



# **PROBABILIDAD**

## **para Machine Learning**

## **con Python**



**PRIMERA EDICIÓN**

AprendeIA

Ligdi González  
2022

# PROBABILIDAD

## para Machine Learning

VERSIÓN 1



## INDICE

|  |           |
|--|-----------|
| <b>CAPÍTULO 1 .....</b>  | <b>6</b>  |
| INTRODUCCIÓN .....   | 6         |
| <b>CAPÍTULO 2 .....</b>  | <b>9</b>  |
| INTRODUCCIÓN A LA PROBABILIDAD .....   | 9         |
| <i>Probabilidad de un suceso .....</i>   | 10        |
| <i>Teoría de la probabilidad.....</i>  | 10        |
| <i>Resumen .....</i>   | 11        |
| <b>CAPÍTULO 3 .....</b>  | <b>12</b> |
| PROBABILIDAD PARA MACHINE LEARNING .....   | 12        |
| <i>Manejar la incertidumbre en Machine Learning .....</i>                            | 12        |
| <i>Ruido en las observaciones .....</i>  | 13        |
| <i>Cobertura incompleta del dominio .....</i>  | 14        |
| <i>Modelo imperfecto del problema.....</i>   | 15        |
| <i>Gestionar la incertidumbre .....</i>  | 15        |
| <i>Resumen .....</i>   | 16        |
| <b>CAPÍTULO 4 .....</b>  | <b>17</b> |
| FUNDAMENTOS DE LA PROBABILIDAD .....   | 17        |
| <i>Teoría de conjuntos .....</i>   | 17        |
| <i>Tipos de probabilidades .....</i>   | 18        |
| <i>Probabilidad conjunta .....</i>   | 18        |
| <i>Probabilidad marginal.....</i>  | 20        |
| <i>Probabilidad condicional.....</i>   | 21        |
| <i>Independencia estadística.....</i>  | 22        |
| <i>Exclusividad .....</i>  | 23        |
| <i>Resumen .....</i>   | 24        |
| <b>CAPÍTULO 5 .....</b>  | <b>25</b> |
| VARIABLES ALEATORIAS .....   | 25        |
| <i>Diferencia entre la teoría y la práctica .....</i>                                | 25        |
| <i>VARIABLES ALEATORIAS .....</i>  | 27        |
| <i>Distribución de probabilidad .....</i>  | 27        |
| <i>Resumen .....</i>   | 28        |
| <b>CAPÍTULO 6 .....</b>  | <b>29</b> |
| DISTRIBUCIONES DE PROBABILIDAD DISCRETA.....   | 29        |
| <i>Definición .....</i>  | 29        |
| <i>Distribución Bernoulli.....</i>   | 30        |
| <i>Distribución binomial.....</i>  | 31        |
| <i>Diferencia entre la Distribución de Bernoulli y la Distribución binomial.....</i> | 32        |
| <i>Distribución categórica .....</i>   | 32        |
| <i>Distribución multinomial.....</i>   | 33        |
| <i>Distribución de probabilidad discretas en Machine Learning .....</i>              | 34        |
| <i>Resumen .....</i>   | 35        |
| <b>CAPÍTULO 7 .....</b>  | <b>36</b> |
| DISTRIBUCIONES DE PROBABILIDAD CONTINUA .....  | 36        |
| <i>Definición .....</i>  | 36        |
| <i>Distribución normal .....</i>   | 37        |
| <i>Distribución exponencial.....</i>   | 39        |
| <i>Distribución de Pareto .....</i>  | 41        |



|   |           |
|---|-----------|
| <i>Resumen</i> .....  | 43        |
| <b>CAPÍTULO 8 .....</b>   | <b>44</b> |
| ESTIMACIÓN DE LA DENSIDAD DE PROBABILIDAD.....                            | 44        |
| <i>Definición</i> .....   | 44        |
| <i>La densidad con un histograma</i> .....                                | 45        |
| <i>Estimación de la densidad paramétrica</i> .....                        | 47        |
| <i>Estimación de la densidad no paramétrica</i> .....                     | 49        |
| <i>Resumen</i> .....  | 53        |
| <b>CAPÍTULO 9 .....</b>   | <b>54</b> |
| TEOREMA DE BAYES.....   | 54        |
| <i>Definición</i> .....   | 54        |
| <i>Teorema de Bayes y Machine Learning</i> .....                          | 56        |
| <i>Teorema de Bayes de las hipótesis de modelización</i> .....            | 56        |
| <i>Resumen</i> .....  | 58        |
| <b>CAPÍTULO 10 .....</b>  | <b>59</b> |
| ESTIMACIÓN .....  | 59        |
| <i>Problema de la estimación de la densidad de la probabilidad.</i> ..... | 59        |
| <i>Estimación de máxima verosimilitud</i> .....                           | 60        |
| <i>Estimación de máxima verosimilitud y Machine Learning</i> .....        | 61        |
| <i>Máxima a posteriori</i> .....  | 62        |
| <i>Máxima a posteriori y Machine Learning</i> .....                       | 63        |
| <i>Resumen</i> .....  | 64        |
| <b>CAPÍTULO 11 .....</b>  | <b>65</b> |
| ENTROPÍA DE LA INFORMACIÓN.....   | 65        |
| <i>Teoría de la información</i> .....                                     | 66        |
| <i>Divergencia de Kullback-Leibler.</i> .....                             | 66        |
| <i>Entropía cruzada</i> .....   | 67        |
| <i>Diferencia entre entropía cruzada y divergencia KL</i> .....           | 68        |
| <i>Resumen</i> .....  | 69        |
| <b>CAPÍTULO 12 .....</b>  | <b>70</b> |
| MÉTRICAS DE PUNTUACIÓN DE PROBABILIDADES.....                             | 70        |
| <i>Puntuación de la pérdida logarítmica</i> .....                         | 70        |
| <i>Puntuación de Brier</i> .....  | 72        |
| <i>Predicción de probabilidades</i> .....                                 | 74        |
| <i>Curvas ROC AUC</i> .....   | 75        |
| <i>Curva PR (curva precision-recall)</i> .....                            | 78        |
| <i>¿Cuándo utilizar las curvas ROC y PR?</i> .....                        | 80        |
| <i>Resumen</i> .....  | 81        |
| <b>CAPÍTULO 13 .....</b>  | <b>82</b> |
| OBTENER MÁS INFORMACIÓN .....   | 82        |
| <i>Consejo general</i> .....  | 82        |
| <i>Ayuda con Python</i> .....   | 83        |
| <i>Ayuda con NumPy</i> .....  | 83        |
| <b>ANEXO 1 .....</b>  | <b>84</b> |
| NUMPY .....   | 84        |
| <i>Introducción a NumPy</i> .....   | 84        |
| <i>Matriz n-dimensional NumPy</i> .....                                   | 85        |
| <i>Funciones para crear matrices</i> .....                                | 86        |



|   |    |
|---|----|
| <i>Combinación de matrices</i> .....      | 87 |
| <i>Convertir listas en matrices</i> ..... | 89 |
| <i>Indexación de matrices</i> .....       | 90 |
| <i>Slicing de matrices</i> .....          | 92 |
| <i>Reajuste de matrices</i> .....         | 95 |
| <i>Resumen</i> .....                      | 97 |



# Capítulo 1

## INTRODUCCIÓN

---

Hola, te doy la bienvenida a esta ebook *Probabilidad para Machine Learning*. Espero que estés preparado para aprender todos los temas bases que necesitas saber sobre la probabilidad y que son implementados dentro de Machine Learning. Estos conocimientos serán de mucha ayuda al momento para ejecutar con éxito cada una de las etapas que tienes que desarrollar dentro de un proyecto de Machine Learning.

El ajuste de modelos en un conjunto de datos de entrenamiento no tiene sentido sin la probabilidad. La interpretación del rendimiento de un algoritmo de Machine Learning no tiene sentido sin la probabilidad. Un profesional de Machine Learning no puede ser eficaz sin comprender y apreciar los conceptos y métodos básicos del campo de la probabilidad.

Y si aún tienes dudas, te comento que los conceptos más simples cuando se trabaja con datos provienen del campo de la probabilidad, como el valor más probable y la idea de una distribución de probabilidad sobre las observaciones. A partir de estas ideas sencillas, los marcos probabilísticos, como la estimación de máxima probabilidad, subyacen a una serie de algoritmos de Machine Learning, desde la Regresión Logística hasta las Redes Neuronales.

### ¿A quién va dirigido este ebook?

Este ebook está dirigido a profesionales que se están iniciando en Machine Learning o ya experimentados pero que quieren tener una base más sólida para entender



mejor cómo funcionan los algoritmos y así obtener mejores resultados en sus proyectos.

El objetivo de este ebook es que conozca todos los temas relacionados a la probabilidad que se utilizan dentro de Machine Learning para que puedes implementar estos conocimientos en los proyectos que estés desarrollando, así como optimizar los que ya hayas desarrollado con anterioridad. Este ebook está escrito para personas de cualquier edad y conocimientos previos pero que quieren tener una mejor base y un mejor entendimiento sobre cómo funciona los algoritmos que están dentro de Machine Learning.

Este ebook es para ti, si estas interesado en conocer con más detalle en cómo funcionan los algoritmos de Machine Learning, así como también en cómo tener un mejor manejo de los datos.

## ¿Qué puedes aprender?

El propósito de este ebook es que conozcas todos los temas relacionados a la probabilidad que se utilizan dentro de Machine Learning. Aprenderás de forma eficiente la parte teórica de cada una de las áreas y a su vez la parte práctica utilizando Python como lenguaje de programación.

A medida que vayas avanzando en la lectura y práctica de este ebook, irás aprendiendo sobre los siguientes temas:

- Cómo se relaciona Machine Learning y cómo aprovechar el pensamiento probabilístico en un proyecto de Machine Learning.
- Considerar los datos en términos de variables aleatorias y cómo reconocer y muestrear a partir de funciones de distribución de probabilidad discretas y continuas comunes.
- Cómo utilizar los métodos probabilísticos para evaluar los modelos de Machine Learning directamente sin evaluar su rendimiento en un conjunto de datos de prueba.
- Cómo calcular y considerar la probabilidad desde la perspectiva bayesiana y calcular la probabilidad condicional con el teorema de Bayes para escenarios comunes.
- Cómo cuantificar la incertidumbre utilizando medidas de información y entropía del campo de la teoría de la información y calcular cantidades como la entropía cruzada y la información mutua.
- Cómo evaluar los modelos de clasificación que predicen probabilidades y calibrar las predicciones probabilísticas.

Una vez que hayas finalizado de trabajar con este ebook, entenderás muchos de los procedimientos que se realizan dentro de Machine Learning, impactando tu práctica de Machine Learning de las siguientes maneras:



- Calcular y manejar con confianza tanto la probabilidad frecuentista (recuentos) como la probabilidad bayesiana (creencias) en general y en el contexto de los conjuntos de datos de Machine Learning.
- Seleccionar y utilizar con confianza las funciones de pérdida y las medidas de rendimiento al entrenar algoritmos de Machine Learning, con el respaldo de un conocimiento del marco probabilístico subyacente y las relaciones entre las métricas.
- Evaluar con confianza los modelos de predicción de clasificación, incluyendo el establecimiento de una línea de base robusta en el rendimiento, las medidas de rendimiento probabilístico y las probabilidades de predicción calibradas.

## Notaciones previas

Para efectos de este ebook, se utilizará Python para realizar las explicaciones prácticas, por lo que es requerido que se cree un entorno profesional en tu computador para que puedas empezar a practicar.

Por lo general, lo que necesitas para crear este entorno es lo siguiente:

- *Lenguaje de programación - Python 3.x*: hay muchos practicantes que continúan usando R, especialmente si tienen una fuerte formación en estadística. Pero en general, Python es un lenguaje de programación más versátil y popular que facilita la solución de una gama más amplia de problemas, desde el web scraping y la limpieza de datos hasta el modelado y la construcción de cuadros de mando o la producción de sus modelos. Hoy en día, la mayoría de los expertos están usando Python 3, con sus respectivas actualizaciones.
- *Editores de textos - IDE*: hay un montón de magníficos editores de textos para editar código Python. Uno de los más populares es Jupyter Notebook y si apenas te estás iniciando y no tienes ninguna experiencia con algún IDE este sería una buena opción. Otra opción de IDE es Spyder, que últimamente se ha vuelto muy popular entre los desarrolladores de Machine Learning. La selección del IDE es algo personal y puedes utilizar con el que te sientes más cómodo.



Para efectos de este ebook, todas las explicaciones prácticas se realizarán utilizando la librería de NumPy en conjunto con el IDE Jupyter Notebook. En el anexo 1 de este ebook encontrarás una explicación breve sobre NumPy.



# Capítulo 2

## INTRODUCCIÓN A LA PROBABILIDAD

---

La probabilidad es una herramienta universalmente aceptada para expresar el grado de confianza o duda sobre alguna proposición en presencia de información incompleta o incertidumbre. Por convención, las probabilidades se calibran en una escala de 0 a 1; asignar a algo una probabilidad de cero equivale a expresar la creencia de que lo consideramos imposible, mientras que asignar una probabilidad de uno equivale a considerarlo en certeza. La mayoría de las proposiciones se sitúan en un punto intermedio.

Las declaraciones de probabilidad que hacemos pueden basarse en nuestra experiencia pasada o en nuestros juicios personales. Independientemente de que nuestras afirmaciones sobre la probabilidad se basen en experiencias pasadas o en juicios personales subjetivos, obedecen a un conjunto de reglas comunes, que podemos utilizar para tratar las probabilidades en un marco matemático, y también para tomar decisiones sobre predicciones, para entender sistemas complejos, o como experimentos intelectuales y para el entretenimiento.

La teoría de la probabilidad es una de las ramas más aplicables de las matemáticas. Se utiliza como herramienta principal para el análisis de metodologías estadísticas, se emplea de forma rutinaria en casi todas las ramas de la ciencia, como la biología, la física, las finanzas y muchas otras. La formación en la teoría, los modelos y las



aplicaciones de la probabilidad es casi una parte de la educación básica. Así de importante es.

## Probabilidad de un suceso

La probabilidad es una medida que cuantifica la posibilidad de que se produzca un suceso. Por ejemplo, podemos cuantificar la probabilidad de que se produzca un terremoto en una región, que gane un candidato la presidencia o que compres un producto. La probabilidad de un suceso puede calcularse directamente de la siguiente manera:

$$\text{probabilidad} = \frac{\text{ocurrencias}}{\text{no ocurrencias} + \text{ocurrencias}}$$

Solamente se tiene que contar las ocurrencias del suceso y dividirla por el total de resultados posibles del mismo.



La probabilidad asignada es un valor fraccionario y siempre está en el rango entre 0 y 1, donde 0 indica que no hay probabilidad y 1 representa la probabilidad total.

En conjunto, la probabilidad de todos los sucesos posibles suma el valor de probabilidad uno. Si todos los sucesos posibles son igual de probables, la probabilidad de que ocurran es 1 dividido por el total de sucesos o ensayos posibles. Por ejemplo, cada uno de los números del 1 al 6 son igualmente probables al lanzar un dado justo, por lo que cada 1 tiene una probabilidad de 1/6 o 0,166 de ocurrir.

La probabilidad suele escribirse con una p minúscula y puede expresarse como porcentaje multiplicando el valor por 100. Por ejemplo, la probabilidad del ejemplo anterior puede expresarse como 16%, dado  $0,166 \times 100$ . Una probabilidad del 50% de un suceso, que suele denominarse probabilidad 50-50, significa que es probable que ocurra la mitad de las veces.

Aunque las probabilidades suelen tener su propia notación de victorias a pérdidas, escrita como victorias:pérdidas, por ejemplo, 1:3 para 1 victoria y 3 pérdidas o 1/4 (25%) de probabilidades de ganar.

## Teoría de la probabilidad

En general, la probabilidad es una extensión de la lógica que puede utilizarse para cuantificar, gestionar y aprovechar la incertidumbre. Como campo de estudio, suele denominarse teoría de la probabilidad para diferenciarla de la probabilidad de un acontecimiento concreto.



La teoría de la probabilidad tiene tres conceptos importantes:

- Evento (A): un resultado al que se le asigna una probabilidad.
- Espacio muestral (S): el conjunto de posibles resultados o eventos.
- Función de probabilidad (P): función utilizada para asignar una probabilidad a un suceso.

La probabilidad de que un suceso (A) se extraiga del espacio muestral (S) viene determinada por la función de probabilidad (P). La forma o distribución de todos los sucesos en el espacio muestral se denomina distribución de probabilidad. En muchos ámbitos, la distribución de las probabilidades de los sucesos tiene una forma conocida, como la uniforme, si todos los sucesos tienen la misma probabilidad, o la gaussiana, si la probabilidad de los sucesos tiene una forma normal o de campana.

La probabilidad constituye la base de muchos campos aplicados de las matemáticas, incluida la estadística, y es un fundamento importante de muchos campos de estudio de nivel superior, como la física, la biología y la informática.

## Resumen

En este capítulo, aprendiste una breve introducción de lo que se trata la probabilidad. Específicamente, se explicó:

- La certeza es inusual y el mundo es desordenado, lo que requiere operar bajo la incertidumbre.
- La probabilidad cuantifica la posibilidad o la creencia de que se produzca un acontecimiento.
- La teoría de la probabilidad es la matemática de la incertidumbre.



# Capítulo 3

## PROBABILIDAD PARA MACHINE LEARNING

---

Machine Learning requiere gestionar la incertidumbre. Como ya debes saber, hay muchas fuentes de incertidumbre en un proyecto de Machine Learning como, por ejemplo, la varianza en los valores de los datos específicos, la muestra de datos recopilados del dominio y en la naturaleza imperfecta de cualquier modelo desarrollado a partir de dichos datos.

La gestión de la incertidumbre que es inherente a Machine Learning para el modelado predictivo puede lograrse a través de las herramientas y técnicas de la probabilidad, un campo específicamente diseñado para manejar la incertidumbre.

### **Manejar la incertidumbre en Machine Learning**

El área de Machine Learning requiere sentirse cómodo con la incertidumbre. Cuando hablamos de incertidumbre se refiere a trabajar con información imperfecta o incompleta.





La incertidumbre es fundamental en el campo de Machine Learning, pero es uno de los aspectos que más dificultades causa a los principiantes, especialmente a los que provienen de un entorno de desarrollo.

Para los ingenieros y desarrolladores de software, los computadores son deterministas. Acá se escribe un programa y el computador hace lo que se le ha escrito. En cambio, los algoritmos se analizan en función de la complejidad espacial o temporal y pueden elegirse para optimizar lo que sea más importante para el proyecto, como la velocidad de ejecución o las limitaciones de memoria.

Cuando se habla del modelado predictivo dentro de Machine Learning implica ajustar un modelo para asignar ejemplos de entradas a una salida, como un número en el caso de un problema de regresión o una etiqueta de clase en el caso de un problema de clasificación.

Es normal, si estás empezando, que te hagas preguntas de cómo definir cuáles son las mejores características del conjunto de datos o cuál es el mejor algoritmo a utilizar para un problema específico. La realidad es que las respuestas a estas preguntas se desconocen e incluso pueden ser desconocidas, al menos exactamente.

Esta es la principal causa de dificultad para los principiantes. La razón por la que se desconocen las respuestas es la incertidumbre, y la solución es evaluar sistemáticamente diferentes soluciones hasta descubrir un conjunto de características y/o un algoritmo bueno o suficiente para un problema de predicción específico.

Machine Learning tiene distintas fuentes de incertidumbre las principales son las siguientes:

- Ruido en las observaciones
- Cobertura incompleta del dominio
- Modelo imperfecto del problema

A continuación, se explica cada una de ellas.

## Ruido en las observaciones

El ruido se refiere a la variabilidad de la observación. La variabilidad puede ser natural, pero también puede ser un error, como un desliz al medir o una errata al escribir. Esta variabilidad no solo afecta a las entradas o mediciones, sino también a las salidas, por ejemplo, una observación podría tener una etiqueta de clase



incorrecta. Esto significa que, aunque tengamos observaciones para el dominio, debemos esperar cierta variabilidad o aleatoriedad.

El mundo real, y a su vez los datos reales, son desordenados o imperfectos. Como profesionales, debemos permanecer escépticos ante los datos y desarrollar sistemas para esperar e incluso aprovechar esta incertidumbre. Por eso se dedica tanto tiempo a revisar las estadísticas de los datos y a crear visualizaciones que ayuden a identificar esos casos aberrantes o inusuales, a esto se le conoce como preprocesamiento de datos.

## Cobertura incompleta del dominio

Las observaciones que se utilizan dentro de un conjunto de datos para entrenar un modelo son solamente una muestra de una población, por lo tanto, tenemos que estar consciente que se encuentra incompleta, solamente es una representación pequeña de un todo.

En estadística, una muestra aleatoria se refiere a una colección de observaciones elegidas del dominio sin sesgo sistemático, por ejemplo, uniformemente al azar. Sin embargo, siempre habrá alguna limitación que introduzca un sesgo. Por ejemplo, podemos elegir verificar los alumnos que continuarán estudiando en la universidad, los estudiantes se seleccionan al azar, pero el alcance se limita a una universidad. El alcance puede ampliarse a otras universidades de una ciudad, de un país e inclusive del continente.

Se requiere un nivel adecuado de varianza y sesgo en la muestra, de manera que ésta sea representativa de la tarea o el proyecto para el que se utilizarán los datos o el modelo. Nuestro objetivo es recoger u obtener una muestra aleatoria adecuadamente representativa de observaciones para entrenar y evaluar un modelo de Machine Learning.

A menudo, tenemos poco control sobre el proceso de muestreo. En su lugar, accedemos a una base de datos o a un archivo CSV y los datos que tenemos son los que debemos trabajar. En todos los casos, nunca tendremos todas las observaciones. Si las tuviéramos, no sería necesario un modelo predictivo. Esto significa que siempre habrá algunos casos no observados. Habrá una parte del dominio del problema para la que no tengamos cobertura.

Por muy bien que fomentemos la generalización de nuestros modelos, solo podemos esperar que podamos cubrir los casos del conjunto de datos de entrenamiento y los casos destacados que no están. Por esa razón dividimos un conjunto de datos en conjuntos de entrenamiento y de prueba o utilizamos métodos de remuestreo como la validación cruzada k-fold. Hacemos esto para manejar la incertidumbre en la representatividad de nuestro conjunto de datos y estimar el rendimiento de un procedimiento de modelado en datos no utilizados en ese procedimiento.



## Modelo imperfecto del problema

Siempre tienes que tener en cuenta que todos los modelos de Machine Learning tendrá algún error. Esto se suele resumir en que todos los modelos son erróneos. Esta afirmación no se aplica solo al modelo sino a todo el procedimiento utilizado para prepararlo, incluyendo la elección y preparación de los datos, la elección de los hiperparámetros de entrenamiento y la interpretación de las predicciones del modelo.

El error del modelo podría significar predicciones imperfectas, como la predicción de una cantidad en un problema de regresión que es muy diferente a lo que se esperaba, o la predicción de una etiqueta de clase que no coincide con lo que se esperaría. Este tipo de error en la predicción es esperable dada la incertidumbre que tenemos sobre los datos que acabamos de discutir, tanto en términos de ruido en las observaciones como cobertura incompleta del dominio.

Otro tipo de error es el de omisión. Omitimos detalles o los abstraemos para poder generalizar a nuevos casos. Esto se consigue seleccionando modelos más sencillos, pero más robustos a las especificidades de los datos, frente a modelos complejos que pueden estar muy especializados a los datos de entrenamiento. Por ello, podemos elegir, y a menudo lo hacemos, un modelo que se sabe que comete errores en el conjunto de datos de entrenamiento con la esperanza de que el modelo se generalice mejor a los nuevos casos y tenga un mejor rendimiento general.

No obstante, se necesitan predicciones. Dado que sabemos que los modelos cometerán errores, manejamos esta incertidumbre buscando un modelo que sea lo suficientemente bueno. Esto suele interpretarse como la selección de un modelo que sea hábil en comparación con un método ingenuo u otros modelos de aprendizaje establecidos, por ejemplo, un buen rendimiento relativo.

## Gestionar la incertidumbre

La incertidumbre en Machine Learning aplicado se gestiona mediante la probabilidad. La probabilidad es el campo de las matemáticas diseñado para manejar, manipular y aprovechar la incertidumbre.

Los métodos y herramientas de la probabilidad proporcionan la base y la forma de pensar sobre la naturaleza aleatoria o estocástica de los problemas de modelado predictivo que se abordan con Machine Learning, por ejemplo:

- En términos de observaciones ruidosas, la probabilidad y la estadística nos ayudan a entender y cuantificar el valor esperado y la variabilidad de las variables en nuestras observaciones del dominio.



- En cuanto a la cobertura incompleta del dominio, la probabilidad ayuda a comprender y cuantificar la distribución y la densidad esperadas de las observaciones del dominio.
- En cuanto al error del modelo, la probabilidad ayuda a comprender y cuantificar la capacidad y la varianza esperadas en el rendimiento de nuestros modelos predictivos cuando se aplican a nuevos datos.

Pero esto es solo el principio, ya que la probabilidad proporciona la base para el entrenamiento iterativo de muchos modelos de Machine Learning, llamado estimación de máxima verosimilitud, que está detrás de modelos como la Regresión Lineal, la Regresión Logística, las redes neuronales artificiales y mucho más.

La probabilidad también proporciona la base para el desarrollo de algoritmos específicos, como Naive Bayes, así como subcampos enteros de estudio en Machine Learning, como los modelos gráficos como la red de creencia bayesiana.

Los procedimientos que utilizamos en Machine Learning aplicado se eligen cuidadosamente para hacer frente a las fuentes de incertidumbre que hemos discutido, pero entender por qué se eligieron los procedimientos requiere una comprensión básica de la probabilidad y la teoría de la probabilidad.

## Resumen

En este capítulo, descubriste la relación entre las probabilidades y Machine Learning. Específicamente, se explicó:

- La incertidumbre es la mayor fuente de dificultad para los principiantes en Machine Learning, especialmente para los desarrolladores.
- El ruido en los datos, la cobertura incompleta del dominio y los modelos imperfectos son las tres principales fuentes de incertidumbre en Machine Learning.
- La probabilidad proporciona la base y las herramientas para cuantificar, manejar y aprovechar la incertidumbre en Machine Learning.



# Capítulo 4

## FUNDAMENTOS DE LA PROBABILIDAD

---

La probabilidad para una sola variable aleatoria es sencilla, aunque puede complicarse cuando se consideran dos o más variables. Cuando tienes solo dos variables, puedes estar interesado en la probabilidad de dos sucesos simultáneos (probabilidad conjunta), en la probabilidad de un suceso dada la ocurrencia de otro (probabilidad condicional) o simplemente en la probabilidad de un suceso independientemente de otras variables (probabilidad marginal).

Estos tipos de probabilidad son fáciles de definir, pero la intuición que hay detrás de su significado puede tardar en asimilarse, por lo que se necesitan algunos ejemplos trabajados con los que se pueda jugar.

### Teoría de conjuntos

Un conjunto es simplemente una colección de cosas. Estas cosas pueden ser números, pero también pueden ser alfabetos, objetos o cualquier cosa. La teoría de conjuntos es una herramienta matemática que define las operaciones con conjuntos. Nos proporciona la aritmética básica para combinar, separar y descomponer conjuntos.



Por su parte, la probabilidad es una medida del tamaño de un conjunto. Sí, la probabilidad no es solo un número que nos dice la frecuencia relativa de los sucesos, es un operador que toma un conjunto y nos dice cuán grande es el conjunto.

De acuerdo a esto, un conjunto es una colección de elementos, por su parte, dado un conjunto, a menudo queremos especificar una porción del conjunto, a esto se le subconjunto, podemos escribir:

$$B \subseteq A$$

para denotar que B es un subconjunto de A.

## Tipos de probabilidades

El cálculo de la probabilidad es relativamente sencillo cuando se trabaja con una sola variable aleatoria, pero cuando se consideran dos o más variables aleatorias, la cosa se pone más interesante.

Hay tres tipos principales de probabilidades que nos puede interesar calcular cuando trabajamos con dos o más variables aleatorias:

- Probabilidad conjunta: la probabilidad de eventos simultáneas.
- Probabilidad marginal o incondicional: la probabilidad de un suceso con independencia de las demás variables.
- Probabilidad condicional: probabilidad de un suceso en función de la presencia de otros sucesos.

El significado y el cálculo de estos diferentes tipos de probabilidades varían en función de si las dos variables aleatorias son independientes, más simples, o dependientes, más complicadas. Expliquemos mejor cada una de ellas.

## Probabilidad conjunta

La probabilidad conjunta es una medida estadística que calcula la probabilidad de que dos sucesos ocurran juntos y en el mismo momento. La probabilidad conjunta es la probabilidad de que el suceso Y ocurra al mismo tiempo que el suceso X.

La notación de la probabilidad conjunta puede adoptar diferentes formas. La siguiente fórmula representa la probabilidad de intersección de eventos:

$$P(X \cap Y)$$

donde:

X, Y = dos sucesos diferentes que se cruzan



$P(X \text{ e } Y)$ ,  $P(XY)$  = la probabilidad conjunta de X e Y.

La probabilidad es un campo estrechamente relacionado con la estadística que se ocupa de la probabilidad de que se produzca un suceso o fenómeno. Se cuantifica como un número entre 0 y 1, ambos inclusive, donde 0 indica una probabilidad imposible de que ocurra y 1 denota el resultado seguro de un evento.

Por ejemplo, la probabilidad de sacar una carta roja de una baraja es  $\frac{1}{2} = 0,5$ . Esto significa que hay la misma probabilidad de sacar un rojo que de sacar un negro; como hay 52 cartas en una baraja, de las cuales 26 son rojas y 26 son negras, hay una probabilidad de 50-50 de sacar una carta roja frente a una negra.



La probabilidad conjunta es una medida de dos sucesos que ocurren al mismo tiempo, y solo puede aplicarse a situaciones en las que puede ocurrir más de una observación al mismo tiempo.

Por ejemplo, de una baraja de 52 cartas, la probabilidad conjunta de coger una carta que sea a la vez roja y 6 es  $P(6 \cap \text{rojo}) = 2/52 = 1/26$ , ya que una baraja tiene dos cartas 6 rojo: el 6 de corazones y el 6 de diamantes. Como los sucesos “6” y “rojo” son independientes en este ejemplo, también se puede utilizar la siguiente fórmula para calcular la probabilidad conjunta:

$$P(6 \cap \text{rojo}) = P(6) \times P(\text{rojo}) = 4/52 \times 26/52 = 1/26$$

El símbolo “ $\cap$ ” es una probabilidad conjunta y se denomina intersección. La probabilidad de que se produzca el suceso X y el suceso Y es lo mismo que el punto en el que se cruzan X e Y. Por tanto, la probabilidad conjunta también se denomina intersección de dos o más sucesos.

Un diagrama de Venn es quizás la mejor herramienta visual para explicar una intersección.



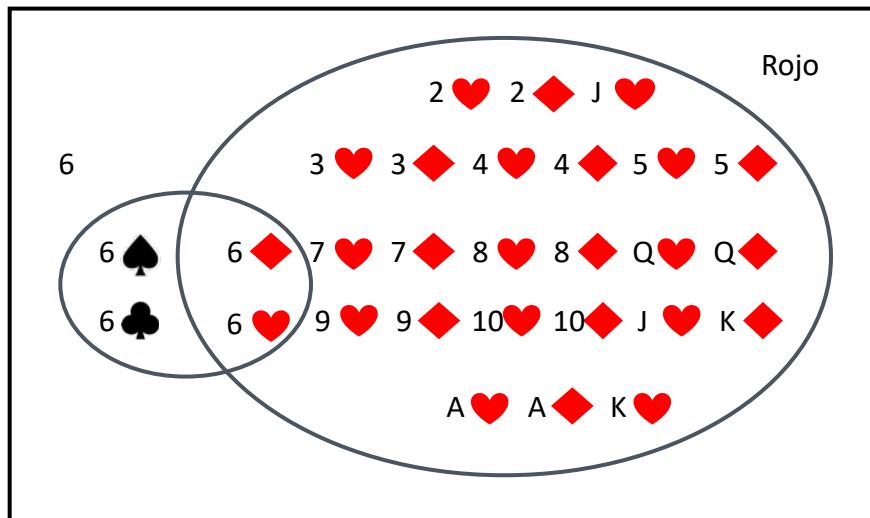


Figura 1. Diagrama de Venn para ilustrar las relaciones lógicas

En la figura anterior, el punto en el que ambos círculos se superponen es la intersección, que tiene dos observaciones: el 6 de corazones y el 6 de diamantes.

## Probabilidad marginal

Una probabilidad marginal o incondicional es la posibilidad de que se produzca un único resultado entre varios posibles. El término se refiere a la probabilidad de que un acontecimiento tenga lugar con independencia de que se hayan producido otros acontecimientos o de que se den otras condiciones.

La probabilidad de que llueve en Bogotá, el día de Navidad, sin tener en cuenta los patrones meteorológicos históricos y los datos climáticos de Bogotá para esas fechas de diciembre es un ejemplo de probabilidad marginal.

La probabilidad incondicional puede contrastarse con la probabilidad condicional.

La probabilidad marginal de un suceso puede determinarse sumando los resultados del suceso y dividiendo por el número total de resultados posibles.

$$P(A) = \frac{\text{Número de veces que } A \text{ ocurre}}{\text{Número total de los resultados posibles}}$$



La probabilidad marginal también se conoce como probabilidad incondicional y mide la posibilidad de que ocurra un suceso ignorando cualquier conocimiento obtenido de sucesos anteriores o externos. Como esta probabilidad ignora la información nueva, permanece constante.



Como ejemplo podemos evaluar el lanzamiento de una moneda, la probabilidad de obtener cara es igual a 0,5 y la probabilidad de obtener cruz es igual a 0,5. Esto es cierto para cada lanzamiento, no importa cuántas veces se lance la moneda o cuáles hayan sido los resultados anteriores. Cada lanzamiento de la moneda es único y no hay manera de conectarlo con ningún otro. En consecuencia, el resultado de cada lanzamiento de una moneda es un evento estadísticamente independiente de los resultados de cualquier otro lanzamiento de ella.

## Probabilidad condicional

La probabilidad condicional se define como la probabilidad de que se produzca un suceso o resultado, basada en la ocurrencia de un suceso o resultado anterior. La probabilidad condicional se calcula multiplicando la probabilidad del suceso anterior por la probabilidad actualizada del suceso posterior o condicional.

La probabilidad condicional puede contrastarse con la probabilidad marginal. La probabilidad marginal o incondicional se refiere a la probabilidad de que un suceso tenga lugar independientemente de que se hayan producido otros sucesos o de que se den otras condiciones.

Las probabilidades condicionales están supeditadas a que se produzca un resultado o acontecimiento previo. Una probabilidad condicional consideraría dichos sucesos en relación unos con otros.



La probabilidad condicional es la probabilidad de que se produzca un suceso o resultado en función de la ocurrencia de algún otro suceso o resultado anterior.

Se dice que dos sucesos son independientes si la ocurrencia de uno de ellos no afecta a la probabilidad de que ocurra el otro. Sin embargo, si la ocurrencia o no de un suceso afecta a la probabilidad de que ocurra el otro, se dice que los dos sucesos son dependientes. Si los sucesos son independientes, entonces la probabilidad de algún suceso B no depende de lo que ocurra con el suceso A. Una probabilidad condicional, por tanto, se refiere a aquellos sucesos que son dependientes entre sí.

La probabilidad condicional se suele representar como la “probabilidad de A dada B”, anotada como  $P(A|B)$ .

$$P(B|A) = \frac{P(A \cap B)}{P(B)}$$

donde:

P = probabilidad



A = evento A

B = evento B

Por ejemplo, suponte que sacas tres pelotas, roja, azul y verde, de una bolsa. Cada pelota tiene la misma probabilidad de ser extraída.

¿Cuál es la probabilidad condicional de sacar la pelota roja después de haber sacado la azul?

En primer lugar, la probabilidad de sacar una pelota azul es de aproximadamente el 33%, ya que es un resultado posible de entre tres. Suponiendo que se produzca este primer suceso, quedarán dos pelotas, cada una de las cuales tendrá una probabilidad del 50% de ser extraída. Por lo tanto, la probabilidad de sacar una pelota azul después de haber sacado una pelota roja sería de un 16,5% (33% x 50%).

Veamos otro ejemplo para que comprendas mejor este concepto. Considera que se ha lanzado un dado y se te pide que des la probabilidad de que haya salido un cinco. Hay seis resultados igualmente probables, así que la respuesta es 1/6.

Pero imagínate que antes de responder, recibes información adicional de que el número lanzado era impar. Dado que solo hay tres números impares posibles, uno de los cuales es el cinco, seguramente revisarías tu estimación de la probabilidad de que salga un conde de 1/6 a 1/3.

Esta probabilidad revisada de que se haya producido un suceso A, teniendo en cuenta la información adicional de que otro suceso B ha ocurrido definitivamente en este ensayo del experimento, se llama probabilidad condicional de A dado B y se denota por  $P(A|B)$ .

## Independencia estadística

Si una variable no depende de una segunda variable, se denomina independencia o independencia estadística. Esto influye en el cálculo de las probabilidades de las dos variables.

Por ejemplo, podemos estar interesados en la probabilidad conjunta de los sucesos independientes A y B, que es lo mismo que la probabilidad de A y la probabilidad de B. Las probabilidades se combinan utilizando la multiplicación, por lo que la probabilidad conjunta de los sucesos independientes se calcula como la probabilidad del suceso A multiplicada por la probabilidad del suceso B. Esto puede enunciarse formalmente somo sigue:

$$\text{Probabilidad conjunta: } P(A \cap B) = P(A) \times P(B)$$



Como puedes intuir, la probabilidad marginal de un suceso para una variable aleatoria independiente es simplemente la probabilidad del suceso. Es la idea de probabilidad de una única variable aleatoria con la que estamos familiarizados:

*Probabilidad marginal:  $P(A)$*

Nos referimos a la probabilidad marginal de una probabilidad independiente como simplemente la probabilidad. Del mismo modo, la probabilidad condicional de A dada B cuando las variables son independientes es simplemente la probabilidad de A ya que la probabilidad de B no tiene ningún efecto. Por ejemplo:

*Probabilidad condicional:  $P(A|B) = P(A)$*

Puedes que estés familiarizado con la noción de independencia estadística del muestreo. Esto supone que una muestra no se ve afectada por las muestras anteriores y no afecta a las muestras futuras. Muchos algoritmos de Machine Learning suponen que las muestras de un dominio son independientes entre sí y proceden de la misma distribución de probabilidad, denominada independiente e idénticamente distribuida.

## Exclusividad

Si la ocurrencia de un evento excluye la ocurrencia de otros eventos, se dice que los eventos son mutuamente excluyentes. La probabilidad de los sucesos se dice que disjunta, lo que significa que no pueden interactuar, son estrictamente independientes. Si la probabilidad del suceso A es mutuamente excluyente con el suceso B, entonces la probabilidad conjunta del suceso A y del suceso B es cero.

$$P(A \cap B) = 0,0$$

En cambio, la probabilidad de un resultado puede describirse como el suceso A o el suceso B, expresado formalmente como sigue:

$$P(A \text{ o } B) = P(A) + P(B)$$

La o también se llama unión y se denota como una letra U mayúscula ( $\cup$ ). Por ejemplo:

$$P(A \text{ o } B) = P(A \cup B)$$

Si los sucesos no son mutuamente excluyentes, podemos estar interesados en el resultado de cualquiera de ellos. La probabilidad de los sucesos no mutuamente excluyentes se calcula como la probabilidad del suceso A y la probabilidad del suceso B menos la probabilidad de que ambos sucesos ocurran simultáneamente. Esto se puede enunciar formalmente de la siguiente manera:



$$P(A \cup B) = P(A) + P(B) - (A \cap B)$$

## Resumen

En este capítulo, descubriste una introducción a la probabilidad conjunta, marginal y condicional para múltiples variables aleatorias. Específicamente se aprendiste:

- La probabilidad conjunta es la probabilidad de que dos o más sucesos ocurran simultáneamente.
- La probabilidad marginal es la probabilidad de un suceso independientemente del resultado de otras variables.
- La probabilidad condicional es la probabilidad de que un suceso ocurra en presencia de otro u otros sucesos.



# Capítulo 5

## VARIABLES ALEATORIAS

---

La probabilidad puede utilizarse para algo más que para calcular la probabilidad de un suceso, puede resumir la probabilidad de todos los resultados posibles. Una cosa de interés en probabilidad se llama variable aleatoria, y la relación entre cada resultado posible para una variable aleatoria y sus probabilidades se llama distribución de probabilidad.

Las distribuciones de probabilidad son un concepto fundacional importante en probabilidad y los nombres y formas de las distribuciones de probabilidad más comunes te resultarán familiares. La estructura y el tipo de la distribución de probabilidad varía en función de las propiedades de la variable aleatoria, como continua o discreta, y esto, a su vez, influye en cómo se podría resumir la distribución o cómo calcular el resultado más probable y su probabilidad.

### Diferencia entre la teoría y la práctica

Cuando se trabaja en un problema de análisis de datos, uno de los mayores retos es la disparidad entre las herramientas teóricas que aprendemos en la escuela o por nuestra cuenta, y los datos reales con los que trabajamos en vida real. Por datos reales se entiende una colección de números, quizás organizados o quizás no.



Cuando se entrega el conjunto de datos, lo primero que se hace seguramente no sería definir el campo y luego definir la medida. En su lugar, normalmente se calcula la media, la desviación estándar y quizás algunas puntuaciones sobre la simetría.

Por un lado, se tienen herramientas de probabilidad bien definidas, pero, por otro lado, se tienen un conjunto de “habilidades de batalla” prácticas para procesar los datos. A menudo se ven como dos entidades separadas. Comúnmente, se utilizan las últimas en vez de utilizar las herramientas predefinidas para salvar la distancia entre la teoría y la práctica.

El punto de partida que se debe considerar es un espacio de probabilidad. Este es un concepto abstracto, y ciertamente no es fácil de usar, pero por esa razón se debe reconocer que el espacio muestral y el espacio de sucesos se basan en enunciados, por ejemplo, obtener una cara al lanzar una moneda o ganar el juego. Estos enunciados no son números, por esa razón se requiere convertirlos en unos y allí es donde entra el concepto de variables aleatorias.



Las variables aleatorias son mapeos de eventos a números.

Supongamos que se ha construido una variable aleatoria que traduce enunciados a números. La siguiente tarea es dotar a la variable aleatoria de probabilidad. Mas concretamente, se tiene que asignar probabilidades a la variable aleatoria para poder realizar cálculos. Para ello se utiliza el concepto denominado función de masa de probabilidad (PMF).

Las funciones de masa de probabilidad son los histogramas ideales de las variables aleatorias. La mejor manera de pensar en una función de masa de probabilidad es un histograma, este cuenta con dos ejes: el eje X denota el conjunto de datos y el eje Y denota la probabilidad. Para cada uno de los estados que posee la variable aleatoria, el histograma indica la probabilidad de obtener un estado concreto.

La función de masa de probabilidad es el histograma ideal de una variable aleatoria. Proporciona una caracterización completa de la variable aleatoria. Si tenemos una variable aleatoria, se debe especificar su función de masa de probabilidad. A la inversa, si se dice la función de masa de probabilidad, ha especificado una variable aleatoria.

La última etapa se refiere a la obtención de información de la función de masa de probabilidad, como la media y la desviación estándar. Entender la diferencia entre la media que se obtiene de una función de masa de probabilidad y la media que se obtiene de un histograma, ayudará inmediatamente a tender un puente desde la teoría a la práctica.



## Variables aleatorias

Una variable aleatoria es una cantidad producida por un proceso aleatorio. En probabilidad, una variable aleatoria puede tomar uno de los muchos valores posibles, por ejemplo, eventos del espacio de estados. Se puede asignar una probabilidad a un valor específico o a un conjunto de valores de una variable aleatoria.

Una variable aleatoria a menudo se denota como una letra mayúscula, por ejemplo,  $X$  y los valores de la aleatoria se denotan con una letra minúscula y un índice, por ejemplo,  $x_1, x_2, x_3$ .

Los valores que puede tomar una variable aleatoria se llama su dominio, y el dominio de una variable aleatoria puede ser discreto o continuo. Expliquemos mejor esto.

- Variable aleatoria discreta: los valores se extraen de un conjunto finito de estados, por ejemplo, los colores de las paredes de una casa.
- Variable aleatoria booleana: los valores se extraen de un conjunto finito de estados, por ejemplo, el lanzamiento de una moneda.
- Variable aleatoria continua: los valores se extraen de un rango de valores numéricos reales, por ejemplo, la altura de las personas.

El valor de una variable aleatoria puede especificarse mediante un operador de igualdad, por ejemplo,  $X = \text{True}$ . La probabilidad de una variable aleatoria se denota como una función utilizando las mayúsculas  $P$ , por ejemplo,  $P(X)$  es la probabilidad de todos los valores de la variable aleatoria  $X$ . La probabilidad de un valor de una variable aleatoria puede denotarse  $P(X = \text{True})$ , indicando en este caso la probabilidad de que la variable aleatoria  $X$  tenga el valor Verdadero ( $\text{True}$ ).

## Distribución de probabilidad

Una distribución de probabilidad es un resumen de las probabilidades de los posibles valores de una variable aleatoria. Como distribución, la asignación de los valores de una variable aleatoria a una probabilidad tiene una forma cuando todos los valores de la variable aleatoria están alineados.

La distribución también tiene propiedades generales que se pueden medir. Dos propiedades importantes de una distribución de probabilidad son el valor esperado y la varianza. Matemáticamente se denominan primer y segundo momento de la distribución. Otros momentos son la asimetría (tercer momento) y la curtosis (cuarto momento).



Es posible que la media y la varianza ya te resulten familiares en estadística, donde los conceptos se generalizan a distribuciones de variables aleatorias distintas de las distribuciones de probabilidad. Ahora, el valor esperado es el valor medio o promedio de una variable aleatoria  $X$ , este es el valor más probable o el resultado con mayor probabilidad. Se suele denotar como una función de la letra  $E$  en mayúsculas con corchetes, por ejemplo,  $E[X]$  para el valor esperado  $X$ .

Por otro lado, la varianza es la dispersión de los valores de una variable aleatoria con respecto a la media. Se suele denotar como una función  $\text{Var}$ , por ejemplo,  $\text{Var}(X)$  es la varianza de la variable aleatoria  $X$ . La raíz cuadrada de la varianza normaliza el valor y se denomina desviación estándar. La varianza entre dos variables se denomina covarianza y resume la relación lineal de cómo dos variables aleatorias cambian juntas.

Cada variable aleatoria tiene su propia distribución de probabilidad, aunque la distribución de probabilidad de muchas variables aleatorias diferentes puede tener la misma forma. Las distribuciones de probabilidad más comunes pueden definirse utilizando unos pocos parámetros y proporcionan procedimientos para calcular el valor esperado y la varianza. La estructura de la distribución de probabilidad será diferente según la variable aleatoria sea discreta o continua.

## Resumen

En este capítulo, aprendiste sobre las distribuciones de probabilidad. Específicamente, se explicó:

- Las variables aleatorias en probabilidad tienen un dominio definido y pueden ser continuas o discretas.
- Las distribuciones de probabilidad resumen la relación entre los valores posibles y su probabilidad de una variable aleatoria.



# Capítulo 6

## DISTRIBUCIONES DE PROBABILIDAD DISCRETA

---

La probabilidad de una variable aleatoria discreta puede resumirse con una distribución de probabilidad discreta. Las distribuciones de probabilidad discretas se utilizan en Machine Learning, sobretodo en el modelado de problemas de clasificación binaria y multiclase, pero también en la evaluación del rendimiento de los modelos de clasificación binaria, como el cálculo de los intervalos de confianza, y en el modelado de la distribución de palabras en el texto para el Procesamiento del Lenguaje Natural.

El conocimiento de las distribuciones de probabilidad discretas también es necesario en la elección de las funciones de activación en la capa de salida de las redes neuronales de Deep Learning para las tareas de clasificación y la selección de una función de pérdida adecuada.

Las distribuciones de probabilidad discretas desempeñan un papel importante en Machine Learning y hay algunas distribuciones que un profesional debe conocer.

### Definición

Una distribución discreta es una distribución de probabilidad que representa la ocurrencia de resultados discretos (individualmente contables), como 1, 2, 3... o



cero frente a uno. La distribución binomial, por ejemplo, es una distribución discreta que evalúa la probabilidad de que se produzca un resultado “si” o “no” a lo largo de un número determinado de ensayos, dada la probabilidad del evento en cada ensayo, como lanzar una moneda cien veces y que el resultado sea cara.

Los estadísticos pueden identificar el desarrollo de una distribución discreta o continua por la naturaleza de los resultados a medir. A diferencia de la distribución normal, que es continua y da cuenta de cualquier resultado posible a lo largo de la recta numérica, una distribución discreta se construye a partir de datos que solo pueden seguir un conjunto finito o discreto de resultados.

Los dos tipos de variables aleatorias discretas más utilizadas en Machine Learning son:

- Variable aleatoria binaria
- Variable aleatoria categórica

Una variable aleatoria binaria es una variable aleatoria discreta cuyo conjunto finito de resultados está en  $\{0,1\}$ . Una variable aleatoria categórica es una variable aleatoria discreta donde el conjunto finito de resultados está en  $\{1, 2, \dots, K\}$ , donde  $K$  es el número total de resultados únicos. Cada resultado o evento de una variable aleatoria discreta tiene una probabilidad.

Hay muchas distribuciones de probabilidad discreta comunes. Las más comunes son las distribuciones Bernoulli y Multinoulli para variables aleatorias discretas binarias y categóricas, respectivamente, y las distribuciones Binomial y Multinomial que generalizan cada una a múltiples ensayos independientes.

## Distribución Bernoulli

La distribución de Bernoulli es una distribución de probabilidad discreta que cubre un caso en el que un evento tendrá un resultado binario como un 0 o un 1. Piensa en cualquier tipo de experimento que plantea una pregunta de si o no, por ejemplo, ¿está moneda saldrá cara cuando la lance? ¿sacaré un seis con este dado? ¿aprobará el estudiante X su examen de matemáticas?

En los ensayos de Bernoulli, los dos resultados posibles pueden considerarse como “éxito” o “fracaso”, pero estas etiquetas no deben tomarse literalmente. En este contexto, “éxito” significa simplemente obtener un resultado positivo, por ejemplo, cara, sacar un seis, etc.

La distribución de Bernoulli es, esencialmente, un cálculo que permite crear un modelo para el conjunto de posibles resultados de un ensayo de Bernoulli. Por tanto, siempre que tengas un evento que tenga solo dos resultados posibles, la distribución de Bernoulli te permite calcular la probabilidad de cada resultado.



# Distribución binomial

La distribución binomial es una distribución de probabilidad utilizada en estadísticas que resume la probabilidad de que un valor tome uno de los valores independientes bajo un conjunto de parámetros o supuestos dados.

Los supuestos subyacentes de la distribución binomial son que solo hay un resultado para cada ensayo, que cada ensayo tiene la misma probabilidad de éxito y que cada ensayo es mutuamente excluyente o independiente de otro.

Para empezar, el “binomio” en la distribución binomial significa dos términos. No nos interesa solo el número de aciertos, ni solo el número de intentos, sino ambos. Cada uno es inútil para nosotros sin el otro.

La distribución binomial es una distribución discreta habitual en estadística, a diferencia de una distribución continua, como la normal. Esto se debe a que la distribución binomial solo cuenta con dos estados, normalmente representados como 1 (para un éxito) o 0 (para un fracaso) dado un número de ensayos en los datos. Así, la distribución binomial representa la probabilidad de  $x$  éxitos en  $n$  ensayos, dada una probabilidad de éxito  $p$  para cada ensayo.



La distribución binomial resume el número de ensayos u observaciones cuando cada ensayo tiene la misma probabilidad de alcanzar un valor determinado. La distribución binomial determina la probabilidad de observar un número específico de resultado exitosos en un número específico de ensayos.

El rendimiento de un algoritmo de Machine Learning en un problema de clasificación binaria puede analizarse como un proceso de Bernoulli, donde la predicción del modelo sobre un ejemplo de un conjunto de pruebas es un ensayo Bernoulli (correcto o incorrecto). La distribución binomial resume el número de aciertos en un número determinado de ensayos Bernoulli  $k$ , con una probabilidad de éxito dada para cada ensayo  $p$ . Podemos demostrarlo con un proceso Bernoulli en el que la probabilidad de éxito es del 30% o  $P(x = 1) = 0,3$  y el número total de ensayos es 100 ( $k = 100$ ).

Podemos simular el proceso Bernoulli con casos generados aleatoriamente y contar el número de aciertos sobre el número de ensayos dado. Esto se puede conseguir mediante la función NumPy `binomial()`. Esta función toma el número total de ensayos y la probabilidad de éxito como argumentos y devuelve el número de resultados exitosos a través de los ensayos para una simulación.



```
from numpy.random import binomial

#Se define los parámetros de la distribución
p = 0.3
k = 100
exito = binomial(k, p)
print('Éxito total: ', exito)
```

Éxito total: 24

Es de esperar que 24 de cada 100 casos tengan éxito dados los parámetros elegidos. Cada vez que ejecutes el código se obtendrá una secuencia aleatoria diferente de 100 ensayos, por lo que los resultados específicos serán diferentes. Intenta ejecutar el ejemplo varias veces. En este caso, puedes ver que se obtuvo un poco más de 20 ensayos exitosos esperados.

## Diferencia entre la Distribución de Bernoulli y la Distribución binomial

En términos muy simplistas, una distribución de Bernoulli es un tipo de distribución binomial. Sabemos que la distribución de Bernoulli se aplica a sucesos que tienen un ensayo ( $n = 1$ ) y dos resultados posibles, por ejemplo, el lanzamiento de una moneda, eso es el ensayo, y un resultado de cara o cruz. Cuando tenemos más de un ensayo, por ejemplo, lanzamos una moneda cinco veces, la distribución binomial da la distribución de probabilidad discreta del número de éxitos en esa secuencia de lanzamientos o ensayos independientes.

Así que, para continuar con el ejemplo de lanzar una moneda, la distribución de Bernoulli te da la probabilidad de éxito, digamos de salir cara, cuando lanzas la moneda una sola vez, ese es tu ensayo de Bernoulli. Si lanzas la moneda cinco veces, la distribución binomial calculará la probabilidad de éxito, salir cara, en los cinco lanzamientos.

## Distribución categórica

Una distribución categórica es una distribución de probabilidad discreta que describe la probabilidad de que una variable aleatoria tome un valor que pertenezca a una de K categorías, donde cada categoría tiene una probabilidad asociada.

Para una distribución se clasifique como categórica, debe cumplir los siguientes criterios:

- Las categorías son discretas.
- Hay dos o más categorías potenciales.
- La probabilidad de que la variable aleatoria tome un valor en cada categoría debe estar entre 0 y 1.



- La suma de las probabilidades de todas las categorías debe ser igual a 1.

El ejemplo más obvio de una distribución categórica es la distribución de resultados asociada al lanzamiento de un dado, hay  $K = 6$  resultados potenciales y la probabilidad de cada resultado es  $1/6$ , o aproximadamente 0,166 o alrededor del 16,6%.

## Distribución multinomial

La distribución multinomial describe el cálculo de los resultados de experimentos que implican eventos independientes que tienen dos o más resultados posibles y definidos.

La distribución multinomial se aplica a los experimentos en los que se dan las siguientes condiciones:

- El experimento consiste en ensayos repetidos, como lanzar un dado cinco veces en lugar de una sola.
- Cada ensayo debe ser independiente de los demás. Por ejemplo, si se lanzan dos dados, el resultado de uno de ellos no influye en el resultado del otro.
- La probabilidad de cada resultado debe ser la misma en cada caso del experimento. Por ejemplo, si se utiliza un dado justo de seis caras, debe haber una probabilidad de uno entre seis de que se dé cada número en cada tirada.
- Cada prueba debe producir un resultado específico, como un número entre dos y 12 si se lanzan dos dados de seis caras.

Siguiendo con los datos, supongamos que hacemos un experimento en el que lanzamos dos dados 500 veces. Nuestro objetivo es calcular la probabilidad de que el experimento produzca los siguientes resultados en los 500 ensayos:

- El resultado será “2” en el 15% de los ensayos;
- el resultado será “5” en el 12% de los ensayos;
- el resultado será “7” en el 17% de los ensayos; y
- el resultado será “11” en el 20% de los ensayos.

La distribución multinomial nos permitirá calcular la probabilidad de que se produzca la combinación de resultados anterior. Aunque estos números se eligieron de forma arbitraria, el mismo tipo de análisis puede realizarse para experimentos significativos en ciencia, inversión y otras áreas.

Podemos demostrar todo esto con un pequeño ejemplo con 3 categorías ( $K = 3$ ) con igual probabilidad ( $p = 33,33\%$ ) y 100 ensayos. En primer lugar, podemos utilizar la función NumPy `multinomial( )` para simular 100 ensayos independientes y resumir el número de veces que el evento resultó en cada una de las categorías dadas. La



función toma tanto el número de ensayos como las probabilidades de cada categoría como una lista.

```
from numpy.random import multinomial

#Se define los parámetros de la distribución
p = [1.0/3.0, 1.0/3.0, 1.0/3.0]
k = 100
casos = multinomial(k, p)

#Se resumen los casos
for i in range(len(casos)):
    print('Caso', i+1, ':', casos[i])

Caso 1 : 29
Caso 2 : 30
Caso 3 : 41
```

Es de esperar que cada categoría tenga unos 33 eventos. La ejecución del ejemplo muestra cada caso y del número de eventos. Cada vez que se ejecute el código se obtendrá una secuencia aleatoria diferente de 100 pruebas, por lo que los resultados específicos serán diferentes, prueba a ejecutar el ejemplo varias veces. En este caso, puedes ver una dispersión de casos tan alta como 41 y tan baja como 29.

## Distribución de probabilidad discretas en Machine Learning

Las distribuciones de probabilidad, como la distribución de Bernoulli, no solo son útiles para los matemáticos y los estadísticos, sino que también desempeñan un papel crucial en el análisis de datos, la ciencia de datos y Machine Learning. Los analistas y científicos de datos trabajan con grandes volúmenes de datos, y observar la distribución de un conjunto de datos determinado es una parte esencial del análisis exploratorio de datos, es decir, obtener una comprensión inicial de los datos antes de seguir investigando.



En Machine Learning, muchos modelos se basan en suposiciones de distribución, y las distribuciones de probabilidad discretas se utilizan principalmente en el modelado de problemas de clasificación binarios y multiclas.

Algunos ejemplos de modelos de clasificación binaria son los filtros de spam que detectan si un correo electrónico debe clasificarse como “spam” o “no spam”, los modelos que pueden predecir si un cliente realizará una determinada acción o no, o la clasificación de un producto como, por ejemplo, un libro o una película. Un



ejemplo de modelo de clasificación multiclasaría ser un modelo que identifique qué categoría de productos será más relevante para un cliente concreto.

Como una de las distribuciones más sencillas, la distribución Bernoulli suele servir como punto de partida para distribuciones más complejas. Por ejemplo, el proceso de Bernoulli sienta las bases de la distribución binomial, la distribución geométrica y la distribución binomial negativa, que desempeñan un papel crucial en Deep Learning.

Por lo tanto, si quieras profundizar en el análisis de datos, la Ciencia de Datos o Machine Learning, las distribuciones de probabilidad, como la distribución Bernoulli, son un buen punto de partida.

## Resumen

En este capítulo, aprendiste las distribuciones de probabilidad discretas utilizadas en Machine Learning. Específicamente, se explicó:

- La probabilidad de los resultados de las variables aleatorias discretas se puede resumir utilizando distribuciones de probabilidades discretas.
- Un único resultado binario tiene una distribución Bernoulli, y una secuencia de resultados binarios tiene una distribución binomial.
- Un único resultado categórico tiene una distribución categórica y una secuencia de resultados categóricos tiene una distribución multinomial.



# Capítulo 7

## DISTRIBUCIONES DE PROBABILIDAD CONTINUA

---

La probabilidad de una variable aleatoria continua puede resumirse con una distribución de probabilidad continua. Las distribuciones de probabilidad continuas se encuentran en Machine Learning, sobretodo en la distribución de las variables numéricas de entrada y salida de los modelos y en la distribución de los errores cometidos por los modelos.

El conocimiento de la distribución de probabilidad continua normal también se requiere de forma más general en la estimación de la densidad y de los parámetros realizada por muchos modelos de Machine Learning. Como tal, las distribuciones de probabilidad continuas juegan un papel importante en Machine Learning y hay algunas distribuciones que un profesional debe conocer.

### Definición

Una variable aleatoria es una cantidad producida por un proceso aleatorio. Una variable aleatoria continua es una variable aleatoria que tiene un valor numérico real. A cada resultado numérico de una variable aleatoria continua se le puede asignar una probabilidad.



La relación entre los sucesos de una variable aleatoria continua y sus probabilidades se denomina distribución de probabilidad continua y se resume mediante una función de densidad de probabilidad.

Hay muchas distribuciones de probabilidad continuas comunes. La más común es la distribución de probabilidad normal. Prácticamente todas las distribuciones de probabilidad continuas de interés pertenecen a la llamada familia de distribuciones exponenciales, que no son más que una colección de distribuciones de probabilidad parametrizadas, por ejemplo, distribuciones que cambian en función de los valores de los parámetros.

Las distribuciones de probabilidad continuas desempeñan un papel importante en Machine Learning, desde la distribución de las variables de entrada a los modelos, la distribución de los errores cometidos por los modelos, y en los propios modelos al estimar el mapeo entre entradas y salidas.

## Distribución normal

La distribución normal, también conocida como distribución gaussiana, es una distribución de probabilidad simétrica con respecto a la media, que muestra que los datos cercanos a la media son más frecuentes que los datos alejados de la misma.

La distribución cubre la probabilidad de eventos de valor normal de muchos dominios de problemas diferentes, lo que la convierte en una distribución común y conocida, de ahí el nombre de normal. Una variable aleatoria continua que tiene una distribución normal se dice que es normal o que tiene una distribución normal. Algunos ejemplos de dominios que tienen eventos normalmente distribuidos incluyen: la altura de las personas, el peso, las puntuaciones de un examen.

La distribución normal tiene varias características y propiedades clave que la definen.

En primer lugar, su media (promedio), su mediana (punto medio) y su moda (observación más frecuente) son iguales entre sí. Además, todos estos valores representan el pico, o punto más alto, de la distribución. La distribución cae simétricamente alrededor de la media, cuya anchura viene definida por la desviación estándar.

Una distribución normal con una media de cero y una desviación estándar de 1 se denomina distribución normal estándar, y a menudo los datos se reducen o estandarizan a ésta para su análisis con el fin de facilitar su interpretación y comparación. Podemos conseguirlo utilizando la función NumPy `normal()`, pero también con la función `norm()` de SciPy, con esta última se puede obtener la función de densidad de probabilidad.



Se puede comprobar si una muestra de datos es normal representándola y comprobando la conocida forma normal, o utilizando pruebas estadísticas. Si las muestras de observaciones de una variable aleatoria se distribuyen normalmente, entonces pueden resumirse solo con la media y la varianza, calculadas directamente sobre las muestras.

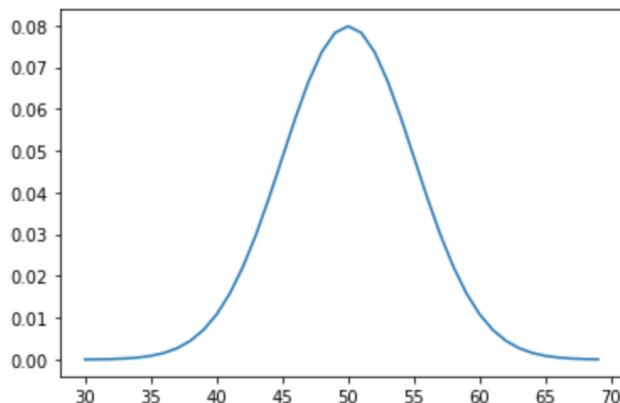
Puedes calcular la probabilidad de cada observación mediante la función de densidad de probabilidad. Un gráfico de estos valores nos daría la reveladora forma de campana.

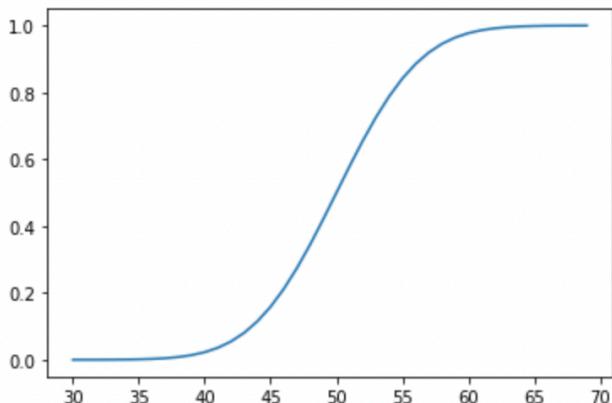
```
from scipy.stats import norm
from matplotlib import pyplot

#Se define los parámetros de la distribución
mu = 50
sigma = 5
dist = norm(mu, sigma)

#Se grafica la distribución
valores = [valor for valor in range(30, 70)]
probabilidad = [dist.pdf(valor) for valor in valores]
pyplot.plot(valores, probabilidad)
pyplot.show()

#Se grafica la función de densidad de probabilidad
cprobs = [dist.cdf(valor) for valor in valores]
pyplot.plot(valores, cprobs)
pyplot.show()
```





Al ejecutar el ejemplo, primero se calcula la probabilidad para los enteros en el rango  $[30, 70]$  y se crea un gráfico de líneas de valores y probabilidades. El gráfico muestra la forma gaussiana o de campana con el pico de mayor probabilidad alrededor del valor esperado o la media de 50 con una probabilidad de alrededor del 8%.

A continuación, se calculan las probabilidades acumulativas de las observaciones en el mismo intervalo, lo que muestra que en la media hemos cubierto alrededor del 50% de los valores esperados y muy cerca del 100% después del valor de aproximadamente 65 o 3 desviaciones estándar de la media ( $50 + (3 \times 5)$ ).

De hecho, la distribución normal tiene una heurística o regla empírica que define el porcentaje de datos cubiertos por un rango dado por el número de desviaciones estándar de la media. Se llama regla del 68-95-99,7, que es el porcentaje aproximado de los datos cubiertos por los rangos definidos por 1, 2 y 3 desviaciones estándar de la media.

Por ejemplo, en nuestra distribución con una media de 50 y una desviación estándar de 5, esperas que el 95% de los datos estuvieron cubiertos por valores que se encuentran a 2 desviaciones estándar de la media, o  $50 - (2 \times 5)$  y  $50 + (2 \times 5)$  o entre 40 y 60. Podemos confirmarlo calculando los valores exactos mediante la función de punto porcentual. El 95% medio estaría definido por el valor de la función de punto porcentual para el 2,5% en el extremo inferior y el 97,5% en el extremo superior, donde 97,5 a 2,5 da el 95% medio.

## Distribución exponencial

La distribución exponencial es una distribución de probabilidad continua que suele referirse a la cantidad de tiempo que transcurre hasta que se produce algún acontecimiento específico. Se trata de un proceso en el que los sucesos se producen de forma continua e independiente a una velocidad media constante.



La distribución exponencial tiene la propiedad clave de no tener memoria. La variable aleatoria exponencial puede tener más valores pequeños o menos variables grandes.

En la teoría de la probabilidad y la estadística, la distribución exponencial es una distribución de probabilidad continua que suele referirse a la cantidad de tiempo que transcurre hasta que se produce algún acontecimiento específico. Se trata de un proceso en el que los sucesos se producen de forma continua e independiente a una velocidad media constante. La distribución exponencial tiene la propiedad clave de no tener memoria.

La variable aleatoria exponencial puede tener más valores pequeños o menos variables grandes. Por ejemplo, la cantidad de dinero que gasta el cliente en un viaje al supermercado sigue una distribución exponencial.

La distribución exponencial puede definirse mediante un parámetro:

→ Escala (Beta o  $\beta$ ): la media y la desviación estándar de la distribución.

A veces la distribución se define más formalmente con un parámetro lambda o tasa. El parámetro beta se define recíproco del parámetro lambda ( $\beta = \frac{1}{\lambda}$ )

→ Tasa (lambda o  $\lambda$ ): tasa de cambio de la distribución.

Podemos definir una distribución con una media de 50 y muestrear números aleatorios de esta distribución. Podemos conseguirlo utilizando la función NumPy `exponential()`. De igual forma, se puede definir una distribución exponencial utilizando la función `expon()` de SciPy y luego calcular propiedades. El siguiente ejemplo define un rango de observaciones entre 50 y 70 y calcula la probabilidad y la probabilidad acumulada para cada una y traza el resultado.

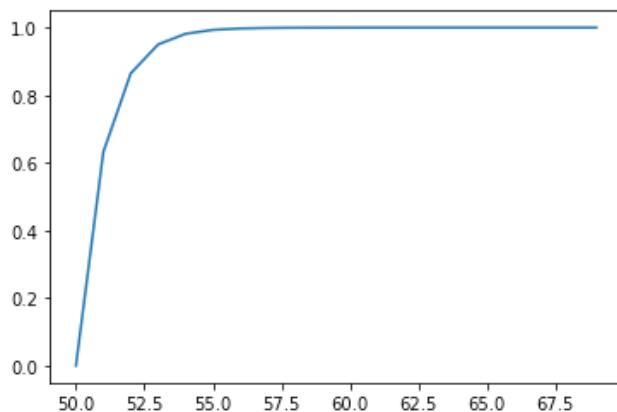
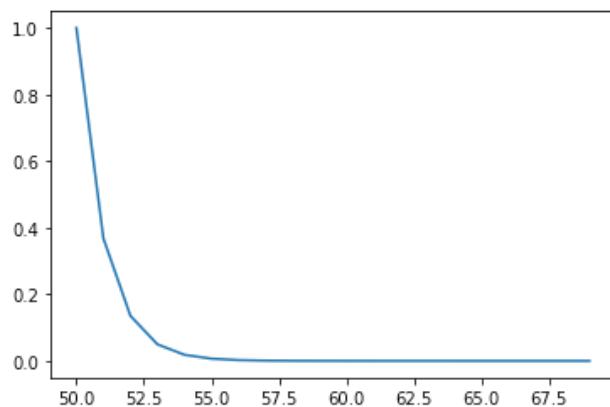
```
from scipy.stats import expon
from matplotlib import pyplot

#Se define los parámetros de la distribución
beta = 50
dist = expon(beta)

#Se grafica la distribución
valores = [valor for valor in range(50, 70)]
probabilidad = [dist.pdf(valor) for valor in valores]
pyplot.plot(valores, probabilidad)
pyplot.show()

#Se grafica la función de distribución acumulativa
cprobs = [dist.cdf(valor) for valor in valores]
pyplot.plot(valores, cprobs)
pyplot.show()
```





Al ejecutar las líneas de código se crea primero un gráfico de líneas de los resultados frente a las probabilidades, mostrando una forma de distribución de probabilidad exponencial familiar. A continuación, se calcula las probabilidades acumulativas de cada resultado y se grafican como un gráfico de líneas, mostrando que después de quizás un valor de 55 se observará casi el 100% de los valores esperados.

## Distribución de Pareto

La distribución de Pareto debe su nombre al economista y sociólogo italiano Vilfredo Pareto. A veces se denomina Principio de Pareto o Regla del 80-20.

Esta distribución para las variables aleatorias continuas que siguen una distribución Pareto, en la que el 80% de los sucesos están cubiertos por el 20% del rango de resultados, es decir, la mayoría de los sucesos se extraen de solo el 20% del rango de resultados, es decir, la mayoría de los sucesos se extraen de solo el 20% del rango de la variable continua.

La distribución puede definirse mediante un parámetro:

- Forma ( $\alpha$  o  $\alpha$ ): la inclinación de la caída de la probabilidad.



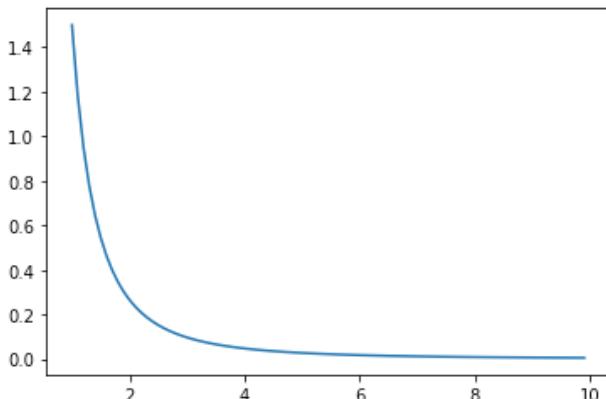
Los valores del parámetro de forma suelen ser pequeños, entre 1 y 3, y el principio de Pareto se da cuando alfa se fija en 1,161. Podemos definir una distribución con una forma de 1,1 y muestrear números aleatorios de esta distribución. Podemos conseguirlo utilizando la función NumPy `pareto()`. También puedes definir la distribución de Pareto utilizando la función `pareto()` de SciPy y luego calcular sus propiedades. El siguiente ejemplo define un rango de observaciones entre 1 y alrededor de 10 y calcula la probabilidad y la probabilidad acumulada para cada una y traza el resultado.

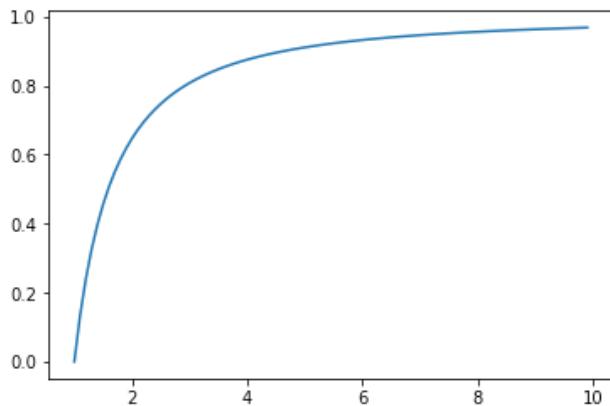
```
from scipy.stats import pareto
from matplotlib import pyplot

#Se define los parámetros de la distribución
alpha = 1.5
dist = pareto(alpha)

#Se grafica la distribución
valores = [valor/10.0 for valor in range(10, 100)]
probabilidad = [dist.pdf(valor) for valor in valores]
pyplot.plot(valores, probabilidad)
pyplot.show()

#Se grafica la función de densidad de probabilidad
cprobs = [dist.cdf(valor) for valor in valores]
pyplot.plot(valores, cprobs)
pyplot.show()
```





Al ejecutar las líneas de código se crea primero un gráfico de líneas de los resultados frente a las probabilidades, que muestra la forma familiar de la distribución de probabilidad de Pareto. A continuación, se calcula las probabilidades acumuladas de cada resultado y se grafican como un gráfico de líneas, mostrando un aumento que es menos pronunciado que la distribución exponencial, explicada anteriormente.

## Resumen

En este capítulo, aprendiste las distribuciones de probabilidad continua utilizadas para Machine Learning. Específicamente, se explicó:

- La probabilidad de los resultados de las variables aleatorias continuas se puede resumir utilizando distribuciones de probabilidad continuas.
- Cómo parametrizar, definir y muestrear aleatoriamente a partir de distribuciones de probabilidad continuas comunes.
- Cómo crear gráficos de densidad de probabilidad y densidad acumulativa para distribuciones de probabilidad continuas comunes.

# Capítulo 8

## ESTIMACIÓN DE LA DENSIDAD DE PROBABILIDAD

---

La densidad de probabilidad es la relación entre las observaciones y su probabilidad. Algunos resultados de una variable aleatoria tendrán una densidad de probabilidad baja y otros resultados tendrán una densidad de probabilidad alta. La forma general de la densidad de probabilidad se denomina distribución de probabilidad, y el cálculo de las probabilidades de resultados específicos de una variable aleatoria se realiza mediante una función de densidad de probabilidad.

Conocer la función de densidad de probabilidad de una muestra de datos es útil para saber si una determinada observación es improbable, o tan improbable como para considerarla un valor atípico o una anomalía, y si debe eliminarse. También es útil para elegir los métodos de aprendizaje adecuados que requieren que los datos de entrada tengan una distribución de probabilidad específica. Es poco probable que se conozca la función de densidad de probabilidad una muestra aleatoria de datos. Por ello, la densidad de probabilidad debe aproximarse mediante un proceso conocido como estimación de la densidad de probabilidad.

### Definición

La densidad de probabilidad o simplemente densidad, es la relación entre los resultados de una variable aleatoria y su probabilidad. En caso de que una variable



aleatoria es continua, la probabilidad puede calcularse a través de la función de densidad de probabilidad. La forma de la función de densidad de probabilidad en el dominio de una variable aleatoria se denomina distribución de probabilidad y las distribuciones de probabilidad comunes tienen nombres como uniforme, normal, exponencial, entre otras.

Conocer la distribución de probabilidad de una variable aleatoria puede ayudar a calcular los momentos de la distribución, como la media y la varianza, pero también puede ser útil para otras consideraciones más generales, como determinar si una observación es improbable o muy improbable y podría ser un valor atípico o una anomalía.

El problema es que es muy común no conocer la distribución de probabilidad de una variable aleatoria, ya que no tenemos acceso a todos los resultados posibles de una variable aleatoria. De hecho, lo único a lo que tenemos acceso es a una muestra de observaciones. Por tanto, debemos seleccionar una distribución de probabilidad.

Este problema se denomina estimación de la densidad de probabilidad, o simplemente estimación de la densidad, ya que estamos utilizando las observaciones de una muestra aleatoria para estimar la densidad general de las probabilidades más allá de la muestra de datos que tenemos disponible.

El proceso de estimación de la densidad de una variable aleatoria consta de varios pasos. El primer paso es revisar la densidad de las observaciones en la muestra aleatoria con un simple histograma. A partir del histograma, es posible que se pueda identificar una distribución de probabilidad común y bien entendida que pueda utilizarse, como una distribución normal. Si no es así, puede que se tenga que ajustar un modelo para estimar la distribución.

## La densidad con un histograma

El primer paso en la estimación de la densidad es crear un histograma de las observaciones de la muestra aleatoria.



Un histograma es un gráfico que consiste en agrupar primero las observaciones en intervalos y contar el número de eventos que caen en cada intervalo. Los recuentos, o las frecuencias de las observaciones, en cada intervalo se trazan como un gráfico de barras con los intervalos en el eje X y la frecuencia en el eje Y.

La elección del número de contenedores es importante, ya que controla el grado de grosor de la distribución, número de barras y, a su vez, el grado de representación de la densidad de las observaciones.



Es una buena idea experimentar con diferentes tamaños de contenedores para una muestra de datos determinada para obtener múltiples perspectivas o puntos de vista sobre los mismos datos.

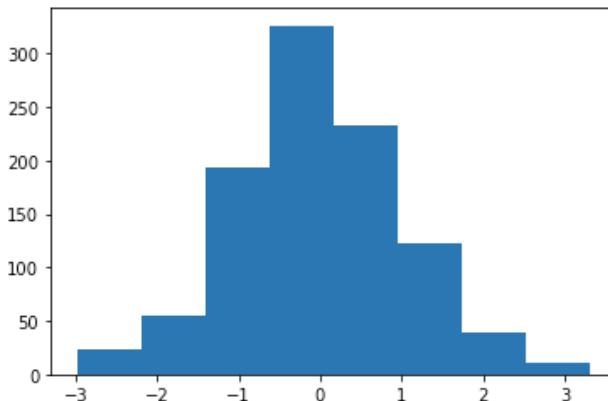
Se puede crear un histograma utilizando la librería Matplotlib y la función `hist()`. Los datos se proporcionan como primer argumento, y el número de contenedores se especifica a través del argumento `bins`, ya sea como un número entero o como una secuencia de los límites de cada contenedor.

Se puede crear una muestra aleatoria extraída de una distribución normal y fingir que no se conoce la distribución, para luego crear un histograma de los datos. La función NumPy `normal()` logrará esto y se generan 1000 muestras con una media de 0 y una desviación estándar de 1.

```
from matplotlib import pyplot
from numpy.random import normal

#Se genera una muestra
muestra = normal(size=1000)

#Se grafica el histograma de la muestra
pyplot.hist(muestra, bins=8)
pyplot.show()
```



Al ejecutar las líneas de código se extrae una muestra de observaciones aleatorias y se crea el histograma con 8 intervalos. Puedes ver claramente la forma de la distribución normal. Ten en cuenta que los resultados serán diferentes dada la naturaleza aleatoria de la muestra datos. Puedes intentar ejecutar el ejemplo varias veces, inclusive cambiar el número de intervalos.

Revisar un histograma de una muestra de datos con un rango de diferentes números de intervalos ayudará a identificar si la densidad se parece a una distribución de probabilidad común o no.

En la mayoría de los casos, verás una distribución unimodal, como la conocida forma de campana de la normal, la forma plana de la uniforme, o la forma



descendente o ascendente de una distribución exponencial o de Pareto. También puedes ver distribuciones complejas, como dos picos que no desaparecen con diferentes números de intervalos, lo que se conoce como una distribución bimodal, o múltiples picos, lo que se conoce como una distribución multimodal. También puedes ver un gran pico en la densidad para un valor dado o un pequeño rango de valores que indican valores atípicos, que a menudo se producen en la cola de una distribución lejos del resto de la densidad.

## Estimación de la densidad paramétrica

La forma de un histograma de la mayoría de las muestras aleatorias coincidirá con una distribución de probabilidad bien conocida, por esta razón es importante que te familiarices con las distribuciones de probabilidad comunes, ya que te ayudará a identificar una determinada distribución a partir de un histograma.

Una vez identificada, puedes intentar estimar la densidad de la variable aleatoria con una distribución de probabilidad elegida. Esto se puede conseguir estimando los parámetros de la distribución a partir de una muestra aleatoria de datos.

Por ejemplo, si se tiene una distribución normal, esta cuenta con dos parámetros: la media y la desviación estándar. Con estos dos parámetros, ahora se conoce la función de distribución de la probabilidad. Este proceso se denomina estimación de la densidad paramétrica.



La razón es que estamos utilizando funciones predefinidas para resumir la relación entre las observaciones, y su probabilidad que pueden ser controladas o configuradas con parámetros, de ahí que sea paramétrica.

Una vez que se ha estimado la densidad, se puede comprobar si es un buen ajuste. Esto se puede hacer de muchas maneras como, por ejemplo:

- Trazando la función de densidad y comparando la forma con el histograma.
- Muestreando la función de densidad y comparando la muestra generada con la muestra real.
- Utilizando una prueba estadística para confirmar que los datos se ajustan a la distribución.

Se puede generar una muestra aleatoria de 1000 observaciones de una distribución normal con una media de 50 y una desviación estándar de 5. Sabiendo que tiene una distribución normal, se calcula los parámetros de la distribución, la media y la desviación típica. A continuación, se ajusta la distribución con estos parámetros, la llamada estimación de la densidad paramétrica de la muestra de datos. En este caso, se puede utilizar la función *norm( )* de SciPy. A continuación, se puede



muestrear las probabilidades de esta distribución para un rango de valores en el dominio, en este caso entre 30 y 70. Por último, se puede trazar un histograma de la muestra de datos y superponer un gráfico de líneas de las probabilidades calculadas para el rango de valores de la densidad de la probabilidad.

Es importante que convirtamos los recuentos o las frecuencias de cada casilla del histograma en una probabilidad normalizada para garantizar que el eje Y del histograma coincida con el eje Y del gráfico de líneas. Esto puede lograrse estableciendo el argumento de la densidad a True en la llamada `hist()`.

Veamos esto aplicado en un ejemplo con Python.

```
from matplotlib import pyplot
from numpy.random import normal
from numpy import mean
from numpy import std
from scipy.stats import norm

#Se genera una muestra
muestra = normal(loc=50, scale=5, size=1000)

#Se calcula los parámetros
media = mean(muestra)
desviacion = std(muestra)
print('Media=', media)
print('Desviación Estándar=', desviacion)

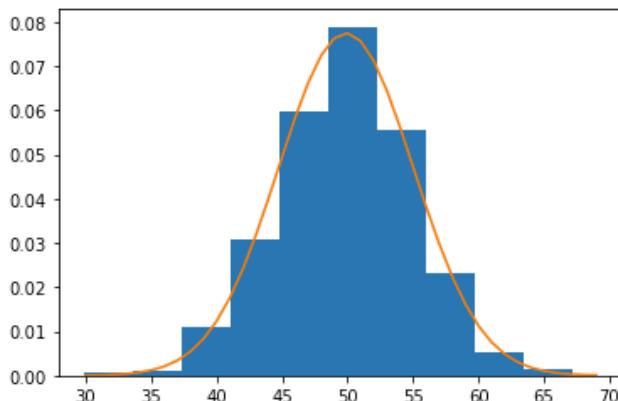
#Definir la distribución
dist = norm(media, desviacion)

#Probabilidad de la muestra
valores = [valor for valor in range(30, 70)]
probabilidad = [dist.pdf(valor) for valor in valores]

#Se grafica el histograma de la muestra
pyplot.hist(muestra, bins=10, density=True)
pyplot.plot(valores, probabilidad)
pyplot.show()

Media= 49.881918811149774
Desviación Estándar= 5.14965306828093
```





Ejecutando las líneas de código anteriores se genera primero la muestra de datos y luego estima los parámetros de la distribución de probabilidad normal. En este caso, puedes ver que la media y la desviación estándar son ligeramente diferentes de los valores esperados de 50 y 5, respectivamente. Seguidamente se ajusta la estimación de la densidad paramétrica utilizando los parámetros estimados y se compara el histograma de los datos con 10 intervalos con las probabilidades para un rango de valores muestreados.

Toma en cuenta que, probablemente, tus resultados sean diferentes por la naturaleza aleatoria de la muestra de datos.

Es posible que los datos se ajusten a una distribución de probabilidad común, pero que requieran una transformación antes de la estimación de la densidad paramétrica. Por ejemplo, puedes tener valores atípicos que estén lejos de la media o del centro de masa de la distribución. Esto puede tener el efecto de dar estimaciones incorrectas de los parámetros de la distribución y, a su vez, provocar un mal ajuste de los datos. Estos valores atípicos deben eliminarse antes de estimar los parámetros de la distribución.

Otro ejemplo es que los datos pueden tener un sesgo o estar desplazados a la izquierda o a la derecha. En este caso, es posible que tengas que transformar los datos antes de estimar los parámetros.

Estos tipos de modificaciones de los datos pueden no ser obvios y la estimación efectiva de la densidad paramétrica puede requerir un proceso iterativo de hasta que el ajuste de la distribución a los datos sea lo suficientemente bueno.

## Estimación de la densidad no paramétrica

En algunos casos, una muestra de datos puede no parecerse a una distribución de probabilidad común o no puede hacerse fácilmente que se ajuste a la distribución. Esto suele ocurrir cuando los datos tienen dos picos, distribución bimodal, o muchos picos, distribución multimodal. En este caso, la estimación de la densidad



paramétrica no es factible y se tienen que utilizar métodos alternativos que no utilicen una distribución común.

Para estos casos, se utiliza un algoritmo para aproximar la distribución de probabilidad de los datos sin una distribución definida, lo que se denomina método no paramétrico. Las distribuciones seguirán teniendo parámetros, pero no son directamente controlables de la misma manera que las distribuciones de probabilidad simples.

El enfoque no paramétrico más común para estimar la función de densidad de probabilidad de una variable aleatoria continua es la estimación de densidad de kernel. Este es un método no paramétrico para utilizar un conjunto de datos para estimar las probabilidades de nuevos puntos. En este caso, el kernel es una función matemática que devuelve una probabilidad para un valor dado de una variable aleatoria.



El kernel suaviza o interpola las probabilidades en el rango de resultados de una variable aleatoria, de forma que la suma de las probabilidades sea igual a uno, un requisito de las probabilidades bien manejadas.

La función kernel pondera la contribución de las observaciones de una muestra de datos en función de su relación o distancia con una muestra de consulta determinada para la que se solicita la probabilidad. Un parámetro, denominado parámetro de suavización o ancho de banda, controla el alcance, o la ventana de observaciones, de la muestra de datos que contribuye a estimar la probabilidad para una muestra determinada.

El siguiente parámetro es el de suavización (ancho de banda), este controla el número de muestras o ventana de muestras utilizadas para estimar la probabilidad de un nuevo punto. Una ventana grande puede dar lugar a una densidad gruesa con pocos detalles, mientras que una ventana pequeña puede tener demasiados detalles y no ser lo suficientemente suave o general para cubrir correctamente los ejemplos nuevos o no vistos. La contribución de las muestras dentro de la ventana puede modelarse utilizando diferentes funciones, a veces denominadas funciones de base con diferentes efectos sobre la suavidad de la función de densidad resultante.

El siguiente parámetro es la función base (kernel), esta es la función elegida para controlar la contribución de las muestras del conjunto de datos a la estimación de la probabilidad de un nuevo punto.

Puede ser útil experimentar con diferentes tamaños de venta y diferentes funciones de contribución y evaluar los resultados contra los histogramas de los datos.

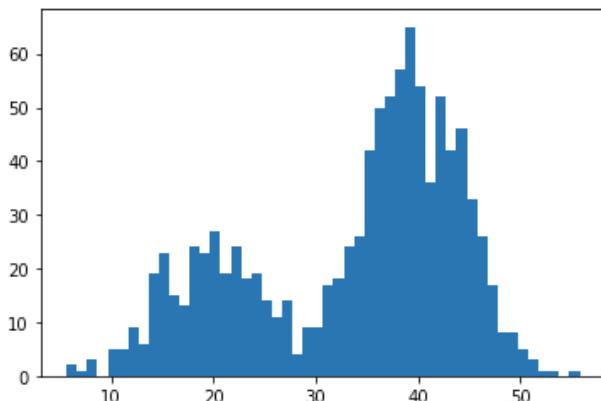


Veamos esto en un ejemplo, se puede construir una distribución bimodal combinando muestras de dos distribuciones normales diferentes. En concreto, 300 ejemplos con una media de 20 y una desviación estándar de 5, el pico más grande. Las medias se eligieron muy próximas entre sí para garantizar que las distribuciones se solapen en la muestra combinada. Este es el código para la creación de esta muestra con una distribución bimodal y el trazado del histograma.

```
from matplotlib import pyplot
from numpy.random import normal
from numpy import hstack

#Se genera una muestra
muestra1 = normal(loc=20, scale=5, size=300)
muestra2 = normal(loc=40, scale=5, size=700)
muestra = hstack((muestra1, muestra2))

#Se grafica el histograma de la muestra
pyplot.hist(muestra, bins=50)
pyplot.show()
```



Al ejecutar las líneas de código del ejemplo crea la muestra de datos y traza el histograma. Observando la gráfica se tiene menos muestras con una media de 20 que muestras con una media de 40. Los datos con esta distribución no encajan bien en una distribución de probabilidad común, por diseño. Este es un buen caso para utilizar un método no paramétrico de estimación de la densidad del kernel.

Toma en cuenta que, probablemente, tus resultados sean diferentes por la naturaleza aleatoria de la muestra de datos.

Scikit Learn proporciona la función *KernelDensity* que implementa la estimación de la densidad del núcleo. En primer lugar, esta función se construye con el ancho de banda (*bandwidth*) deseado y los argumentos del kernel. Es buena idea probar diferentes configuraciones según los datos. La clase se ajusta a una muestra de datos mediante la función *fit( )*. Seguidamente se puede evaluarlo bien que la estimación de la densidad se ajusta a los datos calculando las probabilidades para un rango de observaciones y comparando la forma con el histograma. La función *score\_samples( )* de *KernelDensity* calculará la probabilidad logarítmica para un



conjunto de muestras. Se puede crear un rango de muestras de 1 a 60, aproximadamente el rango del dominio, calcular las probabilidades logarítmicas, y luego invertir la operación logarítmica calculando el exponente o `exp( )` para devolver los valores al rango 0-1 para las probabilidades normales. Veamos todo este ejemplo junto al histograma con frecuencias normalizadas y un gráfico de líneas superpuestas de valores a probabilidades estimadas.

```

from matplotlib import pyplot
from numpy.random import normal
from numpy import hstack
from numpy import asarray
from numpy import exp
from sklearn.neighbors import KernelDensity

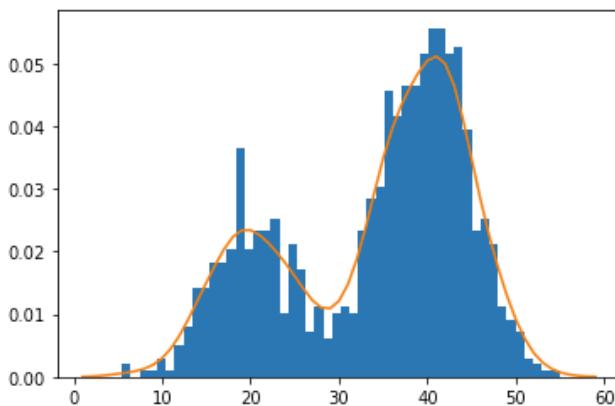
#Se genera una muestra
muestral = normal(loc=20, scale=5, size=300)
muestra2 = normal(loc=40, scale=5, size=700)
muestra = hstack((muestral, muestra2))

#Se entrena la densidad
modelo = KernelDensity(bandwidth=2, kernel='gaussian')
muestra = muestra.reshape((len(muestra), 1))
modelo.fit(muestra)

#Probabilidad de la muestra
valores = asarray([valor for valor in range(1, 60)])
valores = valores.reshape((len(valores), 1))
probabilidad = modelo.score_samples(valores)
probabilidad = exp(probabilidad)

#Se grafica el histograma de la muestra
pyplot.hist(muestra, bins=50, density=True)
pyplot.plot(valores[:,], probabilidad)
pyplot.show()

```



Al ejecutar las líneas de código se crea la distribución de datos, se ajusta el modelo de estimación de la densidad del núcleo, y luego se traza el histograma de la muestra de datos y la estimación de la densidad paramétrica. En este caso se puede ver que la estimación de la densidad paramétrica se ajusta bien al histograma. No es muy sueve y podría serlo más ajustando el argumento del ancho de banda a 3



muestras o más. Puedes experimentar con diferentes valores del ancho de banda y de la función kernel.

Toma en cuenta que, probablemente, tus resultados sean diferentes por la naturaleza aleatoria de la muestra de datos.

## Resumen

En este capítulo, aprendiste una introducción a la estimación de la densidad de probabilidad. Específicamente, se explicó:

- Los gráficos de histograma proporcionan una forma rápida y fiable de visualizar la densidad de probabilidad de una muestra de datos.
- La estimación paramétrica de la densidad de probabilidad implica la selección una distribución común y la estimación de los parámetros de la función de densidad a partir de una muestra de datos.
- La estimación no paramétrica de la densidad de probabilidad implica el uso de una técnica para ajustar un modelo a la distribución arbitraria de los datos, como la estimación de la densidad del kernel.



# Capítulo 9

## TEOREMA DE BAYES

---

El Teorema de Bayes proporciona una forma de calcular una probabilidad condicional basada en principios. Se trata de un cálculo aparentemente sencillo, que proporciona un método fácil de calcular para escenarios en los que la intuición suele fallar.

La mejor manera de desarrollar una intuición para el Teorema de Bayes es pensar en el significado de los términos de la ecuación y aplicar el cálculo muchas veces en una serie de escenarios diferentes del mundo real. Esto proporcionará el contexto de lo que se está calculando y ejemplos que se pueden utilizar como punto de partida al aplicar el cálculo en nuevos escenarios en el futuro.

### Definición

El Teorema de Bayes, llamado así por el matemático británico del siglo XVIII Thomas Bayes, es una fórmula matemática para determinar la probabilidad condicional. La probabilidad condicional es la probabilidad de que se produzca un resultado, basándose en que un resultado anterior se haya producido en circunstancias similares. El Teorema de Bayes permite revisar las predicciones o teorías existentes a partir de pruebas nuevas o adicionales.





El teorema de Bayes se basa en la incorporación de distribuciones de probabilidad a priori para generar probabilidades a posteriori.

La probabilidad a priori,  $P(A)$ , en la inferencia estadística bayesiana, es la probabilidad de que se produzca un evento antes de que se recojan nuevos datos. En otras palabras, representa la mejor evaluación racional de la probabilidad de un resultado concreto basada en los conocimientos actuales antes de realizar un experimento.

La probabilidad posterior,  $P(A|B)$ , es la probabilidad revisada de que se produzca un suceso tras tener en cuenta la nueva información. La probabilidad posterior se calcula actualizando la probabilidad anterior mediante el teorema de Bayes. En términos estadísticos, la probabilidad posterior es la probabilidad de que se produzca el suceso A si se ha producido el suceso B.

La fórmula del teorema de Bayes es la siguiente:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}$$

donde:

$P(A)$ : la probabilidad de que ocurra A

$P(B)$ : la probabilidad de que ocurra B

$P(A|B)$ : la probabilidad de A dada B

$P(B|A)$ : la probabilidad de B dada A

$P(A \cap B)$ : la probabilidad de que ocurran tanto A como B

Consideremos un ejemplo en el que recibimos un correo electrónico y el detector de spam lo pone en la carpeta de spam, ¿cuál es la probabilidad de que sea spam? Supongamos algunos detalles, como que el 2% del correo electrónico que recibimos es spam  $P(A)$ . Supongamos que el detector de spam es realmente bueno y cuando un correo electrónico es spam lo detecta  $P(B|A)$  con una precisión del 99%, y cuando un correo electrónico no es spam, lo marcará como spam con una tasa muy baja del 0,1%  $P(B|\neg A)$ .

Si se introduce esta información a la fórmula del teorema de Bayes, se tiene:

$$P(\text{spam}|\text{detectado}) = \frac{P(\text{detectado}|\text{spam}) \times P(\text{spam})}{P(\text{detectado})} = \frac{0,99 \times 0,02}{P(\text{detectado})}$$

No se cuenta con el valor de  $P(\text{detectado})$  o  $P(B)$ , pero se puede calcular de la siguiente forma:



$$P(B) = P(B|A)x P(A) + P(B|no A) x P(no A)$$

$$P(\text{detectado}) = P(\text{detectado|spam}) \times P(\text{spam}) + P(\text{detectado|no spam}) \times P(\text{no spam})$$

donde:

$$P(\text{detectado|no spam}) = 0,1\%$$

$$P(\text{no spam}) = 1 - P(\text{spam}) = 1 - 0,02 = 0,98$$

Con esta información se puede calcular  $P(\text{detectado})$

$$P(\text{detectado}) = 0,99 \times 0,02 + 0,001 \times 0,98 = 0,02078$$

Es decir, alrededor del 2% de todo el correo electrónico se detecta como spam, independientemente de que lo sea o no. Ahora se calcula la respuesta a nuestra pregunta original:

$$P(\text{spam|detectado}) = \frac{0,99 \times 0,02}{0,02078} = 0,9528$$

Esto quiere decir, si un correo electrónico está en la carpeta de spam, hay un 95,2% de probabilidades de que sea efectivamente spam.

## Teorema de Bayes y Machine Learning

El teorema de Bayes proporciona un método útil para pensar en la relación entre un conjunto de datos y una probabilidad. En otras palabras, el teorema dice que la probabilidad de que una determinada hipótesis sea cierta a partir de unos datos específicos observados puede enunciarse como el hallazgo de la probabilidad de observar los datos dada la hipótesis multiplicada por la probabilidad de que la hipótesis sea cierta independiente de los datos, dividida por la probabilidad de observar los datos independientes de la hipótesis.

De igual forma, Machine Learning puede entenderse utilizando el Teorema de Bayes como la selección del modelo que mejor describe los datos observados. Esta perspectiva proporciona la base del marco probabilístico de máximo a posteriori que puede demostrarse que subyace en muchos modelos de Machine Learning, como la Regresión Lineal y la Regresión Logística. Este marco de Machine Learning también da lugar a la noción de cómo hacer predicciones óptimas para un nuevo ejemplo utilizando un clasificado óptimo de Bayes, que a su vez expone el error de Bayes, el error mínimo esperado al hacer predicciones sobre un conjunto de datos.

## Teorema de Bayes de las hipótesis de modelización

El teorema de Bayes proporciona una forma de pensar en la relación entre los datos y un modelo. Un algoritmo o modelo de Machine Learning es una forma específica de pensar en las relaciones estructuradas de los datos. De este modo, un modelo



puede considerarse como una hipótesis sobre las relaciones en los datos, como la relación entre la entrada (X) y la salida (y).

La práctica de Machine Learning aplicado consiste en probar y analizar diferentes hipótesis (modelos) sobre un conjunto de datos determinado. El teorema de Bayes proporciona un modelo probabilístico para describir la relación proporcional entre los datos (D) y una hipótesis (h):

$$P(h|D) = \frac{P(D|h) \times P(h)}{P(D)}$$

Si se desglosa esto, dice que la probabilidad de que una hipótesis determinada se mantenga o sea verdadera dados unos datos observados puede calcularse como la probabilidad de observar los datos dada la hipótesis multiplicada por la probabilidad de que la hipótesis sea verdadera independientemente de los datos, dividida por la probabilidad de observar los datos independientemente de la hipótesis.

De acuerdo a esto, cada pieza del cálculo tiene un nombre específico:

- $P(h|D)$ : probabilidad posterior de la hipótesis, lo que queremos calcular.
- $P(h)$ : probabilidad previa de la hipótesis.

Esto proporciona un marco útil para pensar y modelar un problema de Machine Learning. Si tienes algún conocimiento previo del dominio sobre la hipótesis, éste se recoge en la probabilidad previa. Si no se tiene, todas las hipótesis, pueden tener la misma probabilidad previa. Si la probabilidad de observar los datos  $P(D)$  aumenta, entonces la probabilidad de que la hipótesis se mantenga dados los datos  $P(h|D)$  disminuye. A la inversa, si la probabilidad de la hipótesis  $P(h)$  y la probabilidad de observar los datos dada la hipótesis aumentan, la probabilidad de que la hipótesis se mantenga dados los datos  $P(h|D)$  aumenta.

El objetivo es localizar una hipótesis que explique mejor los datos observados. Esta simplificación proporciona la base de un procedimiento de optimización para la búsqueda de un modelo y un conjunto de parámetros que se ajusten mejor a los datos, lo que se conoce generalmente como estimación de la densidad.

Un problema común de modelización consiste en cómo estimar una distribución de probabilidad conjunta para un conjunto de datos. Por ejemplo, dada una muestra de observaciones (X) de un dominio, donde cada observación se extrae de forma independiente del dominio con la misma distribución de probabilidad. La estimación de la densidad consiste en seleccionar una función de distribución de probabilidad y los parámetros de esa distribución que mejor expliquen la distribución de probabilidad conjunta de los datos observados (X). A menudo, la estimación de la densidad es demasiado difícil, en su lugar, se conforme con una estimación puntual de la distribución objetivo, como la media.



Hay muchas técnicas para resolver este problema, una de ellas es la máxima a posteriori, un método bayesiano, que será explicado en el siguiente capítulo.

## Resumen

En este capítulo, aprendiste cómo calcular el teorema de Bayes y una perspectiva bayesiana del modelado y la predicción de Machine Learning. Específicamente, se explicó:

- El teorema de Bayes es una técnica para calcular una probabilidad condicional.
- Cómo la selección de un modelo de Machine Learning puede enmarcarse en la resolución del teorema de Bayes.



# Capítulo 10

## ESTIMACIÓN

---

La estimación de la densidad es el problema de estimar la distribución de probabilidad de una muestra de observaciones de un dominio de problemas. Hay muchas técnicas para resolver la estimación de la densidad, aunque un marco común utilizado en todo el campo de Machine Learning es la estimación de máxima verosimilitud. La estimación de máxima verosimilitud implica definir una función de verosimilitud para calcular la probabilidad condicional de observar la muestra de datos dada una distribución de probabilidad y unos parámetros de distribución.

Este enfoque puede utilizarse para buscar un espacio de posibles distribuciones y parámetros. Este marco probabilístico flexible también constituye la base de muchos algoritmos de Machine Learning, incluidos métodos importantes como la Regresión Lineal y la Regresión Logística para predecir valores numéricos y etiquetas de clase, respectivamente, pero también, de forma más general, para las redes neuronales artificiales de Deep Learning.

### Problema de la estimación de la densidad de la probabilidad

Un problema común de modelización consiste en cómo estimar una distribución de probabilidad conjunta para un conjunto de datos. La estimación de la densidad consiste en seleccionar una función de distribución de probabilidad y los parámetros



de esa distribución que mejor expliquen la distribución de probabilidad conjunta de los datos observados.

Este problema se hace más difícil si la muestra ( $X$ ) extraída de la población es pequeña y tiene ruido, lo que significa que cualquier evaluación de una función de densidad de probabilidad estimada y sus parámetros tendrá algún error.

Cada una de las funciones de densidad tienen su respectivo significado y, en consecuencia, un conjunto de técnicas de estimación diferentes, dos enfoques comunes son:

- Estimación de máxima verosimilitud (EMV)
- Máxima a posteriori (MAP)

## Estimación de máxima verosimilitud

Una solución a la estimación de la densidad de probabilidad se denomina estimación de máxima verosimilitud, esta implica tratar el problema como un problema de optimización o de búsqueda, en el que buscamos un conjunto de parámetros que dé como resultado el mejor ajuste para la probabilidad conjunta de la muestra de datos ( $X$ ).

En primer lugar, se trata de definir un parámetro llamado theta ( $\theta$ ) que define tanto la elección de la función de densidad de probabilidad como los parámetros de esa distribución. Puede ser un vector de valores numéricos cuyos valores cambian suavemente y se asigna a diferentes distribuciones de probabilidad y sus parámetros.

En la estimación de máxima verosimilitud, deseamos maximizar la probabilidad de observar los datos de la distribución de probabilidad conjunta dada una distribución de probabilidad específica y sus parámetros, enunciada formalmente como:

$$P(X|\theta)$$

Esta probabilidad condicional se expresa a menudo utilizando la notación de punto y coma (;) en lugar de la notación de barra (|) porque  $\theta$  no es una variable aleatoria, sino un parámetro desconocido. La probabilidad condicional resultante se denomina probabilidad de observar los datos dados los parámetros del modelo y se escribe utilizando la notación  $L( )$  para denotar la función de probabilidad.

El objetivo de la estimación de máxima verosimilitud es encontrar el conjunto de parámetros ( $\theta$ ) que maximicen la función de verosimilitud, es decir, que den lugar al mayor valor de verosimilitud.

$$\max L(X; \theta)$$



La distribución de probabilidad conjunto puede replantearse como la multiplicación de la probabilidad condicional de observar cada ejemplo dados los parámetros de la distribución.

$$\prod_{i=1}^n P(x_i; \theta)$$

La multiplicación de muchas probabilidades pequeñas puedes ser numéricamente inestable en la práctica, por lo tanto, es común replantear este problema como la suma de las probabilidades condicionales logarítmicas de observar cada ejemplo dados los parámetros del modelo.

$$\sum_{i=1}^n \log P(x_i; \theta)$$

Dado el uso frecuente del logaritmo en la función de verosimilitud, se suele denominar función de log-verosimilitud. Es común en los problemas de optimización preferir minimizar la función de coste, en lugar de maximizarla. Por lo tanto, se utiliza el negativo de la función de logaritmo-verosimilitud, que se denomina generalmente función de logaritmo-verosimilitud negativa.

$$\min - \sum_{i=1}^n \log P(x_i; \theta)$$

## Estimación de máxima verosimilitud y Machine Learning

Se puede enmarcar el problema de ajustar un modelo de Machine Learning como el problema de la estimación de la densidad de probabilidad. En concreto, la elección del modelo y de los parámetros del modelo se denomina hipótesis de modelización, y el problema consiste en encontrar una hipótesis ( $h$ ) que explique mejor los datos  $X$ .

$$\max \sum_{i=1}^n \log P(x_i; h)$$

Esto proporciona la base para estimar la densidad de probabilidad de un conjunto de datos, que se suele utilizar en algoritmos de Aprendizaje no Supervisado, como lo son los algoritmos de clustering.

El marco de estimación de máxima verosimilitud es también una herramienta útil para el Aprendizaje Supervisado. Se aplica a los datos en los que tenemos variables



de entrada y de salida, donde la variable de salida puede ser un valor numérico o una etiqueta de clase en el caso del modelado predictivo de regresión y clasificación a posteriori. Se puede plantear como la probabilidad condicional de la salida ( $y$ ) dada la entrada ( $X$ ) a partir de la hipótesis de modelización ( $h$ ).

$$\max \sum_{i=1}^n \log P(y_i|x_i; h)$$

Esto significa que el mismo marco de estimación de máxima verosimilitud que se utiliza generalmente para la estimación de la densidad puede utilizarse para encontrar un modelo de Aprendizaje Supervisado y sus parámetros. Esto proporciona la base para las técnicas fundamentales de modelado lineal.

En el caso de la Regresión Lineal, el modelo está restringido a una línea e implica encontrar un conjunto de coeficientes para la línea que mejor se ajuste a los datos observados. Afortunadamente, este problema puede resolverse analíticamente, por ejemplo, utilizando directamente el álgebra lineal.

En el caso de la Regresión Logística, el modelo define una línea e implica encontrar un conjunto de coeficientes para la línea que mejor separa las clases. Esto no puede resolverse analíticamente y a menudo se resuelve utilizando en el espacio de los posibles valores de los coeficientes utilizando un algoritmo de optimización.

Ambos métodos también pueden resolverse de forma menos eficiente utilizando un algoritmo de optimización más general, como el descenso de gradiente estocástico.



La mayoría de los modelos de Machine Learning pueden enmarcarse en el marco de la estimación de máxima verosimilitud, lo que proporciona una forma útil y coherente de abordar el modelado predictivo como un problema de optimización.

Una ventaja importante del estimador de máxima verosimilitud en Machine Learning es que, a medida que aumenta el tamaño del conjunto de datos, la calidad del estimados sigue mejorando.

## Máxima a posteriori

Si se recuerda el teorema de Bayes, este proporciona una manera de calcular una probabilidad condicional. Se trata de calcular la probabilidad condicional de un resultado dado otro resultado sin utilizar la probabilidad conjunta de los resultados, y se expresa de la siguiente forma:



$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

La cantidad que estamos calculando se suele denominar probabilidad posterior de A dada B y  $P(A)$  se denomina probabilidad a priori de A.

Se puede eliminar la constante de normalización de  $P(B)$  y demostrar que la probabilidad posterior es proporcional a  $\propto$  la probabilidad de B dada A multiplicada por la probabilidad a priori. Por tanto, para nuestros fines, podemos simplificar el cálculo como:

$$P(A|B) = P(B|A) \times P(A)$$

Esta es una simplificación útil ya no se interesa estimar una probabilidad, sino en optimizar una cantidad. Una cantidad proporcional es suficiente para este propósito. Ahora podemos relacionar este cálculo con nuestro deseo de estimar una distribución y unos parámetros ( $\theta$ ) que expliquen mejor el conjunto de datos (X). Esto se puede plantear como:

$$P(\theta|X) = P(X|\theta) \times P(\theta)$$

La maximización de esta cantidad en un rango de  $\theta$  resuelve un problema de optimización para estimar la tendencia central de la probabilidad posterior, por ejemplo, el modelo de la distribución. Como tal, esta técnica se denomina estimación máxima a posteriori, o estimación MAP para abreviar, y a veces simplemente estimación máxima a posterior.

$$\max P(X|\theta) \times P(\theta)$$

Observa que esto es muy similar a la estimación de máxima verosimilitud, con la adición de la probabilidad a priori sobre la distribución y los parámetros. De hecho, si se asume que todos los valores de  $\theta$  son igualmente probables porque no tenemos ninguna información previa, por ejemplo, una prioridad uniforme, entonces ambos cálculos son equivalentes. Debido a esta equivalencia, tanto la estimación de máxima verosimilitud como la máxima a posteriori suelen converger al mismo problema de optimización para muchos algoritmos de Machine Learning.

## Máxima a posteriori y Machine Learning

En Machine Learning, la optimización máxima a posteriori proporciona un marco probabilístico bayesiano para el ajuste de los parámetros del modelo a los datos de entrenamiento y una alternativa y hermano del marco de estimación de máxima verosimilitud, quizás más común.

Un marco no es mejor que otro y, en muchos casos, ambos marcos enmarcan el mismo problema de optimización desde perspectivas diferentes.





La máxima a posteriori es apropiado para aquellos problemas en los que existe cierta información previa, por ejemplo, cuando se puede establecer una prioridad significativa para ponderar la elección de diferentes distribuciones y parámetros o parámetros del modelo.

La estimación de máxima verosimilitud es más apropiada cuando no existe tal información previa.

De hecho, la adición de la hipótesis a priori a la estimación de máxima verosimilitud puede considerarse como un tipo de regularización del cálculo de la estimación de máxima verosimilitud. Esta idea permite que otros métodos de regularización, por ejemplo, la norma L2 en modelos que utilizan una suma ponderada de entradas, se interpreten en el marco de la inferencia bayesiana de máxima a posteriori. Por ejemplo, L2 es un sesgo o una prioridad que supone que un conjunto de coeficientes o pesos tiene un valor de suma cuadrada pequeño.

Al igual que la estimación de máxima verosimilitud, la resolución del problema de optimización depende de la elección del modelo. Para los modelos más sencillos, como la Regresión Lineal, existen soluciones analíticas. Para modelos más complejos, como la Regresión Logística, se requiere una optimización numérica que haga uso de las derivadas de primer y segundo orden. Para los problemas más complicados, pueden ser necesarios algoritmos de optimización estocástica.

## Resumen

En este capítulo, aprendiste una introducción a la estimación, considerando dos enfoques. Específicamente, se explicó:

- La estimación de máxima verosimilitud para resolver el problema de la estimación de la densidad.
- Cómo se puede utilizar el marco probabilístico de máxima a posteriori para optimizar un modelo para un conjunto de datos.



# Capítulo 11

## ENTROPÍA DE LA INFORMACIÓN

---

La entropía de la información da una poderosa perspectiva sobre cómo pasa la información entre los experimentos, y resulta importante en ciertos algoritmos de Machine Learning.

Por su parte, la entropía cruzada se utiliza habitualmente en Machine Learning como función de pérdida. Esta es una medida del campo de la teoría de la información que se basa en la entropía y que generalmente calcula la diferencia entre dos distribuciones de probabilidad. Está estrechamente relacionada con la divergencia de Kullback-Leibler, pero es diferente de ésta, que calcula la entropía relativa entre dos distribuciones de probabilidad, mientras que la entropía cruzada puede considerarse que calcula la entropía total entre las distribuciones.

La entropía cruzada también está relaciona y a menudo se confunde con la pérdida logística, llamada pérdida logarítmica. Aunque las dos medidas se derivan de una fuente diferente, cuando se utilizan como funciones de pérdida para los modelos de clasificación, ambas medidas calculan la misma cantidad y pueden utilizarse indistintamente.



# Teoría de la información

La teoría de la información es un campo de estudio que se ocupa de cuantificar la información para la comunicación. Es un subcampo de las matemáticas y se ocupa de temas como la compresión de datos y los límites de procesamiento de señales.

La entropía del conjunto X se define como:

$$H(X) = \sum_x P(x) \log_2 \frac{1}{P(x)}$$

Las unidades de entropía son los bits y se considera las siguientes propiedades de la entropía de la información:

$$H(X) \geq 0$$

con igualdad si y solo si  $P(X) = 1$  para exactamente un  $X$ , intuitivamente, esto significa que cuando solo uno de los elementos del conjunto se conoce absolutamente, es decir, con  $P(X) = 1$ , la incertidumbre se colapsa a cero.

Observa también que la entropía se maximiza cuando P se distribuye uniformemente entre los elementos del conjunto. En otras palabras, la entropía de la información se maximiza cuando las dos alternativas en conflicto son igualmente probables.

Lo más importante es que el concepto de entropía se extiende de forma conjunta como sigue:

$$H(X, Y) = \sum_x P(x, y) \log_2 \frac{1}{P(x, y)}$$

Si y solo si X e Y son independientes, la entropía se vuelve aditiva:

$$H(X, Y) = H(X) + H(Y)$$

## Divergencia de Kullback-Leibler

Las nociones de entropía de la información conducen a nociones de distancia entre distribuciones de probabilidad que serán importantes para los métodos de Machine Learning. La divergencia de Kullback-Leibler entre dos distribuciones de probabilidad P y Q definidas sobre el mismo conjunto se define como:

$$D_{KL}(P||Q) = \sum_x P(x) \log_2 \frac{P(x)}{Q(x)}$$



Donde el operador  $\|$  indica divergencia o P divergencia de Q. Observa que  $D_{KL}(P||Q) \geq 0$  con igualdad si y solo si  $P = Q$ .

La intuición de la puntuación de la divergencia KL es que cuando la probabilidad de un evento de P es grande, pero la probabilidad del mismo evento en Q es pequeña, hay una gran divergencia. Cuando la probabilidad de P es pequeña y la de Q es grande, también hay una gran divergencia, pero no tan grande como el primer caso. Puede utilizarse para medir la divergencia entre distribuciones de probabilidad discretas o continuas, donde en este último caso se calcula la integral de los sucesos en lugar de la suma de las probabilidades de los sucesos discretos.

Cuando la puntuación es 0, sugiere que ambas distribuciones son idénticas, de lo contrario la puntuación es positiva. Es importante destacar que la puntuación de la divergencia KL no es simétrica, por ejemplo:

$$KL(P||Q) \neq KL(Q||P)$$

Recibe el nombre de los dos autores del método, Solomon Kullback y Richard Leibler, y a veces se denomina entropía relativa.

Si se está intentando aproximar una distribución de probabilidad desconocida, entonces la distribución de probabilidad objetivo de los datos es P y Q es nuestra aproximación de la distribución. En este caso, la divergencia KL resume el número de bits adicionales, es decir, calculados con el logaritmo de base 2, necesarios para representar un evento de la variable aleatoria. Cuanto mejor sea nuestra aproximación, menos información adicional será necesaria.

## Entropía cruzada

La entropía cruzada es una medida de la diferencia entre dos distribuciones de probabilidad para una variable aleatoria determinada o un conjunto de eventos. En concreto, se basa en la idea de entropía de la teoría de la información y calcula el número medio de bits necesarios para representar o transmitir un evento de una distribución en comparación con la otra distribución.

La intuición para esta definición viene si se considera una distribución de probabilidad objetivo o subyacente P y una aproximación de la distribución objetivo Q, entonces la entropía cruzada de Q a partir de P es el número de bits adicionales para representar un evento usando Q en lugar de P. La entropía cruzada entre dos distribuciones de probabilidad, como Q a partir de P, se puede enunciar formalmente como:

$$H(P, Q)$$

Donde:



$H( )$  es la función de entropía cruzada

P puede ser la distribución objetivo

Q es la aproximación de la distribución objetivo

La entropía cruzada puede calcularse utilizando las probabilidades de los eventos de P y Q, como sigue:

$$H(P, Q) = - \sum_x P(x) \log_2(Q(x))$$

Donde:

$P(x)$  es la probabilidad del suceso x en P.

$Q(x)$  es la probabilidad del suceso x en Q.

Este cálculo es para distribuciones de probabilidad discretas, aunque se puede utilizar un cálculo similar para distribuciones de probabilidad continuas utilizando la integral a través de los eventos en lugar de la suma. El resultado será un número positivo medido en bits y será igual a la entropía de la distribución si las dos distribuciones de probabilidad son idénticas.

## Diferencia entre entropía cruzada y divergencia KL

La entropía cruzada no es la divergencia KL. La entropía está relacionada con las medidas de divergencia, como la divergencia de Kullback Leibler, o KL, que cuantifica cuánto difiere una distribución de otra.

En concreto, la divergencia KL mide una cantidad similar a la entropía cruzada. Mide el número medio de bits adicionales necesarios para representar un mensaje con Q en lugar de P, no el número total de bits.

Por ello, la divergencia KL suele denominarse entropía relativa.

- Entropía cruzada: se refiere al número de bits totales para representar un evento de Q en lugar de P.
- Entropía relativa o divergencia KL: es el número medio de bits adicionales para representar un evento de Q en lugar de P.



Tanto la entropía cruzada como la divergencia de KL calculan la misma cantidad cuando se utilizan como funciones de pérdida para optimizar un modelo predictivo de clasificación. Es en este contexto en el que a veces se puede ver que la entropía cruzada y la divergencia de KL son lo mismo.



## Resumen

En este capítulo, aprendiste una introducción a la entropía de la información. Específicamente, se explicó:

- La teoría de la información se ocupa de la compresión y transmisión de datos y se basa en la probabilidad y apoya a Machine Learning.
- La información proporciona una forma de cuantificar la cantidad de sorpresa de un evento medido en bits.
- La entropía proporciona una medida de la cantidad media de información necesaria para representar un evento extraído de una distribución de probabilidad para una variable aleatoria.



# Capítulo 12

## MÉTRICAS DE PUNTUACIÓN DE PROBABILIDADES

---

La predicción de probabilidades en lugar de etiquetas de clase para un problema de clasificación puede proporcionar un matiz adicional e incertidumbre para las predicciones. El matiz añadido permite utilizar métricas más sofisticadas para interpretar y evaluar las probabilidades predichas.

En general, los métodos para la evaluación de la precisión de las probabilidades predichas se denominan reglas de puntuación o funciones de puntuación.

### Puntuación de la pérdida logarítmica

La pérdida logarítmica, también llamada pérdida logística o entropía cruzada, puede utilizarse como medida para evaluar las probabilidades predichas. Cada probabilidad predicha se compara con el valor real de salida de la clase (0 ó 1) y se calcula una puntuación que penaliza la probabilidad en función de la distancia al valor esperado. La penalización es logarítmica, ofreciendo una puntuación pequeña para diferencias pequeñas (0,1 ó 0,2) y enorme para diferencias grandes (0,9 ó 1,0). Un modelo con una habilidad perfecta tiene una puntuación de pérdida logarítmica de 0,0.



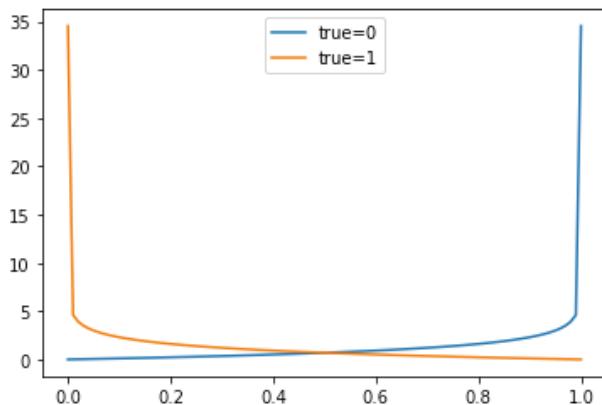
Para resumir la destreza de un modelo utilizando la pérdida logarítmica, se calcula la pérdida logarítmica para cada probabilidad predicha y se informa de la pérdida media.

La pérdida logarítmica puede implementarse en Python utilizando la función `log_loss()` de Scikit Learn. Veamos un ejemplo, en el caso de la clasificación binaria, la función toma una lista de valores de resultados verdaderos y una lista de probabilidades como argumentos y calcula la pérdida logarítmica media para las predicciones. Dado un resultado específico conocido de 0, se puede predecir valores de 0,0 a 1,0 en incrementos de 0,01 (101 predicciones) y calcular la pérdida logarítmica para cada uno. El resultado es una curva que muestra cuánto se penaliza cada predicción a medida que la probabilidad se aleja del valor esperado. Se puede repetir esto para un resultado conocido de 1 y ver la misma curva a la inversa. Implementemos esto en Python.

```
from sklearn.metrics import log_loss
from matplotlib import pyplot

#Predicciones de 0 a 1 en incrementos de 0,01
yhat = [x*0.01 for x in range(0, 101)]

#Evaluar las predicciones para un valor True 0
perdidas_0 = [log_loss([0], [x], labels=[0,1]) for x in yhat]
perdidas_1 = [log_loss([1], [x], labels=[0,1]) for x in yhat]
pyplot.plot(yhat, perdidas_0, label='true=0')
pyplot.plot(yhat, perdidas_1, label='true=1')
pyplot.legend()
pyplot.show()
```



Al ejecutar las líneas de código se crea un gráfico de líneas que muestra las puntuaciones de pérdida para las predicciones de probabilidad de 0,0 a 1,0 tanto para el caso en el que la etiqueta verdadera es 0 como para el caso en el que es 1. Esto ayuda a construir una intuición para el efecto que tiene la puntuación de pérdida cuando se evalúan las predicciones.

La habilidad del modelo se presenta como la pérdida logarítmica media de las predicciones en un conjunto de datos de prueba. Como media, se puede esperar



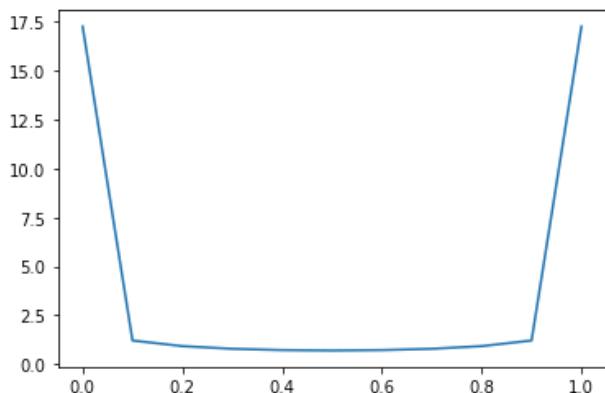
que la puntuación sea adecuada con un conjunto de datos equilibrado y engañosa cuando hay un gran desequilibrio entre las dos clases en el conjunto de pruebas. Esto se debe a que la predicción de 0 o de pequeñas probabilidades dará lugar a una pequeña pérdida. Se puede demostrar comparando la distribución de los valores de pérdida al predecir diferentes probabilidades constantes para un conjunto de datos equilibrado y otro desequilibrado. El siguiente ejemplo predice valores de 0,0 a 1,0 en incrementos de 0,1 para un conjunto de datos equilibrados de 50 ejemplos de clase 0 y 1.

```
from sklearn.metrics import log_loss
from matplotlib import pyplot

#Definir un conjunto de datos desbalanceado
testy = [0 for x in range(50)] + [1 for x in range(50)]

#Pérdida para la predicción de diferentes valores de probabilidad fijos
predicciones = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
perdidas = [log_loss(testy, [y for x in range(len(testy))]) for y in predicciones]

#Grafica las predicciones y las pérdidas
pyplot.plot(predicciones, perdidas)
pyplot.show()
```



Si se ejecuta el código, se puede ver que un modelo predice mejor los valores de probabilidad que no son agudos, cerca del borde, y que están de vuelta hacia el centro de la distribución. La penalización de equivocarse con una probabilidad elevada es muy grande.

## Puntuación de Brier

La puntuación de Brier, llamada así por Glenn Brier, calcula el error cuadrático medio entre las probabilidades predichas y los valores esperados. La puntuación resume la magnitud del error en las predicciones de probabilidad.

La puntuación del error está siempre entre 0,0 y 1,0 donde un modelo con una habilidad perfecta tiene una puntuación de 0,0. Las predicciones que se alejan más de la probabilidad esperada se penalizan, pero de forma menos severa como en el



caso de la pérdida logarítmica. La habilidad de un modelo puede resumir como la puntuación media de Brier de todas las probabilidades predichas para un conjunto de datos de prueba.

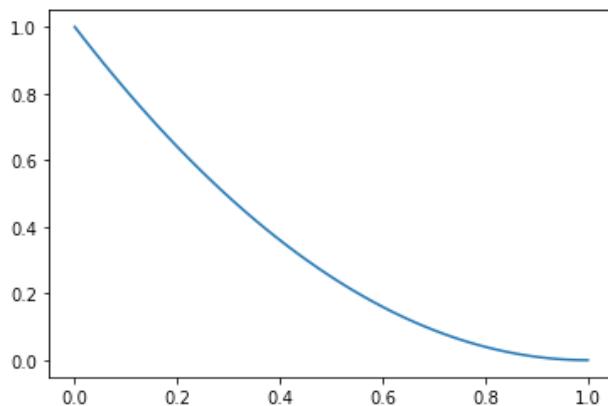
La puntuación Brier puede calcularse en Python utilizando la función `brier_score_loss()` de Scikit Learn. Toma los valores de clase (0,1) y las probabilidades predichas para todos los ejemplos de un conjunto de datos de prueba como argumentos y devuelve la puntuación media de Brier. Se puede evaluar el impacto de los errores de predicción comparando la puntuación de Brier para las predicciones de probabilidad única en un error creciente de 0,0 a 1,0. A continuación se muestra un ejemplo completo utilizando Python.

```
from sklearn.metrics import brier_score_loss
from matplotlib import pyplot

#Predicciones de 0 a 1 en incrementos de 0,01
yhat = [x*0.01 for x in range(0, 101)]

#Evaluar las predicciones para un valor True de 1
perdidas = [brier_score_loss([1], [x], pos_label=[1]) for x in yhat]

#Graficar la entrada a la pérdida
pyplot.plot(yhat, perdidas)
pyplot.show()
```



La habilidad del modelo se presenta como la media de Brier en las predicciones de un conjunto de datos de prueba. Al igual que con la pérdida logarítmica, se puede esperar que la puntuación sea adecuada con un conjunto de datos equilibrado y engañosa cuando haya un gran desequilibrio entre las dos clases en el conjunto de prueba. Se puede demostrar comparando la distribución de los valores de pérdida al predecir diferentes probabilidades constantes para un conjunto de datos equilibrado y otro desequilibrado. En primer lugar, el ejemplo siguiente predice valores de 0,0 a 1,0 en incrementos de 0,1 para un conjunto de datos equilibrado de 50 ejemplos de clase 0 y 1.



```

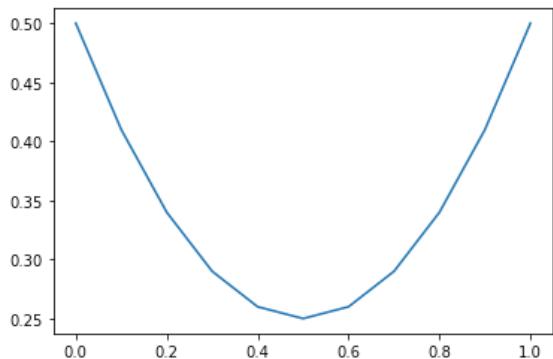
from sklearn.metrics import brier_score_loss
from matplotlib import pyplot

#Definir un conjunto de datos desbalanceado
testy = [0 for x in range(50)] + [1 for x in range(50)]

#Puntuación de brier para la predicción de diferentes valores de probabilidad fija
predicciones = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
perdidas = [brier_score_loss(testy, [y for x in range(len(testy))]) for y in predicciones]

#Grafica las predicciones y las pérdidas
pyplot.plot(predicciones, perdidas)
pyplot.show()

```



Al ejecutar las líneas de código del ejemplo, se puede observar que un modelo es mejor para predecir valores de probabilidades intermedias como 0,5. A diferencia de la pérdida logarítmica, que es bastante plana para las probabilidades cercanas, la forma parabólica muestra el claro aumento cuadrático de la penalización de la puntuación a medida que aumenta el error.

## Predictión de probabilidades

En un problema de clasificación, se puede decidir predecir los valores de la clase directamente. Pero también puede ser más flexible predecir las probabilidades de cada clase. La razón de esto es proporcionar la capacidad de elegir e incluso calibrar el umbral de cómo interpretar las probabilidades predichas.

Por ejemplo, por defecto se podría utilizar un umbral de 0,5, lo que significa que una probabilidad en [0,0, 0,49] es un resultado negativo (0) y una probabilidad en [0,5, 1,0] es un resultado positivo (1). Este umbral puede ajustarse para afinar el comportamiento del modelo para un problema específico. Un ejemplo sería reducir más uno u otro tipo de error.

Al hacer una predicción para un problema de clasificación binaria o de dos clases, hay dos tipos de errores que se podría cometer:

- Falso positivo: predecir un evento cuando no lo hubo.
- Falso negativo: predecir que no hay ningún evento cuando en realidad sí lo hubo.



Al predecir las probabilidades y calibrar un umbral, el desarrollador del modelo puede elegir un equilibrio entre estas dos preocupaciones. Por ejemplo, en un sistema de predicción de lluvias, es más preocupante tener pocos falsos negativos que pocos falsos positivos. Un falso negativo significaría no avisar de un día de lluvia cuando en realidad es un día lluvioso, lo que provocaría muchos inconvenientes a la población que no puede tomar precauciones. Un falso positivo significa que el público tomaría medidas de precaución cuando no es necesario. Una forma habitual de comparar modelos que predicen probabilidades para problemas de dos clases es utilizar una curva ROC.

## Curvas ROC AUC

Una herramienta útil a la hora de predecir la probabilidad de un resultado binario es la curva ROC (Receiver Operating Characteristic). Se trata de un gráfico de la tasa de falsos positivos frente a la tasa de verdaderos positivos para un número de diferentes valores de umbral candidatos entre 0,0 y 1,0. Dicho de otro modo, representa la tasa de falsas alarmas frente a la tasa de aciertos. La tasa de verdaderos positivos se calcula como el número de verdaderos positivos dividido por la suma del número de verdaderos positivos y el número de falsos negativos. Describe lo bueno que es el modelo para predecir la clase positiva cuando el resultado real es positivo. La tasa de verdaderos positivos también se denomina sensibilidad:

$$Sensibilidad = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos}$$

La tasa de falsos positivos se calcula como el número de falsos positivos dividido por la suma del número de falsos positivos y el número de verdaderos negativos. También se denomina tasa de falsa alarma, ya que resume la frecuencia con la que se predice una clase positiva cuando el resultado real es negativo. La tasa de falsos positivos también se denomina especificidad invertida, donde la especificidad es el número total de verdaderos negativos dividido por la suma del número de verdaderos negativos y falsos positivos.

$$Especificidad = \frac{Verdaderos\ negativos}{Verdaderos\ negativos + Falsos\ positivos}$$

Donde:

$$Tasa\ de\ falsos\ positivos = 1 - Especificidad$$

La curva ROC es una herramienta útil por varias razones:

- Las curvas de diferentes modelos pueden compararse directamente en general o para diferentes umbrales.



- El área bajo la curva (AUC) puede utilizarse como resumen de la habilidad del modelo.

La forma de la curva contiene mucha información, incluyendo lo más importante para un problema, la tasa de falsos positivos esperada y la tasa de falsos negativos. Para que quede claro:

- Los valores más pequeños en el eje X indican menos falsos positivos y más verdaderos negativos.
- Los valores más grandes en el eje Y indican mayores verdaderos positivos y menores falsos negativos.



Si se te dificulta entender esto, recuerda que cuando predecimos un resultado binario, es una predicción correcta (verdadero positivo) o no (falso positivo).

Existe una tensión entre estas opciones, lo mismo que con los verdaderos negativos y los falsos negativos. Un modelo hábil asignará una mayor probabilidad a un resultado verdadero positivo elegido al azar que a un resultado negativo, por término medio. A esto se refiere cuando se dice que el modelo tiene habilidad. Por lo general, los modelos hábiles se representan con curvas que se inclinan hacia la parte superior izquierda del gráfico.

Un modelo sin habilidad se representa en el punto (0,5, 0,5). Un modelo sin habilidad en cada umbral está representado por una línea diagonal desde la parte inferior izquierda del gráfico hasta la parte superior izquierda y luego cruza la parte superior hasta la parte superior derecha. Un desarrollador puede trazar la curva ROC para el modelo final y elegir un umbral que ofrezca un equilibrio deseable entre los falsos positivos y los falsos negativos.

Se puede trazar una curva ROC para un modelo en Python utilizando la función `roc_curve()` de Scikit Learn. La función toma tanto los resultados verdaderos (0,1) del conjunto de pruebas como las probabilidades predichas para la clase 1, la función devuelve las tasas de falsos positivos y verdaderos positivos para cada umbral.

Por otra parte, el AUC para el ROC puede calcularse utilizando la función `roc_auc_score()`. Al igual que la función `roc_curve()`, la función AUC toma tanto los resultados verdaderos (0,1), devuelve la puntuación AUC entre 0,1 y 1,0 para la ausencia de habilidad y la habilidad perfecta, respectivamente.

A continuación, se muestra un ejemplo completo de cálculo de la curva ROC y de la curva ROC AUC para un modelo de Regresión Logística en un pequeño problema de prueba.



```

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

#Se genera un conjunto de datos de 2 clases
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)

#Se divide el conjunto en entrenamiento y en prueba
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
ns_probs = [0 for _ in range(len(testy))]

#Se genera el modelo y se entrena
modelo = LogisticRegression(solver='lbfgs')
modelo.fit(trainX, trainy)

#Se predice las probabilidades
lr_probs = modelo.predict_proba(testX)

#Se mantiene las probabilidades sólo para el resultado positivo
lr_probs = lr_probs[:, 1]

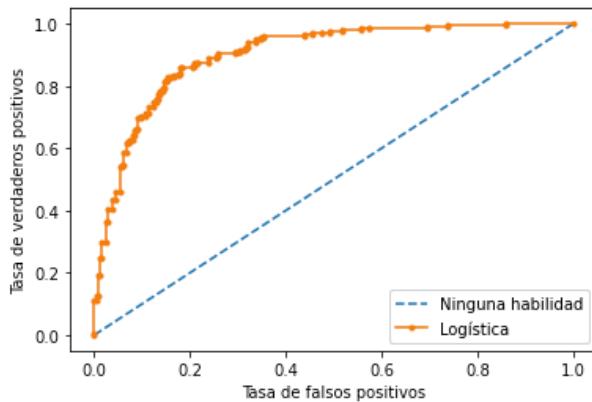
#Se calcula el puntaje
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
print('No hay habilidad: ROC AUC =', ns_auc)
print('Logística: ROC AUC =', lr_auc)

#Se calcula la curva ROC AUC
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)

#Se gráfica la curva ROC AUC del modelo
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='Ninguna habilidad')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logística')
# axis labels
pyplot.xlabel('Tasa de falsos positivos')
pyplot.ylabel('Tasa de verdaderos positivos')
pyplot.legend()
pyplot.show()

```

No hay habilidad: ROC AUC = 0.5  
 Logística: ROC AUC = 0.9028205128205128



Al ejecutar las líneas de código se imprime el área bajo la curva ROC para el modelo de Regresión Logística y el clasificador sin habilidad que solo produce 0 para todos los ejemplos.

## Curva PR (curva precision-recall)

Hay muchas formas de evaluar la habilidad de un modelo de predicción. Un enfoque en el campo relacionado de la recuperación de información mide la precisión y la sensibilidad. Estas medidas también son útiles en Machine Learning para evaluar modelos de clasificación binarios.

La precisión es un cociente del número de verdaderos positivos dividido por la suma de los verdaderos positivos y los falsos positivos. Describe lo bueno que es un modelo para predecir la clase positiva. La precisión se denomina valor predictivo positivo.

$$\text{Precisión} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos positivos}}$$

La sensibilidad se calcula como el cociente del número de verdaderos positivos dividido por la suma de los verdaderos positivos y los falsos negativos.

$$\text{Sensibilidad} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos negativos}}$$



Revisar tanto la precisión con la sensibilidad es útil en los casos en los que hay un desequilibrio en las observaciones entre las dos clases.

En concreto, hay muchos ejemplos de ningún evento (clase 0) y solo unos pocos ejemplos de un evento (clase 1). Esto se debe a que, normalmente, el gran número de ejemplos de la clase 0 significa que estamos menos interesados en la habilidad del modelo para predecir correctamente la clase 0, por ejemplo, un alto número de verdaderos negativos. La clave para el cálculo de la precisión y la sensibilidad es que los cálculos no hacen uso de los verdaderos negativos. Solo se tiene en cuenta la predicción correcta de la clase minoritaria, la clase 1.

Una curva PR (curva precisión-sensibilidad) es un gráfico de la precisión (eje Y) y la sensibilidad (eje X) para diferentes umbrales, muy parecido a la curva ROC. Un clasificador sin habilidad es aquel que no puede discriminar entre las clases y predeciría una clase aleatoria o una clase constante en todos los casos. La línea de no-habilidad cambia en función de la distribución de las clases positivas y negativas. Es una línea horizontal con el valor de la proporción de casos positivos en el



conjunto de datos. Para un conjunto de datos equilibrado, es 0,5. Un modelo con una habilidad perfecto se representa como un punto en (1,1). Un modelo hábil se representa con una curva que se inclina hacia (1,1) por encima de la línea plana de ausencia de habilidad.

También hay puntuaciones compuestas que intentan resumir la precisión y la sensibilidad, dos ejemplos son:

- Puntuación F1: calcula la media armónica de la precisión y la sensibilidad, media armónica porque la precisión y la sensibilidad son cocientes.
- Área bajo la curva: al igual que la curva ROC AUC, resume la integral o una aproximación del área bajo la curva de precisión-sensibilidad.

En términos de selección de modelos, F1 resume la habilidad del modelo para un umbral de probabilidad específico (0,5), mientras que el área bajo la curva resume la habilidad de un modelo a través de umbrales, como el AUC ROC. Esto hace que la precisión-sensibilidad y un gráfico de precisión frente a la sensibilidad y las medidas de resumen sean útiles para los problemas de clasificación binaria que tienen un desequilibrio en las observaciones de cada clase.

Tanto la precisión como la sensibilidad se pueden calcular en Scikit Learn a través de las funciones `precision_score()` y `recall_score()`. La precisión y la sensibilidad se pueden calcular para los umbrales utilizando la función `precision_recall_curve()` que toma los valores de salida verdaderos y las probabilidades para la clase positiva como salida y devuelve los valores de precisión, sensibilidad y umbral. Por su parte, la puntuación F1 puede calcularse llamando la función `f1_score()` que toma como argumentos los valores de la clase verdadera y los valores de la clase predicha.

Veamos esto en un ejemplo con Python.



```

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot

#Se genera un conjunto de datos de 2 clases
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)

#Se divide el conjunto en entrenamiento y en prueba
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)

#Se genera el modelo y se entrena
modelo = LogisticRegression(solver='lbfgs')
modelo.fit(trainX, trainy)

#Se predice las probabilidades
lr_probs = modelo.predict_proba(testX)

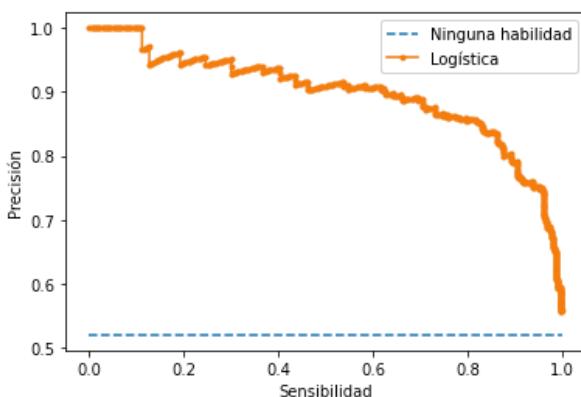
#Se mantiene las probabilidades sólo para el resultado positivo
lr_probs = lr_probs[:, 1]

#Se predice los valores de la clase
yhat = modelo.predict(testX)
lr_precision, lr_recall, _ = precision_recall_curve(testy, lr_probs)
lr_f1, lr_auc = f1_score(testy, yhat), auc(lr_recall, lr_precision)
print('Logística: f1 = ', lr_f1, ' auc = ', lr_auc)

#Se grafica la curva PR
no_skill = len(testy[testy==1]) / len(testy)
pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='Ninguna habilidad')
pyplot.plot(lr_recall, lr_precision, marker='.', label='Logística')
pyplot.xlabel('Sensibilidad')
pyplot.ylabel('Precisión')
pyplot.legend()
pyplot.show()

```

Logística: f1 = 0.8413001912045889 auc = 0.8977244980325833



## ¿Cuándo utilizar las curvas ROC y PR?

En general, el uso de las curvas ROC y PR es la siguiente:



- Las curvas ROC deben utilizarse cuando hay un número aproximadamente igual de observaciones para cada clase.
- Las curvas PR deben utilizarse cuando hay un desequilibrio de clases de moderado a grande.

La razón de esta recomendación es que las curvas ROC presentan una imagen optimista del modelo en conjuntos de datos con desequilibrio de clases. Sin embargo, las curvas ROC pueden presentar una visión demasiado optimista del rendimiento de un algoritmo si hay un desequilibrio en la distribución de clases.

Inclusive algunos expertos van más allá y sugieren que el uso de una curva ROC con un conjunto de datos desequilibrado podría ser engañoso y conducir a interpretaciones incorrectas de la habilidad del modelo.

La razón principal de esta imagen optimista se debe al uso de verdaderos negativos en la tasa de falso positivos en la curva ROC y a la cuidadosa evitación de esta tasa en la curva PR (curva precision-recall).

## Resumen

En este capítulo, aprendiste varios métodos de puntuación que puedes utilizar para evaluar las probabilidades predichas en los problemas de modelado predictivo de clasificación. Específicamente, se explicó:

- La puntuación de pérdida logarítmica que penaliza fuertemente las probabilidades predichas lejos de su valor esperado.
- La puntuación de Brier, que es más suave que la pérdida logarítmica, pero sigue penalizando de forma proporcional a la distancia del valor esperado.
- Las curvas ROC, que resumen el equilibrio entre la tasa de verdaderos positivos y la tasa de falsos positivos de un modelo de predicción utilizando diferentes umbrales de probabilidad.
- Las curvas PR (curva precisión-recall), que resumen el equilibrio entre la tasa de verdaderos positivos y el valor predictivo positivo para un modelo de predicción que utiliza diferentes umbrales de probabilidad.
- Las curvas ROC son apropiadas cuando las observaciones están equilibradas entre cada clase, mientras que las curvas de precisión-sensibilidad son apropiadas para conjuntos de datos desequilibrados.



# Capítulo 13

## OBTENER MÁS INFORMACIÓN

---

Este ebook es solo el comienzo de tu camino para aprender las matemáticas que necesitas para Machine Learning. A medida que se desarrolle nuevos proyectos, es posible que necesites ayuda. En este capítulo se indica algunas de las mejores fuentes de ayuda de Python que puedes encontrar.

### Consejo general

En general, en la página de AprendelA podrás encontrar mucha más información sobre Machine Learning. Desde información teórica como ejercicios prácticos, de esta forma podrás aumentar tus habilidades dentro de este tema. Toda la información contenida acá es en español.



Para mayor información se puede revisar la siguiente información:

<https://bit.ly/3xZXZZD>



De igual forma, la documentación oficial de Python y NumPy es excelente. Tanto las guías de usuario como la documentación de API son una excelente ayuda para aclarar dudas. Acá podrás tener una comprensión más completa de la configuración más profunda que puedes explorar. La información publicada es en inglés.

Otro recurso muy útil son los sitios de preguntas y respuestas, como StackOverflow. Se puede buscar mensajes de error y problemas que se tenga y encontrar ejemplos de códigos e ideas que pueden ayudar en los proyectos. Esta página se encuentra tanto en español como en inglés, aunque en esta última se encuentra la mayor cantidad de información.

## Ayuda con Python

Python es un lenguaje de programación con todas las funciones. Como tal, cuanto más aprendas sobre él, mejor podrás usarlo. Si es relativamente nuevo en la plataforma Python, aquí hay algunos recursos valiosos para ir un paso más profundo:

Documentación oficial Python 3:

<https://docs.python.org/3/>

## Ayuda con NumPy

Es una buena idea familiarizarse con el ecosistema NumPy más amplio, por lo que es recomendable revisar la documentación de NumPy, sobre todo cuando se tenga problemas con estas librerías.

Documentación oficial NumPy:

<https://docs.scipy.org/doc/numpy/user/>



# ANEXO 1

## NUMPY

---

Antes de iniciar con las explicaciones sobre el álgebra lineal, es bueno explicar un poco cómo funciona la librería de Python, NumPy.

NumPy es una librería de Python que puede ser utilizada para aplicaciones científicas y numéricas y es la herramienta a utilizar para operaciones de álgebra lineal.

Las matrices son la principal estructura de datos utilizada en Machine Learning. En Python, los arreglos de la biblioteca NumPy, llamados arreglos de N-dimensiones o *ndarray*, se usan como la estructura de datos principal para representar los datos. En este ebook, descubrirás la matriz N-dimensional en NumPy para representar datos numéricos y manipular datos en Python.

### Introducción a NumPy

NumPy se refiere a la abreviatura de Numerical Python y es una librería de Python diseñada para una eficiente computación científica básica. Se utiliza en casi todos los campos de la ciencia y la ingeniería. Es el estándar universal para trabajar con datos numéricos en Python, y es el núcleo de los ecosistemas científicos Python y PyData.

Los usuarios de NumPy incluyen a todos, desde los codificadores principiantes hasta los investigadores experimentados que realizan investigación y desarrollo



científico e industrial de vanguardia. El API de NumPy se utiliza ampliamente en Pandas, SciPy, matplotlib, scikit-learn y en la mayoría de los demás paquetes de datos científicos de Python.

Para la computación intensiva de datos, NumPy nos proporciona una amplia gama de métodos que hacen que la manipulación de datos en Python sea muy rápida y fácil.

NumPy es un poderoso paquete científico de código abierto que nos proporciona:

- Métodos para trabajar con array N-dimensionales y estructuras de datos matriciales.
- Formas de realizar operaciones matemáticas complejas en matrices, como rutinas trigonométricas, estadísticas y algebraicas. Por lo tanto, la librería contiene un gran número de funciones matemáticas, algebraicas y de transformación.
- Clases y funciones para probar el álgebra lineal, las series de Fourier y las capacidades de generación de números aleatorios.
- Herramientas para integrar el código C/C++ y Fortran.
- NumPy es una extensión de Numeric y Numarray.
- Los objetos de Pandas dependen en gran medida de los objetos de NumPy, en otras palabras, Pandas extienden NumPy.

Además de sus obvios usos científicos, NumPy también puede utilizarse como un eficiente contenedor multidimensional de datos genéricos. Se pueden definir tipos de datos arbitrarios. Esto permite que NumPy se integre sin problemas y rápidamente con una amplia variedad de bases de datos.

Ofrece una gran alternativa a las listas en Python, ya que las matrices de NumPy son más compactas, permiten un acceso más rápido en la lectura y escritura de elementos, y son más convenientes y más eficientes en general.

Dado que es uno de los paquetes fundamentales e imprescindible para la computación científica, NumPy es uno de los paquetes que debes ser capaz de usar si quieras desarrollarte dentro de la Inteligencia Artificial y Machine Learning con Python.

## Matriz n-dimensional NumPy

La principal estructura de datos en NumPy es en *ndarray*, que es un nombre abreviado de matriz o array n-dimensional. *ndarray* es un arreglo de tamaño fijo en memoria que contiene datos del mismo tipo, como números enteros o valores de punto flotante.



Veamos cómo funciona NumPy, en el siguiente ejemplo se crea una lista en Python de 3 valores de punto flotante, luego creamos un *ndarray* y a partir de la lista accedemos a la forma y tipo de los datos de la matriz.

```
from numpy import array

#Crear una matriz
a = [1.0, 2.0, 3.0]
b = array(a)

#Mostrar la matriz
print(b)

#Mostrar forma de la matriz
print(b.shape)

#Mostrar el tipo de datos de la matriz
print(b.dtype)
```

```
[1. 2. 3.]
(3,)
float64
```

Ejecutado el ejemplo, puedes observar el contenido del *ndarray*, seguidamente la forma, que es de 3 elementos y, por último, el tipo de datos, que es una coma flotante de 64 bits.

## Funciones para crear matrices

Hay muchas funciones convenientes para crear matrices de tamaño fijo que puedes utilizar en los proyectos. Veamos algunas de ellas.

### Vacío

La función *empty()* creará una nueva matriz de la forma especificada. El argumento de la función es una tupla que especifica la longitud de cada dimensión de la matriz a crear. Los valores o el contenido de la matriz serán aleatorios y deberán ser asignados antes de su uso.

En el siguiente ejemplo se crea una matriz bidimensional vacío de 4 X 4.

```
from numpy import empty

#Crear una matriz vacía 4x4
a = empty([4, 4])
print(a)
```

```
[[2.05833592e-312 2.33419537e-312 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]]
```



Ejecutado el ejemplo, puedes observar el contenido de la matriz vacía. Toma en cuenta que, el contenido específico de la matriz variará en caso de que ejecutes, por tu cuenta, este mismo código.

### Ceros

La función `zeros( )` creará una nueva matriz del tamaño especificado con los contenidos rellenados con valores cero. El argumento de la función es una tupla que especifica la longitud de cada dimensión de la matriz a crear.

En el ejemplo siguiente se crea una matriz bidimensional de 4 X 8.

```
from numpy import zeros

#Crear una matriz de ceros
a = zeros([4, 8])
print(a)

[[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

Al ejecutar el código se imprime el contenido de la matriz cero creada.

### Unos

La función `ones( )` creará una nueva matriz del tamaño especificado con los contenidos rellenados con un valor. El argumento de la función es una tupla que especifica la longitud de cada dimensión de la matriz a crear.

En el ejemplo siguiente se crea una matriz de 6 elementos de una dimensión.

```
from numpy import ones

#Crear una matriz de unos
a = ones([6])
print(a)

[1. 1. 1. 1. 1. 1.]
```

Como puedes observar, una vez que ejecutamos el código, se imprime el contenido de la matriz de unos creada.

## Combinación de matrices

NumPy proporciona muchas funciones para crear nuevas matrices a partir de matrices existentes. Veamos dos de las funciones más populares que puedes necesitar o encontrar.



### *Pila vertical*

Dados dos o más matrices existentes, puedes apilarlos verticalmente usando la función `vstack()`.

Por ejemplo, dadas dos matrices unidimensionales, puedes crear una nueva bidimensional con dos filas apilándolas verticalmente. Esto se demuestra en el siguiente ejemplo:

```
from numpy import array
from numpy import vstack

#Crear la primera matriz
a1 = array([1,2,3])

#Crear la segunda matriz
a2 = array([4,5,6])

#Crear la pila vertical
a = vstack((a1,a2))
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

Al ejecutar el ejemplo, las matrices (`a1` y `a2`) se apilan verticalmente, lo que da como resultado una matriz (`a`) de 2 X 3, cuyo contenido se muestra al ejecutar el código.

### *Pila horizontal*

Dados dos o más matrices existentes, puedes apilarlos horizontalmente usando la función `hstack()`.

Por ejemplo, dadas dos matrices unidimensionales, puedes crear una nueva unidimensional o una fila con las columnas de la primera y segunda matriz concatenados. Esto se demuestra en el siguiente ejemplo:

```
from numpy import array
from numpy import hstack

#Crear la primera matriz
a3 = array([10,20,30])

#Crear la segunda matriz
a4 = array([40,50,60])

#Crear la pila horizontal
a = hstack((a3,a4))
print(a)
```

```
[10 20 30 40 50 60]
```



Al ejecutar el código las matrices ( $a_3$  y  $a_4$ ) se apilan horizontalmente lo que resulta en una nueva matriz unidimensional con 6 elementos, cuyo contenido ( $a$ ) se imprime.

## Convertir listas en matrices

En general, recomendamos cargar los datos desde un archivo usando Pandas (librería de Python) o incluso utilizando las funciones de NumPy.

Veamos cómo convertir los datos en listas a matrices de NumPy.

### *Lista unidimensional a matriz*

Puedes cargar los datos o generar los datos y tener acceso a ellos en forma de lista. Puedes convertir una lista unidimensional de datos en una matriz llamando a la función `array()` de NumPy.

```
from numpy import array

#Lista de datos
lista = [11, 22, 33, 44, 55]

#Convertir una lista en una matriz
matriz = array(lista)
print(matriz)
print(type(matriz))

[11 22 33 44 55]
<class 'numpy.ndarray'>
```

La ejecución del ejemplo convierte la lista unidimensional guardada en la variable `lista` en una matriz de NumPy, que se almacena en la variable `matriz`.

### *Lista bidimensional a matriz*

Es más probable que los datos de Machine Learning tenga datos bidimensionales. Es decir, una tabla de datos en la que cada fila representa una nueva observación y cada columna una nueva característica.

Tal vez se generaron los datos o se cargaron usando código personalizado y ahora tienes una lista de listas. Cada lista representa una nueva observación. Puedes convertir la lista de listas en una matriz de NumPy de la misma manera que arriba, llamando a la función `array()`.



```

from numpy import array

#Lista de datos
lista = [[11, 22],
          [33, 44],
          [55, 66]]

#Convertir una lista en una matriz
matriz = array(lista)
print(matriz)
print(type(matriz))

[[11 22]
 [33 44]
 [55 66]]
<class 'numpy.ndarray'>

```

Como puedes observar, una vez que se ejecuta el código las listas que estaban almacenadas en la variable *lista* se convirtieron con éxito en una matriz.

## Indexación de matrices

Una vez que los datos se representan mediante una matriz de NumPy, se puede acceder a ellos mediante la indexación. Veamos algunos ejemplos de acceso a los datos mediante la indexación.

### *Indexación unidimensional*

Puedes acceder a los elementos utilizando el operador de corchetes [ ] que especifica el índice de desplazamiento cero para el valor a recuperar.

```

from numpy import array

#Definir la matriz
a = array([11, 22, 33, 44, 55])

#indexar los datos
print(a[0])
print(a[3])

11
44

```

Al ejecutar el ejemplo se imprimen el primer y el penúltimo valor de la matriz.



Recuerda que Python considera que el primer elemento o índice de una estructura de datos está en la posición 0, no en la posición 1.



En caso de que solicites un valor mayor del límite de la matriz causará un error.

```
from numpy import array

#Definir la matriz
a = array([11, 22, 33, 44, 55])

#indexar los datos
print(a[6])

-----
IndexError                                     Traceback (most recent call last)
/var/folders/9j/7r8cq911210_97kqgwdskt4000gn/T/ipykernel_1001/1099547174.py in <module>
      5
      6 #Indexar los datos
----> 7 print(a[6])

IndexError: index 6 is out of bounds for axis 0 with size 5
```

Puedes usar índices negativos para recuperar valores del final de la matriz. Por ejemplo, el índice -1 se refiere al último elemento de la matriz. El índice -2 devuelve el penúltimo elemento hasta -5 para el primer elemento del ejemplo actual.

```
from numpy import array

#Definir la matriz
a = array([11, 22, 33, 44, 55])

#indexar los datos
print(a[-1])
print(a[-5])

55
11
```

Al ejecutar el ejemplo puedes observar que se imprime el último y el primer elemento de la matriz.

#### *Indexación bidimensional*

La indexación de datos bidimensionales es similar a la indexación de datos unidimensionales, excepto que se utiliza una coma para separar el índice de cada dimensión.



```

from numpy import array

#Definir la matriz
a = array([[11, 22],
           [33, 44],
           [55, 66]])

#indexar los datos
print(a[0,0])
print(a[1,0])

```

11  
33

Al ejecutar el ejemplo se imprime el primer elemento del set de datos, así como también el primer elemento de la segunda fila.

En caso de que te interese obtener todos los elementos de la tercera fila, podríamos dejar el índice de la segunda dimensión vacío, por ejemplo:

```

from numpy import array

#Definir la matriz
a = array([[11, 22],
           [33, 44],
           [55, 66]])

#indexar los datos
print(a[2,:])

```

[55 66]

Como puedes observar, una vez ejecutado el código se imprime todos los datos de la tercera fila.

## Slicing de matrices

Hasta ahora, crear e indexar matrices parece familiar. Ahora llegamos al rebanado o *slice* de matrices, y esta es una de las características que causa problemas a los principiantes en las matrices de Python y NumPy.

Estructuras como listas y matrices NumPy pueden ser rebanadas (*slice*). Esto significa que se puede indexar y recuperar una subsecuente estructura.



El rebanado o *slicing* de matrices es muy útil en Machine Learning cuando se especifican variables de entrada y salida, o cuando se dividen las filas de entrenamiento de las filas de prueba.



El rebanado se especifica usando el operador de dos puntos: con un índice de origen y destino antes y después de la columna respectiva.

### *Slicing unidimensional*

Puedes acceder a todos los datos de una dimensión de una matriz especificando el segmento ":" sin índices.

```
from numpy import array

##Definir la matriz
data = array([11, 22, 33, 44, 55])
print(data[:])
```

Al ejecutar el ejemplo se imprimen todos los elementos de la matriz.

El primer elemento de la matriz se puede obtener especificando que comience en el índice 0 y termine en el índice 1, un elemento antes del índice que se quiere obtener.

```
from numpy import array

##Definir la matriz
a = array([11, 22, 33, 44, 55])
print(a[0:1])

[11]
```

Al ejecutar este ejemplo devuelve un subconjunto con el primer elemento de la matriz original.

También podemos utilizar índices negativos en las rebanadas. Por ejemplo, podemos cortar los dos últimos elementos de la lista empezando en -2 (el penúltimo elemento) y no especificando un índice al final, esto lleva la rebanada al final de la dimensión.

```
from numpy import array

##Definir la matriz
a = array([11, 22, 33, 44, 55])
print(a[-2:])
```

[44 55]

La ejecución del ejemplo devuelve un subconjunto con solo las dos últimas posiciones de la matriz original.

### *Slicing bidimensional*

Veamos los dos ejemplos de rebanado bidimensional que más probablemente usarás en Machine Learning.



Es común dividir los datos cargados en variables de entrada ( $X$ ) y la variable de salida ( $y$ ). Podemos hacer esto cortando todas las filas y todas las columnas, y luego indexando por separado la última columna. Para las características de entrada, podemos seleccionar todas las filas y todas las columnas excepto la última, mientras que para la característica de salida utilizamos la última columna, esto se escribiría de la siguiente forma:

Colocando todo esto junto, podemos separar un conjunto de datos de 2D de 3 columnas en datos de entrada y salida de la siguiente manera:

```
from numpy import array

#Definir la matriz
a = array([[11, 22, 33],
           [44, 55, 66],
           [77, 88, 99]])

#Separar los datos
x = a[:, :-1]
y = a[:, -1]
print(x)
print()
print(y)
```

```
[[11 22]
 [44 55]
 [77 88]]

[33 66 99]
```

Al ejecutar el código se imprimen los elementos de  $X$  y  $y$  separados. Observa que  $X$  es una matriz 2D y la variable  $y$  es un vector 1D.

Es común, en Machine Learning, dividir un conjunto de datos que ha sido cargado en datos de entrenamiento y de prueba. Esta es una división de filas donde alguna porción será usada para entrenar el modelo y la porción restante será usada para estimar la habilidad del modelo entrenado. Esto implicaría cortar todas las columnas especificando: en el índice de la segunda dimensión. El conjunto de datos de entrenamiento sería todas las filas desde el principio hasta el punto de división.

Poniendo todo esto junto, podemos dividir el conjunto de datos en el punto de división artificial de 2.



```
from numpy import array

#Definir la matriz
a = array([[11, 22, 33],
           [44, 55, 66],
           [77, 88, 99]])

#Separar los datos
split = 2
train = a[:split,:]
test = a[split:,:]
print(train)
print()
print(test)
```

```
[[11 22 33]
 [44 55 66]]
```

```
[[77 88 99]]
```

Al ejecutar el ejemplo se seleccionan las dos primeras filas para los datos de entrenamiento y la última fila para el conjunto de datos de pruebas.

*Toma en cuenta que lo explicado acá es simplemente un ejemplo de cómo deberías escribir tu código al momento de separar los datos, en los proyectos de Machine Learning, se tiene un conjunto de datos mucho mayor al que se muestra acá.*

## Reajuste de matrices

Después de cortar los datos, puedes que tengas que volver a darles forma. Por ejemplo, algunas librerías, como scikit-learn, pueden requerir que una matriz unidimensional de variables de salida ( $y$ ) se forme como una matriz bidimensional con una columna y resultados para cada columna.

Algunos algoritmos de Machine Learning requieren que la entrada sea especificada como un arreglo tridimensional compuesto de muestras, pasos de tiempo y características.

Es importante saber cómo remodelar los arreglos de NumPy para que los datos cumplan con las expectativas de las librerías específicas de Python. Veamos estos dos ejemplos.

### Forma de los datos

Las matrices NumPy tienen un atributo *shape* que devuelve una tupla de la longitud de cada dimensión de la matriz.



```

from numpy import array

#Definir la matriz
a = array([11, 22, 33, 44, 55])

print(a.shape)

(5,)

```

Ejecutando el código puedes observar que nos devuelva una tupla de una dimensión con el número de columnas, 5.

En caso de que se tenga un arreglo bidimensional, se devuelve una tupla con dos longitudes, veamos un ejemplo.

```

from numpy import array

#Definir la matriz
a = array([[11, 22],
           [33, 44],
           [55, 66]])

print(a.shape)

(3, 2)

```

Una vez que ejecutamos el código podemos observar que se imprime una tupla con el número de filas, 3, y el número de columnas, 2.

#### *Cambio de la forma de la matriz 1D a la matriz 2D*

Es común la necesidad de reajustar un arreglo unidimensional en un arreglo bidimensional con una columna y múltiples arreglos. NumPy proporciona la función *reshape()* en el objeto de matriz NumPy que puede ser usada para reajustar los datos. La función *reshape()* toma un único argumento que especifica la nueva forma de la matriz.

```

from numpy import array

#Definir la matriz
a = array([11, 22, 33, 44, 55])
print(a.shape)

#Reajustar los datos
b = a.reshape((a.shape[0], 1))
print(b.shape)

(5, )
(5, 1)

```

Ejecutando el código puedes ver que se imprime la forma de la matriz unidimensional, para luego reajustar la matriz para tener 5 filas con 1 columna.



### *Cambio de la forma de la matriz 2D a la matriz 3D*

Es común la necesidad de reajustar datos bidimensionales en los que cada fila representa una secuencia en una matriz tridimensional para algoritmos que esperan múltiples muestras de uno o más pasos de tiempo y una o más características.

La función de reajuste puede ser usada directamente, especificando la nueva dimensionalidad. Esto es claro con un ejemplo donde cada secuencia tiene múltiples pasos de tiempo con una observación (característica) en cada paso de tiempo. Podemos usar los tamaños en el atributo de forma en la matriz para especificar el número de muestras (filas) y columnas (paso de tiempo) y fijar el número de características en 1.

```
from numpy import array

#Definir la lista de los datos
a = ([[11, 22],
       [33, 44],
       [55, 66]])

#Definir la matriz de los datos
b = array(a)
print(b.shape)

print()

#Reajustar los datos
c = b.reshape(b.shape[0], b.shape[1], 1)
print(c.shape)

(3, 2)

(3, 2, 1)
```

Al ejecutar el código, primero se imprime el tamaño de cada dimensión de la matriz 2D, se reajusta la matriz y luego se imprime la forma de la nueva matriz 3D.

## Resumen

En este capítulo, descubriste la matriz N-dimensional en NumPy para representar numéricamente y manipular datos en Python. Específicamente, se explicó:

- Qué es la matriz N y cómo crear e inspeccionar una matriz en Python.
- Funciones claves para crear nuevas matrices vacías y matrices con valores por defecto.
- Cómo combinar matrices existentes para crear nuevas matrices.
- Cómo convertir los datos de una lista en matrices de NumPy.
- Cómo acceder a los datos mediante la indexación y el slicing o rebanado en Python.



- Cómo cambiar el tamaño de los datos para satisfacer las expectativas de algunas API de Machine Learning.
- El problema de la aritmética con matrices de diferentes tamaños.



AprendelA

Ligdi González  
2022

# PROBABILIDADES

## para Machine Learning

VERSIÓN 1

