de decisión que comienza con la raíz.

Como podemos ver, la raíz contiene 94 muestras de clases mixtas. Nuestra primera variable de división es la característica *config.php* y nuestro valor de división es ≤ a 0,5. Un total de 64 muestras coinciden con este criterio (por lo tanto, la condición es verdadera). Por lo tanto, estas muestras se han copiado al nodo hijo *404* a la izquierda. La puntuación de Gini de 0,4824 indica que este nodo contiene muestras de ambas clases, por lo que se requerirá una división adicional.

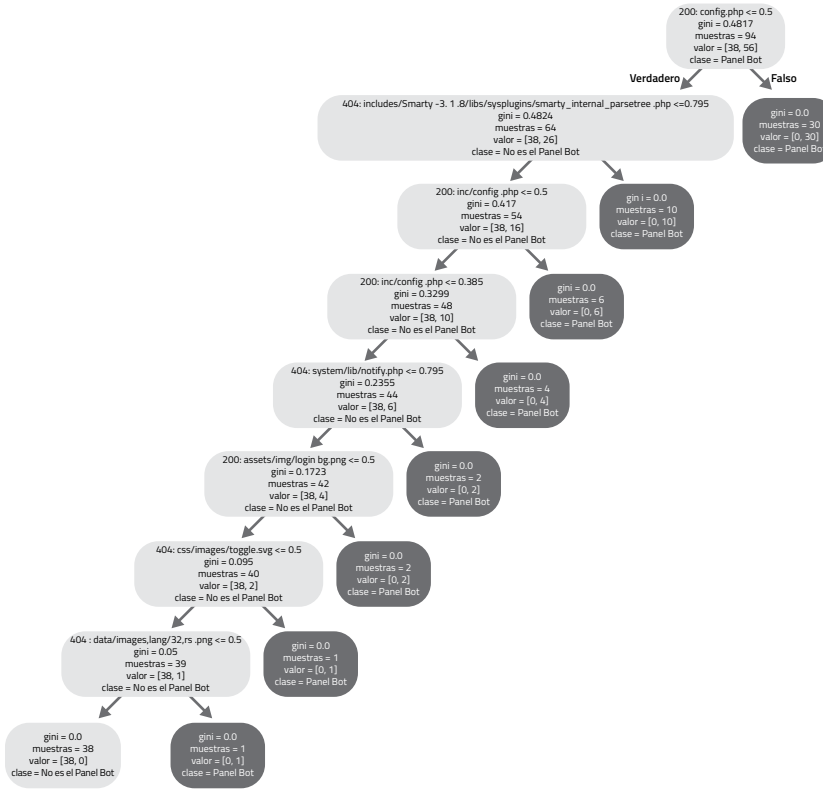


FIGURA 2.12: Detalles del árbol de decisión

Por el contrario, podemos ver que 30 de las muestras de raíz *no* coinciden con el criterio de división (es decir, la condición es falsa). Por lo tanto, estas muestras se copian en el nodo hijo a la derecha. La puntuación de Gini de 0,0 para este nodo indica que hemos alcanzado la pureza perfecta del nodo. El recuento de muestras indica que ninguna de estas muestras pertenece a la Clase 0 (no se trata de un panel de botnet) y 30 pertenecen a la Clase 1 (hay un panel de botnet). Por lo tanto, hemos encontrado nuestro primer nodo terminal, que no requiere procesamiento adicional.

Continuamos siguiendo nuestro árbol de rama en rama hacia abajo hasta que alcanzamos nuestra octava característica, que produce nuestro conjunto final de nodos terminales. El

nodo hijo de la izquierda contiene 38 muestras de servidores benignos. El nodo hijo a la derecha contiene una sola muestra de un sistema de C&C. Ahora que todas nuestras muestras han sido clasificadas, nuestro árbol de decisión está completo y podemos exportar nuestro archivo de modelo para su implementación. Nombraremos nuestro archivo modelo *bot_model.mdl*.

Este modelo es sumamente eficiente ya que solo necesita ocho características, en lugar de las 357.947 con las que empezamos, para clasificar las muestras con un alto grado de precisión. Además:

- Cuando comencemos a recopilar nuevas muestras sin etiquetar para clasificar, podremos limitar nuestras solicitudes HTTP exclusivamente a aquellas relevantes para estas ocho características.
- El proceso de extracción de características también procederá de manera mucho más rápida y eficiente, ya que podremos ignorar las 357.939 características restantes que hemos extraído de nuestro conjunto de entrenamiento original.

Usemos nuestro modelo para clasificar sitios web activos a fin de comprobar si son sistemas C&C. Para hacer esto, ejecutamos *classify_panel.py* con la URL del servidor y el nombre de nuestro archivo modelo como argumentos.

Advertencia: Le recomendamos encarecidamente que* no *pruebe ninguno de nuestros modelos contra servidores C&C circulando libremente a menos que haya tomado las precauciones adecuadas de antemano para evitar que su sistema se infecte con malware.

Una vez más, emitimos solicitudes HTTP para recopilar códigos de respuesta y desplazamientos. Sin embargo, solo

necesitamos ocho solicitudes, ya que estamos clasificando solo con ocho características.

```
bwall@highwind:~/code/IDPanel$ python classify_panel.py bot_model.mdl https://bwall.github.io
Identifying panels we can actually reach
We can reach https://bwall.github.io/
Making 8 total requests to 1 servers
https://bwall.github.io/, Not Panel, [ 1.  0.]
bwall@highwind:~/code/IDPanel$
```

A continuación, repetimos el proceso de extracción de características que utilizamos anteriormente, vectorizamos los resultados y luego ejecutamos los vectores a través en el modelo.

Como podemos ver, nuestro modelo pudo determinar con solo ocho solicitudes que el servidor web era benigno. Intentemos nuevamente con un servidor que sepamos que es un sistema de C&C.

```
bwall@highwind:~/code/IDPanel$ python classify_panel.py bot_model.mdl http://112.213.88.68/Panel/
Identifying panels we can actually reach
We can reach http://112.213.88.68/Panel/
Making 8 total requests to 1 servers
http://112.213.88.68/Panel/, Botnet Panel, [ 0.  1.]
bwall@highwind:~/code/IDPanel$
```

Una vez más, emitimos ocho solicitudes, extraemos y vectorizamos los resultados, y luego ejecutamos los vectores a través del modelo.

Como podemos ver, el modelo ha clasificado correctamente el servidor como un sistema de C&C.

Flujo de trabajo de la sesión: Clasificación con Regresión Logística

El procedimiento de clasificación que acabamos de realizar se diseñó originalmente utilizando métodos de árbol de decisión. Sin embargo, podemos resolver el mismo problema de clasificación utilizando la regresión logística. Para entrenar el modelo de RL, usaremos el script *train_lr_model.py* y ajustaremos nuestra configuración de hiperparámetros usando los argumentos disponibles de la línea de comando.

A continuación, aplicamos el argumento *–h* nuevamente para mostrar la lista de valores predeterminados y argumentos opcionales para *train_lr_model.py*.

```
bwall@highwind:~/code/IDPanel$ python train_lr_model.py -h
usage: train_lr_model.py [-h] [-p {l1,l2}] [-d] [-C C] [-f]
                         [-i INTERCEPT_SCALING] [-m MAX_ITER]
                         [-s {newton-cg,lbfgs,liblinear,sag}] [-t TOL]

Train Logistic Regression Model

optional arguments:
  -h, --help            show this help message and exit
  -p {l1,l2}, --penalty {l1,l2}
  -d, --dual
  -C C
  -f, --fit-intercept
  -i INTERCEPT_SCALING, --intercept-scaling INTERCEPT_SCALING
  -m MAX_ITER, --max-iter MAX_ITER
  -s {newton-cg,lbfgs,liblinear,sag}, --solver {newton-cg,lbfgs,liblinear,sag}
  -t TOL, --tol TOL
bwall@highwind:~/code/IDPanel$
```

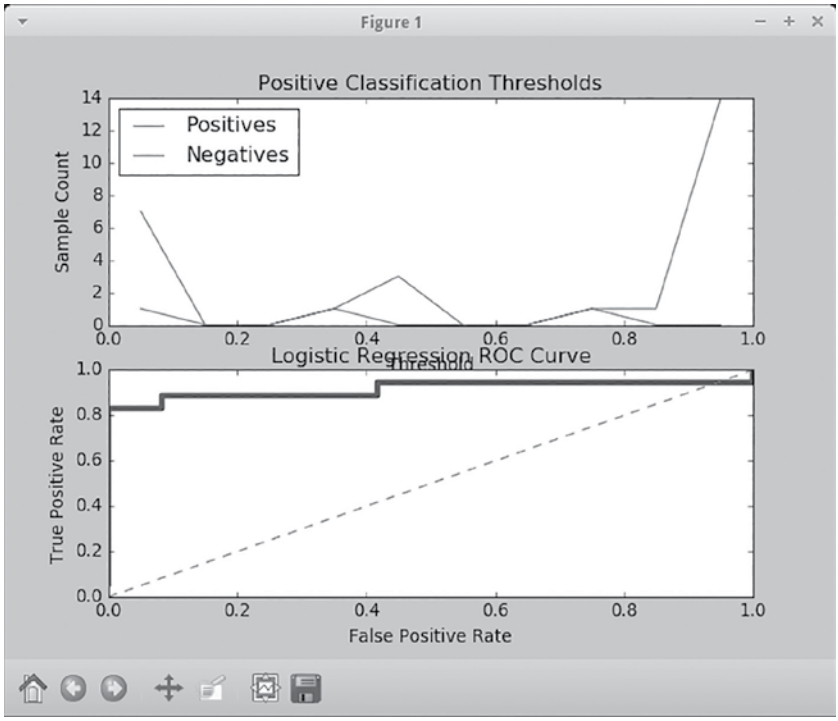


FIGURA 2.13: Umbrales de clasificación y curva ROC

Como hicimos anteriormente, ejecutaremos nuestro clasificador sin cambiar la configuración predeterminada de hiperparámetros. Una vez que se completa el procesamiento, el script generará nuestro modelo de RL *bot_model.lrmdl* junto con su gráfico de umbral asociado y la curva ROC.

Podemos ver que la configuración de hiperparámetros predeterminada ha generado una vez más un modelo preciso. Veamos si podemos mejorarlo aún más experimentando con algunas configuraciones de hiperparámetros alternativas. Intentaremos cambiar la penalización del "L2" predeterminado a "L1".

```
bwall@highwind:~/code/IDPanel$ python train_lr_model.py -p l1
Creating training and testing sets
94 samples in training set, 2 labels in training set
30 samples in training set, 2 labels in testing set
Confusion Matrix:
[[11  1]
 [ 5 13]]
```

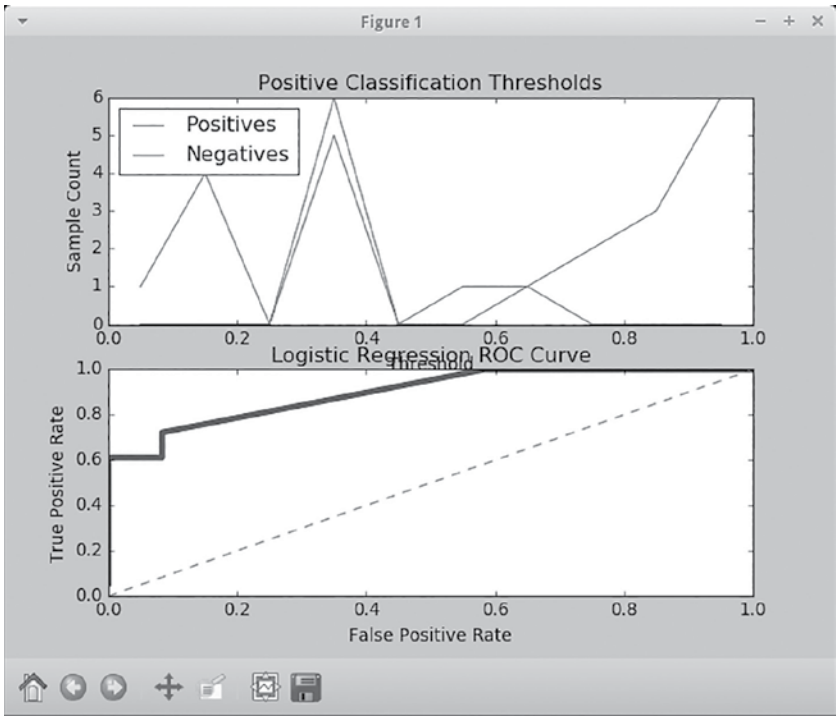


FIGURA 2.14: Umbrales de clasificación y curva ROC

Este ajuste aparentemente pequeño ha producido un modelo algo menos preciso que el anterior. Antes de continuar con la clasificación, repetiremos el entrenamiento de nuestro modelo con los valores predeterminados.

Ahora podemos aplicar nuestro modelo a nuevos vectores sin etiquetar para clasificarlos. Usaremos el mismo comando que antes pero con *bot_model.lrmdl* especificado en el argumento.

```
bwall@highwind:~/code/IDPanel$ python classify_panel.py bot_model.lrmdl https://bwall.github.io
Identifying panels we can actually reach
We can reach https://bwall.github.io/
Making 4789 total requests to 1 servers
https://bwall.github.io/, Not Panel, [ 0.99699357  0.00300643]
bwall@highwind:~/code/IDPanel$
```

La regresión logística no es tan agresiva como los árboles de decisión para reducir las características, por lo que necesitamos

casi 5.000 solicitudes para clasificar nuestro objetivo como benigno. Sin embargo, nuestra clasificación sigue siendo correcta y segura. A continuación, probaremos nuestro modelo nuevamente con un sistema conocido como C&C.

Una vez más emitimos nuestra solicitud HTTP, extraemos y vectorizamos los resultados, y luego ejecutamos los vectores en el modelo. Como podemos ver, el modelo ha clasificado correctamente el objetivo como un sistema de C&C.

```
bwall@highwind:~/code/IDPanel$ python classify_panel.py bot_model.lrmdl http://112.213.88.68/Panel/
Identifying panels we can actually reach
We can reach http://112.213.88.68/Panel/
Making 4789 total requests to 1 servers
http://112.213.88.68/Panel/, Botnet Panel, [  5.99004234e-04   9.99400916e-01]
bwall@highwind:~/code/IDPanel$
```

Conclusiones de la clasificación

Como hemos visto, la clasificación es un método de aprendizaje supervisado potente y eficaz que se puede aplicar productivamente a una amplia gama de problemas de seguridad. Estos son algunos de los puntos clave que cubrimos en este capítulo:

- La clasificación es un método de aprendizaje supervisado que puede funcionar bien cuando hay una cantidad suficiente de datos etiquetados; las muestras reflejan con precisión la proporción de casos reales positivos y negativos; y los datos subyacentes son inherentemente compatibles con el análisis de clasificación.
- La clasificación se lleva a cabo a través de una metodología de entrenamiento, validación, prueba e implementación de cuatro fases. En este capítulo, limitamos nuestra discusión al entrenamiento, la prueba y la implementación. La pertenencia a la clase para todas las muestras de entrenamiento y prueba siempre se conoce de antemano, lo que permite a cada

clasificador crear un modelo basándose en el grado en que cada característica y valor de característica predice la pertenencia a una clase ya conocida de cada muestra.

- La regresión logística y los árboles de decisión son muy efectivos para realizar la clasificación, pero adoptan enfoques muy diferentes al hacerlo. La RL funciona calculando las ponderaciones de regresión y los valores de tendencia que determinan el grado en que cada característica influye finalmente en una predicción de clase. Los árboles de decisión lo hacen al calcular variables de división y dividir valores para definir nodos con miembros de clase cada vez más homogéneos.
- Los clasificadores de regresión logística son más propensos a un ajuste insuficiente de los datos ya que deben dividir el espacio de características utilizando líneas rectas y planos. Los árboles de decisión dividen el espacio de características utilizando rectángulos y son más propensos al sobreajuste, ya que, a menos que se apliquen criterios de detención, crean reglas de decisión para cada vector independientemente de su prevalencia real en el entorno de datos subyacente. Se puede sospechar de un sobreajuste cuando un modelo de AD que hace un excelente trabajo al clasificar los datos de entrenamiento muestra una marcada disminución en la precisión cuando se aplica a los datos de prueba.
- Los analistas utilizan diversos métodos de validación rigurosos para evaluar la precisión de sus modelos. Entre ellos, se incluyen matrices de confusión, diagramas de umbral de probabilidad y curvas ROC, entre otros. Sin embargo, un clasificador solo puede predecir la *probabilidad* de que una nueva muestra no etiquetada pertenezca a una clase determinada. Casi siempre habrá

un cierto número de asignaciones de clase de falso positivo y falso negativo. Por lo tanto, la clasificación es más adecuada para situaciones en las que las tasas de error caen dentro del margen de tolerancia aceptable que refleja la naturaleza del problema de clasificación que se aborda.

Probabilidad

En el capítulo de clasificación, vimos cómo los algoritmos de regresión logística y árbol de decisión utilizaron puntuaciones de probabilidad para asignar muestras a una de dos clases. Si una muestra obtuvo una puntuación por encima del umbral de probabilidad de 0,5, se le asignó a la Clase 1. Si obtuvo una puntuación inferior a 0,5, se asignó a la Clase 0. En este capítulo, tomaremos una perspectiva más amplia de la probabilidad como una categoría de métodos de Aprendizaje Automático para hacer predicciones. En particular, examinaremos el clasificador bayesiano ingenuo y el algoritmo de agrupamiento del modelo de mezclas Gaussianas y consideraremos cómo se pueden aplicar para resolver problemas de detección y reparación relacionados con la seguridad.

¿Qué es exactamente la probabilidad?

Como humanos, estamos acostumbrados a vivir en un mundo incierto en el que nuestro éxito en el logro de los objetivos se basa en nuestra capacidad de evaluar y predecir con éxito los eventos. Examinamos las posibilidades y consideramos las probabilidades. Pero nunca podemos estar del todo seguros de que cualquier decisión que tomemos sea la correcta. Siempre hay un elemento aleatorio que puede obstaculizarnos. ¿Deberíamos lavar nuestro automóvil cuando el pronóstico del clima prevee un 25 % de probabilidad de lluvia? ¿Deberíamos comprar boletos de avión para unas vacaciones planificadas ahora o esperar unas semanas para ver si podemos obtener una mejor oferta?

Los modelos probabilísticos son adecuados para resolver problemas como estos en los que tenemos un conocimiento insuficiente o imperfecto. Son particularmente efectivos para modelar la incertidumbre. Si se utilizan de manera adecuada, nos permiten reducir la incertidumbre hasta el punto en que podemos tomar decisiones con un alto grado de confianza.

EJEMPLOS COMUNES DE PROBLEMAS DE PROBABILIDAD

Consideremos el caso de una ruleta como la que aparece a continuación, con cuatro cuadrantes del mismo tamaño etiquetados como rojo, amarillo, verde y azul, respectivamente.

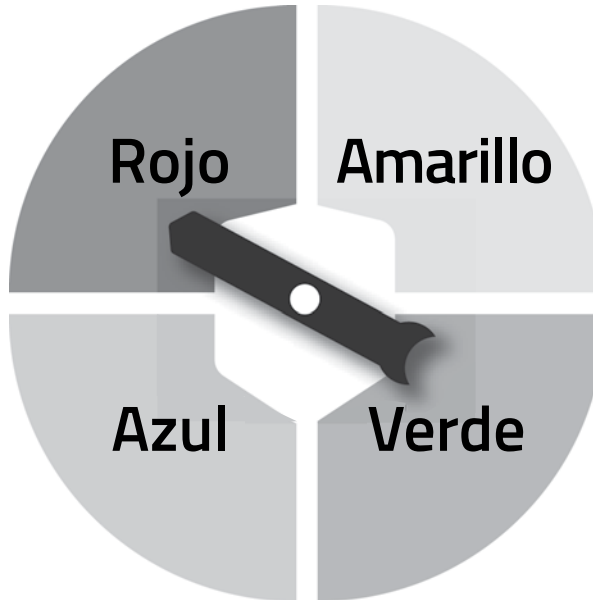


FIGURA 3.1: Ruleta de color

Suponiendo que pueda girar libremente, ¿cuál es la probabilidad de que un giro dado se detenga en amarillo, rojo, azul o verde? Para responder a esta pregunta, tendremos que poner en marcha la ruleta. En la terminología de probabilidad, cada giro se conoce como un *experimento* o una *prueba*. La evidencia producida por cada prueba se conoce como un *resultado*. En este caso, tenemos cuatro resultados posibles: amarillo, rojo, azul y verde.

Para calcular la probabilidad de cada resultado, dividimos la cantidad de formas en que puede ocurrir cada resultado por el número total de resultados posibles, tal como se muestra a continuación.

Resultados de interés	Número de formas en que este resultado puede ocurrir	Número de posibles resultados	Cálculo de probabilidad
P (amarillo)	1	4	1/4 = 0,25
P (rojo)	1	4	1/4 = 0,25
P (azul)	1	4	1/4 = 0,25
P (verde)	1	4	1/4 = 0,25

Como podemos ver, la probabilidad de cada resultado es la misma: P=0,25.

Ahora consideremos un ejemplo un poco más complicado en el que tiramos un dado de seis caras como el que se muestra a continuación.

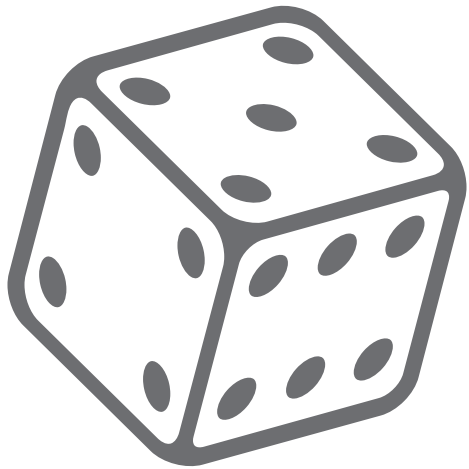


FIGURA 3.2: Dado

Tomemos el mismo enfoque y pronostiquemos la probabilidad de cada uno de los seis resultados posibles

Resultados de interés	Número de formas en que este resultado puede ocurrir	Número de posibles resultados	Cálculo de probabilidad
P (1)	1	6	1/6 = 0,16
P (2)	1	6	1/6 = 0,16
P (3)	1	6	1/6 = 0,16
P (4)	1	6	1/6 = 0,16
P (5)	1	6	1/6 = 0,16
P (6)	1	6	1/6 = 0,16

Si nuestro dado no ha sido alterado, podemos esperar probabilidades iguales para cada uno de los seis posibles resultados (es decir, P= 0,16).

Ahora, realicemos un experimento para predecir la probabilidad de sacar un número impar o par. En cualquier prueba dada, podemos producir un resultado *impar* al lanzar un 1, 3 o 5 y un número *par* al lanzar un 2, 4 o 6. Usamos el término *evento* para referirnos a un resultado que puede ocurrir de diferentes maneras. El evento *impar*, por ejemplo, puede ser producido por los resultados 1, 3 o 5, mientras que el evento *par* puede ser producido por los resultados 2, 4 o 6. Ahora, calculemos la probabilidad de un evento par o impar según nuestros seis resultados posibles.

Eventos de interés	Número de resultados que producen un evento impar/par	Número de posibles resultados	Cálculo de probabilidad
P (evento par)	3	6	3/6 = 0,5
P (evento impar)	3	6	3/6 = 0,5

Como podemos ver, la probabilidad de cada evento es la misma en ambos casos. Por lo tanto, nuestro análisis de probabilidad no nos ayuda a predecir los resultados de ninguna prueba en particular. Los resultados serán aleatorios.

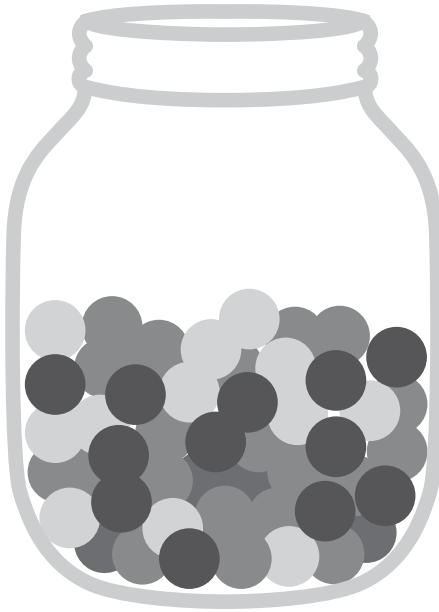


FIGURA 3.3: Frasco de canicas

A menudo, sin embargo, los problemas que queremos resolver involucran resultados que *no* son igualmente probables. Consideremos este tipo de ejemplo usando un frasco de canicas como el que se muestra a continuación.

Cuando contamos las canicas, descubrimos que tres son amarillas, seis son rojas, ocho son azules y cinco son verdes. Si elegimos una canica al azar del frasco, ¿cuál es la probabilidad de que la canica sea amarilla? La siguiente tabla proporciona la probabilidad de seleccionar cada uno de los cuatro resultados posibles.

Eventos de interés	Número de resultados que producen este evento	Número de posibles resultados	Cálculo de probabilidad
P (amarillo)	3	22	3/22 = 0,13
P (rojo)	6	22	6/22 = 0,27
P (azul)	8	22	8/22 = 0,36
P (verde)	5	22	5/22 = 0,22

Como podemos ver, los cuatro eventos posibles ocurren con diferentes probabilidades. Para cualquier prueba dada, es más probable que elijamos una canica azul (P=0,36) y menos probable que elijamos una amarilla (P=0,13).

Concluiremos esta sección examinando un caso en el que todos los resultados tienen la misma probabilidad de ocurrir, pero los eventos que conducen a estos resultados, no. En este experimento, una prueba consiste en buscar con la mano en una caja que contiene cinco tarjetas numeradas del uno al cinco. Nuestro objetivo será resolver dos problemas de probabilidad diferentes. Primero, calcularemos la probabilidad de elegir uno de los cinco resultados posibles. A continuación, calcularemos la probabilidad de elegir un evento de número impar o par.

Una vez más, dividimos el número de formas en que se puede seleccionar un número en particular (1) por el número total de resultados posibles (5). Como antes, observamos que todos los resultados tienen la misma probabilidad de ocurrir.

Eventos de interés	Número de resultados que producen este evento	Número de posibles resultados	Cálculo de probabilidad
P (1)	1	5	1/5 = 0,2
P (2)	1	5	1/5 = 0,2
P (3)	1	5	1/5 = 0,2
P (4)	1	5	1/5 = 0,2
P (5)	1	5	1/5 = 0,2

Sin embargo, cuando se trata de calcular las probabilidades de eventos, los resultados serán diferentes porque tres resultados (1, 3 y 5) producen un evento impar mientras que solo dos (2 y 4) producen uno par.

	Número de formas en que este evento puede ocurrir	Número total de resultados	Cálculo de probabilidad
P (par)	2	5	2/5 = 0,4
P (impar)	3	5	3/5 = 0,6

Como resultado, en cualquier prueba dada, tenemos un 60 % de probabilidad (P=0,6) de elegir una tarjeta impar y solo un 40 % de probabilidad (P=0,4) de elegir una pareja con número par.

PROBABILIDAD CONDICIONAL FRENTE A PROBABILIDAD CONJUNTA

En los ejemplos anteriores, cada prueba se consideró de manera independiente. Por ejemplo, no consideramos cómo lanzar al aire una moneda para producir el resultado de *cara* en una prueba puede afectar la probabilidad de obtener un resultado de *cruz* en el siguiente. Sin embargo, muchos problemas de interés requieren que consideremos el grado en que dos resultados están relacionados:

- En los problemas de *probabilidad condicional* nuestro objetivo es determinar la probabilidad de que el Evento B siga al Evento A, por ejemplo, la probabilidad de que se produzca un aumento en lo cantidad de accidentes automovilísticos después de una tormenta de nieve.
- En los problemas de *probabilidad conjunta* nuestro objetivo es determinar la probabilidad de que los Eventos A y B ocurran al mismo tiempo, por ejemplo, si al lanzar los dados, obtendremos un par de cincos.

Consideremos primero un ejemplo de *probabilidad condicional*. Imagine que tiene una bolsa que contiene tres monedas de veinticinco centavos y dos monedas de diez centavos. ¿Cuál es la probabilidad de seleccionar una de las monedas de

veinticinco centavos en su primera prueba? Para calcular esto, dividimos tres (el número de posibles eventos de selección de monedas de veinticinco centavos) por cinco (el número de resultados posibles) para una puntuación de probabilidad de P=0,6.

De hecho, seleccionamos una moneda de veinticinco centavos, dejando dos monedas de veinticinco centavos y dos monedas de diez centavos en la bolsa. ¿Cuál es la probabilidad de que seleccionemos una moneda de veinticinco centavos en nuestra próxima prueba? Una vez más, dividimos el número de posibles eventos de monedas de veinticinco centavos (reducido de 3 a 2) por el número de resultados posibles (reducido de 5 a 4) para una puntuación de probabilidad de P=0,5. Como podemos ver, el resultado de nuestra segunda prueba dependió del resultado de nuestra primera prueba.

Luego, consideremos un ejemplo de *probabilidad conjunta*. Imagine que está jugando a los dados y ha apostado sus ahorros de toda la vida a sacar un par de cincos. ¿Cuál es su probabilidad de ganar? Sabemos por nuestro ejemplo anterior con un solo dado que nuestra posibilidad de producir un cinco en cualquier lanzamiento es P=0,16. Esto no cambia si lanzamos dos dados al mismo tiempo, o mil. Para cada dado, la probabilidad de lanzar un cinco siempre será P=0,16. Los dos eventos son independientes el uno del otro.

Intuitivamente, sin embargo, sabemos que la probabilidad de lanzar dos cincos al mismo tiempo debe ser menor que la probabilidad de lanzar un cinco y algún otro número. En los problemas de probabilidad conjunta, calculamos esto multiplicando los valores de probabilidad de los dos eventos entre sí para producir una puntuación de probabilidad conjunta como se muestra a continuación:

Probabilidad de sacar un cinco en el dado n.° 1: 0,16
Probabilidad de sacar un cinco en el dado n.° 2: 0,16
Probabilidad de lanzar dos cincos: 0,16 * 0,16 = 0,0256

Como podemos ver, nuestras perspectivas de ganar son de hecho muy bajas.

Clasificación con el algoritmo bayesiano ingenuo

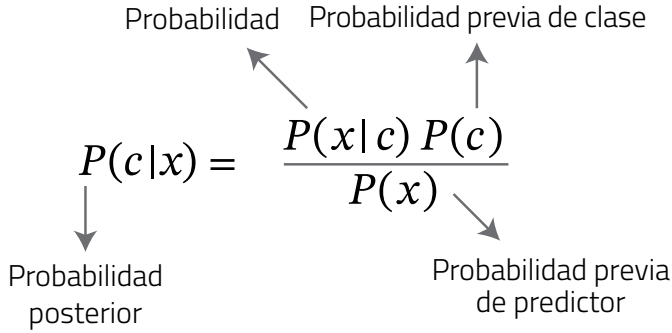
El algoritmo bayesiano ingenuo se deriva del Teorema de Bayes, que fue concebido por primera vez en el siglo XVIII por el estadístico inglés Thomas Bayes y luego perfeccionado a su forma actual por el matemático francés Pierre-Simon Laplace. El Teorema de Bayes proporciona los medios para calcular la probabilidad de que un evento determinado *A* ocurra cuando la condición *B* es verdadera. En un problema de clasificación, el Teorema de Bayes nos permite calcular la probabilidad condicional de que una muestra pertenezca a una clase particular dados los atributos de sus características. Como analizamos en el Capítulo 2, la clasificación es un ejemplo de un método de *aprendizaje supervisado*. En consecuencia, una sesión de clasificación normalmente se realizará a través de una secuencia de entrenamiento, validación y prueba.

El Teorema de Bayes puede ser difícil de resolver porque tiene en cuenta las probabilidades condicionales. Es decir, calcula las relaciones de probabilidad entre cada característica en el conjunto de datos mientras produce una decisión de clasificación. A medida que aumenta la cantidad de características, el número y la complejidad de estas relaciones pueden incrementarse exponencialmente. Los costos en términos de tiempo de cómputo y recursos pueden tornarse rápidamente prohibitivos.

El Teorema bayesiano Ingenuo simplifica dramáticamente este proceso al asumir la *independencia condicional de las*

clases. En otras palabras, ignora los efectos potenciales de las probabilidades condicionales cuando asigna muestras a las clases. En casi todos los casos, esta suposición es falsa. Es en este sentido que esta formulación del Teorema de Bayes es ingenua. Sorprendentemente, sin embargo, este teorema a menudo produce excelentes resultados y con gran eficiencia ya que solo requiere cuatro componentes para clasificar una muestra.

Naïve Bayes



$$P(c\,|\,x) = P(x_1\,|\,c) \times P(x_2\,|\,c) \times \cdots \times P(x_n\,|\,c) \times P(c)$$

FIGURA 3.4: Ecuación de probabilidad con clasificador bayesiano ingenuo

1. **Probabilidad a posteriori:** [P (c | x)] Esta es la probabilidad de que una muestra pertenezca a una clase particular (c) dados los atributos de sus características (x). En la práctica, esto se calcula varias veces, una vez por cada posible asignación de clase. (De hecho, calculamos esto una vez menos que el número de clases, ya que podemos derivar la puntuación de probabilidad restante al sumar las anteriores y luego restar el total de 1). El resultado con la puntuación de probabilidad más alta determina qué clase se asigna.

2. **Probabilidad a priori de clase:** [P (c)] La probabilidad a priori de clase (PPC) se refiere a la prevalencia de la *clase* en el conjunto de datos. Por ejemplo, si tenemos diez muestras en nuestro conjunto de datos y siete de ellas pertenecen a la Clase 1, entonces la PPC para la Clase 1 = 7/10 = 0,7. También podemos expresar esto diciendo que una muestra aleatoria tiene un 70 % de probabilidades de pertenecer a la Clase 1.
3. **Probabilidad a priori predictiva:** [P (x)] La probabilidad a priori predictiva (PPP) se refiere a la prevalencia del *atributo de la característica* en el conjunto de datos. Por ejemplo, si tenemos 20 muestras en nuestro conjunto de datos y 5 de ellas tienen el atributo de característica *Préstamo*, entonces la PPP de *Préstamo* = 5/20 = 0,25.
4. **Probabilidad:** [P (x | c)] La probabilidad es la posibilidad de encontrar el atributo de característica *X* dado *C*. Se calcula dividiendo el número de muestras con un valor de atributo y una etiqueta de clase particular por el número total de muestras que componen esa clase. Como ejemplo, considere el estudio hipotético de precipitaciones en dos ciudades de los EE. UU., una con un invierno frío y otra con un invierno cálido. Los patrones de precipitación en estas dos ciudades se observaron durante 10 días para cada ciudad. Los investigadores registraron la cantidad de días soleados, días nublados y días lluviosos o nevados. Con el ejemplo de Tabla de probabilidad a continuación, se puede calcular la puntuación de probabilidad para el atributo Lluvioso/Nevado en la Ciudad 1 con un invierno más cálido = 1/10 = 0,1. La puntuación de probabilidad para este atributo en la Ciudad 2 con un invierno más frío es = 7/10 = 0,7. La Tabla de probabilidad también calcula:

- La PPP, al dividir los totales por fila entre el número total de muestras.
- La PPC, al dividir los totales por columna entre el número total de muestras.

Tabla de probabilidad

Clima	Ciudad 1 (ciudad más cálida)	Ciudad 2 (ciudad más fría)	Total	Probabilidad
Soleado	6	1	7	(=7/20)
Nublado	3	2	5	(=5/20)
Lluvioso/ Nevado	1	7	8	(=8/20)
Total	10	10	20	
Probabilidad	(=10/20)	(=10/20)		

Existen diferentes variantes del clasificador bayesiano ingenuo, cada una adaptada a un conjunto de datos y la situación de un problema en particular. Tres de los más populares son:

- **Bayesiano ingenuo con distribución de Bernoulli** (BIB) Este método es ideal para la detección de spam y otros problemas basados en texto en los que los vectores han sido codificados para indicar la presencia de una cadena de texto en particular, como "consolidación de deuda". Si se encuentra una sola instancia de esa cadena, el mensaje se clasificaría como spam.
- **Bayesiano ingenuo con distribución multinomial** (BIM) Este método es muy adecuado para los problemas basados en texto en los que los vectores se han codificado para indicar la *frecuencia* con la que aparece una cadena de texto. Esto hace que el BIM sea un método apropiado para problemas tales como la clasificación de documentos, por ejemplo, para determinar si un documento es un

contrato o no en función de la frecuencia con la que se encuentran las cadenas de texto "arbitraje" y "violación" en el texto.

- Bayesiano ingenuo gaussiano (BIG) Este método es apropiado con datos continuos que se encuentran dentro de una distribución *normal* o *gaussiana*, por ejemplo, datos que describen la estatura y el peso promedio de hombres y mujeres en los Estados Unidos. Examinaremos las distribuciones gaussianas y sus características más adelante en este capítulo cuando analicemos el algoritmo de agrupamiento del modelo de mezclas gaussianas.

EL CLASIFICADOR BAYESIANO INGENUO EN ACCIÓN

Ahora, echemos un vistazo al clasificador bayesiano ingenuo en acción. Nuestro objetivo será determinar si una computadora remota determinada ejecuta Windows o Linux en función de los servicios de red que brinda. Adquiriremos los datos de entrenamiento que necesitamos explorando una red de computadoras con sistemas operativos conocidos.

Una vez que los datos de entrenamiento se recopilan, vectorizan e importan a scikit-learn, podemos aplicar el clasificador bayesiano ingenuo para crear nuestro modelo de clasificación. El clasificador comenzará analizando los vectores en la tabla a continuación para determinar:

- La cantidad de sistemas etiquetados como Windows (4) y Linux (3).
- Los servicios HTTP, SSH, SMB y FTP proporcionados por estos servidores. Por ejemplo, podemos ver que el primer servidor de Windows proporciona solo servicios de SMB y FTP, mientras que el tercer servidor de Windows proporciona solo HTTP y FTP.

Etiqueta	HTTP	SSH	SMB	FTP
Windows	0	0	1	1
Windows	1	0	1	0
Linux	1	1	0	0
Windows	1	0	0	1
Linux	0	1	0	1
Windows	1	0	1	0
Linux	0	1	0	0

A continuación, el clasificador bayesiano ingenuo calcula la Probabilidad a priori de clase y la Probabilidad a priori predictiva como se muestra en la Tabla de probabilidades a continuación.

Etiqueta	HTTP	SSH	SMB	FTP	Total de muestras por clase	Probabilidad a priori de clase
Windows	3	0	3	2	4	4 / 7 = 0,571428571
Linux	1	3	0	1	3	3 / 7 = 0,428571429
Total	4	3	3	3	7	
Probabilidad a priori predictiva	4 / 7 = 0,571428571	2 / 7 = 0,428571429	3 / 7 = 0,428571429	3 / 7 = 0,428571429		

Para calcular la PPC, el clasificador bayesiano ingenuo divide el número de incidentes de cada clase (Windows o Linux) por el número total de muestras en nuestro conjunto de entrenamiento. En este caso, hemos calculado la Probabilidad a priori de clase de Windows como aproximadamente 0,57 y la de Linux como aproximadamente 0,43. Esto es equivalente a predecir que una muestra seleccionada aleatoriamente tiene más probabilidades de ser un sistema Windows (57 %) que un sistema Linux (43 %).

La tabla de probabilidad también muestra la Probabilidad a priori predictiva, que se calcula dividiendo la suma de los

incidentes de cada característica (en todas las muestras) por el número total de muestras. Por ejemplo, podemos ver que la PPP para la característica HTTP es aproximadamente 0,57.

No importa cuán cuidadosamente realicemos nuestro muestreo, existe una alta probabilidad de que nuestro conjunto de datos contenga combinaciones de características/clases que no hayamos podido captar en nuestro conjunto de entrenamiento. Si no compensamos esto, el clasificador bayesiano ingenuo calculará una puntuación de probabilidad de *0* de que una muestra con esos atributos de característica pertenezca a esa clase. Este es un problema conocido con el clasificador bayesiano ingenuo que los analistas suelen corregir utilizando un método conocido como "suavizado".

El método de suavizado compensa los valores de las características que, de otro modo, se calcularían con una probabilidad del 0 % de incidencia para cualquier clase sin afectar sustancialmente la influencia en la puntuación de probabilidad. Se pueden aplicar varias técnicas de suavizado. En el suavizado *aditivo* (to de *Laplace*) agregamos un valor *alfa* a los recuentos de características. (En este caso, usaremos un valor alfa de 1). También aumentamos los recuentos de clases por un valor igual a d * alfa, en el cual d es igual a la cantidad de clases. Los recuentos revisados se muestran en la tabla a continuación. Como podemos ver, el recuento de clases de Windows se ha incrementado de cuatro a seis y el recuento de clases de Linux de tres a cinco.

Etiqueta	HTTP	SSH	SMB	FTP	Recuento de clases
Windows	4	1	4	3	6
Linux	2	4	1	2	5

Ahora podemos recalcular nuestros valores de probabilidad como lo hicimos antes. En el caso de HTTP y Windows, por ejemplo, la probabilidad es:

$$4 / 6 = 0{,}6667$$

La tabla de probabilidad con nuestros valores suavizados aparece a continuación.

Etiqueta	HTTP	SSH	SMB	FTP
Windows	0,6667	0,1667	0,6667	0,5
Linux	0,4	0,8	0,2	0,4

Ahora estamos listos para aplicar estas puntuaciones de probabilidad a nuestro conjunto de datos de prueba para ver con qué precisión podemos predecir la pertenenciaa una clase de una nueva muestra. En otras palabras, calcularemos la *Probabilidad a posteriori* de cada muestra. Comencemos con una muestra con los valores de las características que se muestran a continuación. (Por ahora, simularemos que no sabemos que la muestra fue etiquetada como Linux).

Etiqueta	HTTP	SSH	SMB	FTP
Linux	1	1	0	1

Para predecir la clase de la muestra, debemos calcular su probabilidad a posteriori dos veces; una vez para Windows y una vez para Linux. La fórmula es:

P (Característica | Clase) ^ ValorCaracterística * (1 – P (característica | clase)) ^ (1 – ValorCaracterística)

Procederemos de la siguiente manera:

HTTP: 0,6667 ^ 1 * 0,3333 ^ 0 = 0,6667 * 1,0 = 0,6667
SSH: 0,1667 ^ 1 * 0,8333 ^ 0 = 0,1667 * 1,0 = 0,1667
SMB: 0,6667 ^ 0 * 0,3333 ^ 1 = 1,0 * 0,3333 = 0,3333
FTP: 0,5 ^ 1 * 0,5 ^ 0 = 0,5 * 1 = 0,5

Ahora que tenemos los cuatro valores de las características, podemos generar nuestra puntuación de probabilidad de Windows al encontrar el producto de estas puntuaciones y el valor de presuavizado de la Probabilidad a priori de clase.

(0,6667 * 0,1667 * 0,3333 * 0,5) * 0,571428571 = 0,010583598

A continuación, repetiremos el mismo cálculo para Linux:

HTTP: 0,4 ^ 1 * 0,6 ^ 0 = 0,4 * 1,0 = 0,4
SSH: 0,8 ^ 1 * 0,2 ^ 0 = 0,8 * 1,0 = 0,8
SMB: 0,2 ^ 0 * 0,8 ^ 1 = 1,0 * 0,8 = 0,8
FTP: 0,4 ^ 1 * 0,6 ^ 0 = 0,4 * 1,0 = 0,4
(0,4 * 0,8 * 0,8 * 0,4) * 0,428571429 = 0,043885714

Ahora podemos comparar los resultados de Probabilidad a posteriori:

Probabilidad de Windows: 0,010583598
Probabilidad de Linux: **0,043885714**

Como podemos ver, el clasificador bayesiano ingenuo predijo correctamente que la muestra pertenece a la clase *Linux*.

PROCESO DE LA SESIÓN DE CLASIFICADOR BAYESIANO INGENUO

Como método de aprendizaje supervisado, el análisis bayesiano ingenuo procede a través de la misma secuencia de fases de entrenamiento, validación y prueba que describimos

anteriormente para la regresión logística y los árboles de decisión en el Capítulo 2. Cuando se completa el entrenamiento, el modelo resultante toma la forma de una Tabla de probabilidad y sus valores asociados de Probabilidad a priori de clase y predictiva. Durante las fases siguientes de validación y prueba, el modelo se expone a datos de prueba y se evalúa su precisión con matrices de confusión y curvas ROC. Una vez que este proceso se completa con éxito, el modelo entrenado se aplica a datos nuevos sin etiqueta para hacer predicciones de clase.

ESCOLLOS Y LIMITACIONES DEL CLASIFICADOR BAYESIANO INGENUO

El clasificador bayesiano ingenuo es sorprendentemente eficaz en la producción de clasificaciones precisas basadas en antecedentes computados, aunque con algunas limitaciones:

- Como se señaló anteriormente, el clasificador bayesiano ingenuo supone que las características son condicionalmente independientes. En la mayoría de las situaciones de problemas del mundo real, esta suposición no es cierta. A pesar de esto, el clasificador bayesiano ingenuo a menudo produce excelentes resultados.
- Cuando el conjunto de datos disponible es escaso, es posible que no podamos captar todas las combinaciones de características/clases reales que existen en el entorno de datos subyacente. Afortunadamente, podemos mejorar estos efectos con Laplace y otras técnicas de suavizado.

Agrupamiento con el algoritmo del modelo de mezclas gaussianas

En el Capítulo 2, presentamos el concepto de aprendizaje no supervisado y describimos cómo los algoritmos *k*-means y DBSCAN asignan vectores a grupos. Como vimos, ninguno de los algoritmos utilizó medidas de probabilidad para realizar estas asignaciones. En cambio, los vectores se asignaron a grupos conforme a sus ubicaciones relativas en el espacio de características. En esta sección, examinaremos una técnica de agrupamiento diferente que *utiliza* medidas de probabilidad. Específicamente, consideraremos cómo el algoritmo del modelo de mezclas gaussianas (MMG) utiliza las puntuaciones de probabilidad para asignar vectores a los grupos y las ventajas de este enfoque en ciertas situaciones de problemas.

Considere el ejemplo de agrupamiento en la Figura 3.5. El gráfico de la izquierda muestra dos conjuntos de vectores, uno en gris claro y el otro en gris oscuro. El gráfico en el medio muestra los resultados del agrupamiento después de usar el algoritmo *k*-means. El gráfico a la derecha muestra los resultados del agrupamiento

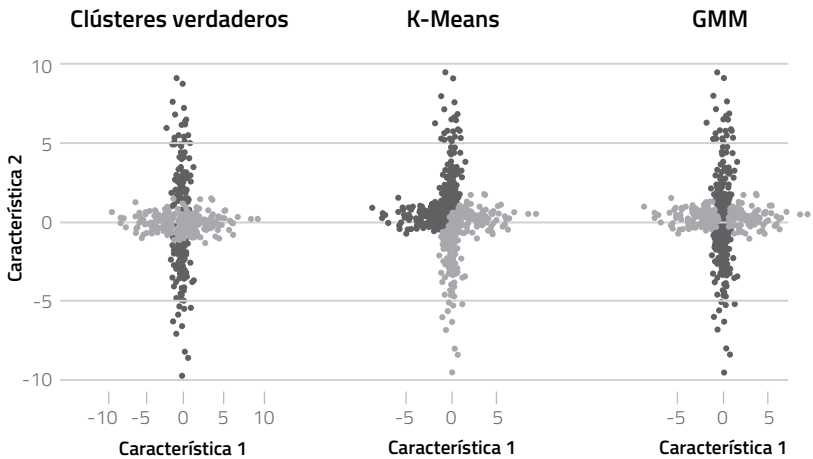


FIGURA 3.5: Comparación de K-Means y MMG

producidos por el MMG. Como podemos ver, *k*-means, al utilizar su método de distancia euclidiana predeterminado, no puede detectar grupos que se superponen o poseen una forma no circular. Por el contrario, el MMG puede hacerlo.

Antes de que podamos considerar cómo funciona el MMG, primero debemos entender varios principios clave, comenzando con la naturaleza de las distribuciones *gaussianas* o *normales*.

DISTRIBUCIONES GAUSSIANAS

Imagine que estamos interesados en analizar la distribución de estatura entre un grupo de 20.000 alumnos de escuela secundaria en un distrito escolar hipotético conocido como Municipio de Jefferson. Tenemos varias opciones para representar estos datos. Por ejemplo, podemos utilizar un histograma como se muestra en la Figura 3.6, con la estatura de los alumnos en centímetros que va en aumento de izquierda a derecha a lo largo del eje x y la cantidad de alumnos con la estatura correspondiente que se muestran en el eje y.

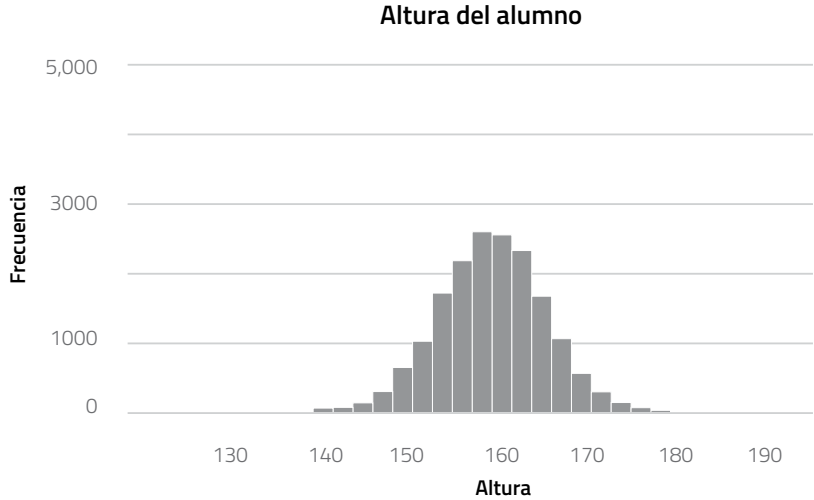


FIGURA 3.6: Estatura de los alumnos de la escuela secundaria en el Municipio de Jefferson

Cada barra vertical representa un conjunto de alumnos que comparten un rango común de estatura.

Como podemos ver, la mayoría de los alumnos se ubica cerca del punto central de la distribución a una estatura de 160 cm. También podemos expresar esto diciendo que 160 cm es el valor de estatura más frecuente entre nuestra población estudiantil. El conjunto a la derecha inmediata del centro representa el número de alumnos con una estatura de entre 160-162 cm. El conjunto a la izquierda del centro representa a aquellos alumnos entre 158 a 160 cm.

El punto central se denomina de forma diversa como *media*, *mediana* o *modo*. Los términos tienen significados ligeramente diferentes pero, para nuestros propósitos, el término *media* será suficiente. La media representa el valor promedio de la característica que estamos midiendo en nuestra distribución. En este ejemplo, podemos calcular la estatura media de nuestros alumnos sumando todas las medidas de estatura en todos los conjuntos y luego dividiendo la suma entre el número total de alumnos. Supongamos que lo hemos hecho y descubrimos que la media realmente es igual a 160 cm.

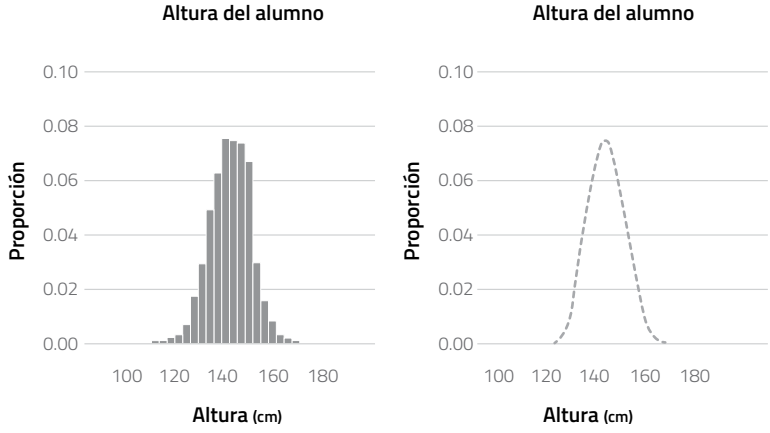


FIGURA 3.7: Versiones de Histograma y Curva suave de distribuciones gaussianas expresadas como proporciones

Como se muestra en la Figura 3.7, tenemos otras opciones para trazar nuestra distribución gaussiana. Aquí, el eje *y* indica la *proporción* de alumnos en cada valor de estatura. Las proporciones se calcularon dividiendo el número de alumnos en cada conjunto (la frecuencia) por el número total de alumnos.

El gráfico de la izquierda es un histograma de la estatura de los alumnos expresado como proporciones, mientras que el gráfico de la derecha es la curva suave en forma de campana que se ajusta al histograma. Una vez más, podemos ver que la mayor proporción de alumnos (alrededor del 75 %) se encuentra más cerca de la media de 160 cm, mientras que los alumnos más bajos y más altos se distribuyen simétricamente en ambos lados.

Si bien la mayoría de los alumnos mide 160 cm, también hay algunos muy altos y muy bajos. A medida que nos alejamos de la media hacia la izquierda y la derecha, la cantidad de alumnos con mayor y menor estatura disminuye simétricamente hasta desaparecer. Esta simetría alrededor de la media es una de las cualidades más reconocibles de una distribución gaussiana y explica su característica curva en forma de campana.

Cálculo de varianza y desviación estándar

Cada distribución gaussiana puede identificarse de manera única por su media y su *varianza* (o *desviación estándar)*. Tanto la varianza como las medidas de desviación estándar indican el ancho de la distribución y su variabilidad con respecto a la media. Si la desviación estándar o la varianza es pequeña, podemos esperar que la distribución sea estrecha, con los puntos de datos densamente agrupados alrededor de la media. Si la varianza y la desviación estándar son amplias, la distribución de datos se distribuirá en consecuencia. La única diferencia entre las dos métricas se refiere a sus unidades de medida. La varianza es igual a SD^2. Por lo tanto, en nuestro ejemplo de la estatura

de los alumnos, la desviación estándar (DE) y la varianza se expresarían en cm y cm^2 respectivamente.

Comencemos calculando la *varianza* para dos alumnos de 155 cm y 165 cm de estatura de una población con una estatura promedio de 160 cm. El proceso es simple:

1. Primero calculamos la desviación restando el promedio de la población de la estatura de cada sujeto y luego elevando el resultado al cuadrado:

$$(155-160)^2=25$$
$$(165-160)^2=25$$

2. Para finalizar, simplemente calculamos el promedio de las desviaciones al cuadrado:

$$(5^2+5^2)/2 = 25\ cm^2$$

Una vez que conocemos la varianza, es fácil convertir esto a una DE tomando su raíz cuadrada. Basado en el ejemplo anterior, la DE sería $\sqrt{25} = 5$ cm. Esto significa que una DE abarcará muestras que caen dentro del área 5 cm a cada lado de la media de 160 cm. Dos DE abarcarán muestras que ocupen los siguientes 5 cm a cada lado después de eso, etc. La figura 3.8 muestra los puntos de datos que caen dentro de una DE a cada lado de la media.

La DE es útil para ayudarnos a visualizar la probabilidad de encontrar una muestra con un valor de atributo dado. Como podemos ver, es muy probable que encontremos alumnos con una estatura en el rango de 155 cm a 165 cm, mientras que es significativamente menos probable hallar alumnos con valores de estatura en el extremo izquierdo y derecho.

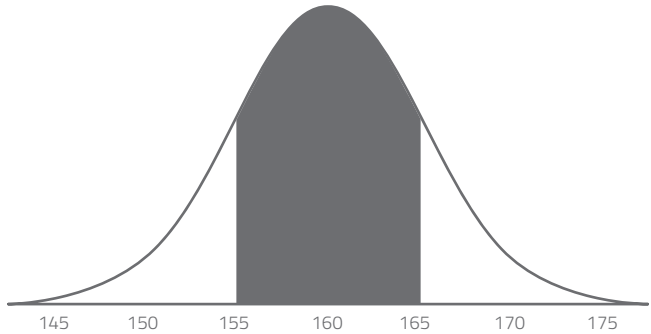


FIGURA 3.8: El área sombreada de la curva muestra la porción de la distribución que cae dentro de una desviación estándar a cada lado de la media.

Aplicación de los conceptos de distribución gaussiana al agrupamiento

Considere una situación en la cual recolectamos datos erróneos sobre la estatura de los alumnos de escuela secundaria y primaria, y no registramos a qué nivel pertenecía cada uno de estos alumnos. Al mezclar nuestras muestras, ¿sería posible "deshacer la mezcla" de algún modo?

Sería razonable esperar que los alumnos de primaria, en promedio, sean más bajos que sus contrapartes de la escuela secundaria. También es previsible que la distribución de la estatura entre los dos grupos sea algo diferente, ya que las escuelas primarias suelen incluir alumnos desde el jardín de infantes hasta el quinto grado, mientras que las escuelas secundarias abarcan a niños de sexto a octavo grado.

Las variables continuas, como la estatura de los alumnos, generalmente se pueden ajustar a una distribución gaussiana. También sabemos que cada distribución gaussiana puede identificarse de manera única por sus parámetros de media y varianza. Por lo tanto, debería ser teóricamente posible distinguir los dos grupos de alumnos entre sí. En términos de Aprendizaje Automático, deberíamos poder asignar a cada

alumno al grupo de escuela primaria o secundaria según el valor de la característica, la *estatura del alumno*.

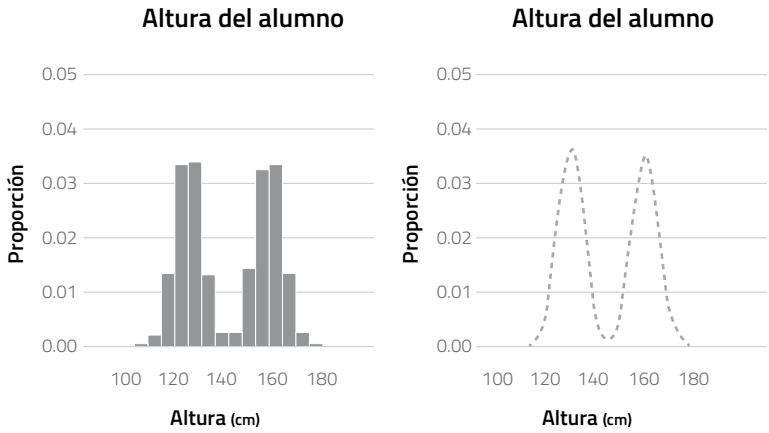


FIGURA 3.9: GMM divide a los alumnos de primaria y secundaria en grupos basados en sus exclusivas distribuciones gaussianas

El MMG puede lograr esto al analizar el conjunto de datos de estatura de los alumnos e identificar los parámetros únicos de media y varianza que definen cada grupo. A continuación, utiliza estos y muchos otros parámetros que examinaremos en breve para calcular la probabilidad de que cada alumno pertenezca a cada uno de los grupos. Luego, se asigna al alumno al grupo con la puntuación de probabilidad más alta.

En la Figura 3.9, podemos ver que el MMG ha reestructurado con éxito las distribuciones gaussianas por separado para los alumnos de primaria y secundaria. Como antes, los gráficos de la izquierda y la derecha muestran estas distribuciones gaussianas como histogramas y curvas suavizadas, respectivamente.

Tenga en cuenta que hemos explicado estos conceptos con ilustraciones y ejemplos simples para mayor claridad y comprensión. Por ejemplo:

- En las Figuras 3.7 y 3.8, nos referimos a las medidas en el eje *y* como *proporciones*. Sin embargo, por razones técnicas, el término más preciso es *densidad*. Para obtener más información sobre la densidad y su función en la resolución de problemas probabilísticos, consulte https://es.wikipedia.org/wiki/Funci%C3%B3n_de_densidad_de_probabilidad
- Nuestro análisis sobre la varianza se basó en una situación problemática utilizando solo una característica única. Sin embargo, los problemas probabilísticos típicamente involucran múltiples características y requieren un análisis de sus relaciones de *covarianza* asociadas. Para obtener más información sobre la covarianza y su función en la resolución de problemas probabilísticos, consulte https://es.wikipedia.org/wiki/Covarianza

Echemos un vistazo más de cerca al proceso que el MMG utiliza para asignar puntos de datos a los grupos.

MAXIMIZACIÓN DE EXPECTATIVAS

Para realizar el agrupamiento, el MMG emplea la *Maximización de Expectativas* (ME), un proceso iterativo de optimización en dos pasos que también es utilizado por muchos otros algoritmos de Aprendizaje Automático para resolver problemas probabilísticos. Para comenzar, el analista establece cuatro valores iniciales:

- **Hiperparámetro fijo: Cantidad de mezclas** Esto determina cuántos grupos se crearán. En el MMG, usamos el término *mezclas* en lugar de grupos porque la asignación de puntos de datos a grupos se logra "deshaciendo la mezcla" de una colección de distribuciones gaussianas o *componentes de mezclas* discretas. La elección del

número de mezcla suele ser intuitiva en función de la naturaleza del problema, el grado de precisión requerido y el conocimiento de dominios del analista. En un problema de detección de spam, por ejemplo, lo más probable es que el analista esté interesado en definir solo dos mezclas, una para el spam y otra para el correo legítimo. Sin embargo, también se pueden usar métodos más avanzados para optimizar el número de componentes de la mezcla.

- **Parámetro 1: Proporción de mezcla (PM)** Esta es una estimación de la proporción de muestras que pertenecen a cada componente de la mezcla. En nuestro ejemplo de la estatura de los alumnos, el analista puede decidir establecer inicialmente que la PM para la mezcla de la escuela primaria sea igual a dos tercios de los alumnos y asignar el tercio restante a los alumnos de la escuela secundaria. La PM se expresa en términos de probabilidad, con valores que van de 0 a 1. En consecuencia, a las mezclas de las escuelas primaria y secundaria se les asignaría una PM de alrededor de 0,66 y 0,34 respectivamente.
- **Parámetro 2: Media** Este es el valor promedio de la distribución de datos para cada componente de la mezcla, por ejemplo, la estatura promedio de los alumnos en nuestros componentes de mezcla de escuela primaria y secundaria, respectivamente.
- **Parámetro 3: Varianza (Desviación estándar o DE)** Define la concentración de los datos con respecto a la media. El analista a menudo estima esto para cada componente de la mezcla en función de su experiencia y la naturaleza del conjunto de datos. Por ejemplo, si la característica utilizada para agrupar exhibe una

amplia gama de valores, se elegiría un parámetro de DE igualmente amplio.

Es deseable, pero no esencial, que el conjunto inicial de valores de PM, Media y DE/Varianza sean realistas. Sin embargo, en el transcurso de múltiples iteraciones del MMG, estos parámetros se perfeccionarán progresivamente hasta que el analista esté satisfecho con el rendimiento del modelo de agrupamiento o se haya alcanzado el número máximo de iteraciones.

Paso 1: Expectativa

En este paso, el MMG usa los parámetros de media, varianza y proporción de la mezcla de cada mezcla para asignar una puntuación de probabilidad entre 0 y 1 a cada punto de datos. Esto indica cuál de las distribuciones gaussianas es *responsable* de ese punto de datos: la probabilidad de que pertenezca a cada uno de los componentes de la mezcla. En el caso de la estatura de nuestros alumnos, a cada alumno se le asignarán *dos* puntuaciones de responsabilidad: una para los componentes de mezcla de la escuela primaria, y otra para secundaria. Luego, el punto de datos se asigna al componente de la mezcla con la puntuación de responsabilidad más alta.

Una vez que el MMG termina de asignar puntuaciones de responsabilidad a todos los puntos de datos, pasa a la etapa de Maximización.

Paso 2: Maximización

Durante este paso, los promedios ponderados se calculan y luego se usan para modificar los parámetros de media y varianza. Este proceso procede de la siguiente manera:

1. El modelo MMG recalcula las proporciones de la mezcla sumando las puntuaciones de responsabilidad para cada

mezcla y luego dividiendo la suma entre la cantidad total de puntos de datos. En nuestro ejemplo de estatura, el MMG sumaría las puntuaciones de responsabilidad para el componente de la mezcla de la escuela secundaria y luego dividiría esto entre el número total de alumnos. Luego repetiría este proceso para el componente de la mezcla de la escuela primaria.

2. El MMG actualiza la media calculando un *promedio ponderado*.
    - El MMG multiplica el valor de característica para cada miembro de la mezcla por su puntuación de responsabilidad correspondiente para producir un valor ponderado. Por ejemplo, a un alumno con una estatura de 120 cm y una puntuación de responsabilidad de 0,9 de pertenecer al componente de la mezcla de la escuela primaria se le asignaría un nuevo valor de estatura ponderada de 108 cm (120 cm*0,9 = 108 cm).
    - El MMG calcula una nueva media ponderada para cada componente de la mezcla. Esto se logra sumando los valores ponderados totales para ese componente de la mezcla y luego dividiendo el total por la suma de las puntuaciones de responsabilidad de esa mezcla. A continuación, repite el mismo proceso con cada uno de los componentes de la mezcla restantes.
3. El MMG calcula una nueva varianza para cada componente de la mezcla. Anteriormente, explicamos la varianza como un promedio simple de las diferencias al cuadrado entre las estaturas y la media. Ahora, sin embargo, usamos el promedio ponderado para calcular esto en su lugar.

Paso 3: Iteración del ciclo de ME

Con los parámetros revisados, el MMG repite el ciclo de ME varias veces, produciendo nuevas puntuaciones de responsabilidad y luego aplicándolas para modificar los parámetros de la mezcla. El proceso continúa hasta que ocurre una de dos cosas. O bien los valores dejan de cambiar dentro de un nivel de tolerancia predefinido (es decir, los medios ponderados permanecen constantes dentro de un rango de 1 %), o se ha completado el número predefinido de iteraciones de ME. En ese momento, a cada punto de datos se le habrá añadido una puntuación de responsabilidad que indica su pertenencia a la mezcla más probable.

La tolerancia aceptable para la precisión será determinada por el analista según su conocimiento de dominios, la naturaleza del problema de agrupamiento y el riesgo asociado. Por ejemplo, una tolerancia a los errores amplia puede ser aceptable para detectar un correo electrónico como spam pero no para detectar una transacción bancaria como fraudulenta.

El modelo MMG resultante consiste en un vector que contiene la proporción de la mezcla, la media y los valores de varianza que definen cada distribución gaussiana. El modelo se puede aplicar a nuevos datos para generar puntuaciones de responsabilidad que indiquen la pertenencia más probable a una mezcla de cada punto de datos.

ESCOLLOS Y LIMITACIONES DEL MODELO MMG

Como hemos visto, el uso del MMG de los métodos probabilísticos para definir grupos tiene varias ventajas sobre los algoritmos *k*-means y DBSCAN que analizamos en el Capítulo 1, aunque con ciertas limitaciones:

1. El analista debe elegir la cantidad correcta de componentes de mezcla para crear.

2. El MMG requiere datos continuos, como la estatura de los alumnos que utilizamos en nuestro ejemplo. También funciona mejor cuando los datos se ajustan a una distribución gaussiana.
3. Nuestro ejemplo de MMG utilizó solo una característica (estatura) para simplificar. En la práctica, el MMG funciona mejor con espacios de características de seis dimensiones o menos. Luego, el MMG se vuelve cada vez más costoso a nivel computacional y, en última instancia, puede no lograr la convergencia.

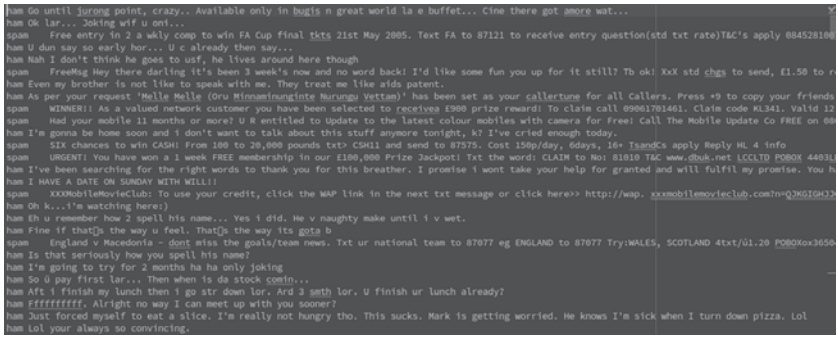
Detección de spam de SMS con modelos de mezclas gaussianas y clasificador bayesiano ingenuo

El spam de SMS es un intento intrusivo, ofensivo y, a menudo, fraudulento por parte de comerciantes inescrupulosos para obligar a los usuarios de teléfonos móviles a realizar compras o, lo que es peor, a proporcionar información personal a los atacantes que cometen fraudes de phishing. Afortunadamente, los desarrolladores de aplicaciones se han vuelto cada vez más expertos en la utilización de métodos probabilísticos para detectar estos mensajes. Consideraremos dos ejemplos de detección probabilística de spam de SMS con los algoritmos MMG y bayesiano ingenuo provistos con scikit-learn.

Primero necesitaremos un conjunto de datos para trabajar. Utilizaremos el conjunto de datos de SMS Spam Collection que se puede descargar en formato de archivo comprimido desde el sitio web del Centro de Aprendizaje Automático y Sistemas Inteligentes en https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection.

A continuación, extraemos y cargamos el archivo de texto *SMSSpamCollection*. Cuando examinamos el contenido del

archivo, notamos que la primera palabra en cada línea es en realidad una etiqueta que indica si el mensaje es *spam* o *ham* (un mensaje benigno). Como se muestra a continuación, el archivo de texto contiene significativamente más mensajes ham que spam. Realizaremos el agrupamiento de forma no supervisada, por lo que no utilizaremos estas etiquetas para asignar muestras a los grupos. Sin embargo, las etiquetas nos permitirán luego evaluar la precisión de nuestros resultados de agrupamiento.



Para preparar estos mensajes para el análisis, analizaremos las transiciones entre cada conjunto secuencial de dos caracteres en cada cadena de caracteres, un proceso conocido como *análisis de bigrama*. Por ejemplo, en el primer mensaje de spam que comienza con "Free Entry" (Entrada libre), observaremos las transiciones entre las letras *F* y *r*, *r* y *e*, *e* y *e*, etc. Cuando calculemos la frecuencia con la que cada tipo de transición ocurre, los modelos que entrenamos podrán determinar si un mensaje es benigno o un intento de publicitar o subvertir.

Vectorización

Una forma de analizar cadenas de texto es considerar que cada carácter pertenezca a una *clase de caracteres* en particular. Utilizaremos cuatro clases de caracteres en nuestro ejemplo:

- Letras
- Dígitos
- Puntuación
- Espacio en blanco

Con estas definiciones de clase de caracteres, convertiremos la cadena de texto "abc123" en una *secuencia de clase de caracteres*. Asignaremos la clase de caracteres 0 a las letras y la clase de caracteres 1 a los dígitos. Esto produce la siguiente secuencia:

Cadena de caracteres	a	b	c	1	2	3
Secuencia de clase de caracteres	0	0	0	1	1	1

Ahora, podemos convertir esta secuencia en un conjunto de bigramas.

0, 0
0, 0
0, 1
1, 1
1, 1

A continuación, contamos el número de bigramas únicos y descubrimos que hay *dos* instancias de (*0, 0*), *una* instancia de (*0, 1*), *dos* instancias de (*1, 1*), y ninguna instancia de (*1,0*). Usamos estas sumas para crear la siguiente matriz:

2	**1**
2	0

Nuestro último paso de la preparación de datos es aplanar la matriz en un vector adecuado para su uso por los algoritmos de MMG y bayesiano ingenuo. Cada uno de los vectores resultantes estará compuesto por 16 dimensiones (cuatro posibles clases de

caracteres para el primer carácter * cuatro posibles clases de caracteres para el segundo carácter). Al limitar el número de dimensiones de esta manera, nos aseguramos de que nuestros cálculos basados en el modelo MMG y el clasificador bayesiano ingenuo se ejecutarán fácilmente en una computadora personal sin sacrificar indebidamente nuestra clasificación y precisión de agrupamiento.

Ejemplo 1: Identificación de spam de SMS con MMG

Ahora que tenemos nuestros vectores, estamos listos para comenzar nuestra sesión de análisis de MMG. Como lo hemos hecho a lo largo de este libro, utilizaremos los scripts que desarrollamos expresamente para ilustrar conceptos de Aprendizaje Automático basados en las funciones proporcionadas por la biblioteca scikit-learn. Este script de MMG en particular se llama *cluster_with_gmm.py*.

Como podemos ver, esta versión del algoritmo MMG acepta diversos hiperparámetros, aunque solo se requiere la ruta al conjunto de datos.

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py -h
usage: cluster_with_gmm.py [-h] [-n N_COMPONENTS] [-r]
                           [-c {full,tied,diag,spherical}]
                           dataset

Cluster SMS messages with Gaussian Mixture Models

positional arguments:
  dataset               Path to dataset to read

optional arguments:
  -h, --help            show this help message and exit
  -n N_COMPONENTS, --n_components N_COMPONENTS
                        Number of clusters to produce
  -r, --print-results   Print out results per sample
  -c {full,tied,diag,spherical}, --covariance-type {full,tied,diag,spherical}
                        Covariance type
```

Si ejecutamos *cluster_with_gmm.py* con su configuración predeterminada, solo produce un grupo.

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py data/SMSSpamCollection
Cluster 0 - Total Samples: 5574 - Percent Ham: 86.5984930032 - Percent Spam: 13.4015069968
```

Las proporciones de ham/spam de este grupo son interesantes, pero lo que *realmente* queremos saber es qué mensajes son spam y cuáles son benignos. Por lo tanto, estableceremos *n* (*número de componentes*) en *dos* y luego ejecutaremos el comando nuevamente.

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py -n 2 -c full data/SMSSpamCollection
Cluster 0 - Total Samples: 4401 - Percent Ham: 98.9775051125 - Percent Spam: 1.02249488753
Cluster 1 - Total Samples: 1173 - Percent Ham: 40.1534526854 - Percent Spam: 59.8465473146
```

El aspecto positivo es que podemos ver que el Grupo 0 se compone casi exclusivamente de mensajes ham. Sin embargo, la proporción para el Grupo 1 es 60/40 entre los mensajes de spam y los mensajes ham respectivamente, por lo que será necesario un mayor refinamiento de nuestra configuración de hiperparámetros. Vamos a experimentar con los ajustes de covarianza *vinculada*, *diagonal* y *esférica*.

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py -n 2 -c diag data/SMSSpamCollection
Cluster 0 - Total Samples: 1209 - Percent Ham: 41.7700578991 - Percent Spam: 58.2299421009
Cluster 1 - Total Samples: 4365 - Percent Ham: 99.0148911798 - Percent Spam: 0.98510882016
```

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py -n 2 -c spherical data/SMSSpamCollection
Cluster 0 - Total Samples: 3006 - Percent Ham: 98.9354624085 - Percent Spam: 1.06453759148
Cluster 1 - Total Samples: 2568 - Percent Ham: 72.1573208723 - Percent Spam: 27.8426791277
```

```
bwall@highwind:~/code/private/probability_example$ python cluster_with_gmm.py -n 2 -c tied data/SMSSpamCollection
Cluster 0 - Total Samples: 5028 - Percent Ham: 95.7239459029 - Percent Spam: 4.27605409706
Cluster 1 - Total Samples: 546 - Percent Ham: 2.5641025641 - Percent Spam: 97.4358974359
```

La configuración *vinculada* produjo los mejores resultados, con aproximadamente el 95 % de las muestras ham asignadas al Grupo 0 y el 97 % de las muestras de spam asignadas al Grupo 1. Si bien realizamos la agrupación de manera no supervisada, sabemos por las etiquetas proporcionadas con nuestro conjunto de datos que estas asignaciones de grupo son bastante precisas. Terminaremos exportando las muestras en cada grupo para que podamos ver los mensajes reales. Utilizaremos el indicador *–r* (*resultados de salida por muestra*) para esto.

Como podemos ver, casi todos los mensajes spam y ham han sido clasificados correctamente.

Ejemplo 2: Identificación de spam de SMS con el modelo bayesiano ingenuo

Esta vez, tomaremos un enfoque de aprendizaje supervisado para la detección de spam utilizando las etiquetas suministradas con nuestro conjunto de datos. También podemos reutilizar los mismos vectores y secuencias de clases de caracteres que usamos antes. Por lo general, comenzaríamos dividiendo nuestras muestras en conjuntos de entrenamiento, prueba y validación por separado. Para simplificar, sin embargo, ilustraremos el proceso de entrenamiento solamente.

Estamos utilizando una función bayesiana ingenua de distribución multinomial provista por scikit-learn llamada *MultinomialNB* para crear nuestro modelo. Una vez más, el único parámetro requerido es la ruta al conjunto de datos.

```
bwall@highwind:~/code/private/probability_example$ python train_nb.py -h
usage: train_nb.py [-h] [-f] [-a ALPHA] dataset

Train model to classify SMS as spam or ham

positional arguments:
  dataset               Path to dataset to read

optional arguments:
  -h, --help            show this help message and exit
  -f, --fit-prior       Learn class prior probabilities
  -a ALPHA, --alpha ALPHA
                        Smoothing Parameter
```

Cuando lo ejecutamos con su configuración predeterminada (*alfa = 1,0*, *fit_prior = Falso*), encontramos que el clasificador ha logrado una precisión promedio de aproximadamente 95 %.

```
bwall@highwind:~/code/private/probability_example$ python train_nb.py data/SMSSpamCollection
Predict(Sunshine Quiz Wkly Q! Win a top Sony DVD player if u know which country the Algarve is in? Txt ansr to 82277. £1.50 SP:Tyrone)
[[  1.32093912e-09   9.99999999e-01]]
Predict(What time you coming down later?)
[[ 0.9859347  0.0140653]]
Classification report for testing set
Accuracy: 0.954088952654
             precision    recall  f1-score   support

          0       0.98      0.97      0.97      1202
          1       0.81      0.88      0.84       192

avg / total       0.96      0.95      0.95      1394
```

Veamos si podemos mejorar la precisión al permitir que el clasificador bayesiano ingenuo aprenda las probabilidades a priori de clase. Esto lo logramos cambiando la configuración del hiperparámetro *fit_prior* a *verdadero*.

```
bwall@highwind:~/code/private/probability_example$ python train_nb.py data/SMSSpamCollection -f
Predict(Sunshine Quiz Wkly Q! Win a top Sony DVD player if u know which country the Algarve is in? Txt ansr to 82277. £1.50 SP:Tyrone)
[[  8.18576852e-09   9.99999992e-01]]
Predict(What time you coming down later?)
[[ 0.99738247  0.00261753]]
Classification report for testing set
Accuracy: 0.963414634146
             precision    recall  f1-score   support

          0       0.98      0.98      0.98      1206
          1       0.85      0.88      0.87       188

avg / total       0.96      0.96      0.96      1394
```

Como podemos ver, hemos logrado aumentar la precisión promedio del modelo a aproximadamente 96 %. Ahora podemos aplicar el modelo a nuevas muestras.

Cuando lo hacemos, el modelo predice que el mensaje debe clasificarse como ham.

```
bwall@highwind:~/code/private/probability_example$ python classify_nb.py "I bought a dog"
[ham]  0.978657044375
[spam]  0.021342955625
Prediction: ham
```

Este segundo mensaje ha sido clasificado como spam.

```
bwall@highwind:~/code/private/probability_example$ python classify_nb.py "Want to buy Male Verility Treatment! Txt ansr to 12345"
[ham]  1.23879674572e-06
[spam]  0.999998761203
Prediction: spam
```

Conclusiones del modelo de probabilidad

En este capítulo, consideramos diversos conceptos de probabilidad e ilustramos cómo los algoritmos del modelo de mezclas gaussianas y bayesiano ingenuo emplean medidas de probabilidad para realizar sus respectivas funciones de clasificación y agrupamiento. Algunos de los puntos clave que se incluyeron:

- La diferencia entre probabilidad *condicional* y *conjunta*. La probabilidad condicional se refiere a determinar la probabilidad de que el Evento B siga al Evento A, mientras que la probabilidad conjunta se refiere a determinar la probabilidad de que ambos eventos ocurran simultáneamente.
- En contraste con el Teorema de Bayes, el clasificador bayesiano ingenuo supone *independencia condicional de clase*, lo que significa que se considera que cada característica no tiene influencia sobre ninguna otra al tomar decisiones de clasificación. Si bien la suposición es casi siempre incorrecta, el clasificador bayesiano ingenuo a menudo produce excelentes resultados y con gran eficiencia ya que solo requiere cuatro componentes para clasificar una muestra. Estos incluyen: *Probabilidad*

a posteriori, *Probabilidad a priori de clase*, *Probabilidad a priori predictiva* y *Probabilidad*.

- Cada distribución gaussiana puede identificarse de manera única por su *media* y su *varianza* (o desviación estándar). El modelo MMG realiza el agrupamiento a través de un proceso iterativo de *maximización de expectativas* que genera puntuaciones que indican qué distribución gaussiana es más probable que sea *responsable* de cada punto de datos. En consecuencia, el MMG puede detectar correctamente grupos que se superponen o poseen una forma no circular. El MMG requiere que el analista configure solo un hiperparámetro: cantidad de mezclas. Luego, el MMG asigna puntos de datos a los grupos al calcular la proporción, la media y la varianza de la mezcla para cada uno de los componentes de la mezcla. Sin embargo, el MMG solo es adecuado para resolver problemas en los que los datos subyacentes se componen de valores continuos y las muestras se ajustan a una distribución Gaussiana.

Aprendizaje profundo

En capítulos anteriores, consideramos cómo se podrían aplicar distintos enfoques de Aprendizaje Automático para resolver un problema de agrupamiento o clasificación. Aunque los detalles variaron, el proceso se llevó a cabo a través de una secuencia de operaciones relativamente simple. Un algoritmo recibía un conjunto de vectores como entrada, realizaba un cálculo y luego producía una etiqueta o asignación de grupo como salida. En el caso de la regresión logística, por ejemplo, el motor de clasificación calculó las ponderaciones de regresión. Para realizar el agrupamiento, *k*-means calcula distancias euclidianas y otras distancias entre los vectores y sus centroides.

Los analistas pueden controlar cómo se realizaron estos cálculos y los resultados que obtuvieron utilizando una variedad de parámetros e hiperparámetros. Con la regresión logística, por ejemplo, los analistas pueden aplicar parámetros de

regularización y penalización que influyen en cómo se calculan las ponderaciones. Asimismo, podrían aplicarse distintas funciones de optimización para determinar aspectos tales como el tamaño mínimo de cada corrección de ponderación y lo que constituye un nivel suficiente de convergencia. Sin embargo, estos fueron esencialmente métodos para ajustar el comportamiento de cada algoritmo de forma muy parecida a como se controla el motor de un automóvil manipulando el flujo de combustible o la velocidad de combustión. Si bien son esenciales, estas funciones auxiliares no produjeron una decisión de clasificación o agrupamiento por sí mismas, ni se envió la salida de un tipo de algoritmo a la entrada de otro.

El aprendizaje profundo se basa en un enfoque fundamentalmente diferente que incorpora capas de procesamiento, donde cada capa realiza un tipo de cálculo diferente. Las muestras se procesan capa por capa de forma gradual, y la salida de cada capa proporciona la entrada para la siguiente. Al menos una de estas capas de procesamiento estará "oculta". Es este enfoque multicapa, que emplea capas ocultas, lo que distingue el aprendizaje profundo de todos los demás métodos de Aprendizaje Automático.

El término *aprendizaje profundo* en realidad abarca una amplia gama de métodos de aprendizaje no supervisados, semisupervisados, supervisados y reforzados basados principalmente en el uso de *redes neuronales*, una clase de algoritmos llamados así porque simulan la forma en que las redes de neuronas densamente interconectadas interactúan en el cerebro.

Originalmente concebidas en la década de 1950, las redes neuronales recibieron una gran atención en la década de 1980 gracias a su potencial para producir "máquinas inteligentes" que pudieran "pensar" y resolver problemas al igual que los humanos.

Sin embargo, hacia el final de esa década, el entusiasmo por esta tecnología naciente comenzó a menguar. Las computadoras del momento simplemente no eran lo suficientemente potentes como para manejar las cargas de procesamiento requeridas para resolver los problemas más complejos e interesantes. Los investigadores también encontraron difícil obtener los conjuntos de datos masivos necesarios para entrenar adecuadamente los modelos de redes neuronales. En consecuencia, muchos investigadores centraron su atención en los enfoques de Aprendizaje Automático que eran más económicos desde el punto de vista de los datos y la perspectiva del procesamiento.

El interés en este tema se renovó en 2005 y 2006, gracias a los esfuerzos de los investigadores de Inteligencia Artificial Geoff Hinton, Yoshua Bengio, Yann Lecun y Jurgen Schmidhuber, entre otros, que demostraron que las redes neuronales no solo eran prácticas sino también capaces de resolver problemas complejos, como el reconocimiento de voz y la categorización de imágenes, con mucha más precisión que los métodos existentes. Las unidades de procesamiento de gráficos (GPU, por sus siglas en inglés), como las incorporadas en las consolas de juegos, ahora podrían proporcionar la potencia de cálculo necesaria para ejecutar algoritmos de redes neuronales de manera eficiente y asequible. Las bases de datos en línea masivas también se tornaron disponibles gracias al crecimiento explosivo de Internet. Los gigantes del software como Google y Microsoft comenzaron a invertir fuertemente en la investigación de aprendizaje profundo. Un aluvión de nuevos productos que incorporaban tecnologías de aprendizaje profundo comenzó a surgir poco después. Hoy en día, las redes neuronales se han integrado a productos tan comunes como el sistema operativo Android de Google y Siri, el asistente digital activado por voz de Apple. El aprendizaje profundo también ha demostrado ser

muy eficaz para abordar un amplio espectro de problemas de seguridad de redes.

En este capítulo, describiremos dos tipos diferentes de algoritmos de redes neuronales:

1. *Memoria a largo y corto plazo* (*MLCP*), un tipo de *Red neuronal recurrente* (*RNR*)
2. *Redes neuronales convolucionales* (*RNC*)

Tenga en cuenta: Las redes neuronales son sumamente flexibles: algoritmos generales que pueden resolver innumerables problemas en un sinfín de formas. A diferencia de otros algoritmos, por ejemplo, las redes neuronales pueden tener millones o incluso miles de millones de parámetros aplicados para definir un modelo. Para simplificar, nos enfocaremos exclusivamente en las capacidades de las redes neuronales para resolver un problema de clasificación junto con un pequeño conjunto de hiperparámetros representativos. Concluiremos con un ejemplo práctico y concreto que muestra cómo se pueden aplicar los modelos MLCP y RNC para determinar la longitud de la clave de cifrado XOR utilizada para ofuscar una muestra de texto.

Una arquitectura de red neuronal genérica

Como se muestra en la Figura 4.1, las redes neuronales se componen de *nodos* contenidos dentro de las capas de entrada, ocultas y de salida. Cada capa juega un papel particular en el cálculo de una clasificación. En una red *completamente conectada* como la que se muestra aquí, la salida de cada nodo en una capa determinada se conecta a las entradas de cada nodo en la capa siguiente. Este es también un ejemplo de una red neuronal *prealimentada (feedforward)* en la que la información pasa directamente de una capa a la siguiente sin retroceder,

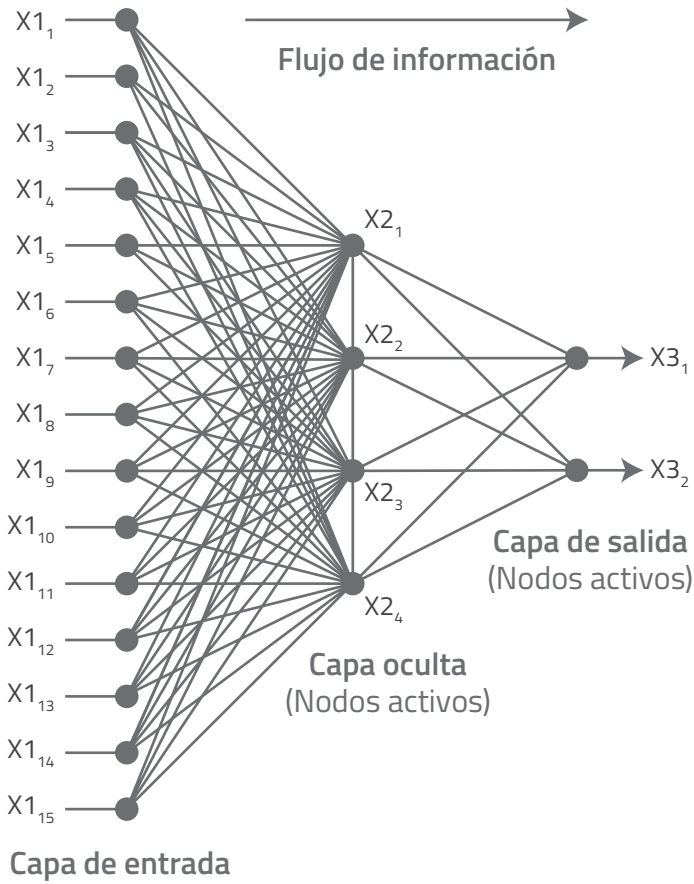


FIGURA 4.1: Una arquitectura de red neuronal genérica

hasta que llega a la capa de salida, donde se asigna una decisión de clasificación.

Como veremos, sin embargo, esta es solo una de las muchas configuraciones posibles. Las redes neuronales también pueden emplear bucles de retroalimentación entre capas y utilizar arquitecturas *parcialmente conectadas* que restringen el flujo de información solo a ciertos nodos. Empecemos, sin embargo, examinando las capas que comprenden uno de los tipos de prealimentados completamente conectados.

CAPA DE ENTRADA

Los nodos en la capa de entrada son pasivos. Simplemente reciben valores de atributo para una muestra en particular y luego los pasan a todos los nodos en la primera capa oculta para su procesamiento. En consecuencia, la capa de entrada debe contener un nodo para cada característica en el conjunto de muestras. Si categorizáramos imágenes con una resolución de 64 píxeles x 64 píxeles, por ejemplo, configuraríamos una capa de entrada con 4.096 nodos de entrada, uno para cada píxel. Al resolver un problema de procesamiento del lenguaje, las características relevantes pueden incluir el número de palabras únicas en la muestra que estamos analizando, la frecuencia con la que aparece cada palabra, etc.

CAPAS OCULTAS

Las capas ocultas están compuestas por nodos como el de la Figura 4.2 que realizan el trabajo pesado del proceso de aprendizaje profundo. Dado que este es el primer nodo en la primera capa oculta, este procesamiento se realiza de la siguiente manera:

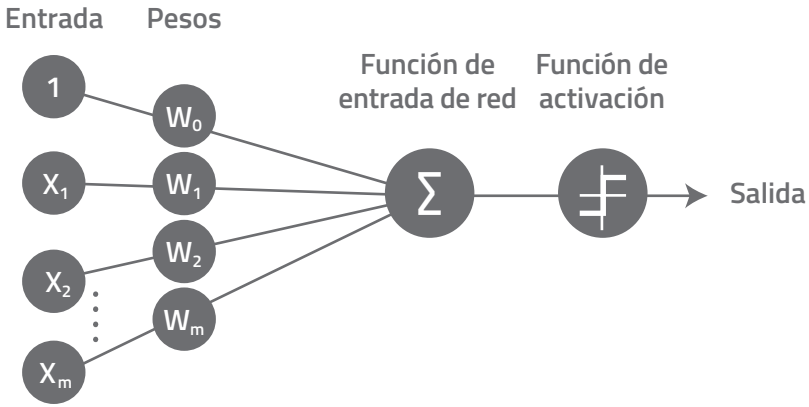


FIGURA 4.2: Nodo n.° 1 en capa oculta n.° 1

- **Recepción de valores de las características de la capa de entrada:** Todos los valores de atributo para la primera muestra se reciben en las entradas x_1-x_m del nodo.
- **Aplicación de ponderaciones:** Cada valor de atributo se multiplica por un valor de ponderación correspondiente, por ejemplo, el atributo en la entrada x_1 se multiplica por la ponderación w_1, el atributo en x_2 se multiplica por la ponderación w_2, etc. Si la magnitud del valor de ponderación es mayor que uno, entonces la contribución de esa característica a la eventual decisión de clasificación aumentará gradualmente. Si la magnitud es menor que uno, su contribución se reducirá en consecuencia. Este proceso es similar a la forma en que se utilizan las ponderaciones de regresión en la regresión logística. Sin embargo, con las redes neuronales, es el procesamiento global en todas las capas ocultas lo que finalmente determina la decisión de clasificación, no el procesamiento dentro de una sola capa oculta.

  Además de las ponderaciones de las características w_1-w_m, notará un valor de ponderación con la etiqueta w_0. Este es el valor de *tendencia* o *intersección en el eje y* para el nodo n.° 1. Como vimos en capítulos anteriores, la tendencia permite que el límite de decisión en el espacio de características se mueva desde el punto de origen en el eje *y* a una ubicación que proporcione el mejor ajuste para los datos de muestras.

  Cada entrada en cada nodo en las capas ocultas se inicializa con diferentes valores de ponderación, que se optimizan incrementalmente hasta alcanzar el nivel deseado de precisión. Los analistas pueden establecer estas ponderaciones iniciales de forma aleatoria, usar funciones para estimar los valores iniciales apropiados

o establecerlos en función de su experiencia previa con problemas y conjuntos de datos similares.

- **Suma de los productos:** Los productos se envían a una función de *entrada de red* que calcula la suma de los productos y pasa el resultado a una *función de activación* para un procesamiento adicional.
- **Aplicación de la función de activación:** La *función de activación* realiza el cálculo particular especificado para esa capa. Los analistas pueden elegir entre un gran conjunto de funciones de activación basadas en la naturaleza del problema y la secuencia de cálculos necesarios para producir una solución.
- **Producción del resultado:** El resultado de esta función de activación es un valor numérico que refleja los efectos globales del procesamiento de ese nodo. Cada uno de estos valores representa una proporción diferente de ponderaciones y combinaciones de atributos de las características. Al procesar todas estas combinaciones y pasar los resultados a capas ocultas adicionales, las redes neuronales pueden determinar, paso a paso, qué combinación de características y ponderaciones predecirá con mayor precisión la asignación de clase de una muestra.

Procesamiento en capas ocultas 2-n

Cada nodo en la capa oculta n.° 2 recibe los valores de salida de *todos* los nodos en la capa oculta n.° 1. Una vez más, cada uno de estos valores se multiplica por un valor de ponderación particular, los productos se suman, y luego los resultados se someten a una función de activación para producir un nuevo resultado para la siguiente capa, donde el proceso se repite. Esto continúa hasta que todas las capas ocultas hayan sido atravesadas y los resultados de esos cálculos lleguen a la capa de salida.

CAPA DE SALIDA

La capa de salida es la capa final en la red neuronal. Como el nuestro es un ejercicio de clasificación, la capa de salida incorporará un nodo para cada posible asignación de clase. Al igual que las capas ocultas, los nodos aquí están activos, lo que significa que también pueden incorporar funciones de activación. Por ejemplo, se puede aplicar una función de activación *logit* o *softmax* para convertir la decisión de clasificación en una puntuación de probabilidad. El nodo con la puntuación más alta determinará qué etiqueta de clase asignar.

Después de cada ciclo de entrenamiento, una *función de pérdida* compara la decisión de clasificación con las etiquetas de clase para determinar cómo se deben modificar las ponderaciones en todas las capas ocultas a fin de producir un resultado más preciso. Este proceso se repite tantas veces como sea necesario antes de que un conjunto de modelos candidatos pueda pasar a las fases de validación y prueba.

Tenga en cuenta: Este documento no describe cómo los optimizadores interactúan con los procesos de *retropropagación* para calcular y aplicar los cambios de ponderación y cómo estos procesos se implementan de forma diferente al entrenar los modelos MLCP y RNC. Los lectores a quienes les gustaría aprender más...

Aumento de los niveles de abstracción

Como se señaló anteriormente, los valores de las características de una muestra solo son visibles para los nodos en la capa de entrada y la primera capa oculta. Todas las capas subsiguientes solo pueden "ver" los valores de salida combinados de los nodos en la capa anterior y, por lo tanto, "observar" muestras globales a niveles crecientes de abstracción. Esto es similar en

concepto a cómo nuestros cerebros perciben e interpretan la información sensorial.

En el caso de la visión, por ejemplo, el papel de la capa de entrada lo desempeña la retina, que dispara señales eléctricas a la corteza visual según la intensidad de la energía de luz que recibe de una "muestra" de fotones. Allí, múltiples "capas ocultas" aplican diferentes tipos de "funciones de activación" visuales. Por ejemplo, una capa puede realizar un procesamiento de bordes, otra puede consolidar bordes en formas y una tercera puede asociar esa forma con una categoría, como "rostro". Las redes neuronales posibilitan este tipo de procesamiento de una manera sumamente granular, pasando de señales de bajo nivel a decisiones complejas a través de una secuencia ordenada de cálculos jerárquicos de varias capas.

¿Cuántas capas ocultas se necesitan?

En términos generales, cuanto más capas ocultas usamos, mayor es la capacidad general que tiene la red neuronal para resolver problemas complejos de uso intensivo de datos. Como siempre, sin embargo, existen concesiones. Las redes neuronales con exceso de capacidad pueden producir modelos que sobreajustan los datos. Las redes neuronales con capacidad insuficiente pueden producir modelos con tasas de error inaceptables.

Durante una sesión, un analista generalmente aumentará la capacidad de un modelo hasta que se observe un sobreajuste y luego reducirá el número de capas en consecuencia. La densidad de capas y nodos también se puede modificar programáticamente aplicando algoritmos auxiliares que, por ejemplo, eliminan capas innecesarias o redundantes. Si bien los mecanismos específicos son algo diferentes, el efecto es similar a la forma en que los hiperparámetros se aplican para eliminar ramas innecesarias de un árbol de decisión.

La red neuronal de memoria a largo y corto plazo (MLCP)

Nuestra discusión, hasta ahora, se ha centrado en las redes neuronales prealimentadas y completamente conectadas, como la que se muestra en Figura 4.1. Esta arquitectura es adecuada cuando el problema que estamos resolviendo no requiere que consideremos la secuencia en la que las muestras llegan a la capa de entrada para su procesamiento. Al clasificar una serie de imágenes de animales, por ejemplo, no es necesario considerar cómo la clasificación de la primera imagen como *gato* puede influir en la clasificación de la segunda imagen como *perro*. Las redes prealimentadas no tienen noción de tiempo. Operan en el momento presente solamente, sin poder "recordar" ninguna de las muestras que procesaron antes.

Sin embargo, muchos problemas interesantes requieren que consideremos cómo el significado de una muestra puede influir en el significado de otra que ocurre más adelante. Por ejemplo, no podemos deducir la trama de una película examinando un solo cuadro. Es la secuencia de fotogramas en el tiempo lo que proporciona el significado y el contexto. Del mismo modo, no podemos identificar una conexión sospechosa al examinar un solo paquete. Es la secuencia de paquetes en el tiempo lo que nos permite determinar si ocurrió una vulnerabilidad de seguridad. Para problemas de *series de tiempo* como estos, una *red neuronal recurrente* (*RNR*) es la opción más apropiada.

Las RNR se distinguen de las del tipo prealimentado por su capacidad para considerar no solo la entrada actual, sino también su relación con la entrada que la precedió inmediatamente. Es en este sentido que se dice que las redes recurrentes poseen "memoria". Para lograr esto, las redes recurrentes emplean bucles de retroalimentación. Como se muestra en la Figura 4.3, los valores de muestra de la capa de salida se copian nuevamente

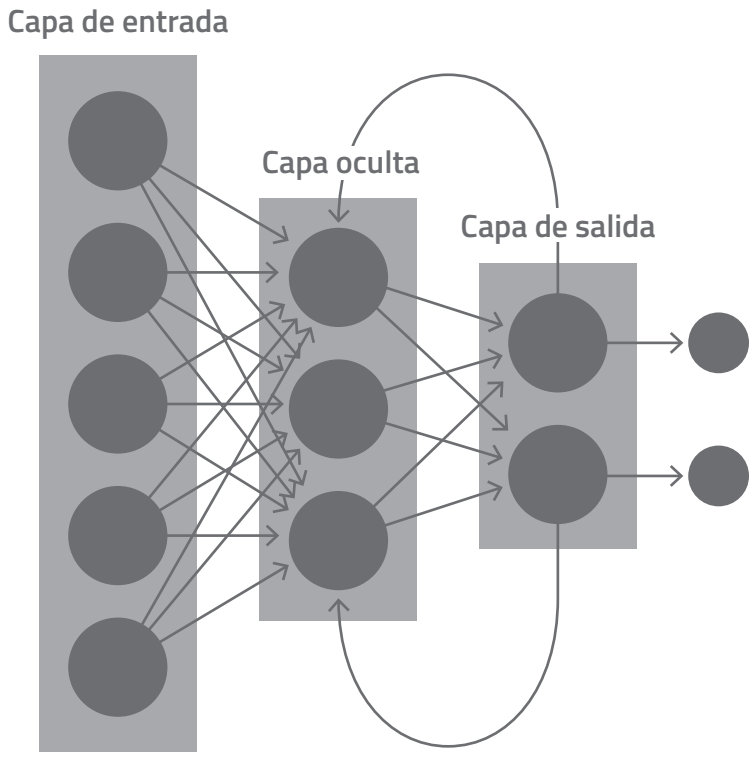


FIGURA 4.3: Una red neuronal recurrente simple

a los nodos en la primera capa oculta. Allí, se combinan con los valores de entrada de la siguiente muestra en la secuencia. Este proceso se repite hasta que todas las muestras hayan sido procesadas.

Una forma más técnica de describir esto es decir que las redes recurrentes rastrean los *estados ocultos* de las muestras en las que un estado comprende todos los valores actuales dentro de una capa oculta. Como se muestra en la Figura 4.4, las redes recurrentes alimentan el estado del paso temporal anterior como entrada al estado oculto del paso temporal actual. Esto permite que la red neuronal rastree cómo están cambiando los estados ocultos de un paso a otro.

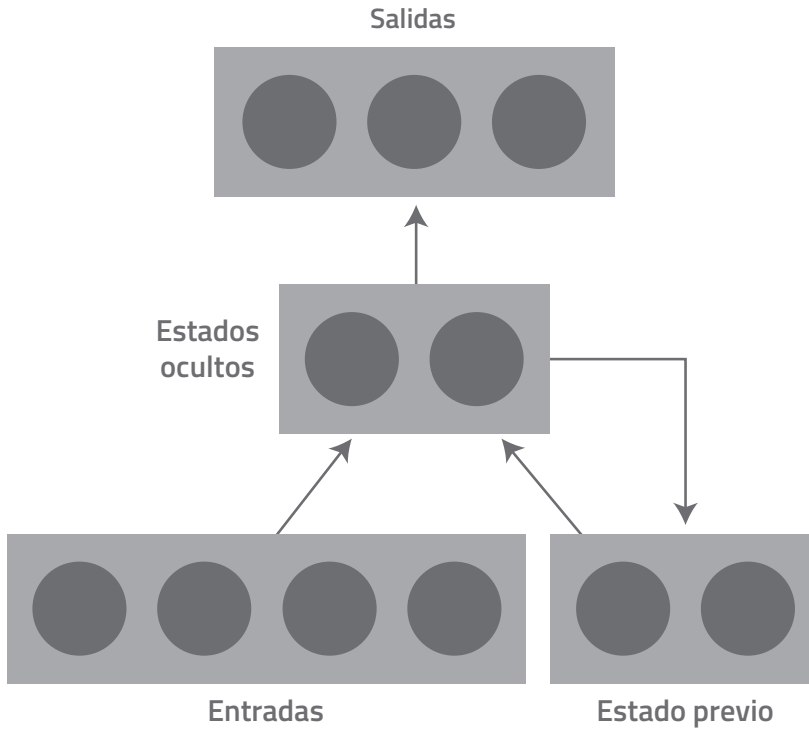


FIGURA 4.4: Seguimiento de estados ocultos

Los tipos más simples de RNR son limitados con respecto a la cantidad de pasos temporales en una secuencia que pueden recordar. En otras palabras, las RNR simples tienen problemas para mantener el *contexto*. Si, por ejemplo, una RNR debe poder recordar los paquetes primero a décimo en una secuencia para clasificar correctamente una conexión como sospechosa, entonces fallará si solo es capaz de mantener el contexto para los paquetes del 1 al 9. Esta limitación se ha abordado modificando la arquitectura de las RNR para incluir *unidades de Memoria a largo y corto plazo (MLCP)*, que dan nombre a esta arquitectura de red neuronal.

En una red MLCP, los nodos en cada capa oculta son reemplazados por *bloques de memoria*, cada uno de los cuales puede contener una o más *celdas de memoria* como la que se muestra en la Figura 4.5. A diferencia de los nodos de capas ocultas, los bloques de memoria no están equipados con funciones de activación. En su lugar, utilizan *puertas*, que determinan cómo y cuándo los estados almacenados en cada celda deberían actualizarse o transmitirse a los bloques de memoria en la capa

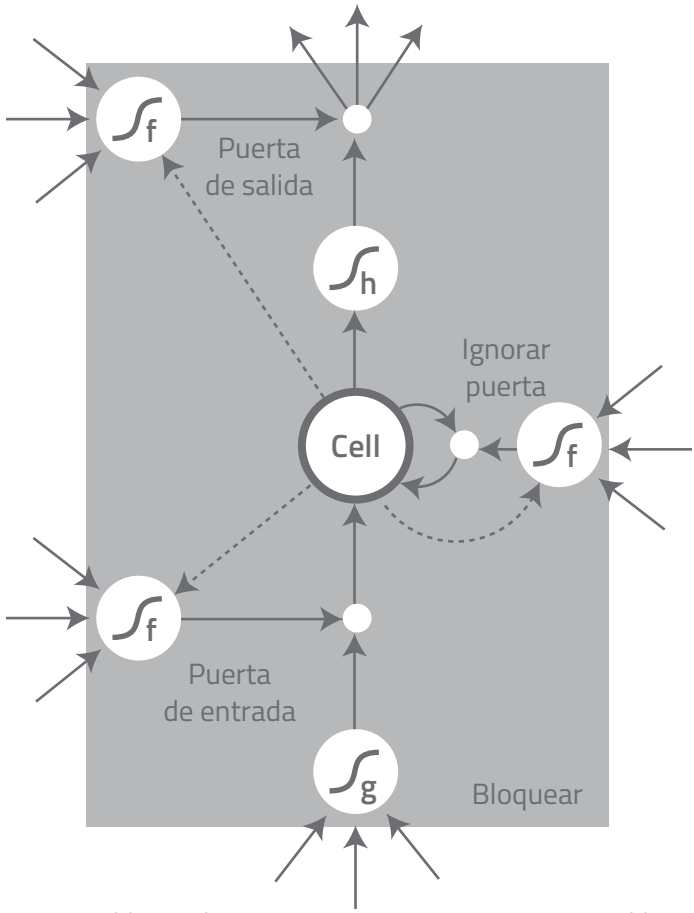


FIGURA 4.5: Un bloque de memoria MLCP que contiene una celda

oculta subsiguiente. Hay tres tipos de puertas, cada una equipada con su propio ajuste de ponderación:

- La *puerta de olvido* multiplica de forma recurrente el estado actual de cada celda por una ponderación igual a 1. Esto tiene el efecto de mantener el estado actual de la celda indefinidamente hasta que esté expuesto a los valores de estado que se originan a partir de muestras nuevas o de bucles de retroalimentación. Si los valores de ponderación de la puerta de olvido son altos, permitirá que ingrese nueva información de estado en la celda, sobrescribiéndola parcial o completamente. Si se establecen bajos, los valores de celda se mantendrán casi sin cambios.
- La *puerta de entrada* se abre para admitir nuevos valores de estado o se cierra para preservar los valores actuales.
- La *puerta de salida* se abre para permitir que los valores de estado pasen a los bloques de memoria en la siguiente capa oculta o se cierra para evitar esto.

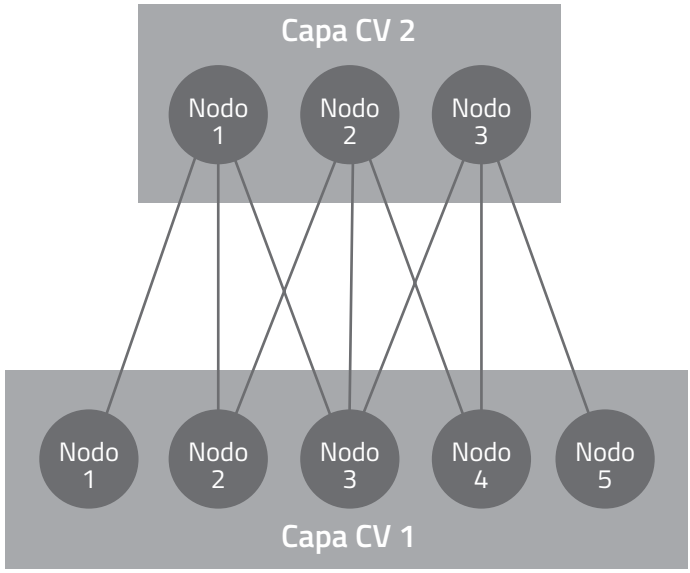
Las puertas permiten a las redes de MLCP retener y reutilizar información relevante distribuida en secuencias muy largas de datos de series temporales. Este proceso es similar a la forma en que aprenden las personas. En el cerebro, las "puertas" determinan si se debe permitir que la información pase del "bloque de memoria" de una neurona al siguiente. Solo aquellos que pasan información significativa pueden forjar conexiones sólidas y transmitir sus valores de estado a bloques de memoria en capas ocultas subsiguientes.

La red neuronal convolucional (RNC)

A diferencia de las redes neuronales que examinamos anteriormente, las RNC *no* utilizan una arquitectura completamente conectada. En cambio, cada capa de convolución conecta sus

nodos solo a conjuntos contiguos de nodos en la capa anterior. En la Figura 4.6, por ejemplo, el Nodo 1 en la Capa 2 está conectado exclusivamente a los Nodos 1-3 en la Capa 1. Desde un punto de vista técnico, diríamos que el Nodo 1 es una *función* de estos tres nodos. Lo mismo sucede con los Nodos 2 y 3 en esa capa. Cada uno de estos es una *función* de los tres nodos contiguos a los que se conectan en la Capa 1.

Para definir estas conexiones, utilizamos los hiperparámetros de RNC *tamaño* e *intervalo*. El *tamaño* determina la cantidad de nodos contiguos conectados de esta manera, mientras que el *intervalo* determina cuántos nodos se saltan. En la Figura 4.6, por ejemplo, el *tamaño* se ha establecido en 3 y el *intervalo* se ha establecido en 1. En consecuencia, ninguno de los nodos de Capa 1 ha sido salteado.

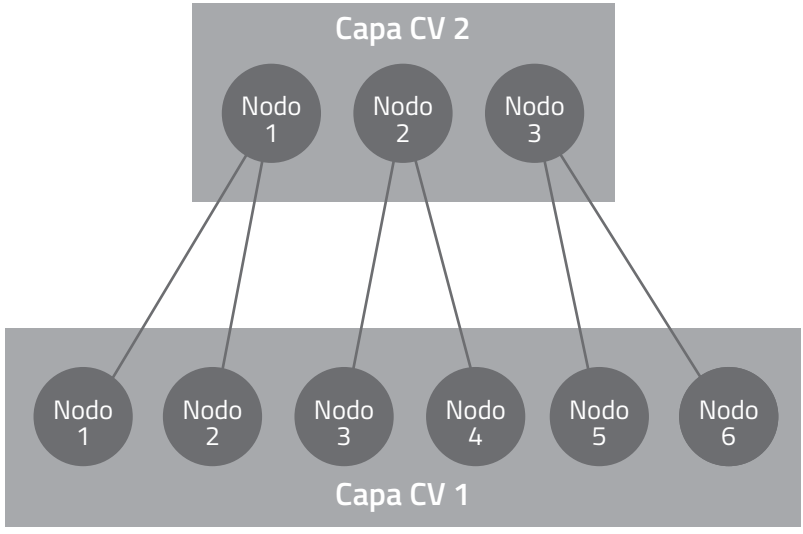


Cada nodo en la Capa 1 conectado
a tres nodos contiguos en la Capa 2

FIGURA 4.6: Tamaño=3, Intervalo=1

En la Figura 4.7, por el contrario, tanto el *tamaño* como el *intervalo* se han establecido en 2. En consecuencia, cada nodo en la Capa 2 es una *función* de solo dos nodos en la Capa 1. Además, la configuración del *intervalo* ha causado que el Nodo 2 en la Capa 2 *saltee* los primeros dos nodos en la Capa 1 y se conecte a los Nodos 3-4 en su lugar. A su vez, el Nodo 3 se conecta solo a los Nodos 5-6.

Anteriormente, describimos cómo cada nodo en una capa oculta está equipado con un conjunto único de ponderaciones, que se ajustan gradualmente hasta que los valores de ponderación globales en todas las capas producen una decisión de clasificación correcta. Con las RNC, podemos "vincular" estos valores de ponderación para cada conjunto de nodos conectados.



Cada nodo en la Capa 1 conectado solo a dos nodos contiguos en la Capa 2

FIGURA 4.7: Tamaño=2, Intervalo=2

La *vinculación de ponderación* hace que todos estos nodos apliquen los mismos valores de ponderación al realizar sus

cálculos. Esto reduce drásticamente el número de parámetros que debe calcular un modelo RNC. La RNC que se muestra en la Figura 4.8, por ejemplo, emplea solo *dos* ponderaciones (W_1 y W_2). Con una arquitectura completamente conectada equipada con el mismo número de nodos, la red tendría que calcular *18* ponderaciones (6 ponderaciones en la Capa 1 * 3 ponderaciones en la Capa 2). Un conjunto de ponderaciones vinculadas como W_1 y W_2 se conoce como *filtro*. Como veremos a continuación, los filtros son sumamente útiles. Consideremos ahora cómo fluye la información de una capa de RNC a la siguiente y el papel que desempeña cada capa al producir una decisión de clasificación. Exploraremos el proceso desde el punto de vista de la clasificación de imágenes, en el que los datos de entrada consisten en una

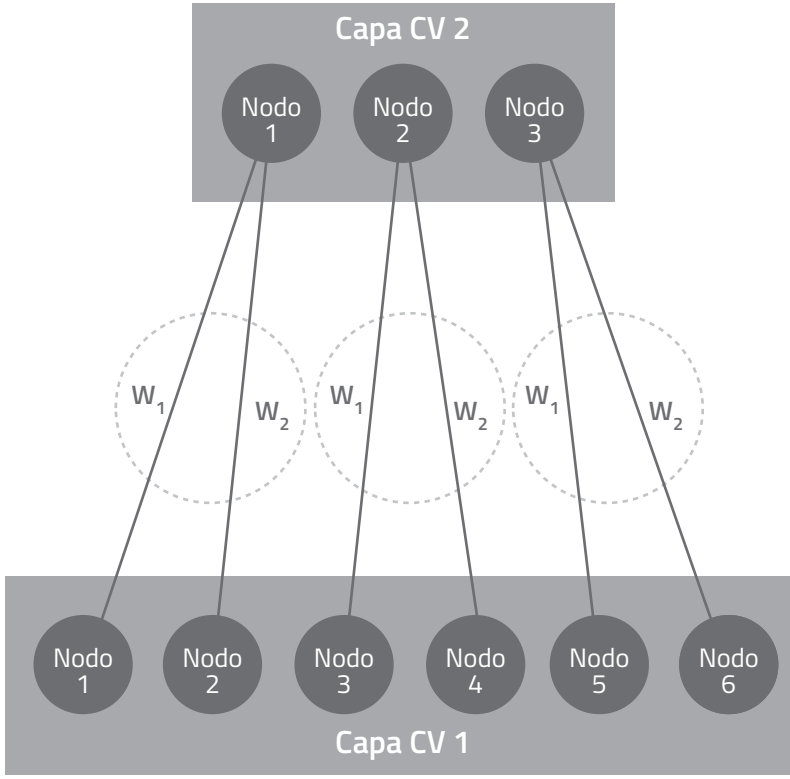


FIGURA 4.8: Ponderaciones vinculadas a través de nodos conectados

matriz bidimensional de valores de píxeles, donde 0 indica un píxel blanco y 1 indica uno negro. Sin embargo, el proceso se aplica casi de la misma manera al procesamiento del lenguaje natural y otros problemas de datos unidimensionales en los que la entrada consiste en una secuencia de caracteres binarios como los de nuestro problema de detección de la longitud de la clave de cifrado XOR.

LA CAPA DE CONVOLUCIÓN

Anteriormente describimos cómo los filtros y sus ajustes de *tamaño* e *intervalo* determinan qué nodos en las capas de convolución adyacentes pueden formar conexiones y cuáles no. Los filtros también desempeñan un papel central en identificar características y añadirlas a representaciones cada vez más abstractas que finalmente conducen a una decisión de clasificación.

En la Figura 4.9, vemos una matriz de valores de las características que corresponden a una imagen en blanco y negro de 5 píxeles por 5 píxeles. Al lado de la imagen hay un filtro con un *campo receptivo* de 3 x 3 y un conjunto de ponderaciones iniciales. Comenzamos "deslizando" el filtro sobre el primer conjunto de píxeles y luego multiplicamos los valores de píxel por las ponderaciones de filtro correspondientes.

A continuación, sumamos los productos de esta *multiplicación de elementos* e insertamos el resultado en la primera celda del *mapa de características asociado*. Una vez que se completa el mapa de características, los valores de *activación* resultantes se enviarán a los nodos conectados en la siguiente capa oculta para la etapa de procesamiento que sigue. El valor de activación de la primera convolución se muestra en la Figura 4.10.

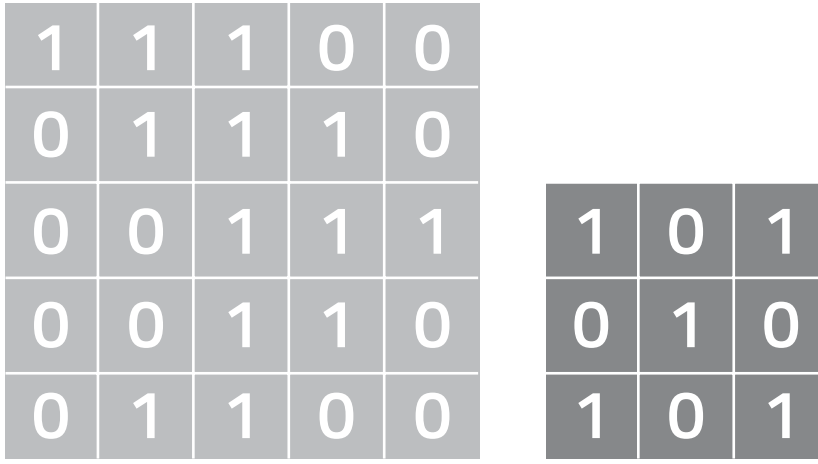


FIGURA 4.9: Imagen y filtro con ponderaciones a aplicar

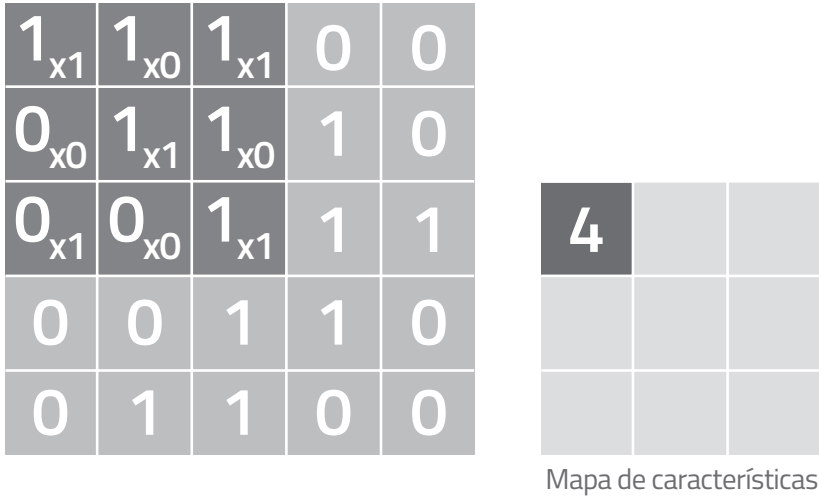


FIGURA 4.10: Convolucionar los primeros nueve pixeles

Nuestra RNC se ha configurado con un *intervalo* de 1. Por lo tanto, realizaremos nuestra próxima convolución deslizando el filtro un píxel a la derecha como se muestra en la Figura 4.11. Una vez más, registramos la activación en nuestro mapa de características.

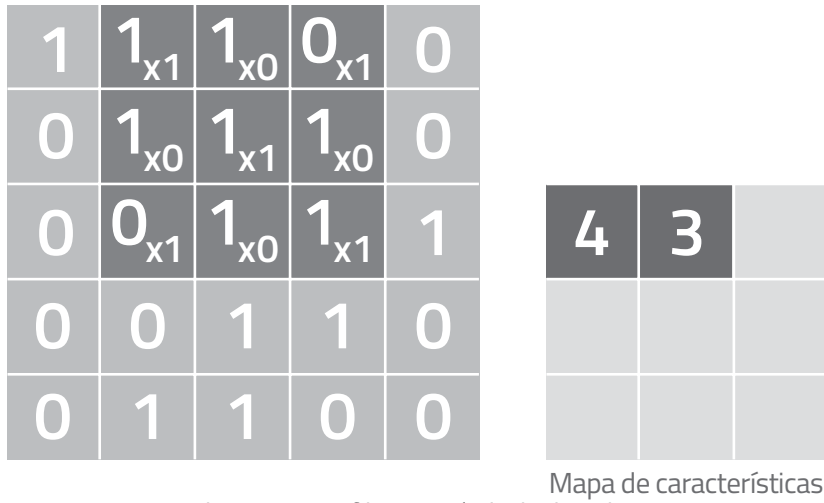


FIGURA 4.11: Deslizar nuestro filtro un píxel a la derecha

Continuamos de esta manera, recorriendo la imagen hasta que, como se muestra en la Figura 4.12, nuestro mapa de características esté completo. Observe cómo el mapa de características representa nuestra imagen de forma más abstracta y con menos parámetros para procesar capas subsiguientes.

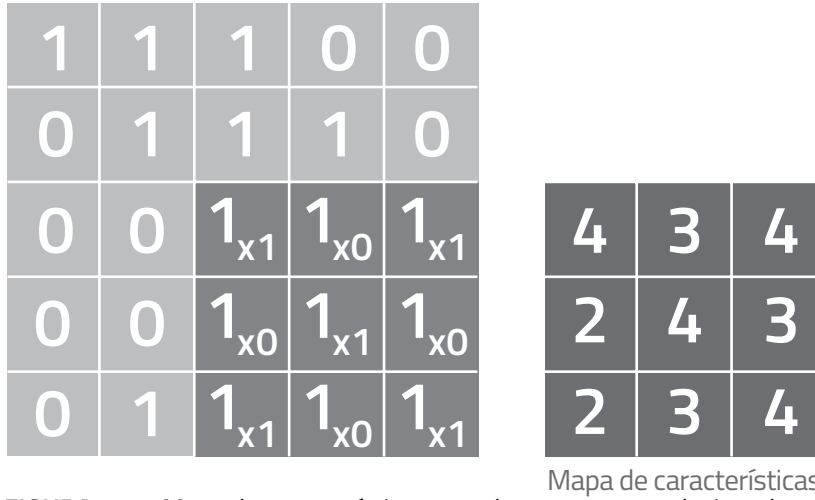


FIGURA 4.12: Mapa de características completamente convolucionadas

Es costumbre emplear cientos o incluso miles de filtros diferentes en cada capa de convolución, donde cada filtro detecta diferentes características y combinaciones de características. Por ejemplo, un conjunto de filtros puede detectar bordes, mientras que otros pueden detectar líneas curvas de varias orientaciones.

Una vez que se han calculado todas las activaciones, los resultados se envían a los nodos conectados en la siguiente capa para su procesamiento.

LA CAPA DE NORMALIZACIÓN POR LOTES

Ya en la década de 1990, los analistas sabían que las redes neuronales son más fáciles de optimizar cuando las entradas al modelo se normalizan: cada entrada al modelo tiene un valor promedio de cero y una desviación estándar de uno. Muchas estrategias para establecer ponderaciones iniciales aleatorias se basan en la suposición de que las activaciones de la capa anterior también se han normalizado de esta manera. En la práctica, sin embargo, esto no ocurre a menudo.

Para ayudar a acelerar la tasa de optimización, podemos colocar una capa de *normalización por lotes* después de cada capa convolucional. La normalización por lotes hace que las activaciones ocultas adquieran valores que están próximos a normalizarse sin imponer en realidad un valor promedio de cero y una desviación estándar de uno. En la práctica, esto puede reducir sustancialmente el tiempo de entrenamiento por hastadiez o más.

LA CAPA DE UNIDADES LINEALES RECTIFICADAS (ULR)

Para simplificar, nuestras ponderaciones de filtro y mapas de características resultantes emplearon números positivos. Sin embargo, sabemos por nuestra análisis de la regresión logística que las ponderaciones a menudo pueden tomar valores negativos.

Con una red RNC, el procesamiento adicional requerido para calcular estos parámetros negativos tal vez no produzca una mejora significativa en la precisión o el rendimiento del modelo. Para abordar esto, podemos incorporar una capa de ULR después de cada capa de convolución, que convierte estos valores negativos en ceros como se muestra en la Figura 4.13.

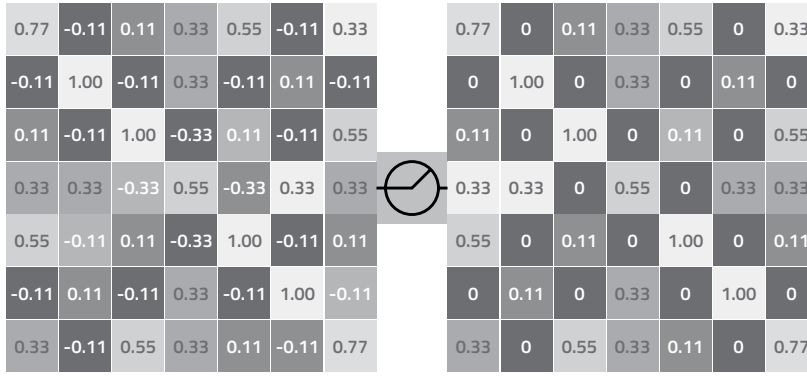


FIGURA 4.13: La capa ULR convierte los valores negativos en ceros

CAPAS DE AGRUPACIÓN

Cuantos más filtros empleemos, más activaciones tendremos que procesar en las capas subsiguientes. Las capas de agrupación nos permiten reducir este procesamiento, junto con la posibilidad de un sobreajuste, al permitir que solo entren ciertas activaciones y luego pasen a través de la capa de agrupamiento a las capas subsiguientes. Este proceso también se conoce como *submuestreo* de entradas. Una de las formas más populares de implementar la agrupación es con la función de activación de *agrupación máxima* que se desarrolla de la siguiente manera.

Primero, se coloca un filtro sobre el mapa de características entrantes y se evalúan las activaciones dentro de su campo receptivo. Normalmente, se usa un filtro configurado con un campo receptivo de 2x2 y un *intervalo* de 2. Si la *agrupación*

FIGURA 4.14: Agrupación máxima aplicada al mapa de características entrantes

máxima está configurada para la operación *1-máx.* solo la activación de mayor magnitud dentro de ese campo receptivo se copiará en un nuevo mapa de características más pequeño. El filtro luego se desliza a su siguiente ubicación (dos celdas a la derecha en este caso), y el proceso se repite hasta que se complete el nuevo mapa de características. Este proceso se ilustra en la Figura 4.14. Al agrupar de esta manera, hemos reducido el número de ponderaciones que se calcularán de 16 a 4.

A menudo se incorpora una capa de *agrupación máxima global* final para submuestrear todas las activaciones entrantes antes de pasarlas a los nodos en la capa de salida. Por ejemplo, si tuviéramos que aplicar una operación de agrupación máxima global al mapa de características original que se muestra en la Figura 4.14, la salida sería un valor único, 8.

CAPA DE SALIDA

La capa de salida está completamente conectada a todos los nodos en la capa anterior. Cada nodo de salida recibe como entrada un valor de activación global que determina qué clasificación debe

asignarse. También se puede aplicar una función de activación softmax para asignar una puntuación de probabilidad a esta asignación de clase.

Una sesión de clasificación de aprendizaje profundo típica

Consideremos ahora cómo un analista podría navegar cada etapa de una sesión de clasificación de aprendizaje profundo. Algunos de estos pasos serán familiares para los lectores de los capítulos anteriores.

ADQUISICIÓN Y VECTORIZACIÓN DE MUESTRAS

Paso 1: Selección de las muestras

Una vez más, comenzamos a partir de conjunto representativo de datos de muestras. Como se trata de una sesión de aprendizaje supervisado, el conjunto de datos incluirá etiquetas que definen la pertenencia a la clase de cada muestra.

Paso 2: Subdivisión de las muestras en subconjuntos

A continuación, separamos entre el 70 al 90 % de nuestro conjunto de datos total para su uso en el entrenamiento de nuestro modelo. Dependiendo de la aplicación, podemos subdividir las muestras restantes en conjuntos separados para validación y prueba. Ninguno de estos conjuntos estará expuesto al modelo hasta que se complete el ciclo de entrenamiento.

Los conjuntos de validación y entrenamiento tienen propósitos ligeramente diferentes. El conjunto de validación se usa generalmente para comparar la precisión y el rendimiento de varios modelos diferentes a fin de determinar cuál de ellos debe ponerse en producción o someterse a prueba. El paso de validación también es útil para ayudarnos a detectar si se está aplicando un sobreajuste. Esto se hace evidente si la precisión

del modelo con los datos de entrenamiento supera ampliamente la precisión que obtiene cuando se expone al conjunto de validación.

Sin embargo, el proceso de selección del modelo aún puede estar sujeto a una tendencia si los conjuntos de entrenamiento y validación no reflejan con precisión la información real del entorno de datos subyacente. Para abordar esto, podemos exponer nuestro modelo elegido a un conjunto de datos de prueba y aplicar varios métodos para evaluar su precisión. Si encontramos anomalías en el rendimiento y la precisión entre los resultados de las fases de validación y prueba, podemos concluir que el modelo tiene un bajo rendimiento y debe ser entrenado nuevamente.

Por otro lado, vale la pena señalar que nuestra elección sobre qué datos incluir en el conjunto de pruebas no tiene que ser arbitraria. Por ejemplo, es posible que deseemos probar el modelo exponiéndolo a un conjunto de pruebas que contiene muestras mucho más difíciles de clasificar que aquellas en los conjuntos de entrenamiento y validación.

Paso 3: Extracción de características y vectorización

La extracción de características y la ingeniería de características se encuentran entre los aspectos más complejos, difíciles y que requieren más tiempo del proceso de Aprendizaje Automático. Si las características están mal definidas o si se excluyen los atributos clave, los modelos resultantes no reflejarán con exactitud la información real y fallarán cuando se expongan a datos del mundo real. Afortunadamente, las redes neuronales tienen la capacidad de automatizar el proceso de extracción de características y aceptar datos de entrenamiento similares a su representación en bruto. Por ejemplo, si nuestro objetivo es aprender secuencias de datos binarios, tal vez no necesitemos

extraer características sobre estas secuencias. En cambio, podemos enviar los datos binarios en bruto directamente a la red neuronal y permitir que la red determine qué secuencias son significativas. En cualquier caso, primero debemos reestructurar nuestras muestras en matrices separadas de vectores y etiquetas para cargar nuestras muestras de entrenamiento en la capa de entrada de la red neuronal.

SELECCIÓN DE MODELO E INSTANCIACIÓN

Con nuestras muestras cargadas, estamos listos para comenzar a seleccionar y crear instancias de un modelo. Determinaremos el tipo de modelo que elegimos y su arquitectura según el tipo de datos que estamos analizando y la naturaleza del problema. Por ejemplo, si trabajamos con datos de imagen, una arquitectura convolucional sería una elección adecuada. Si estamos trabajando con datos secuenciales, como texto, bytes, datos de captura de red, etc., entonces sería conveniente una red neuronal recurrente. Si ya hemos extraído características y simplemente queremos comparar el rendimiento de diferentes tipos de modelos, podríamos emplear una red completamente conectada. Por ahora, asumamos que hemos elegido una arquitectura apropiada y que estamos listos para comenzar el entrenamiento del modelo.

ENTRENAMIENTO DEL MODELO

En esta fase, ejecutamos datos de entrenamiento en el modelo para optimizar incrementalmente las ponderaciones en cada capa oculta hasta que el modelo logre un nivel aceptable de precisión y rendimiento.

El entrenamiento puede comenzar no bien se haya definido la arquitectura del modelo y se haya especificado la ruta al directorio que contiene el conjunto de datos. Suponiendo que

estamos dispuestos a comenzar con el conjunto predeterminado de hiperparámetros, podemos simplemente ejecutar una función como esta:

```
model.fit(train_x,train_y, nb_epochs=10)
```

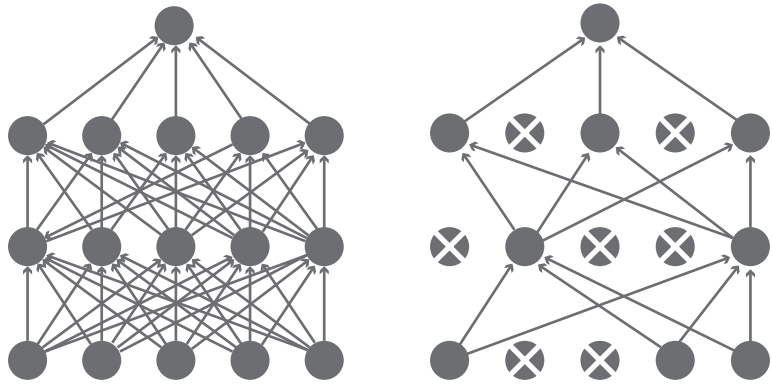
Aquí, *train_x* representa la matriz de datos de entrenamiento, *train_y* representa el vector de etiquetas de entrenamiento, y *nb_epochs* determina cuántas *épocas* deben completarse antes de detenerse y evaluar cómo han cambiado la precisión y el rendimiento del modelo. El término *época* se refiere a un ciclo de cálculo en el cual el modelo está expuesto a todo el conjunto de entrenamiento.

Puede tomar bastante tiempo para que el modelo procese todo el conjunto de entrenamiento. A menudo es más eficiente entrenar con subconjuntos o *lotes* más pequeños. Dado que cada lote lleva menos tiempo en procesarse, podemos descubrir antes cómo se desarrolla nuestro modelo al exponer cada resultado del lote a un segmento del conjunto de validación. Si concluimos que el modelo no está funcionando como se esperaba, podemos modificar los hiperparámetros y reiniciar el proceso de entrenamiento. Trataremos el enfoque por lotes más adelante en este capítulo cuando mostremos cómo clasificar la longitud de una clave de cifrado XOR durante las sesiones de aprendizaje profundo de MLCP y RNC.

Las redes neuronales tienen un número interminable de hiperparámetros que influyen en cómo se crean los modelos. Los más comunes incluyen:

- **Cantidad de capas ocultas.** Podemos aumentar este número si las tasas de error son excesivas o disminuirlo si sospechamos un sobreajuste.

- **Cantidad de nodos por capa oculta.** Por lo general, mantenemos la configuración predeterminada, que especifica un número igual de nodos para cada capa oculta. Sin embargo, podemos experimentar asignando diferentes cantidades de nodos si parece que esto podría ser útil.
- **Tasa de aprendizaje.** Esto determina el tamaño de cada ajuste incremental aplicado a las ponderaciones durante la optimización, es decir, qué tan rápido el modelo se mueve hacia la convergencia en el *espacio de ponderación*. Si elegimos una tasa de aprendizaje demasiado alta, los valores de ponderación rebotan alrededor del espacio de ponderación y la convergencia puede no ocurrir nunca. Si establecemos la tasa de aprendizaje demasiado baja, el optimizador modificará las ponderaciones en pasos tan pequeños que el "tiempo de convergencia" se extiende innecesariamente. Es posible que necesitemos experimentar con la tasa de aprendizaje para encontrar un valor que sea "correcto" si esperamos entrenar a nuestro modelo dentro de un marco de tiempo razonable.
- **Regularización por exclusión (dropout).** Debido a su complejidad y capacidad, las redes neuronales pueden sobreajustar los datos de entrenamiento al establecer cada ponderación en todas las capas ocultas a un valor específico para cada muestra de entrada. Esto es equivalente a un árbol de decisión que crea un nodo para cada muestra en el conjunto de entrenamiento. Para evitar esto, podemos aplicar el hiperparámetro de regularización por exclusión, que hace que el modelo ignore la salida de un porcentaje de los nodos dentro de cada capa oculta durante cada época. Podemos ver el



(a) Red neuronal estándar

(b) Después de aplicar la deserción

FIGURA 4.15: Antes y después de la aplicación de regularización por exclusión

efecto de la aplicación de la regularización por exclusión en la Figura 4.15.

Este hiperparámetro se expresa como un valor de probabilidad que indica la probabilidad de que cada nodo oculto se deshabilite durante el procesamiento de una muestra. Por ejemplo, una configuración de exclusión de 0,2 indica que el 20 % de los nodos ocultos se desactivarán aleatoriamente. Una configuración de exclusión baja como esta dará lugar a un modelo muy *expresivo* que capta cada matiz del conjunto de entrenamiento, pero puede ser propenso a un sobreajuste. Una configuración alta, como 0,8, puede causar tantas exclusiones que la precisión del modelo podría verse comprometida. En la práctica, generalmente comenzamos con un valor predeterminado de 0,5 y luego experimentamos variando la configuración más alta y más baja para evaluar los resultados.

Es habitual aplicar diferentes configuraciones aleatorias a estos y otros hiperparámetros durante el entrenamiento con

el fin de generar varias decenas de modelos diferentes para la evaluación durante las fases de validación y prueba subisguientes.

MODELO DE VALIDACIÓN

Como se señaló anteriormente, podemos alternar los procesos de entrenamiento y validación de lote a lote o esperar a validar un conjunto de modelos una vez completado el entrenamiento. Se puede emplear una amplia variedad de métodos y funciones de validación, incluidas las métricas de *precisión* y *recuperación* que describimos en el capítulo de clasificación. De todos modos, nuestro objetivo es asegurarnos de que no hayamos escogido accidentalmente un modelo que sobreajuste los datos de entrenamiento. Para evitar esto, exponemos nuestros modelos entrenados al conjunto de datos de validación y luego elegimos el que ofrece la mejor combinación de rendimiento y precisión para las pruebas.

PRUEBAS E IMPLEMENTACIÓN DE MODELOS

Una vez que elegimos nuestro "mejor" modelo, estamos listos para someterlo a los datos de prueba y medir los resultados. Como se mencionó anteriormente, no es raro poblar datos de prueba con ejemplos particularmente difíciles para evaluar qué tan bien funcionaría el modelo en el mundo real. Si nuestro objetivo es determinar la longitud de la clave de una secuencia de bytes ofuscada por XOR, por ejemplo, nuestro conjunto de pruebas podría incluir muestras que son particularmente difíciles de desencubrir. Alternativamente, podríamos exponer a un modelo entrenado en caracteres ASCII en inglés ofuscados por XOR a datos de prueba codificados en base64 ofuscados por XOR para evaluar qué tan bien se generaliza en la clasificación de otros tipos de secuencias de bytes.

Ejercicios de aprendizaje prácticos

Vamos a aplicar lo que hemos aprendido para ver cómo las redes de MLCP y RNC pueden resolver un problema de clasificación mediante el análisis de secuencias de datos binarios. En este caso, nuestro objetivo es determinar la longitud de bit de la clave de cifrado XOR utilizada para ofuscar una muestra de texto.

Si bien el conjunto de herramientas *scikit-learn* proporciona algunas implementaciones simples de redes neuronales, utilizaremos las versiones más robustas proporcionadas por la biblioteca *Theano* de Python. Puede descargar los scripts y los datos que usaremos aquí:

https://www.cylance.com/intro-to-ai

Como se señaló anteriormente, se pueden aplicar numerosas configuraciones de hiperparámetros para influir en cómo se entrenan y optimizan los modelos de redes neuronales. Para simplificar, nuestros scripts han sido codificados con configuraciones que sabemos que son útiles. Sin embargo, le recomendamos encarecidamente que modifique estos scripts o cree otros nuevos. Esto le permitirá apreciar plenamente cómo las configuraciones y arquitecturas de hiperparámetros de las redes neuronales influyen en el proceso de clasificación y la precisión y el rendimiento resultantes de los modelos de redes neuronales resultantes.

ENCRIPTAR CON XOR

XOR, u "o" exclusivo, es una operación lógica para cifrar y descifrar datos binarios utilizando claves de longitud variable. El nombre refleja el método utilizado para modificar los valores de bytes. Cada bit en la clave se compara con un bit correspondiente en el conjunto de muestras. A continuación, XOR devuelve un valor

de 1 (verdadero) si los valores de bit son diferentes y 0 (falso) si coinciden. El proceso puede invertirse fácilmente aplicando la misma clave para devolver los bits modificados a sus valores originales. Si bien XOR no es uno de los métodos de cifrado más seguros disponibles, a menudo se utiliza como parte de un esquema de cifrado debido a su fácil implementación y su capacidad para modificar datos rápidamente. Por ejemplo, la misma clave de cifrado XOR puede aplicarse repetidamente en todo un conjunto de datos. Si nuestra clave es una *contraseña* y nuestros datos son *Cifrar estos datos*, XOR aplicará la clave de 8 bits a cada byte en nuestros datos, como se muestra en la siguiente table . (Dado que estamos utilizando datos binarios ASCII, solo se encriptarán siete de los ocho bits. El octavo bit será ignorado).

XOR aplicado a encriptar caracteres binarios ASCII

Clave	Texto sin formato	Encriptado
p	E	0x35
a	n	0x0f
s	c	0x10
s	r	0x01
w	y	0x0e
o	p	0x1f
r	t	0x06
d		0x44
p	t	0x04
a	h	0x09
s	i	0x1a
s	s	0x00
w		0x57
o	d	0x0b
r	a	0x13
d	t	0x10
p	a	0x11

El proceso sería el mismo si estuviéramos utilizando una longitud de clave de dos bytes (16 bits) con un conjunto de datos que consta de 160 caracteres de 8 bits cada uno. En ese caso, la clave se aplicaría secuencialmente diez veces. Por ejemplo, el primer bit de la clave se aplicaría al primer bit del primer byte, al primer bit del tercer byte, etc.

Si conocemos la longitud de la clave de cifrado XOR, podemos intentar adivinar los caracteres originales utilizando una técnica llamada *análisis de frecuencia*. Normalmente, este enfoque solo funcionaría bien con las claves de un solo byte. Sin embargo, si conocemos la longitud de la clave, podemos aplicar la misma técnica a las secuencias de texto cifrado. Una descripción detallada del análisis de frecuencia y su papel en el cifrado y descifrado está fuera del alcance de este capítulo. Aquí, nos enfocaremos exclusivamente en demostrar cómo se puede determinar la longitud de la clave de cifrado XOR usando los algoritmos MLCP y RNC.

GENERACIÓN DE UN CONJUNTO DE DATOS

Como de costumbre, debemos adquirir un conjunto de datos representativo para trabajar. En este caso, nuestro conjunto de datos consistirá en una secuencia de bytes que representan caracteres ASCII en inglés. En consecuencia, cada vector incluirá ocho dimensiones, una para cada uno de los ocho valores de bit posibles (aunque el octavo bit se ignorará como se indicó anteriormente). Dado su formato binario, cada característica puede contener solo uno de dos valores: *0* y *1*.

Para generar nuestro conjunto de datos, comenzaremos por descargar una sección aleatoria de texto sin formato del conjunto de datos *enwik8*. Se puede acceder a esta información, junto con la documentación, en el siguiente enlace:

https://cs.fit.edu/~mmahoney/compression/textdata.html

Reservaremos aproximadamente el 70 % de estos datos de texto sin formato para nuestro conjunto de entrenamiento. Los datos restantes se usarán para la validación durante el entrenamiento. También crearemos algunos datos de validación adicionales para usar cuando estemos listos para probar nuestros modelos. Para crear estos datos, usaremos el script de Python *generate_xor.py*. Este script leerá el texto sin formato de enwik8 y luego lo encriptará con una clave aleatoria XOR de una longitud especificada. Especificaremos una longitud de ocho bits. Más tarde, probaremos nuestros modelos con estos datos cifrados para ver qué tan bien pueden predecir correctamente la longitud de nuestra clave de 8 bits.

Como se muestra, el script devuelve una clave aleatoria de 8 bits junto con la versión encriptada de nuestros datos de texto sin formato restantes. Tanto la clave como los datos se han escrito en la ruta de archivo especificada.

Ahora estamos listos para comenzar a alimentar lotes del conjunto de entrenamiento de texto sin formato a nuestras dos redes neuronales. Comencemos con la red de MLCP.

APLICA UN MODELO CIÓN DE UN MODELO MLCP PARA IDENTIFICAR LA LONGITUD DE LA CLAVE DE CIFRADO XOR

Las redes neuronales recurrentes como MLCP son adecuadas para resolver problemas en los que las relaciones secuenciales entre muestras determinan sus asignaciones de clase. Instanciaremos nuestro modelo y comenzaremos el proceso de entrenamiento con el script python *train_model.py*. Por defecto, esta función crea un modelo MLCP (observe que el operador *conv* se establece en *falso*), junto con una configuración

predeterminada de ajustes de hiperparámetros. Estos se muestran en la captura de pantalla a continuación.

```
root@                                  # KERAS_BACKEND="theano" THEANO_FLAGS=device=gpu,floatX=float32 python train_model.py
Using Theano backend.
Using gpu device 0: GRID K520 (CNMeM is disabled, cuDNN not available)
{'output_dim': 256, 'output_activation': 'softmax', 'activation': 'relu', 'conv': False}
Loading data
Starting vectorization threads
Model Summary
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 64, 8)         0
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 64, 256)       271360      input_1[0][0]
____________________________________________________________________________________________________
lstm_2 (LSTM)                    (None, 64, 256)       525312      lstm_1[0][0]
____________________________________________________________________________________________________
globalmaxpooling1d_1 (GlobalMaxP (None, 256)           0           lstm_2[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 32)            8224        globalmaxpooling1d_1[0][0]
====================================================================================================
Total params: 804,896
Trainable params: 804,896
Non-trainable params: 0
____________________________________________________________________________________________________
```

Como se muestra en la Figura 4.16, nuestro modelo incluirá una capa de entrada MLCP (que contiene 256 nodos) y una capa MLCP oculta con el mismo número de nodos. La salida de esta

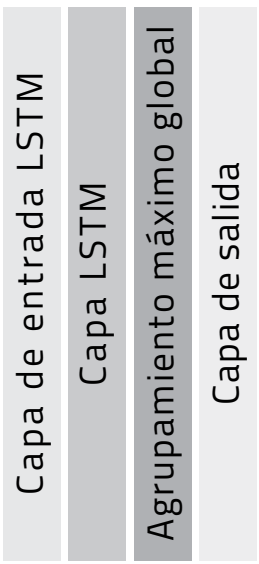


FIGURA 4.16: Nuestra arquitectura MLCP

capa oculta se pasará a una capa de agrupación máxima global seguida de una capa de salida. Allí, se aplicará una función de

activación *softmax* para clasificar cada muestra y predecir la longitud del bit de nuestra clave de cifrado XOR. Ahora estamos listos para comenzar el proceso de entrenamiento.

Entrenamiento y optimización del modelo MLCP

Entrenaremos nuestro modelo MLCP por lotes. Después de cada uno, evaluaremos la precisión de nuestro modelo exponiéndolo a nuestro conjunto de validación encriptado. Si las puntuaciones de precisión no mejoran en diez lotes de entrenamiento y validación, interpretaremos que esto significa que nuestra tasa de aprendizaje es demasiado alta y la reduciremos en consecuencia. Esto nos permitirá continuar ajustando nuestro modelo de forma indefinida o interrumpir el entrenamiento y crear un nuevo modelo con diferentes ajustes de configuración e hiperparámetros. Podemos ver este proceso de entrenamiento y validación en la captura de pantalla en la página siguiente.

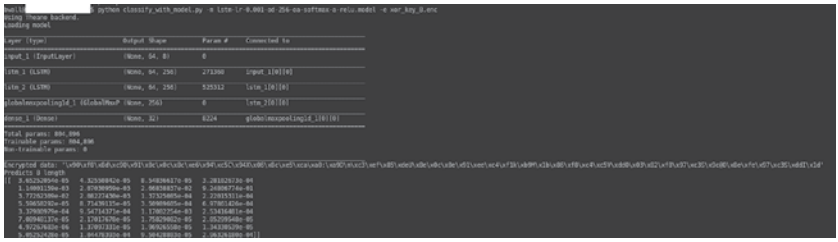
```
Total params: 804,896
Trainable params: 804,896
Non-trainable params: 0

_________________________________________________________________
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 3.4259 - acc: 0.0447
New best validation score: 0.118347167969 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 2.4053 - acc: 0.2164
New best validation score: 0.344116210938 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 1.9004 - acc: 0.3696
New best validation score: 0.543029785156 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 1.2417 - acc: 0.5870
New best validation score: 0.608764648438 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 1.1356 - acc: 0.6335
New best validation score: 0.621643066406 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.8702 - acc: 0.7270
New best validation score: 0.749877929688 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.7887 - acc: 0.7664
Validation score: 0.73828125
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 1.0136 - acc: 0.7015
Validation score: 0.54443359375
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.7927 - acc: 0.7715
New best validation score: 0.762512207031 (saving)
Epoch 1/1
 2144/16384 [==>...........................] - ETA: 73s - loss: 0.7446 - acc: 0.7873
16384/16384 [==============================] - 84s - loss: 0.6921 - acc: 0.8035
New best validation score: 0.83056640625 (saving)
```

El entrenamiento de una red neuronal puede ser un proceso bastante largo. En este punto, nuestro modelo ha logrado una puntuación de validación de aproximadamente 0,83. Si bien esta es una gran mejora con respecto a la puntuación inicial de aproximadamente 0,12, todavía no es lo suficientemente precisa para nuestras necesidades. Por ello, ejecutaremos lotes adicionales hasta lograr una puntuación de precisión de al menos 0,90.

```
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1474 - acc: 0.9625
Validation score: 0.970397949219
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1316 - acc: 0.9655
Validation score: 0.967346191406
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1333 - acc: 0.9656
Validation score: 0.970703125
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1386 - acc: 0.9644
Validation score: 0.968688964844
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1377 - acc: 0.9644
Validation score: 0.970031738281
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1223 - acc: 0.9679
New best validation score: 0.973571777344 (saving)
Epoch 1/1
16384/16384 [==============================] - 84s - loss: 0.1316 - acc: 0.9667
Validation score: 0.970520019531
```

Prueba del modelo MLCP

Después de muchos más lotes, nuestro modelo MLCP ha logrado una puntuación de validación superior a 0,97. Ahora estamos listos para ver qué tan bien es capaz de predecir la longitud de nuestra clave de cifrado XOR. Después de guardar el modelo en el disco, lo expondremos a nuestro conjunto de pruebas XOR utilizando el script *classify_with_model.py* con argumentos que incluyen la ruta a los datos de prueba y el nombre del modelo (*lstm-lr-0.001-od-256-oa-softmax-a-relu.model*).



Como podemos ver, el modelo MLCP predijo correctamente una longitud de clave de ocho bits.

APLICACIÓN DE UN MODELO RNC PARA IDENTIFICAR LA LONGITUD DE LA CLAVE DE CIFRADO XOR

Aunque carecen de las puertas que proporcionan a las RNR su memoria prodigiosa, las RNC también pueden resolver complejos problemas de clasificación en los que el orden de las muestras en el tiempo o su adyacencia en el espacio determina finalmente la decisión de clasificación. En consecuencia, las RNC se usan ampliamente con problemas que van desde la categorización de imágenes (mediante el análisis de vecindarios de píxeles adyacentes) hasta el procesamiento del lenguaje natural (mediante el análisis de vecindarios de palabras). Consideremos primero, a nivel conceptual, cómo se pueden aplicar estas capacidades a la detección de claves de cifrado XOR.

En nuestra capa convolucional inicial, cada nodo de entrada recibirá una serie de muestras que consisten en bytes codificados de 8 bits de caracteres ASCII. Allí, se aplicarán filtros para calcular las ponderaciones de los vecindarios de los caracteres, cuyo tamaño de vecindario quedará definido por el hiperparámetro de *tamaño* de la RNC. A continuación, pasaremos la salida a los nodos conectados en una segunda capa convolucional y aplicaremos funciones adicionales de filtrado y activación para interpretar las muestras en un nivel más abstracto. Después de

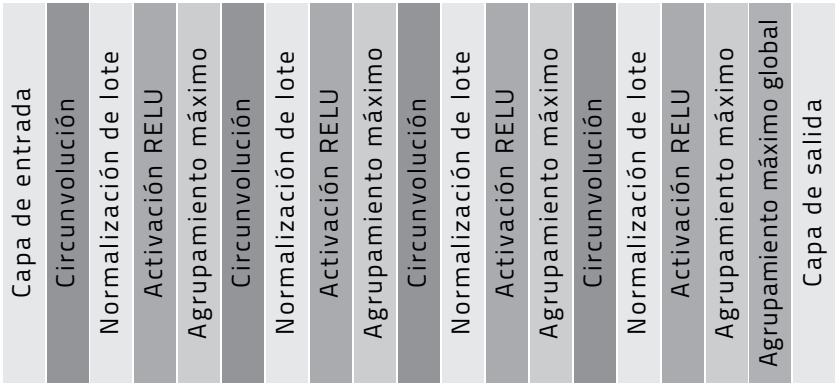


FIGURA 4.17: Nuestra arquitectura RNC

varias etapas más de procesamiento, una capa de agrupación máxima global seleccionará los valores máximos de nodo para los filtros en la capa convolucional final en toda la secuencia de entrada. Finalmente, las muestras se pasarán a través de una capa completamente conectada a la capa de salida, donde se asignará la clasificación de longitud de la clave.

La RNC que usaremos en nuestro ejemplo será un poco más compleja, incorporando cuatro capas de convolución, cada una equipada con 256 nodos. Cada una de estas capas de convolución será seguida por capas para la normalización de lotes, la activación de unidades lineales rectificadas (ULR) y la agrupación máxima. La salida resultante se enviará a una capa global de agrupación máxima y, finalmente, a la capa de salida, donde se aplicará una función de activación *softmax* para producir la predicción de clasificación. La arquitectura de nuestra RNC se muestra en la Figura 4.17: Nuestra arquitectura RNC.

ENTRENAMIENTO Y OPTIMIZACIÓN DEL MODELO RNC

Como antes, instanciaremos nuestro modelo y lo entrenaremos por lotes utilizando el script de Python *train_model.py*. Esta vez, sin embargo, hemos cambiado el operador *conv* a *verdadero*

para producir una RNC en lugar de la MLCP predeterminada. Podemos ver los detalles de configuración de nuestro modelo en la captura de pantalla a continuación.

```
root@                    # KERAS_BACKEND="theano" THEANO_FLAGS=device=gpu,floatX=float32 python train_model.py --conv
Using Theano backend.
Using gpu device 0: GRID K520 (CNMeM is disabled, cuDNN not available)
{'output_dim': 256, 'output_activation': 'softmax', 'activation': 'relu', 'conv': True}
Loading data
Starting vectorization threads
Model Summary
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 64, 8)	0	
convolution1d_1 (Convolution1D)	(None, 64, 256)	33024	input_1[0][0]
batchnormalization_1 (BatchNorma	(None, 64, 256)	1024	convolution1d_1[0][0]
activation_1 (Activation)	(None, 64, 256)	0	batchnormalization_1[0][0]
maxpooling1d_1 (MaxPooling1D)	(None, 32, 256)	0	activation_1[0][0]
convolution1d_2 (Convolution1D)	(None, 32, 256)	1048832	maxpooling1d_1[0][0]
batchnormalization_2 (BatchNorma	(None, 32, 256)	1024	convolution1d_2[0][0]
activation_2 (Activation)	(None, 32, 256)	0	batchnormalization_2[0][0]
maxpooling1d_2 (MaxPooling1D)	(None, 16, 256)	0	activation_2[0][0]
convolution1d_3 (Convolution1D)	(None, 16, 256)	1048832	maxpooling1d_2[0][0]
batchnormalization_3 (BatchNorma	(None, 16, 256)	1024	convolution1d_3[0][0]
activation_3 (Activation)	(None, 16, 256)	0	batchnormalization_3[0][0]
maxpooling1d_3 (MaxPooling1D)	(None, 8, 256)	0	activation_3[0][0]
convolution1d_4 (Convolution1D)	(None, 8, 256)	1048832	maxpooling1d_3[0][0]
batchnormalization_4 (BatchNorma	(None, 8, 256)	1024	convolution1d_4[0][0]
activation_4 (Activation)	(None, 8, 256)	0	batchnormalization_4[0][0]
maxpooling1d_4 (MaxPooling1D)	(None, 4, 256)	0	activation_4[0][0]
globalmaxpooling1d_1 (GlobalMaxP	(None, 256)	0	maxpooling1d_4[0][0]
dense_1 (Dense)	(None, 32)	8224	globalmaxpooling1d_1[0][0]

```
Total params: 3,191,840
Trainable params: 3,189,792
Non-trainable params: 2,048
```

Esta vez, el entrenamiento avanza mucho más rápido.

```
Trainable params: 3,189,792
Non-trainable params: 2,048

Epoch 1/1
16384/16384 [==============================] - 66s - loss: 1.6461 - acc: 0.5364
New best validation score: 0.630859375 (saving)
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.3217 - acc: 0.9221
Validation score: 0.442932128906
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.2563 - acc: 0.9385
New best validation score: 0.857849121094 (saving)
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.2221 - acc: 0.9479
New best validation score: 0.888427734375 (saving)
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.2140 - acc: 0.9478
Validation score: 0.886352539062
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.1919 - acc: 0.9542
New best validation score: 0.934265136719 (saving)
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.1789 - acc: 0.9564
Validation score: 0.893249511719
```

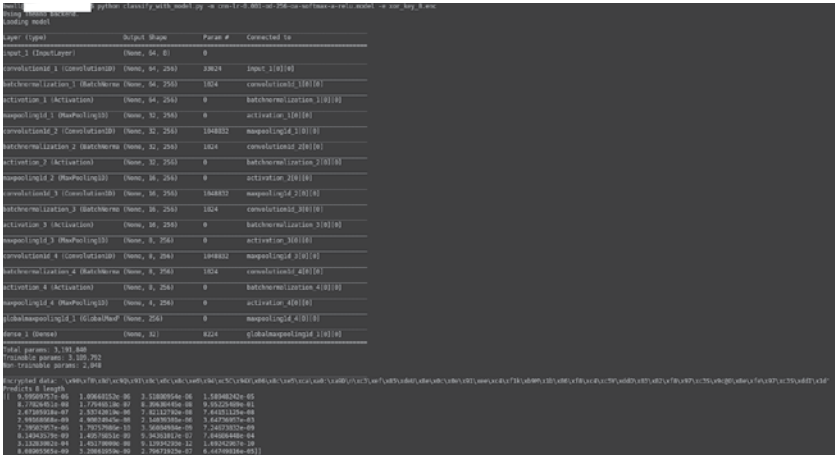
En poco tiempo, la puntuación de validación de nuestro modelo ha aumentado desde su valor inicial de alrededor de 0,63 a más de 0,99.

```
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.0381 - acc: 0.9885
Validation score: 0.991638183594
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.0359 - acc: 0.9893
Validation score: 0.991088867188
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.0409 - acc: 0.9886
Validation score: 0.991577148438
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.0383 - acc: 0.9888
New best validation score: 0.993041992188 (saving)
Epoch 1/1
16384/16384 [==============================] - 66s - loss: 0.0395 - acc: 0.9896
```

En este punto, estamos listos para probar nuestro modelo.

Prueba del modelo RNC

Una vez más, guardaremos nuestro modelo en el disco y lo expondremos a nuestro conjunto de prueba utilizando el script *classify_with_model.py* con los mismos argumentos que antes. La única diferencia es el nombre de nuestro modelo: *cnn-lr-0.001-od-256-oa-softmax-a-relu.model*.



Como podemos ver, el modelo RNC también predijo correctamente que la longitud de la clave de cifrado XOR es de ocho bits.

Conclusiones del aprendizaje profundo

En este capítulo, consideramos cómo se pueden aplicar las redes neuronales para resolver diversos problemas de aprendizaje profundo y examinamos tres arquitecturas de redes neuronales diferentes. Estos son algunos de los puntos clave que cubrimos:

- Las redes neuronales son sumamente flexibles: algoritmos generales que pueden resolver innumerables problemas en un sinfín de formas. A diferencia de otros algoritmos, por ejemplo, las redes neuronales pueden tener millones o incluso miles de millones de parámetros aplicados para definir un modelo.
- Las redes neuronales emplean capas de procesamiento, donde cada capa y su conjunto de nodos realizan un tipo de cálculo en particular. Al menos una de estas capas estará oculta. Es este enfoque multicapa, que emplea capas ocultas, lo que distingue el aprendizaje profundo de todos los demás métodos de Aprendizaje Automático.
- Todos los nodos en cada capa oculta se asignan aleatoriamente a un conjunto de valores de ponderación, uno para cada característica en el conjunto de muestras. Durante el procesamiento, cada nodo multiplica el valor de la característica por su ponderación correspondiente, suma los productos y luego pasa el resultado a través de una función de activación que realiza el cálculo especificado para esa capa. El resultado es un valor de activación que refleja el efecto global del procesamiento de ese nodo.

- Después de cada ciclo de entrenamiento, una función de pérdida compara la decisión de clasificación asignada en la capa de salida a las etiquetas de clase en el conjunto de entrenamiento para determinar cómo se deben modificar las ponderaciones en todas las capas ocultas para producir un resultado más preciso. Este proceso se repite tantas veces como sea necesario antes de que un conjunto de modelos candidatos pueda pasar a las fases de validación y prueba.
- En una red completamente conectada como MLCP, la salida de cada nodo se conecta a las entradas de cada nodo en la capa siguiente. Por el contrario, las RNC emplean una arquitectura parcialmente conectada, en la que los nodos de una capa oculta se conectan solo a un conjunto de nodos contiguos en la capa oculta anterior. Estas conexiones están controladas por el *tamaño* y el *intervalo* de la configuración del filtro.
- En una red prealimentada, la información fluye desde la capa de entrada hasta la capa de salida sin retroceder. Por el contrario, las MLCP y otras redes neuronales recurrentes emplean bucles de retroalimentación y puertas, que determinan cómo y cuándo el contenido de un nodo debe actualizarse o transmitirse a nodos en capas ocultas subsiguientes.
- Las redes prealimentadas son adecuadas para problemas en los que no hay necesidad de recordar el orden en el que aparece una secuencia de muestras en la capa de entrada. Cuando se debe considerar el contexto de muestras sucesivas, una red neuronal recurrente (RNR) como una red MLCP, es una mejor opción. En lugar de las funciones de activación, las MLCP emplean puertas que

les permiten retener y reutilizar la información de estado distribuida en secuencias muy largas de datos de series temporales.

- Las RNC son particularmente adecuadas para resolver problemas, como el reconocimiento de imágenes, donde las características de una muestra están relacionadas espacialmente. Sin embargo, las RNC también pueden funcionar bien con problemas como la clasificación de longitud de clave de cifrado XOR, en la que las relaciones entre una secuencia de caracteres binarios se pueden determinar en función de una serie de conexiones locales entre nodos contiguos.
- Los valores de las características de una muestra solo son visibles para los nodos en la entrada y las primeras capas ocultas. Todas las capas subsiguientes solo pueden "ver" los valores de salida combinados de los nodos en la capa anterior y, por lo tanto, "observar" características globales a niveles crecientes de abstracción. Por ejemplo, una capa puede realizar un procesamiento de bordes, otra puede consolidar bordes en formas y una tercera puede asociar esa forma con una categoría, como "rostro". Las redes neuronales posibilitan este tipo de procesamiento de una manera sumamente granular, pasando de señales de bajo nivel a decisiones complejas a través de una secuencia ordenada de cálculos jerárquicos de varias capas.

Aplicación de la inteligencia artificial a los problemas de seguridad informática

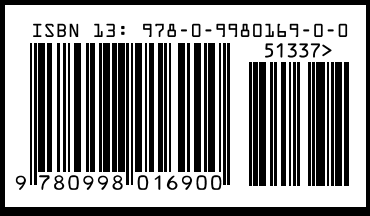
El mundo de la seguridad informática es rico en información. Desde la revisión de registros al análisis de malware, la información está en todas partes y en grandes cantidades, que superan lo que puede cubrir el personal. El campo de estudio de la inteligencia artificial (IA) profundiza en la aplicación de inteligencia a grandes cantidades de datos para obtener resultados significativos. En este libro, cubriremos las técnicas de aprendizaje automático en situaciones prácticas para mejorar su capacidad de triunfar en un mundo impulsado por los datos. Con el agrupamiento, exploraremos cómo agrupar elementos para identificar anomalías. Con la clasificación, veremos cómo entrenar a un modelo para que distinga entre distintas clases de entradas. En la sección de probabilidad, responderemos a la pregunta "¿Cuáles son las chances de que algo ocurra?" y utilizaremos los resultados. Con el aprendizaje profundo, nos meteremos de lleno en los poderosos ámbitos de la IA, inspirados en la biología, que están detrás de algunos de los métodos de aprendizaje automático más eficaces de la actualidad.

Sobre los autores

El Equipo de Cylance Data Science está conformado por expertos en una serie de campos. Entre los miembros de este equipo que colaboraron en este libro están Brian Wallace, un investigador de seguridad convertido en científico de datos con una propensión por crear herramientas que combinan los mundos de la seguridad informática y la ciencia de datos. Sepehr Akhavan-Masouleh es un científico que trabaja en la aplicación de modelos de aprendizaje automático y estadísticos en la seguridad cibernética con un doctorado de la Universidad de California en Irvine. Andrew Davis es un genio de las redes neuronales con un doctorado en ingeniería informática de la Universidad de Tennessee. Mike Wojnowicz es un científico de datos con un doctorado de Cornell University que disfruta de desarrollar e implementar modelos probabilísticos de gran escala debido a su interpretabilidad. El científico de datos John H. Brock investiga las aplicaciones del aprendizaje automático para el análisis y la detección de malware estáticos y cuenta con una maestría en ciencias informáticas de la Universidad de California en Irvine, y se lo suele encontrar depurando código abierto digno de Lovecraft mientras murmura para sí mismo sobre las virtudes de la prueba de unidades.

CYLANCE

ISBN 13: 978-0-9980169-0-0
51337>



Computación / Inteligencia (IA) y Semántica
© 2018 Cylance