Aprendizaje por lotes y en línea

Aprendizaje por lotes

En el aprendizaje por lotes, el sistema es incapaz de aprender de forma incremental: debe ser entrenado utilizando todos los datos disponibles. Esto suele requerir mucho tiempo y recursos informáticos, por lo que suele hacerse fuera de línea. Primero se entrena el sistema, y luego se lanza a la producción y se ejecuta sin aprender más; sólo aplica lo que ha aprendido. Esto se llama aprendizaje offline.

Si se quiere que un sistema de aprendizaje por lotes conozca nuevos datos (como un nuevo tipo de spam), hay que entrenar una nueva versión del sistema desde cero en el conjunto de datos completo (no sólo los nuevos datos, sino también los antiguos), y luego detener el sistema antiguo y sustituirlo por el nuevo.

Afortunadamente, todo el proceso de entrenamiento, evaluación y puesta en marcha de un sistema de aprendizaje automático puede automatizarse con bastante facilidad, por lo que incluso un sistema de aprendizaje por lotes puede adaptarse al cambio. Basta con actualizar los datos y entrenar una nueva versión del sistema desde cero tantas veces como sea necesario.

Esta solución es sencilla y suele funcionar bien, pero el entrenamiento con el conjunto completo de datos puede llevar muchas horas, por lo que normalmente se entrenaría un nuevo sistema sólo cada 24 horas o incluso sólo semanalmente. Si el sistema necesita adaptarse a datos que cambian rápidamente (por ejemplo, para predecir los precios de las acciones), entonces se necesita una solución más reactiva.

Además, el entrenamiento sobre el conjunto completo de datos requiere muchos recursos informáticos (CPU, espacio de memoria, espacio de disco, E/S de disco, E/S de red, etc.). Si tienes muchos datos y automatizas el sistema para entrenar desde cero cada día, acabará costando mucho dinero. Si la cantidad de datos es enorme, puede ser incluso imposible utilizar un algoritmo de aprendizaje por lotes.

Por último, si el sistema tiene que ser capaz de aprender de forma autónoma y tiene recursos limitados (por ejemplo, una aplicación de teléfono inteligente o un Rover en Marte), entonces llevar grandes cantidades de datos de entrenamiento y ocupar muchos recursos para entrenar durante horas todos los días es un obstáculo.

Afortunadamente, una opción mejor en todos estos casos es utilizar algoritmos que sean capaces de aprender de forma incremental.

Aprendizaje en línea

En el aprendizaje en línea, se entrena al sistema de forma incremental alimentándolo con instancias de datos secuencialmente, ya sea individualmente o en pequeños

grupos denominados minilotes. Cada paso de aprendizaje es rápido y barato, por lo que el sistema puede aprender sobre los nuevos datos sobre la marcha, a medida que llegan.

El aprendizaje en línea es estupendo para los sistemas que reciben datos en forma de flujo continuo (por ejemplo, las cotizaciones bursátiles) y necesitan adaptarse a los cambios de forma rápida o autónoma. También es una buena opción si tiene recursos informáticos limitados: una vez que un sistema de aprendizaje en línea ha aprendido sobre nuevas instancias de datos, ya no las necesita, por lo que puede descartarlas (a no ser que quiera poder volver a un estado anterior y "reproducir" los datos). Este puede ahorrar una gran cantidad de espacio.

Los algoritmos de aprendizaje en línea también pueden utilizarse para entrenar sistemas en conjuntos de datos enormes que no caben en la memoria principal de una máquina (esto se llama aprendizaje fuera del núcleo). El algoritmo carga parte de los datos, ejecuta un paso de entrenamiento en esos datos y repite el proceso hasta que se haya ejecutado en todos los datos.

Un parámetro importante de los sistemas de aprendizaje en línea es la rapidez con la que deben adaptarse a los datos cambiantes: esto se llama tasa de aprendizaje. Si se establece una tasa de aprendizaje alta, el sistema se adaptará rápidamente a los nuevos datos, pero también tenderá a olvidar rápidamente los datos antiguos (no se quiere que un filtro de spam marque sólo los últimos tipos de spam que se le han mostrado).

Por el contrario, si establece una tasa de aprendizaje baja, el sistema tendrá más inercia, es decir, aprenderá más lentamente, pero también será menos sensible al ruido de los nuevos datos o a las secuencias de puntos de datos no representativos (valores atípicos, outliers).

Uno de los grandes retos del aprendizaje en línea es que si se introducen datos erróneos en el sistema, el rendimiento de éste disminuirá gradualmente. Si se trata de un sistema vivo, tus clientes lo notarán.

Por ejemplo, los datos erróneos pueden provenir de un sensor que no funciona bien en un robot, o de alguien que hace spam en un motor de búsqueda para tratar de posicionarse en los resultados de búsqueda. Para reducir este riesgo, debes supervisar su sistema de cerca y desactivar rápidamente el aprendizaje (y posiblemente volver a un estado de funcionamiento anterior) si detecta una caída en el rendimiento. También es posible que desee supervisar los datos de entrada y reaccionar ante los datos anómalos (por ejemplo, utilizando un algoritmo de detección de anomalías).

Ejemplos de sesgo de muestreo

Quizás el ejemplo más famoso de sesgo de muestreo ocurrió durante las elecciones presidenciales de Estados Unidos en 1936, que enfrentaron a Landon y Roosevelt: el Literary Digest realizó una encuesta muy grande, enviando correo a unos 10 millones de personas. Obtuvo 2,4 millones de respuestas y predijo con gran

seguridad que Landon obtendría el 57% de los votos. En cambio, Roosevelt ganó con el 62% de los votos. El fallo estaba en el método de muestreo del Literary Digest:

- En primer lugar, para obtener las direcciones a las que enviar las encuestas, el Literary Digest utilizó directorios telefónicos, listas de suscriptores de revistas, listas de miembros de clubes y similares y similares. Todas estas listas tendían a favorecer a las personas más ricas, que tenían más probabilidades de votar a los republicanos (de ahí lo de Landon).
- En segundo lugar, menos del 25% de las personas encuestadas respondieron. Una vez más, esto introdujo un sesgo de muestreo, al descartar potencialmente a las personas que no se preocupan mucho por la política, a las que no les gusta el Literary Digest y a otros grupos clave.

Se trata de un tipo especial de sesgo de muestreo denominado sesgo de falta de respuesta.

Otro ejemplo: supongamos que queremos crear un sistema para reconocer los vídeos de música funk. Una forma de construir su conjunto de entrenamiento es buscar "música funk" en YouTube y utilizar los vídeos resultantes. Pero esto supone que el motor de búsqueda de YouTube devuelve un conjunto de de vídeos que son representativos de todos los vídeos de música funk de YouTube. En realidad, los resultados de la búsqueda probablemente estén sesgados hacia artistas populares (y si vives en Brasil obtendrás muchos vídeos de "funk carioca", que no se parecen en nada a James Brown).

Sobreajuste de los datos de entrenamiento

Supongamos que estás de visita en un país extranjero y el taxista te estafa. Podrías tener la tentación de decir que todos los taxistas de ese país son ladrones. Generalizar en exceso es algo que los humanos hacemos con demasiada frecuencia y, por desgracia, las máquinas pueden caer en la misma trampa si no tenemos cuidado.

En el aprendizaje automático, esto se denomina sobreajuste (**overfitting**): significa que el modelo funciona bien con los datos de entrenamiento, pero no generaliza bien.

Los modelos complejos, como las redes neuronales profundas, pueden detectar patrones sutiles en los datos, pero si el conjunto de entrenamiento es ruidoso, o si es demasiado pequeño (lo que introduce ruido de muestreo), es probable que el modelo detecte patrones en el propio ruido. Obviamente, estos patrones no se generalizarán a nuevas instancias. Por ejemplo, supongamos que alimentamos nuestro modelo de "satisfacción vital" con muchos más atributos, incluidos los no informativos, como el nombre del país. En este caso, un modelo complejo puede detectar patrones como el hecho de que todos los países de los datos de entrenamiento con una w en su nombre (*lee los nombres en inglés*) tienen una satisfacción vital superior a 7: Nueva Zelanda (7,3), Noruega (7,4), Suecia (7,2) y Suiza (7,5). ¿En qué medida confía de que la regla de la satisfacción w se generaliza a Ruanda o Zimbabue? Obviamente, este patrón se produjo en los datos de

entrenamiento por pura casualidad, pero el modelo no tiene forma de decir si un patrón es real o simplemente el resultado del ruido en los datos.

La restricción de un modelo para simplificarlo y reducir el riesgo de sobreajuste se denomina regularización.

Ajuste insuficiente de los datos de entrenamiento

Como se puede adivinar, el ajuste insuficiente (**underfitting**) es lo contrario del sobreajuste: se produce cuando su modelo es demasiado simple para aprender la estructura subyacente de los datos. Por ejemplo, un modelo lineal de "satisfacción vital" es propenso al underfitting; la realidad es simplemente más compleja que el modelo, por lo que sus predicciones son forzosamente inexactas, incluso en los ejemplos de entrenamiento.

Estas son las principales opciones para solucionar este problema:

- Seleccionar un modelo más potente, con más parámetros.
- Alimentar el algoritmo de aprendizaje con mejores datos (ingeniería de datos).
- Reducir las restricciones del modelo (por ejemplo, reducir el hiperparámetro de regularización).

Pruebas y validaciones

La única forma de saber si un modelo se generalizará a nuevos casos es probarlo realmente con nuevos casos. Una forma de hacerlo es poner el modelo en producción y controlar su rendimiento. Esto funciona bien, pero si tu modelo es terriblemente malo, tus usuarios se quejarán, lo que no es la mejor idea.

Una mejor opción es dividir los datos en dos conjuntos: el **conjunto de entrenamiento** y el **conjunto de prueba**. Como estos nombres implican, el modelo se entrena con el conjunto de entrenamiento y se prueba con el conjunto de prueba. La tasa de error en los nuevos casos se denomina error de generalización (o error fuera de la muestra, *out-of-sample*), y al evaluar el modelo en el conjunto de prueba, se obtiene una estimación de este error. Este valor le indica el rendimiento de su modelo en instancias que nunca ha visto antes.

Si el error de entrenamiento es bajo (es decir, su modelo comete pocos errores en el conjunto de entrenamiento) pero el error de generalización es alto, significa que su modelo se está ajustando en exceso a los datos de entrenamiento (*overfitting*).

Ajuste de hiperparámetros y selección de modelos

Evaluar un modelo es bastante sencillo: basta con utilizar un conjunto de pruebas. Pero suponga que está dudando entre dos tipos de modelos (por ejemplo, un modelo lineal y un modelo polinómico): ¿cómo puede decidir entre ellos? Una opción es

entrenar a ambos y comparar su grado de generalización utilizando el conjunto de pruebas.

Supongamos ahora que el modelo lineal generaliza mejor, pero queremos aplicar alguna regularización para evitar el sobreajuste. La cuestión es cómo elegir el valor del hiperparámetro de regularización. Una opción es entrenar 100 modelos diferentes utilizando 100 valores distintos para este hiperparámetro. Supongamos que se encuentra el mejor valor de hiperparámetro que produce un modelo con el menor error de generalización, por ejemplo, sólo un 5% de error. Pone este modelo en producción, pero desgraciadamente no funciona tan bien como se esperaba y produce un error del 15%. ¿Qué ha ocurrido?

El problema es que ha medido el error de generalización varias veces en el conjunto de pruebas y ha adaptado el modelo y los hiperparámetros para producir el mejor modelo para ese conjunto concreto. Esto significa que es poco probable que el modelo funcione igual de bien con los nuevos datos.

Una solución común a este problema se denomina validación por retención: simplemente se retiene parte del conjunto de entrenamiento para evaluar varios modelos candidatos y seleccionar el mejor.

El nuevo conjunto retenido se llama conjunto de validación (o a veces conjunto de desarrollo). Más concretamente, se entrenan múltiples modelos con varios hiperparámetros en el conjunto de entrenamiento reducido (es decir, el conjunto de entrenamiento completo menos el conjunto de validación), y se selecciona el modelo que mejor funciona en el conjunto de validación. Después de este proceso de validación, se entrena el mejor modelo en el conjunto de entrenamiento completo (incluyendo el conjunto de validación), y se obtiene el modelo final. Por último, se evalúa este modelo final en el conjunto de pruebas para obtener una estimación del error de generalización.

Esta solución suele funcionar bastante bien. Sin embargo, si el conjunto de validación es demasiado pequeño, las evaluaciones del modelo serán imprecisas: puede acabar seleccionando un modelo subóptimo por error. Por el contrario, si el conjunto de validación es demasiado grande, el conjunto de entrenamiento restante será mucho más pequeño que el conjunto de entrenamiento completo. ¿Por qué es esto malo? Bueno, como el modelo final se entrenará con el conjunto de entrenamiento completo, no es ideal comparar modelos candidatos entrenados con un conjunto de entrenamiento mucho más pequeño. Sería como seleccionar al velocista más rápido para participar en una maratón. Una forma de resolver este problema es realizar una validación cruzada repetida, utilizando muchos conjuntos de validación pequeños. Cada modelo se evalúa una vez por conjunto de validación después de haber sido entrenado con el resto de los datos. Al promediar todas las evaluaciones de un modelo, se obtiene una medida mucho más precisa de su rendimiento. Sin embargo, hay un inconveniente: el tiempo de entrenamiento se multiplica por el número de conjuntos de validación.