Geron

Outline breve (págs. 409 – 419)

- **Inicio del Cap. 9 – Aprendizaje no supervisado**
    - Motiva su relevancia: la mayoría de los datos no están etiquetados; subraya el potencial inexplorado del no supervisado.
- **Tres tareas centrales**
    - **Agrupamiento (clustering)** – descubrir grupos de instancias similares.
    - **Detección de anomalías / valores atípicos** – modelar lo “normal” para identificar casos raros (fraude, defectos, series temporales). (no entra como tema)
    - **Estimación de densidad** – aproximar la PDF generadora; útil tanto para visualización como para detectar regiones de baja densidad. (no entra)
- **Aplicaciones ilustrativas de clustering**
    - Segmentación de clientes, motores de búsqueda (imágenes similares), sistemas de recomendación, aprendizaje semisupervisado y segmentación de imágenes.
- **Diferencias con clasificación**
    - Demuestra con el dataset Iris: con etiquetas (clasificación) vs. sin etiquetas (clustering) y cómo los algoritmos pueden detectar sub-clústeres invisibles a simple vista.
- **Tipos de algoritmos de clustering**
    - Basados en centroides (k-means), en densidad (DBSCAN), jerárquicos, etc.; cada uno captura nociones distintas de “grupo” y se elige según contexto.

k-means

- **Ejemplo sintético de cinco blobs**
    - Uso de `make_blobs` y `KMeans(n_clusters=5)` para agrupar rápidamente; predicción de etiquetas y centros mostrados.
    - Visualización de límites de decisión → teselación de Voronoi (Fig. 9-3).
- **Asignación dura vs. blanda**
    - `predict()` da etiqueta rígida; `transform()` devuelve distancias/afinidades para clustering flexible (puntuación por centroide).

- **Complejidad y escalabilidad**
    - Normalmente O(m × k × n); lineal en número de instancias, clústeres y dimensiones salvo casos patológicos.
- **Procedimiento iterativo del algoritmo**
    - Inicializar centroides aleatoriamente.
    - **Etiqueta** → asignar cada punto al centroide más cercano.
    - **Actualiza** → reemplazar centroides por la media de puntos asignados.
    - Repetir hasta converger (distancia intra-clúster no puede aumentar).
- **Sensibilidad a la inicialización**
    - Puede converger a óptimos locales; Fig. 9-5 muestra dos soluciones subóptimas provocadas por centroides iniciales desafortunados.

Marsland

Outline breve (págs. 302 – 306)

- **Aprendizaje no supervisado: idea central**
    - Se contrasta con el aprendizaje supervisado: sin etiquetas, el objetivo es descubrir grupos de entradas similares mediante una medida de proximidad interna, no un error externo basado en objetivos.
- **Criterio de error interno**
    - Al no disponer de salidas correctas, el algoritmo debe valerse de un criterio independiente de la tarea (p. ej., minimizar la suma de distancias intra-clúster).
- **Algoritmo *k*-means**
    - Elegir *k* y colocar los centros de clúster al azar.
    - **Asignación:** cada punto se asocia al centro más cercano (distancia euclidiana; puede acelerarse con KD-Tree).
    - **Actualización:** cada centro se mueve a la media de los puntos asignados.
    - Repetir hasta que los centros dejen de moverse.
- **Implementación NumPy**
    - Uso de `np.argmin` para la etapa de asignación y cálculo vectorizado de distancias; código ejemplo incluido.

- **Problemas prácticos**
    - **Mínimos locales:** la solución depende de la inicialización; se recomienda ejecutar varias veces con semillas distintas y quedarse con el menor SSE.
    - **Elección de *k*:** valores incorrectos provocan sub- o sobre-ajuste (Fig. 14.2 muestra 2 y 11 clústeres).
    - **Sobreajuste extremo:** *k* igual al número de puntos lleva a SSE≈0 pero sin capacidad de generalizar.
- **Ruido y valores atípicos**
    - El uso de la media hace a *k*-means sensible a outliers; puede mitigarse sustituyéndola por la **mediana** (k-medians), a costa de mayor costo computacional.

Capítulo 9. Técnicas de aprendizaje no supervisado

Aunque la mayoría de las aplicaciones actuales del aprendizaje automático se basan en el aprendizaje supervisado (y, por lo tanto, es aquí donde se destina la mayor parte de la inversión), la gran mayoría de los datos disponibles no están etiquetados: tenemos las características de entrada X, pero no las etiquetas y. El científico informático Yann LeCun afirmó: «Si la inteligencia fuera un pastel, el aprendizaje no supervisado sería el pastel, el aprendizaje supervisado sería la guinda del pastel y el aprendizaje por refuerzo sería la guinda del pastel». En otras palabras, existe un enorme potencial en el aprendizaje no supervisado, que apenas hemos empezado a explorar.

Supongamos que quieres crear un sistema que tome algunas fotos de cada artículo en una línea de producción y detecte cuáles son defectuosos. Puedes crear fácilmente un sistema que tome fotos automáticamente, lo que podría generar miles de fotos al día. Así, puedes generar un conjunto de datos bastante grande en tan solo unas semanas. ¡Pero espera, no hay etiquetas! Si quieres entrenar un clasificador binario estándar que prediga si un artículo es defectuoso o no, tendrás que etiquetar cada foto como "defectuoso" o "normal". Esto generalmente requiere que expertos humanos revisen manualmente todas las fotos. Es una tarea larga, costosa y tediosa, por lo que normalmente solo se realizará en un pequeño subconjunto de las imágenes disponibles. Como resultado, el conjunto de datos etiquetados será bastante pequeño y el rendimiento del clasificador será decepcionante. Además, cada vez que la empresa realiza algún cambio en sus productos, todo el proceso tendrá que empezar desde cero. ¿No sería fantástico si el algoritmo pudiera explotar los datos sin etiquetar sin necesidad de que los humanos etiqueten cada foto? Ingrese al aprendizaje no supervisado.

En el capítulo 8, analizamos la tarea de aprendizaje no supervisado más común: la reducción de dimensionalidad. En este capítulo, analizaremos algunas más.

tareas no supervisadas:

Agrupamiento

El objetivo es agrupar instancias similares en clústeres. La agrupación en clústeres es una herramienta excelente para el análisis de datos, la segmentación de clientes, los sistemas de recomendación, los motores de búsqueda, la segmentación de imágenes, el aprendizaje semisupervisado, la reducción de dimensionalidad y más.

Detección de anomalías (también llamada detección de valores atípicos)

El objetivo es comprender el aspecto de los datos "normales" y utilizarlo para detectar casos anormales. Estos casos se denominan anomalías o valores atípicos, mientras que los casos normales se denominan valores atípicos. La detección de anomalías es útil en una amplia variedad de aplicaciones, como la detección de fraudes, la detección de productos defectuosos en la fabricación, la identificación de nuevas tendencias en series temporales o la eliminación de valores atípicos de un conjunto de datos antes de entrenar otro modelo, lo que puede mejorar significativamente el rendimiento del modelo resultante.

Estimación de densidad

Esta tarea consiste en estimar la función de densidad de probabilidad (PDF) del proceso aleatorio que generó el conjunto de datos. La estimación de densidad se utiliza comúnmente para la detección de anomalías: las instancias ubicadas en regiones de muy baja densidad tienden a ser anómalas. También es útil para el análisis y la visualización de datos.

¿Listos para un pastel? Empezaremos con dos algoritmos de agrupamiento: k-means y DBSCAN. Después, analizaremos los modelos de mezcla gaussiana y veremos cómo se pueden usar para la estimación de densidad, el agrupamiento y la detección de anomalías.

Algoritmos de agrupamiento: k-medias y DBSCAN. Mientras disfrutas de una caminata por la montaña, te topas con una planta que nunca antes habías visto. Miras a tu alrededor y notas algunas más. No son idénticas, pero son lo suficientemente similares como para que sepas que probablemente pertenecen a la misma especie (o al menos al mismo género). Quizás necesites que un botánico te diga de qué especie se trata, pero ciertamente no necesitas un experto para identificar grupos de objetos de aspecto similar. Esto se llama agrupamiento: consiste en identificar instancias similares y asignarlas a clústeres, o grupos de instancias similares.

Al igual que en la clasificación, cada instancia se asigna a un grupo. Sin embargo, a diferencia de la clasificación, la agrupación en clústeres es una tarea no supervisada. Considere la Figura 9-1: a la izquierda está el conjunto de datos iris (presentado en el Capítulo 4), donde la especie de cada instancia (es decir, su clase) se representa con un marcador diferente. Es un conjunto de datos etiquetado, para el cual los algoritmos de clasificación como la regresión logística, las máquinas de modelado de valores (SVM) o los clasificadores de bosque aleatorio son muy adecuados. A la derecha está el mismo conjunto de datos, pero sin las etiquetas, por lo que ya no se puede utilizar un algoritmo de clasificación. Aquí es donde entran en juego los algoritmos de agrupación en clústeres: muchos de ellos pueden detectar fácilmente el clúster inferior izquierdo. También es bastante fácil ver con nuestros propios ojos, pero no es tan obvio que el clúster superior derecho esté compuesto por dos subclústeres distintos. Dicho esto, el conjunto de datos tiene dos características adicionales (longitud y ancho del sépalo) que no están representadas aquí, y los algoritmos de agrupamiento pueden hacer un buen uso de todas las características, por lo que, de hecho, identifican los tres grupos bastante bien (por ejemplo, utilizando un modelo de mezcla gaussiana, solo 5 instancias de 150 se asignan al grupo incorrecto).

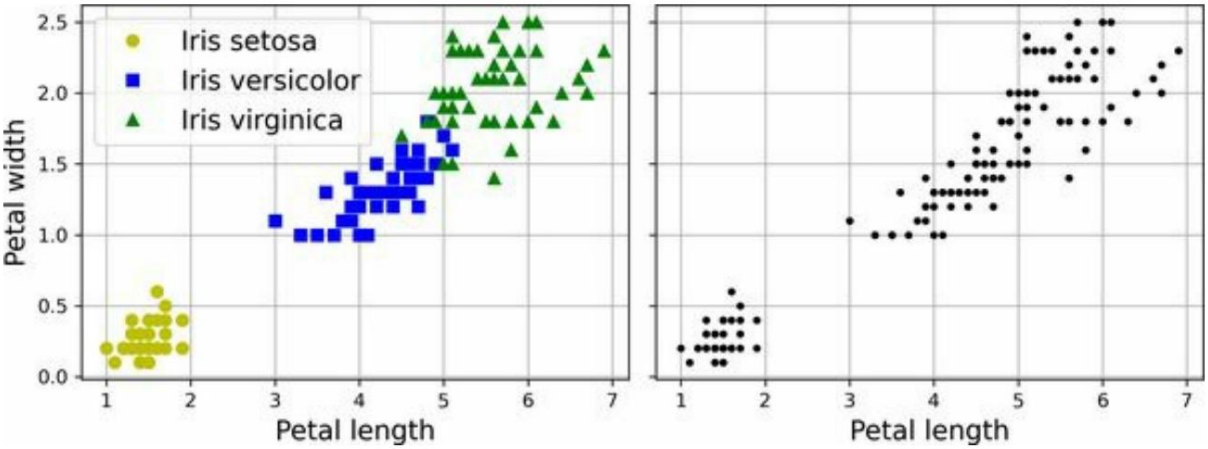

Figura 9-1. Clasificación (izquierda) versus agrupamiento (derecha)

La agrupación en clústeres se utiliza en una amplia variedad de aplicaciones, entre las que se incluyen:

Segmentación de clientes

Puedes agrupar a tus clientes según sus compras y su actividad en tu sitio web. Esto te ayudará a comprender quiénes son tus clientes y qué necesitan, para que puedas adaptar tus productos y campañas de marketing a cada segmento. Por ejemplo, la segmentación de clientes puede ser útil en sistemas de recomendación para sugerir contenido que otros usuarios del mismo grupo disfrutaron.

Análisis de datos

Al analizar un nuevo conjunto de datos, puede ser útil ejecutar un algoritmo de agrupamiento y luego analizar cada grupo por separado.

Reducción de dimensionalidad

Una vez agrupado un conjunto de datos, suele ser posible medir la afinidad de cada instancia con cada clúster; la afinidad es cualquier medida de la afinidad de una instancia con un clúster. El vector de características x de cada instancia puede entonces reemplazarse por el vector de afinidades de sus clústeres. Si hay k clústeres, este vector es k-dimensional. El nuevo vector suele tener una dimensión mucho menor que el vector de características original, pero puede conservar suficiente información para su posterior procesamiento.

Ingeniería de características

Las afinidades de clústeres suelen ser útiles como características adicionales. Por ejemplo, en el capítulo 2, utilizamos k-means para añadir características de afinidad de clústeres geográficos al conjunto de datos de vivienda de California, lo que nos ayudó a obtener un mejor rendimiento.

Detección de anomalías (también llamada detección de valores atípicos)

Cualquier instancia con baja afinidad con todos los clústeres probablemente represente una anomalía. Por ejemplo, si ha agrupado a los usuarios de su sitio web según su comportamiento, puede detectar usuarios con comportamientos inusuales, como un número inusual de solicitudes por segundo.

Aprendizaje semisupervisado

Si solo tiene unas pocas etiquetas, puede agruparlas y propagarlas a todas las instancias del mismo clúster. Esta técnica puede aumentar considerablemente la cantidad de etiquetas disponibles para un algoritmo de aprendizaje supervisado posterior y, por lo tanto, mejorar su rendimiento.

Motores de búsqueda

Algunos motores de búsqueda permiten buscar imágenes similares a una imagen de referencia. Para crear un sistema de este tipo, primero se aplicaría un algoritmo de agrupamiento a todas las imágenes de la base de datos; las imágenes similares se agruparían en el mismo grupo. Luego, cuando un usuario proporciona una imagen de referencia, bastaría con usar el modelo de agrupamiento entrenado para encontrar el grupo de esta imagen y, a continuación, se podrían devolver todas las imágenes de este grupo.

Segmentación de imágenes

Al agrupar los píxeles según su color y luego reemplazar el color de cada píxel con el color promedio de su grupo, es posible reducir considerablemente la cantidad de colores diferentes en una imagen. La segmentación de imágenes se utiliza en muchos sistemas de detección y seguimiento de objetos, ya que facilita la detección del contorno de cada objeto.

No existe una definición universal de lo que es un clúster: realmente depende del contexto y diferentes algoritmos capturarán distintos tipos de clústeres. Algunos algoritmos buscan instancias centradas en un punto específico, llamado centroide. Otros buscan regiones continuas de instancias densamente agrupadas: estos grupos pueden adoptar cualquier forma. Algunos algoritmos son jerárquicos y buscan grupos de grupos. Y la lista continúa.

En esta sección, analizaremos dos algoritmos de agrupamiento populares, k-means y DBSCAN, y exploraremos algunas de sus aplicaciones, como la reducción de dimensionalidad no lineal, el aprendizaje semisupervisado y la detección de anomalías.

k-medias

Considere el conjunto de datos sin etiquetar representado en la Figura 9-2: se pueden ver claramente cinco blobs de instancias. El algoritmo k-means es un algoritmo simple capaz de agrupar este tipo de conjunto de datos de forma rápida y eficiente, a menudo en tan solo unas pocas iteraciones. Fue propuesto por Stuart Lloyd en Bell Labs en 1957 como una técnica para la modulación por código de pulsos, pero solo se publicó. fuera de la empresa en 1982. En 1965, Edward W. Forgy[1] había publicado prácticamente el mismo algoritmo, por lo que a veces se hace referencia a k-means como el algoritmo de Lloyd-Forgy.

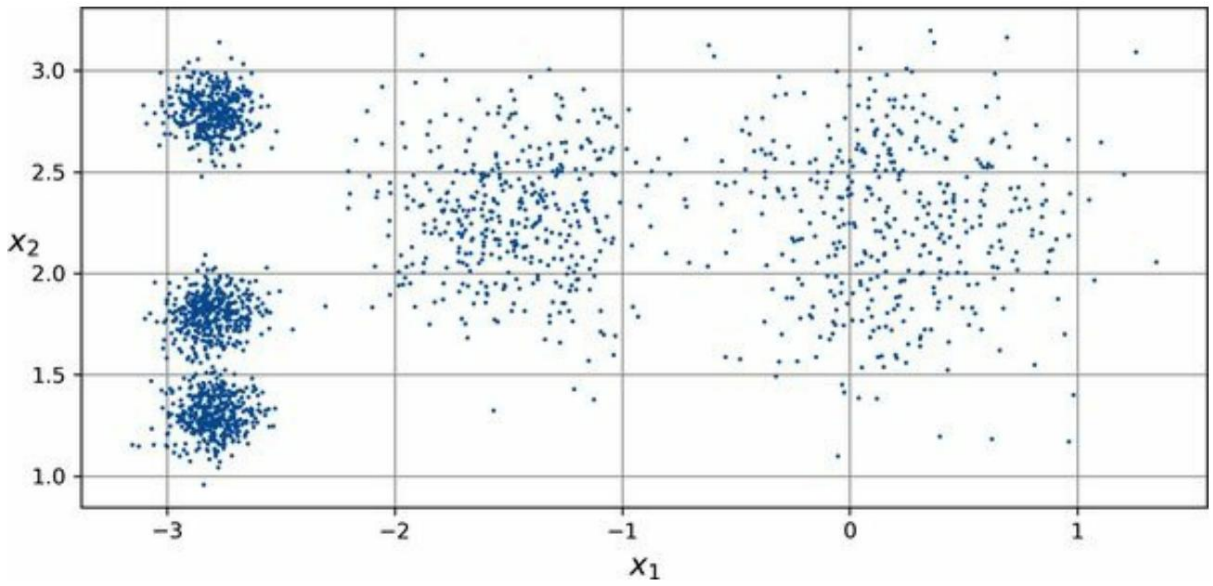

Figura 9-2. Un conjunto de datos sin etiquetar compuesto por cinco blobs de instancias

Entrenemos un clusterizador de k-medias con este conjunto de datos. Intentará encontrar el centro de cada blob y asignar cada instancia al blob más cercano:

```
desde sklearn.cluster importar KMeans desde
sklearn.datasets importar make_blobs

X, y = make_blobs([...]) # crea los blobs: y contiene los ID del clúster, pero
                         # no los usaremos; eso es lo que queremos predecir
k = 5
kmeans = KMeans(n_grupos=k, estado_aleatorio=42) y_pred
= kmeans.fit_predict(X)
```

Tenga en cuenta que debe especificar el número de clústeres k que el algoritmo debe encontrar. En este ejemplo, al observar los datos, es bastante obvio que k debe establecerse en 5, pero en general no es tan fácil. Lo abordaremos en breve.

Cada instancia se asignará a uno de los cinco clústeres. En el contexto de la agrupación en clústeres, la etiqueta de una instancia es el índice del clúster al que el algoritmo asigna esta instancia; esto no debe confundirse con las etiquetas de clase en la clasificación, que se utilizan como objetivos (recuerde que la agrupación en clústeres es una tarea de aprendizaje no supervisado). La instancia de KMeans conserva las etiquetas predichas de las instancias con las que se entrenó, disponibles mediante la variable de instancia labels_:

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32) >>>
y_pred es kmeans.labels_ Verdadero
```

También podemos echar un vistazo a los cinco centroides que encontró el algoritmo:

```
>>> kmeans.cluster_centers_
array([[-2,80389616, 1,80117999],
       [ 0,20876306, 2,25551336],
       [-2,79290307, 2,79641063],
       [-1,46679593, 2,28585348],
       [-2,80037642, 1,30082566]])
```

Puede asignar fácilmente nuevas instancias al clúster cuyo centroide esté más cercano:

```
>>> importar numpy como np
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]]) >>>
kmeans.predict(X_new) array([1, 1,
2, 2], dtype=int32)
```

Si trazas los límites de decisión del grupo, obtienes una teselación de Voronoi: consulta la Figura 9-3, donde cada centroide está representado con una X.

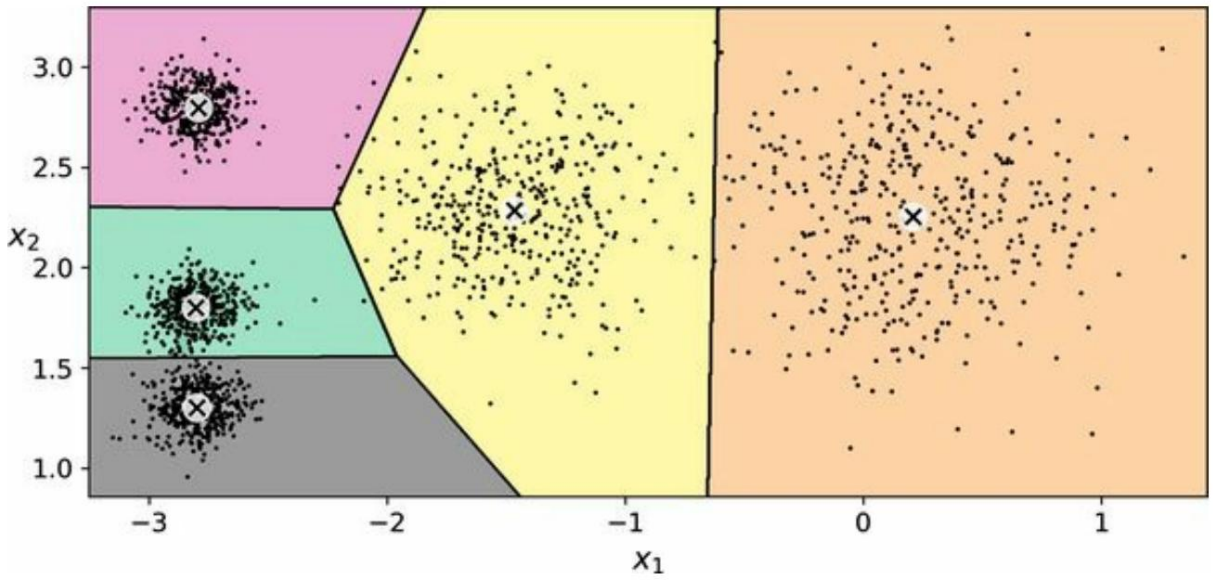

Figura 9-3. Límites de decisión de k-medias (teselación de Voronoi)

La gran mayoría de las instancias se asignaron claramente al clúster apropiado, pero algunas probablemente estaban mal etiquetadas, especialmente cerca del límite entre el clúster superior izquierdo y el clúster central. De hecho, el algoritmo k- medias no funciona muy bien cuando los blobs tienen diámetros muy diferentes, ya que, al asignar una instancia a un clúster, solo se preocupa por la distancia al centroide.

En lugar de asignar cada instancia a un solo clúster, lo que se denomina agrupamiento rígido, puede ser útil asignar a cada instancia una puntuación por clúster, lo que se denomina agrupamiento flexible. La puntuación puede ser la distancia entre la instancia y el centroide o una puntuación de similitud (o afinidad), como la función de base radial gaussiana que usamos en el capítulo 2. En la clase KMeans, el método transform() mide la distancia de cada instancia a cada centroide:

```
>>> kmeans.transform(X_new).round(2)
matriz([[2.81, 0.33, 2.9, 1.49, 2.89],
       [5.81, 2.8, 5.85, 4.48, 5.84],
       [1.21, 3.29, 0.29, 1.69, 1.71],
       [0.73, 3.22, 0.36, 1.55, 1.22]])
```

En este ejemplo, la primera instancia en X_new se ubica a una distancia de aproximadamente 2,81 del primer centroide, 0,33 del segundo centroide, 2,90 del tercer centroide, 1,49 del cuarto centroide y 2,89 del quinto centroide.

Si se tiene un conjunto de datos de alta dimensión y se transforma de esta manera, se obtiene un conjunto de datos de k dimensiones: esta transformación puede ser una técnica de reducción de dimensionalidad no lineal muy eficiente. Como alternativa, se pueden usar estas distancias como características adicionales para entrenar otro modelo, como en el capítulo 2.

El algoritmo k-means

Entonces, ¿cómo funciona el algoritmo? Supongamos que se le proporcionan los centroides. Podría etiquetar fácilmente todas las instancias del conjunto de datos asignándolas al clúster cuyo centroide esté más cercano. Por el contrario, si se le proporcionan todas las etiquetas de las instancias, podría localizar fácilmente el centroide de cada clúster calculando la media de las instancias de ese clúster. Pero no se le proporcionan ni las etiquetas ni los centroides, así que ¿cómo puede proceder? Empiece colocando los centroides aleatoriamente (por ejemplo, seleccionando k instancias al azar del conjunto de datos y usando sus ubicaciones como centroides). Luego, etiquete las instancias, actualice los centroides, etiquete las instancias, actualice los centroides, y así sucesivamente hasta que los centroides dejen de moverse. Se garantiza que el algoritmo convergerá en un número finito de pasos (normalmente bastante pequeño). Esto se debe a que la distancia cuadrática media entre las instancias y sus centroides más cercanos solo puede disminuir en cada paso, y como no puede ser negativa, se garantiza que convergerá.

Puede ver el algoritmo en acción en la Figura 9-4: los centroides se inicializan aleatoriamente (arriba a la izquierda), luego se etiquetan las instancias (arriba a la derecha), luego se actualizan los centroides (centro a la izquierda), las instancias se reetiquetan (centro a la derecha), y así sucesivamente. Como puede ver, en tan solo tres iteraciones, el algoritmo ha alcanzado una agrupación que parece cercana a la óptima.

> NOTA
>
> La complejidad computacional del algoritmo es generalmente lineal con respecto al número de instancias m, el número de clústeres k y el número de dimensiones n.
> Sin embargo, esto solo es cierto cuando los datos tienen una estructura de agrupamiento. De no ser así, en el peor de los casos, la complejidad puede aumentar exponencialmente con el número de instancias. En la práctica, esto rara vez ocurre, y k-means suele ser uno de los algoritmos de agrupamiento más rápidos.

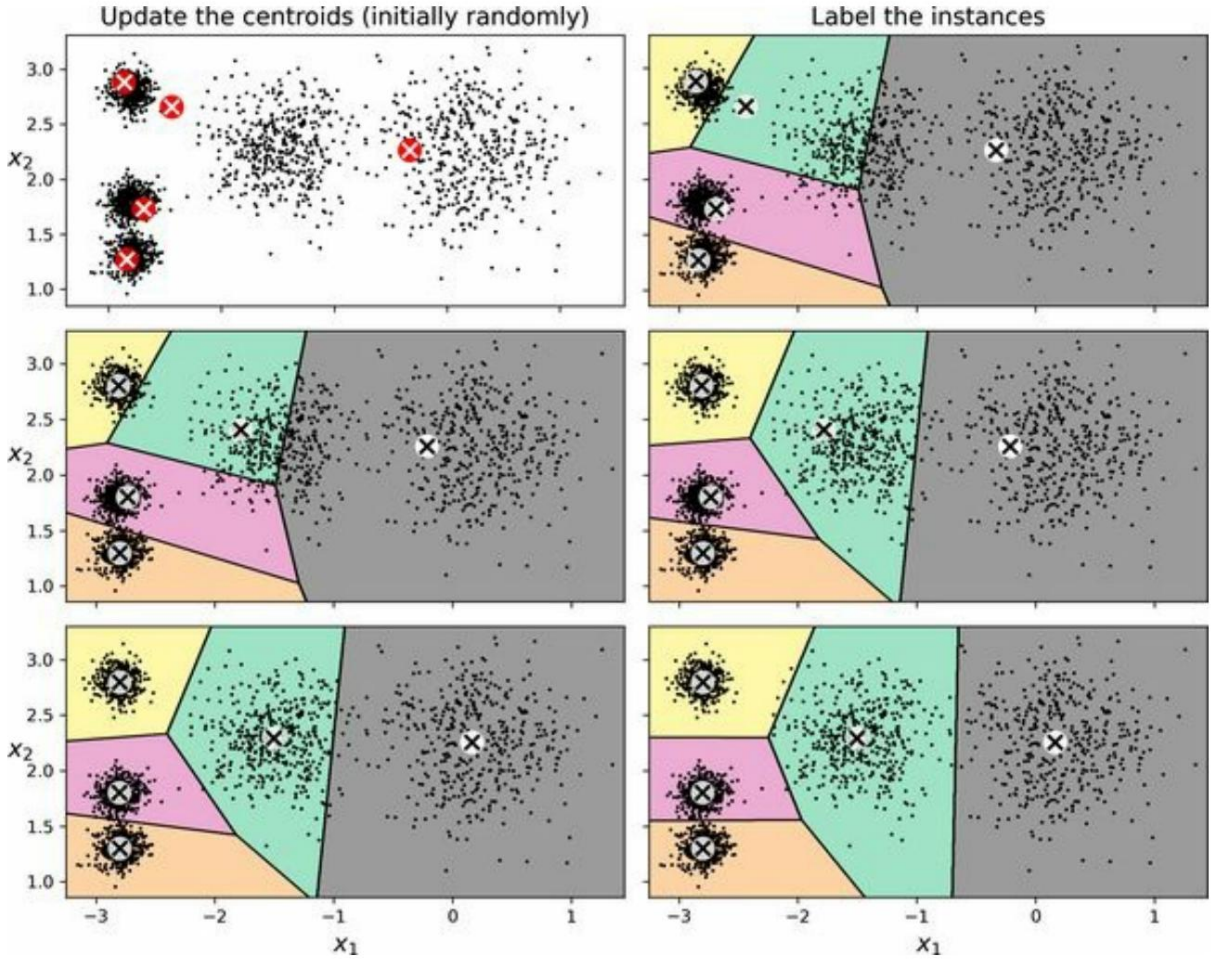

Figura 9-4. El algoritmo k-means

Aunque se garantiza la convergencia del algoritmo, es posible que no converja a la solución correcta (es decir, que converja a un óptimo local): que esto ocurra depende de la inicialización del centroide. La Figura 9-5 muestra dos soluciones subóptimas a las que el algoritmo puede converger si no se tiene suerte con el paso de inicialización aleatoria.

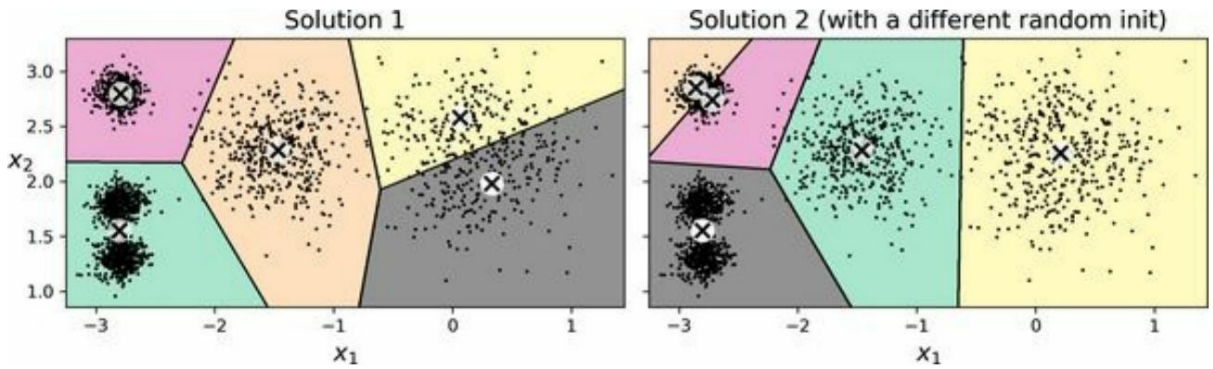

Figura 9-5. Soluciones subóptimas debido a inicializaciones de centroides desafortunadas.

Aprendizaje no supervisado

Muchos de los algoritmos de aprendizaje que hemos visto hasta la fecha utilizan un conjunto de entrenamiento compuesto por una colección de datos objetivo etiquetados o, al menos (para el aprendizaje evolutivo y de refuerzo), algún sistema de puntuación que identifica si una predicción es correcta. Los objetivos son, obviamente, útiles, ya que permiten mostrar al algoritmo la respuesta correcta a las posibles entradas, pero en muchas circunstancias son difíciles de obtener; por ejemplo, podrían implicar que alguien etiquete cada instancia manualmente. Además, no parece ser muy plausible biológicamente: la mayoría de las veces, durante el aprendizaje, no se nos dice exactamente cuál debería ser la respuesta correcta. En este capítulo, consideraremos el caso opuesto, donde no se dispone de información sobre las salidas correctas, y el algoritmo debe detectar por sí mismo alguna similitud entre las diferentes entradas.

El aprendizaje no supervisado es un problema conceptualmente diferente al aprendizaje supervisado. Obviamente, no podemos aspirar a realizar una regresión: desconocemos las salidas de ningún punto, por lo que no podemos adivinar cuál es la función. ¿Podemos entonces aspirar a realizar una clasificación? El objetivo de la clasificación es identificar similitudes entre las entradas que pertenecen a la misma clase. No existe información sobre las clases correctas, pero si el algoritmo puede aprovechar las similitudes entre las entradas para agrupar las entradas similares, podría realizar la clasificación automáticamente. Por lo tanto, el objetivo del aprendizaje no supervisado es encontrar grupos de entradas similares en los datos sin que se le indique explícitamente que estos puntos pertenecen a una clase y aquellos a otra diferente. En su lugar, el algoritmo debe descubrir las similitudes por sí mismo. Ya hemos visto algunos algoritmos de aprendizaje no supervisado en el Capítulo 6, donde el enfoque se centró en la reducción de la dimensionalidad y, por lo tanto, en la agrupación de punt

Los algoritmos de aprendizaje supervisado que hemos analizado hasta ahora han buscado minimizar un criterio de error externo —principalmente el error de suma de cuadrados— basado en la diferencia entre los objetivos y los resultados. Calcular y minimizar este error fue posible gracias a que disponíamos de datos objetivo para calcularlo, lo cual no ocurre con el aprendizaje no supervisado. Esto significa que necesitamos encontrar algo más que impulse el aprendizaje. El problema es más general que el error de suma de cuadrados: no podemos usar ningún criterio de error basado en objetivos u otra información externa (un criterio de error externo); necesitamos encontrar algo interno al algoritmo. Esto significa que la medida debe ser independiente de la tarea, ya que no podemos cambiar todo el algoritmo cada vez que se introduce una nueva tarea. En el aprendizaje supervisado, el criterio de error era específico de la tarea, ya que se basaba en los datos objetivo proporcionados.

Para ver cómo determinar un criterio de error general que podamos usar, necesitamos retomar algunos de los conceptos importantes que se discutieron en la Sección 2.1.1: espacio de entrada y espacio de ponderación. Si dos entradas están cerca, significa que sus vectores son similares, por lo que la distancia entre ellas es pequeña (las medidas de distancia se discutieron en la Sección 7.2.3, pero...

Aquí nos ceñiremos a la distancia euclidiana. Las entradas cercanas se identifican como similares, de modo que se pueden agrupar, mientras que las entradas distantes no se agrupan. Podemos extender esto a los nodos de una red alineando el espacio de ponderaciones con el espacio de entradas. Si los valores de ponderación de un nodo son similares a los elementos de un vector de entrada, ese nodo debería coincidir adecuadamente con la entrada y con cualquier otra entrada similar. Para empezar a aplicar estas ideas en la práctica, analizaremos un algoritmo de agrupamiento simple, el algoritmo k-medias, que se utiliza desde hace mucho tiempo en estadística.

14.1 EL ALGORITMO K-MEANS

Si alguna vez has visto a un grupo de turistas con un par de guías turísticos que sostienen paraguas para que todos puedan verlos y seguirlos, entonces has visto una versión dinámica del algoritmo k-means. Nuestra versión es más sencilla, porque los datos (que representan a los turistas) no se mueven, solo los guías turísticos.

Supongamos que queremos dividir nuestros datos de entrada en k categorías, donde conocemos el valor de k (por ejemplo, tenemos un conjunto de resultados de pruebas médicas de muchas personas para tres enfermedades, y queremos ver qué tan bien las pruebas identifican las tres enfermedades). Asignamos k centros de conglomerados a nuestro espacio de entrada y nos gustaría ubicarlos de manera que haya un centro de conglomerados en el centro de cada conglomerado. Sin embargo, desconocemos dónde se encuentran los conglomerados, y mucho menos dónde está su punto medio, por lo que necesitamos un algoritmo que los encuentre.
Los algoritmos de aprendizaje generalmente intentan minimizar algún tipo de error, por lo que necesitamos pensar en un criterio de error que describa este objetivo. La idea del «punto medio» es lo primero que debemos considerar. ¿Cómo definimos el punto medio de un conjunto de puntos? En realidad, hay dos cosas que debemos definir:

Una medida de distancia . Para hablar de distancias entre puntos, necesitamos una forma de medirlas. Suele ser la distancia euclidiana normal, pero existen otras alternativas; hemos abordado otras alternativas en la Sección 7.2.3.

La media promedio Una vez que tenemos una medida de distancia, podemos calcular el punto central de un conjunto de puntos de datos, que es la media promedio (si no estás convencido, piensa cuál es la media de dos números, es el punto a mitad de la línea entre ellos).
En realidad, esto solo es cierto en el espacio euclidiano, al que estamos acostumbrados, donde todo es plano y uniforme. Todo se complica mucho si tenemos que pensar en espacios curvos; cuando nos preocupamos por la curvatura, la métrica de la distancia euclidiana no es la correcta, y existen al menos dos definiciones diferentes de la media. Así que no nos preocuparemos por ninguno de estos aspectos y asumiremos que el espacio es plano.
Esto es lo que los estadísticos hacen todo el tiempo.

Ahora podemos pensar en una forma adecuada de posicionar los centros de los conglomerados: calculamos el punto medio de cada conglomerado, $\mu c(i)$, y ubicamos allí el centro del conglomerado. Esto equivale a minimizar la distancia euclidiana (que, nuevamente, es el error de suma de cuadrados) desde cada punto de datos hasta su centro.

¿Cómo decidimos qué puntos pertenecen a qué clústeres? Es importante decidirlo, ya que usaremos esta información para posicionar los centros de los clústeres. Lo obvio es asociar cada punto con el centro del clúster más cercano. Esto podría cambiar a medida que el algoritmo itera, pero no hay problema.

Comenzamos posicionando los centros de los clústeres aleatoriamente a través del espacio de entrada, ya que no sabemos dónde colocarlos, y luego actualizamos sus posiciones de acuerdo a los datos.
Decidimos a qué grupo pertenece cada punto de datos calculando la distancia entre cada uno.

punto de datos y todos los centros del clúster, y asignándolo al clúster que esté más cercano.
Tenga en cuenta que podemos reducir el coste computacional de este procedimiento utilizando el algoritmo KD-Tree descrito en la Sección 7.2.2. Para todos los puntos asignados a un clúster, calculamos su media y desplazamos el centro del clúster a esa posición.
Repetimos el algoritmo hasta que los centros de los clústeres dejen de moverse. Aquí está la descripción del algoritmo:

El algoritmo k-medias

- Inicialización
  - elija un valor para k
  - elige k posiciones aleatorias en el espacio de entrada
  - asignar los centros del grupo μj a esas posiciones
- Aprendiendo
  - repetir

    para cada punto de datos xi :

    · Calcular la distancia a cada centro de grupo · Asignar el punto de datos al centro de grupo más cercano con distancia

$$di = \min d(xi , \mu j) .j \tag{14.1}$$

    para cada centro del clúster:

    · mover la posición del centro a la media de los puntos de ese grupo (Nj es el número de puntos en el grupo j):

$$\mu j = \frac{1}{\text{Nueva Jersey}} \sum_{i=1}^{\text{Nueva Jersey}} xi \tag{14.2}$$

  - hasta que los centros de los grupos dejen de moverse
- Uso
  - para cada punto de prueba: calcular la distancia a cada centro del grupo asignar el punto de datos al centro del grupo más cercano con distancia

$$di = \min j \quad d(xi , \mu j). \tag{14.3}$$

La implementación de NumPy sigue estos pasos casi al pie de la letra, y podemos aprovechar la función np.argmin(), que devuelve el índice del valor mínimo, para encontrar el clúster más cercano. El código que calcula las distancias, encuentra el centro del clúster más cercano y las actualiza se puede escribir así:

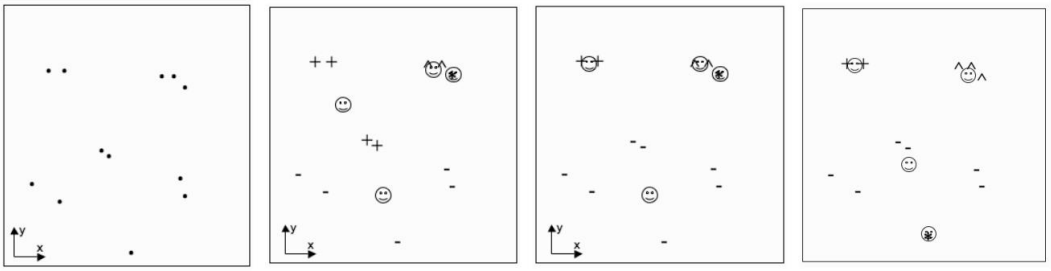

FIGURA 14.1 Izquierda: Un conjunto de datos bidimensional. Derecha: Tres posibles maneras de posicionar 4 centros (dibujados como caras) usando el algoritmo k-means, que es claramente susceptible a las fluctuaciones locales. mínimos.

```
# Calcular distancias
distancias = np.ones((1,self.nData))*np.sum((data-self.centres[0,:])**2, axis=1) para j en

rango(self.k-1):
    distancias = np.append(distancias,np.ones((1,self.nData))*np.sum((data-
    self.centres[j+1,:])**2,eje=1),eje=0)

# Identifica el clúster más
cercano cluster = distances.argmin(axis=0)
cluster = np.transpose(cluster*np.ones((1,self.nData)))

# Actualizar los centros de los
clústeres para j en
    range(self.k): thisCluster = np.where(cluster==j,1,0)
    if sum(thisCluster)>0:
        self.centres[j,:] = np.sum(data*thisCluster,axis=0)/np.sum( thisCluster)
```

Para ver cómo funciona esto en la práctica, las figuras 14.1 y 14.2 muestran algunos datos y diferentes maneras de agruparlos, calculados mediante el algoritmo k-means. Debe quedar claro que el algoritmo es susceptible a los mínimos locales: dependiendo de la posición inicial de los centros en el espacio, se pueden obtener soluciones muy diferentes, muchas de las cuales parecen muy improbables. La figura 14.2 muestra ejemplos de lo que ocurre cuando se elige incorrectamente el número de centros. Ciertamente, hay casos en los que no sabemos de antemano cuántos clústeres veremos en los datos, pero el algoritmo k-means no gestiona esto adecuadamente.

A costa de un gasto computacional adicional significativo, podemos solucionar ambos problemas ejecutando el algoritmo varias veces. Para encontrar un buen óptimo local (o incluso el global), utilizamos diferentes ubicaciones iniciales del centro, y la solución que minimiza el error total de suma de cuadrados probablemente sea la mejor.

Al ejecutar el algoritmo con muchos valores diferentes de k, podemos ver qué valores nos dan la mejor solución. Por supuesto, debemos tener cuidado con esto. Si simplemente medimos el error de suma de cuadrados entre cada punto de datos y su centro de clúster más cercano, entonces, cuando

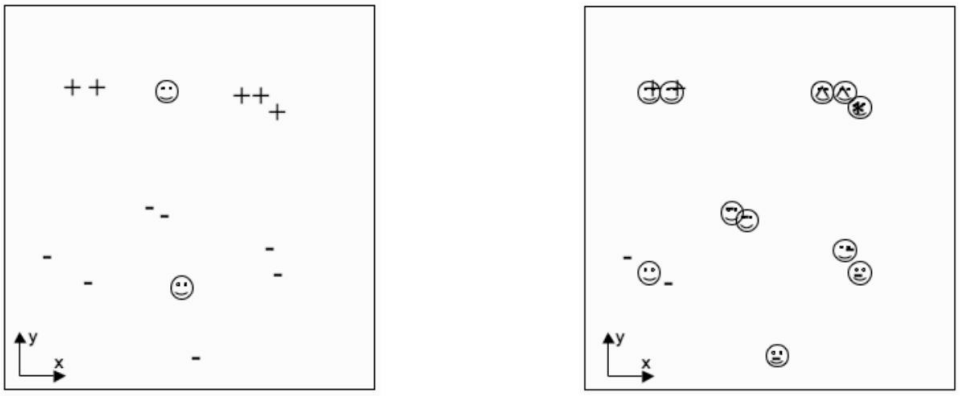

FIGURA 14.2 Izquierda: Una solución con sólo 2 clases, que no coincide bien con los datos. Derecha: Una solución con 11 clases, que muestra un sobreajuste severo.

Si fijamos k como el número de puntos de datos, podemos colocar un centro en cada punto de datos y el error de suma de cuadrados será cero (de hecho, esto no ocurrirá, ya que la inicialización aleatoria implicará que varios clústeres coincidirán). Sin embargo, esta solución no es generalizable: se trata de un caso de sobreajuste grave. No obstante, al calcular el error en un conjunto de validación y multiplicarlo por k, podemos observar la ventaja de añadir cada centro de clúster adicional.

14.1.1 Manejo del ruido

Existen muchas razones para realizar la agrupación en clústeres, pero una de las más comunes es lidiar con lecturas de datos ruidosas. Estas pueden estar ligeramente corruptas o, en ocasiones, simplemente erróneas. Si podemos elegir los clústeres correctamente, entonces habremos eliminado eficazmente el ruido, porque reemplazamos cada punto de datos ruidoso por el centro del clúster (usaremos esta forma de representar los puntos de datos para otros fines en la Sección 14.2). Desafortunadamente, la media promedio, que es fundamental para el algoritmo k-medias, es muy susceptible a los valores atípicos, es decir, mediciones muy ruidosas. Una forma de evitar el problema es reemplazar la media promedio por la mediana, que es lo que se conoce como una estadística robusta, lo que significa que no se ve afectada por los valores atípicos (la media de (1, 2, 1, 2, 100) es 21,2, mientras que la mediana es 2). El único cambio necesario en el algoritmo es reemplazar el cálculo de la media por el cálculo de la mediana. Esto es computacionalmente más costoso, como hemos discutido anteriormente, pero elimina el ruido de manera efectiva.

14.1.2 La red neuronal k-media

El algoritmo k-means funciona claramente, a pesar de sus problemas con el ruido y la dificultad para elegir el número de clústeres. Curiosamente, aunque parezca estar muy lejos de las redes neuronales, no lo está. Si consideramos los centros de los clústeres cuyas posiciones optimizamos como ubicaciones en el espacio de ponderaciones, podríamos posicionar neuronas en esos lugares y usar el entrenamiento de redes neuronales. El cálculo realizado en el algoritmo k-means fue que cada entrada decidía a qué centro de clúster estaba más cercano calculando la distancia a todos los centros. Esto también podríamos hacer dentro de una red neuronal: la ubicación de cada neurona es su posición en el espacio de ponderaciones, que coincide con los valores de sus ponderaciones. Por lo tanto, para cada entrada,