Apuntes de MLOps

por: Fode

31 de marzo de 2024

Capítulo 1

Conceptos MLOps

1.1. Introducción

1.1.1. ¿Qué es MLOps?

MLOps es la abreviación de Machine Learning Operations y describe el conjunto de prácticas para diseñar, implementar y mantener el aprendizaje automático en producción ed manera continua, confiable y eficiente.

Operaciones de Machine Learning

En producción significa que nos centramos en el aprendizaje automático que se utiliza en los procesos empresariales en lugar de en un modelo de aprendizaje automático que solo existe en su computadora portátil. También analizamos algo más que simplemente entrenar un modelo de aprendizaje automático. MLOps se aplica a lo que llamamos el ciclo de vida del aprendizaje automático, que incluye desde el diseño y desarrollo hasta el mantenimiento del aprendizaje automático en producción.

¿Por qué MLOps?

Imagínese a un chef en un restaurante preparando platos maravillosos para sus invitados. Antes de preparar una receta para un plato nuevo, necesita saber qué ingredientes tiene y qué quieren los invitados. A partir de ahí, desarrolla la mejor receta posible. Para entregar el plato a los invitados, se requieren personas en el servicio y también se necesita equipo para preparar el plato. La combinación de todos estos factores lo convierten en un proceso complejo. Para estructurar el proceso, se podrían realizar controles rutinarios de los equipos, una degustación diaria del plato y se podrían probar los platos con diferentes invitados. Del mismo modo, el desarrollo del aprendizaje automático también es un proceso complejo. **Antes de desarrollar un modelo de aprendizaje automático, debemos pensar en los requisitos comerciales y el valor agregado de nuestro modelo. Combinando los datos y los requisitos, necesitamos encontrar el algoritmo adecuado, la receta.** Durante el desarrollo también necesitamos un ordenador, lo que también plantea sus propios retos, al igual que el equipamiento de la cocina. **MLOps tiene como objetivo estructurar el proceso y mitigar los riesgos involucrados en el desarrollo, implementación y mantenimiento del aprendizaje automático.**

El origen de MLOps

MLOps se origina en Development Operations, también llamado DevOps para abreviar. **DevOps describe un conjunto de prácticas y herramientas que se pueden aplicar al desarrollo de software para garantizar que el software se desarrolle de forma continua, confiable y eficiente**. El desarrollo de software tradicional solía ser lento debido a la separación de los equipos de Desarrollo y Operaciones. El equipo de desarrollo está formado por las personas que escriben el código, que fueron separadas del equipo de operaciones, las personas que implementan y dan soporte al código. Es por eso que DevOps es una integración de ambos equipos. De manera similar a cómo se aplica DevOps al desarrollo de software, MLOps se aplica al desarrollo de aprendizaje automático.

¿Qué operaciones?

Al igual que DevOps y MLOps, también existen mejores prácticas y herramientas para departamentos similares que podemos encontrar en una organización de TI, como ModelOps, DataOps y AIOps. Cada Ops se origina en la misma filosofía de DevOps y se centra en un desarrollo continuo, confiable y eficiente. ModelOps puede verse como una extensión de MLOps, con un conjunto de prácticas centradas principalmente en el modelo de aprendizaje automático. DataOps se centra en las mejores prácticas en materia de calidad y análisis de datos. Dado que los datos son parte del aprendizaje automático, esto se superpone con MLOps. AIOps significa Inteligencia Artificial para Operaciones de TI y es más amplio que el simple aprendizaje automático. En AIOps, se utilizan análisis, big data y aprendizaje automático para resolver problemas de TI sin asistencia o intervención humana.

Beneficios de MLOps

El uso de prácticas y herramientas de MLOps tiene múltiples beneficios. Puede, por ejemplo, mejorar la velocidad general de desarrollo y entrega de modelos de aprendizaje automático. Los procesos también se vuelven más confiables y seguros gracias a MLOps. Inherente a MLOps es que tiene como objetivo cerrar la brecha entre el aprendizaje automático y los equipos de operaciones, lo que mejora la colaboración. Durante este curso, analizaremos cómo MLOps pretende proporcionar estos beneficios.

1.1.2. Diferentes fases en MLOps

Ciclo de vida de MLOps

El ciclo de vida del aprendizaje automático es uno de los conceptos fundamentales en MLOps. Consta de tres fases amplias:

1. Diseño.
2. Desarrollo.
3. Implementación.

Este es un proceso iterativo y cíclico en el que no es raro ir y venir entre fases. Es importante dedicar tiempo a cada fase, ya que todas desempeñan un papel importante en el ciclo de vida completo. **Durante cada fase, es importante evaluar constantemente con las partes interesadas si el proyecto de aprendizaje automático debe continuar.** Podría ser que descubramos durante la fase de diseño que solo tenemos datos limitados o que solo podemos aplicar el problema a un grupo pequeño. Esto reduce el valor añadido y, por tanto, requiere una evaluación adicional por parte de las partes interesadas.

¿Por qué el ciclo de vida del aprendizaje automático?

El ciclo de vida del aprendizaje automático es importante porque brinda una descripción general de alto nivel de cómo se debe estructurar un proyecto de aprendizaje automático para brindar valor real y práctico. También define los roles que se requieren en cada paso para que el proyecto sea un éxito. Como veremos, podemos aplicar ciertas prácticas y herramientas a cada fase para optimizar el ciclo de vida.

Fase de diseño

En la fase de diseño, nos centramos en el diseño del proyecto de aprendizaje automático.

1. Definir el contexto del problema.
2. Determinar el valor añadido del uso del aprendizaje automático.
3. Recopilar requisitos comerciales.
4. Establecer métricas clave. (Para rastrear el progreso del ciclo de vida del aprendizaje automático).
5. Recopilar datos.
6. Asegurarnos de que la calidad de los datos sea suficiente para desarrollar un modelo de aprendizaje automático.

Fase de desarrollo

En la fase de desarrollo, nos centramos en desarrollar el modelo de aprendizaje automático. Hacemos esto experimentando con una combinación de datos, algoritmos e hiperparámetros de acuerdo con el diseño de implementación. Durante el experimento, entrenamos y evaluamos uno o más modelos para encontrar el más adecuado. **El objetivo de la fase de desarrollo es terminar con el modelo de aprendizaje automático más adecuado y listo para su implementación.**

Fase de implementación

En la fase de implementación, integramos el modelo de aprendizaje automático que desarrollamos anteriormente en el proceso comercial. Esto podría implicar la **creación de un microservicio** a partir del modelo de aprendizaje automático. **Un microservicio es una pequeña aplicación que incluye el modelo de aprendizaje automático de modo que podamos integrarlo fácilmente en el proceso de negocio.** También pretendemos establecer un seguimiento del modelo de aprendizaje automático. *Podemos configurar alertas cuando encontremos una desviación de datos o cuando nuestro modelo ya no genere una predicción.* La deriva de datos ocurre cuando nuestros datos cambian, lo que afecta el modelo de aprendizaje automático. Analizaremos estos conceptos con mayor detalle más adelante, donde analizaremos los diferentes componentes de cada fase.

1.1.3. Roles en MLOps

A partir de las fases que hemos visto en el ciclo de vida del aprendizaje automático, podemos crear una secuencia de ejemplo de pasos por los que pasa el ciclo de vida. Para cada tarea se requieren diferentes roles. Analizaremos dos categorías de roles, a saber, roles comerciales y roles técnicos.

Roles comerciales

En la categoría empresarial, podemos distinguir dos roles principales,

- El interesado en la empresa.
- El experto en la materia.

Veamos primero el papel de las partes interesadas en el negocio.

Roles empresariales: partes interesadas del negocio A veces también se hace referencia a la parte interesada del negocio como propietario del producto. Son personal directivo que toma decisiones presupuestarias y se asegura de que el proyecto de aprendizaje automático esté alineado con la visión de alto nivel de la empresa. **Están involucrados durante todo el ciclo de vida.** Primero, definen los requisitos comerciales durante la fase de diseño. En la fase de desarrollo también comprueban si los resultados iniciales de los experimentos son satisfactorios. Más adelante en la fase de implementación, vuelven a examinar si el resultado del ciclo de vida es el esperado.

Roles comerciales: experto en la materia En segundo lugar, está el experto en la materia, que tiene conocimientos de dominio sobre el problema que intentamos resolver. En el comercio minorista, podría ser, por ejemplo, alguien del equipo de ventas que conozca las variables que influyen en las ventas. El experto en la materia participa durante todo el ciclo de vida porque puede ayudar a las funciones más técnicas a interpretar los datos y los resultados en cada paso.

Roles técnicos

Hay cinco roles técnicos principales involucrados en el ciclo de vida del aprendizaje automático:

- Ingeniero de datos.
- Científico de datos.
- Ingeniero de software.
- Ingeniero de aprendizaje automático.
- Ingeniero de backend.

Roles técnicos: ingeniero de datos El ingeniero de datos **es responsable de la recopilación, almacenamiento y procesamiento de datos**. Esto también significa que el ingeniero de datos **debe verificar la calidad de los datos e incluir pruebas para que la calidad se mantenga durante todo el proceso**. Por lo tanto, el ingeniero de datos **participa principalmente en tareas que tienen que ver con los datos antes de entrenar el modelo, durante el entrenamiento del modelo y una vez que el modelo se utiliza en producción**.

Roles técnicos: científico de datos El científico de datos **es responsable del análisis de datos y la capacitación y evaluación de modelos**. La evaluación incluye el seguimiento del modelo una vez que se ha implementado para garantizar que las predicciones del modelo sean válidas. Podemos encontrar al científico de datos en todas las fases del ciclo de vida, pero **principalmente durante la fase de desarrollo**.

Roles técnicos: ingeniero de software El ingeniero de software **participa principalmente en la fase de implementación, donde escribe software para ejecutar el modelo, implementar el modelo y monitorear si el modelo está y permanece en línea una vez implementado**. También se aseguran de que el código esté escrito de acuerdo con pautas comunes. Dado que la implementación es una parte importante del ciclo de vida del aprendizaje automático, el ingeniero de software también debe participar en la fase de diseño.

Roles técnicos: ingeniero de ML El ingeniero de aprendizaje automático es un rol relativamente nuevo que es bastante versátil y **está diseñado específicamente para tener experiencia en todo el ciclo de vida del aprendizaje automático**. Es una función multifuncional que se superpone con las demás funciones técnicas. Como tal, el ingeniero de aprendizaje automático **participa en todas las fases**. Saben, por ejemplo, cómo extraer y almacenar datos y desarrollar o implementar un modelo de aprendizaje automático.

Roles técnicos: ingeniero backend El ingeniero de backend es alguien que **participa principalmente en la configuración de la infraestructura de la nube para permitir el desarrollo y la implementación de modelos de aprendizaje automático**. Podría ser una base de datos para almacenar los datos, pero también las computadoras que ejecutan el modelo de aprendizaje automático.

1.2. Diseño de MLOps

1.2.1. Diseño de aprendizaje automático

Ahora analizaremos el inicio de la fase de diseño, en la que investigamos

1. El valor añadido.
2. Los requisitos comerciales.
3. Las métricas clave.

que debemos rastrear.

Valor añadido

Normalmente, el ciclo de vida del aprendizaje automático comienza con la determinación del valor agregado de crear y ejecutar el modelo de aprendizaje automático. Esto **suele expresarse en términos de dinero o tiempo**. Determinar el valor de un modelo de aprendizaje automático puede ser un poco aproximado, pero es aconsejable estimar el potencial que tiene un determinado proyecto.

Estimación del valor agregado Digamos que tenemos un modelo de aprendizaje automático que predice si un cliente va a abandonar. Tenemos una base de clientes de 100.000 clientes que tienen una suscripción de $10 cada mes. Si podemos predecir esto correctamente para el 80 % de los 1000 clientes que normalmente abandonan, podemos enviarles a estos clientes una suscripción con descuento para que permanezcan el 50 % del tiempo. Esto da como resultado 1000 clientes multiplicados por ochenta por ciento multiplicados por cincuenta por ciento, es decir, 400 clientes

que no abandonan. Si les damos a estos 400 clientes una suscripción con descuento de ocho dólares, podemos ahorrar un total de 3200 por mes.

Requisitos comerciales Aparte del valor añadido del modelo de aprendizaje automático, también debemos considerar los requisitos del negocio. Especialmente en la fase de diseño, **es fundamental pensar en el usuario final del modelo de aprendizaje automático**. Digamos que estamos construyendo un modelo de aprendizaje automático que predice el número de ventas de un producto específico, de modo que podamos comprar la cantidad correcta para poner en nuestra tienda. El modelo de aprendizaje automático generará una cantidad prevista de ventas. Debemos considerar la frecuencia de las predicciones y qué tan rápido las necesitamos. También debemos evaluar la precisión del modelo y si sus resultados son explicables para los no expertos. **La transparencia se está convirtiendo en un factor cada vez más importante en el desarrollo del aprendizaje automático, ya que nos permite descubrir por qué el modelo hace sus predicciones, por qué está equivocado y cómo podemos mejorar el modelo. Todos estos requisitos tienen un impacto en el algoritmo que usaremos**.

Dependiendo del problema que estemos tratando de resolver, **también podrían existir requisitos regulatorios y de cumplimiento para el uso del aprendizaje automático**. Por ejemplo, en finanzas, cuando la ley exige una explicación por parte del sistema. **También debemos tener en cuenta el presupuesto disponible y el tamaño del equipo**.

Métricas clave Para ver si el ciclo de vida del aprendizaje automático avanza según lo esperado, suele ser aconsejable realizar un seguimiento del rendimiento del modelo. Sin embargo, como hemos visto anteriormente, los roles involucrados en los procesos MLOps son multidisciplinarios y, por lo tanto, también tienen su propia forma de rastrear el desempeño. El científico de datos analiza la precisión de un modelo, cuántas veces el algoritmo es correcto.

El experto en la materia está interesado en el impacto del modelo en el negocio, por ejemplo, en cómo mejora su trabajo gracias al uso del aprendizaje automático. Están interesados principalmente en métricas específicas de dominio.

La parte interesada del negocio está más interesada en el valor monetario del modelo, en cuántos casos realmente generamos ingresos. Esto suele expresarse en dinero o tiempo. Para aprovechar al máximo el aprendizaje automático, debemos alinear las diferentes métricas para asegurarnos de que todos estén en la misma página.

1.2.2. Calidad e ingesta de datos

La recopilación de datos es parte de la fase de diseño. Durante esta fase, investigamos:

- La calidad de los datos.
- Cómo extraemos los datos requeridos.

¿Qué es la calidad de los datos?

Según DAMA-DMBOK, un marco para la gestión de datos, el término **calidad de los datos se refiere tanto a las características asociadas con datos de alta calidad como a los procesos utilizados para medir o mejorar la calidad de los datos**. La calidad de un modelo de aprendizaje automático depende en gran medida de la calidad de los datos. Los datos son el núcleo del modelo de aprendizaje automático. Por lo tanto, tener una visión clara de la calidad de los datos es crucial para el éxito del ciclo de vida del aprendizaje automático. Tener una mala calidad de los datos es perjudicial para el rendimiento del modelo de aprendizaje automático. **Mejorar la calidad de los datos suele ser el primer paso para mejorar el rendimiento del modelo.**

1.2.3. Dimensiones de la calidad de los datos

La calidad de los datos se puede definir según cuatro dimensiones principales:

- Precisión.- describe el grado en que los datos son exactos o correctos para la tarea en cuestión.
- Exhaustividad.- se refiere al grado en que los datos describen completamente el problema en cuestión.

- Coherencia.- Un departamento podría tener una definición diferente de lo que constituye un cliente activo que otro, lo que hace que los datos sean inconsistentes.
- Puntualidad.- se refiere al plazo en el que los datos estarán disponibles.

Ejemplo de dimensiones de calidad de datos Digamos que estamos construyendo un modelo predictivo para determinar si los clientes abandonarán. Podemos comprobar la calidad de los datos de los clientes repasando las cuatro dimensiones. Un ejemplo de precisión sería si los datos describen correctamente al cliente. Podría ser que los datos indiquen que un cliente tiene 18 años, pero en realidad tiene 32 años. Eso sería inexacto. Para estar completo, nos fijamos principalmente en los datos que faltan, por ejemplo, si nos faltan los apellidos de los clientes. Con coherencia, investigamos si la definición de cliente es coherente en toda la organización. Podría ser que un departamento tenga una definición diferente de cliente activo que otro, lo que hace que los datos sean inconsistentes. Si nos fijamos en la puntualidad, nos interesa la disponibilidad de datos. Por ejemplo, cuando los pedidos de los clientes se sincronizan diariamente, no están disponibles en tiempo real. Tener una calidad de datos inferior en una o más dimensiones no significa que el proyecto esté destinado al fracaso. Hay varias cosas que podemos hacer para abordar la baja calidad de los datos, como recopilar más datos o completar los datos faltantes con otras fuentes.

Ingestión de datos

Normalmente, en la fase de diseño, también analizamos cómo extraer y procesar datos. Esto se hace mediante el uso de un canal de datos automatizado, del cual vemos un ejemplo de alto nivel aquí. Una canalización de datos suele ser una parte del ciclo de vida del aprendizaje automático a través del cual los datos se procesan automáticamente. **Un tipo común de proceso de ingesta de datos es ETL, que significa extraer, transformar y cargar**. Describe los tres pasos seguidos en un proceso ETL. Los datos se extraen de la fuente, se transforman al formato requerido y se cargan en alguna base de datos interna o propietaria. En un proceso ETL, también podemos incluir verificaciones automatizadas, como las expectativas que tenemos sobre ciertas columnas de datos. Por ejemplo, esperamos que la columna de temperatura siempre contenga un número. Incluir estas comprobaciones automatizadas en un proceso de datos ayuda a acelerar la fase de desarrollo e implementación del ciclo de vida, ya que los datos defectuosos o de baja calidad afectarán el modelo de aprendizaje automático.

1.2.4. Ingeniería de funciones y tienda de funciones

Ingeniería de funciones

Después de la fase de diseño, la ingeniería de funciones es el siguiente paso en el proceso de desarrollo del aprendizaje automático. La ingeniería de funciones es un componente clave del desarrollo del aprendizaje automático.

La **ingeniería de características es el proceso de seleccionar, manipular y transformar datos sin procesar en características**. Una característica es una variable, como una columna de una tabla. La forma en que creamos funciones a partir de los datos es una parte importante del desarrollo del aprendizaje automático. Podemos usar las funciones tal como aparecen en los datos sin procesar, pero también podemos crear las nuestras propias. Veamos un ejemplo.

Datos del cliente Por ejemplo, tenemos un conjunto de clientes y cierta información sobre sus pedidos. El número de pedidos y el gasto total de cada cliente son dos características diferentes. Para crear una nueva función, podríamos calcular el gasto promedio por cliente. De esta manera, hemos diseñado una nueva característica. Además de los datos de pedido en este ejemplo, a menudo hay muchos más datos disponibles en proyectos reales de aprendizaje automático y, por lo tanto, muchas posibilidades para diseñar funciones. Por lo tanto, la ingeniería de funciones es una parte importante del desarrollo del aprendizaje automático.

Evaluación de ingeniería de funciones

Una ponderación importante en la ingeniería de funciones es cuándo mantener la ingeniería o cuándo detenerla. Realizar una ingeniería de características integral puede producir un modelo muy preciso o lograr más estabilidad. Sin embargo, realizar una ingeniería integral de funciones también tiene un costo, lo que puede afectar el éxito de nuestro proyecto de aprendizaje automático.

- Más funciones pueden ser más costosas, ya que esto puede requerir costosos pasos de preprocesamiento.
- Más funciones también requieren más mantenimiento.
- Más funciones también pueden generar ruido o ingeniería excesiva.

¿Qué pasa si aumenta la cantidad de proyectos de ML?

Imagine que un científico de datos está trabajando en un nuevo proyecto de aprendizaje automático. Para el primer proyecto, podrían simplemente realizar la ingeniería de funciones una vez y entrenar el modelo. Pero, ¿qué pasa si crece la cantidad de proyectos de aprendizaje automático? **A medida que crece el número de proyectos y, por tanto, el número de modelos de aprendizaje automático, almacenar funciones de forma estructurada y centralizada puede acelerar enormemente el desarrollo de modelos de aprendizaje automático. Para hacer esto, una herramienta importante en MLOps es la tienda de funciones. Un almacén de características, como su nombre lo indica, es una herramienta para almacenar características o variables de uso común relevantes para el modelo de aprendizaje automático.**

La tienda de funciones

La tienda de funciones es el lugar central donde se pueden administrar las funciones. Al utilizar un almacén de funciones, un científico de datos puede encontrar las funciones adecuadas para su proyecto, definir nuevas funciones y utilizarlas para entrenar el modelo. También es el lugar donde se pueden monitorear las funciones. Al mismo tiempo, al utilizar un almacén de funciones, nos aseguramos de que las funciones estén listas para usarse como entrada para el modelo de aprendizaje automático en producción cuando lleguen nuevas muestras.

¿Cuándo utilizar una tienda de funciones? No tenemos que utilizar una tienda de funciones al desarrollar un modelo de aprendizaje automático. En algunos casos, puede resultar redundante utilizar una tienda de funciones. Los factores a considerar cuando decide utilizar una tienda de funciones son el costo computacional de las funciones. A veces, las funciones estarán listas como entrada para el modelo de aprendizaje automático tal como están. El uso de una tienda de funciones también depende de la cantidad de proyectos que tengamos. Las respuestas a estas preguntas determinarán si el desarrollo actual del aprendizaje automático se beneficia de una tienda de funciones o no.

1.2.5. Seguimiento de experimentos

Ahora analizaremos los experimentos de aprendizaje automático, qué es el seguimiento de experimentos y por qué es un concepto importante en MLOps.

El experimento de aprendizaje automático

Parte del desarrollo del modelo de aprendizaje automático consiste en realizar experimentos de aprendizaje automático. En un experimento de aprendizaje automático, entrenamos y evaluamos múltiples modelos de aprendizaje automático para encontrar el mejor. Como en cualquier experimento, probamos diferentes configuraciones para ver cuál funciona mejor.

¿Por qué es importante el seguimiento de experimentos?

Durante los experimentos de aprendizaje automático, podemos configurar diferentes modelos de aprendizaje automático, por ejemplo, regresión lineal o redes neuronales profundas. Podemos alterar los hiperparámetros del modelo, como el número de capas en una red neuronal. Podríamos utilizar diferentes versiones de los datos y diferentes scripts para ejecutar el experimento. También podemos usar diferentes archivos de configuración de entorno por experimento, como qué versión de Python o R se usa y qué bibliotecas. Al alterar cada uno de estos factores durante los experimentos, la cantidad de configuraciones diferentes puede volverse enorme. **Cada experimento también tiene un resultado diferente. Por eso es una buena idea realizar un seguimiento de las configuraciones y los resultados de cada experimento.**

Uso del seguimiento de experimentos en el ciclo de vida del aprendizaje automático

Además de rastrear todas las diferentes configuraciones de experimentos, el seguimiento de experimentos puede ayudar a comparar y evaluar experimentos, reproducir resultados de experimentos anteriores, colaborar con desarrolladores y partes interesadas e informar sobre los resultados a las partes interesadas.

¿Cómo realizar un seguimiento de los experimentos? Dependiendo de la madurez de nuestro desarrollo de aprendizaje automático, existen diferentes opciones para realizar un seguimiento de los experimentos. Podríamos empezar utilizando una hoja de cálculo en Excel, donde anotamos los detalles de cada experimento en cada fila. Si hacemos muchos experimentos, esto requerirá mucho trabajo manual. También podríamos crear nuestra propia plataforma de experimentos que rastree automáticamente el experimento durante el entrenamiento del modelo. Tener nuestra propia plataforma experimental nos permite crear una solución personalizada para nuestro proceso específico. Sin embargo, esto puede costar tiempo y esfuerzo. Esto se puede resolver mediante el uso de herramientas modernas de seguimiento de experimentos, ya que están diseñadas para la mayoría de los casos de uso de seguimiento de experimentos. Tenga en cuenta que esto puede resultar costoso. Los resultados del seguimiento del experimento se pueden almacenar en un almacén de metadatos.

Un experimento de aprendizaje automático y el preceso Imaginemos que estamos desarrollando un clasificador que clasifica si hay un perro o un gato en una imagen. Podemos comenzar el primer experimento entrenando una red neuronal con una capa oculta en mil imágenes. En el segundo experimento, incluimos cachorros y gatitos, aumentamos el conjunto de datos a dos mil imágenes y entrenamos una red neuronal con dos capas ocultas.

Si ejecutamos el experimento en el ejemplo de clasificar un perro o un gato, seguiremos los siguientes pasos.

1. Formulamos una hipótesis basada en los objetivos comerciales.
2. Recopilamos los datos necesarios
3. Definimos los hiperparámetros iniciales que deseamos probar.
4. Configuramos el seguimiento del experimento.
5. Ejecutamos el primer entrenamiento del modelo.
6. Durante el entrenamiento del modelo, ajustamos el modelo de aprendizaje automático al conjunto de datos que configuramos para el entrenamiento durante ese experimento
7. Evaluamos los resultados probando el modelo.
8. Registramos el modelo.
9. Visualizamos e informamos estos resultados al equipo o a las partes interesadas para determinar los próximos pasos.

1.2.6. Preparar el modelo para la implementación

Ahora que hemos analizado el diseño y desarrollo del ciclo de vida del aprendizaje automático, es hora de analizar la última fase: la implementación.

Entorno de ejecución

Durante la fase de desarrollo, los científicos de datos utilizaron datos de entrenamiento para desarrollar un modelo de aprendizaje automático.

Del desarrollo al despliegue

El desarrollo se lleva a cabo en el entorno de desarrollo, que suele ser el ordenador local de un científico de datos o un ordenador virtual que se puede controlar de forma remota, por ejemplo, en la nube. Una vez que se desarrolla el modelo de aprendizaje automático, debemos trasladarlo al entorno de producción. En el entorno de producción, el modelo de aprendizaje automático hará predicciones basadas en datos entrantes reales. Una vez implementado, el modelo está activo

y creará un impacto empresarial real. Sin embargo, implementar un modelo en el entorno de producción no es tan sencillo porque están configurados en diferentes entornos de ejecución. Un entorno de ejecución es el entorno en el que se ejecuta el modelo. Durante el desarrollo, el modelo se ejecuta en el entorno de ejecución de desarrollo.

Trabajar en diferentes entornos de ejecución es similar a trabajar en diferentes cocinas. Si creamos una receta en una cocina, los resultados de la misma receta pueden ser muy diferentes en otra cocina. Hay un juego diferente de sartenes, tal vez una cocina tenga electrodomésticos diferentes o un grupo diferente de chefs esté a cargo de la cocina. Todo esto puede afectar al plato que estemos preparando. De manera similar, en un entorno de ejecución de desarrollo, podemos usar diferentes versiones de Python y ciertas bibliotecas. Esto causa problemas porque es posible que el mismo código no funcione en diferentes entornos de ejecución o produzca resultados diferentes.

Mitigar diferentes entornos

Para mitigar el hecho de tener diferentes entornos, podemos utilizar máquinas separadas pero configuradas de forma idéntica. Esto es similar a una computadora normal. Tiene el hardware físico, el sistema operativo, las bibliotecas y la aplicación. Las bibliotecas y la aplicación contienen el entorno de producción y el modelo de aprendizaje automático. Esta es una solución sencilla, pero difícil de mantener y no escalable. Con cada actualización, tenemos que actualizar toda la máquina.

También podemos usar una o más máquinas virtuales en una máquina separada. Cada máquina virtual es como una versión virtual de una computadora física normal con un sistema operativo, bibliotecas y la aplicación. Una computadora que ejecuta máquinas virtuales tiene un hipervisor. Esto ayuda a distribuir los recursos, el hardware de la computadora, entre diferentes máquinas virtuales. Esto es más fácil de mantener pero requiere muchos recursos, porque necesitamos una computadora virtual para cada aplicación.

Por último, podríamos usar algo llamado contenedor. Esto nos permite ejecutar múltiples aplicaciones en una máquina. Los contenedores utilizan menos recursos que una máquina virtual y son más portátiles que las aplicaciones en una máquina virtual. Puede verse como una versión más ligera de la máquina virtual. **La implementación del modelo de aprendizaje automático como contenedor es actualmente el estándar para MLOps**.

Contenedores de beneficios

El uso de contenedores proporciona numerosos beneficios.

- Son más fáciles de mantener.
- Son muy portátiles.
- Se inician rápidamente.

Sin embargo, estos beneficios no significan que cada aplicación deba implementarse como un contenedor. Si la aplicación ha funcionado bien en una máquina virtual y no sufre problemas de diferentes entornos, está bien no utilizar contenedores.

1.2.7. Arquitectura de implementación de aprendizaje automático

Arquitectura de microservicios

Una vez que nos hayamos ocupado de los diferentes entornos de ejecución, debemos pensar en cómo implementar el modelo de aprendizaje automático. Esto también implica que debemos pensar en cómo configuramos la arquitectura.

Arquitectura monolítica frente a microservicio

Digamos que tenemos una tienda web con un servicio de pago, un servicio para el carrito de compras y un servicio para el inventario. Podemos ejecutar cada servicio en el mismo ordenador, también conocido como instancia única. En el desarrollo de software tradicional, las aplicaciones a menudo se creaban en una arquitectura monolítica. Esto significa que la aplicación es una aplicación uniforme que incluye todos los servicios.

Una solución diferente es la arquitectura de microservicios. La arquitectura de microservicios es, a diferencia de la arquitectura monolítica, una colección de servicios más pequeños que se pueden implementar de forma independiente. Si una aplicación falla en una arquitectura de microservicios,

solo falla el servicio por separado, mientras que en una arquitectura monolítica, fallará toda la aplicación. Una aplicación monolítica puede volverse compleja ya que todos los servicios están entrelazados y no son independientes. Esto también hace que sea más difícil escalar. El uso de una arquitectura de microservicios depende de la aplicación. Si nuestra aplicación es muy pequeña, tener un microservicio separado para cada parte aún más pequeña puede resultar costoso porque cada servicio requiere potencia de cálculo y debe mantenerse de forma independiente.

Inferencia

Es una práctica común implementar el modelo de aprendizaje automático como un microservicio. Esto nos permite utilizar el modelo de aprendizaje automático para hacer predicciones basadas en datos nuevos e invisibles.

Este proceso también se llama inferencia. Es el proceso en el que enviamos nuevos datos, por ejemplo, los datos de entrada de un cliente, para los cuales el modelo de aprendizaje automático inferirá un resultado. En este caso, el resultado es una predicción que contiene la probabilidad de que un cliente abandone.

Interfaz de programación de aplicaciones (API)

Para proporcionar comunicación entre microservicios, se utiliza una interfaz de programación de aplicaciones (o API). Una API es un conjunto de combinaciones de entrada y salida predefinidas que permite que diferentes servicios se comuniquen. Es similar al menú de un restaurante. Si se pide una pizza carciofo usando un menú, la cocina sabe que necesita juntar la base de la pizza, la berenjena, etcétera, y lo entregará una vez hecho para que el servicio pueda entregárselo al huésped. La entrada es el pedido en el menú, que se envía a la cocina, el modelo de aprendizaje automático, que devuelve el artículo pedido. No tener una API sería como no tener un menú y, como tal, no tendríamos una comunicación adecuada entre los servicios.

Flujo de datos API

1. Llegan datos de entrada nuevos e invisibles.
2. Los datos de entrada se envían a la API.
3. La API envía datos de entrada al modelo de aprendizaje automático.
4. El modelo de aprendizjae automático hace una predicción basada en nuevos datos de entrada.
5. La salida del modelo se envía de vuelta a la API.
6. La API comunica las predicciones del modelo a la aplicación.

Integración

Una vez que el modelo se ha implementado como un microservicio y la API nos permite inferir el modelo, se requiere un último paso. El último paso es integrar el modelo dentro del proceso de negocio. Esto es diferente para cada negocio, pero la mayoría de las veces implica conectar la API con el sistema que ya está implementado. Antes de utilizar el modelo de aprendizaje automático en producción, es una práctica común probar primero el modelo con una muestra de los datos para asegurarnos de que todo funcione como se esperaba.

1.2.8. CI/DI y estrategia de implementación

El proceso de CI/CD también forma parte de la fase de implementación.

CI/CD

El uso de integración continua e implementación continua (o CI/CD) es un concepto importante dentro del desarrollo de software. CI/CD se originó en DevOps y **se centra en automatizar la implementación de código. Es una serie de pasos para desarrollar, probar e implementar el código**. Al utilizar una canalización de CI/CD, los desarrolladores de software pueden realizar fácilmente cambios incrementales y luego impulsar estos cambios al entorno de producción. Estos mismos principios se pueden aplicar al desarrollo e implementación de código para modelos de aprendizaje automático.

Integración continua CI **La integración continua es la práctica en la que los cambios de código se integran continuamente de forma rápida y frecuente**. Cada cambio se prueba automáticamente cuando se confirman y fusionan. De esta manera, podemos identificar errores y fallos fácilmente y asegurarnos de que muchos desarrolladores puedan trabajar juntos en el mismo código.

Despliegue continuo CD La implementación continua funciona junto con la integración continua al automatizar la publicación del código que se validó durante el proceso de integración continua. **El objetivo de la práctica de la implementación continua es tener siempre código listo para producción**.

Canalización de CI/CD

Configurar una canalización de CI/CD puede resultar tedioso al principio, pero puede acelerar enormemente el proceso de implementación. Es como tener una lista de comprobaciones a realizar antes del lanzamiento de una nueva receta en un restaurante. Podemos crear fácilmente una nueva receta, realizar cambios en la receta y comprobar si cumple con los procesos actuales. En resumen, la integración continua es un conjunto de prácticas mientras se escribe el código para ejecutar el modelo de aprendizaje automático. La implementación continua es un conjunto de prácticas una vez completado el código.

Estrategias de implementación

Una vez que un modelo de aprendizaje automático está listo para implementarse, podemos elegir diferentes estrategias de implementación. Cada estrategia tiene una forma diferente de reemplazar el antiguo modelo de aprendizaje automático por el nuevo modelo de aprendizaje automático. Analizaremos tres estrategias de implementación:

- Implementación básica.- simplemente reemplazamos el modelo antiguo con el nuevo modelo en producción. Todos los datos de entrada nuevos se enviarán al nuevo modelo en lugar del modelo anterior.

- Implementación paralela.- enviamos nuevos datos tanto al modelo nuevo como al modelo anterior. Todavía utilizamos el modelo antiguo en producción. Se probará el resultado de ambos modelos para garantizar que el nuevo modelo funcione como se espera.

- Implementación canaria.- utilizamos el nuevo modelo en producción, pero solo para una pequeña parte de los nuevos datos entrantes. De esta manera, utilizamos el nuevo modelo de inmediato, pero en caso de que falle, solo un pequeño número de usuarios se verá afectado.

La implementación básica es la más fácil de implementar y utiliza la menor cantidad de recursos porque el nuevo modelo reemplaza completamente al anterior. Esto conlleva un alto riesgo en caso de que el modelo no funcione como se esperaba. La implementación paralela es similar en términos de implementación, pero utiliza más recursos ya que ejecutamos ambos modelos por completo en lugar de reemplazar uno por el otro. No hay riesgo cuando el modelo no funciona como se esperaba. La implementación canary es un poco más difícil de implementar pero utiliza menos recursos que tener dos modelos completamente implementados. Sin embargo, el riesgo es ligeramente mayor cuando el nuevo modelo no funciona como se esperaba.

1.2.9. Automatización y escalado

Hasta ahora, hemos analizado los componentes del diseño, desarrollo e implementación del aprendizaje automático. El ciclo de vida del aprendizaje automático es un proceso experimental, lo que significa que con frecuencia tenemos que ir y venir por las diferentes fases. Por tanto, la automatización puede ayudar enormemente a acelerar el ciclo de vida. Por ejemplo, nos permite repetir fácilmente los mismos experimentos varias veces.

Dado que el aprendizaje automático suele trabajar con grandes cantidades de datos, también es necesario configurar un sistema escalable. Por lo tanto, la automatización y el escalado son conceptos cruciales en MLOps.

Fase de diseño

La fase de diseño es la fase más importante dentro del ciclo de vida del aprendizaje automático. Sin un objetivo decente y datos de alta calidad, las otras dos fases podrían fracasar. Dado que el aprendizaje automático es multidisciplinario, como hemos visto por los diferentes roles involucrados, es importante que todos estén alineados. En términos de automatización y escalamiento, la fase de diseño sigue siendo un proceso manual. Sin embargo, el diseño se puede modelar para obtener el valor agregado, los requisitos comerciales y las métricas clave. Esto convierte la fase de diseño en un proceso estructurado en línea con las prácticas de MLOps. La adquisición de datos y los controles de calidad de los datos se pueden automatizar. Dado que la calidad del modelo de aprendizaje automático depende de la calidad de los datos, la automatización del proceso de adquisición de datos mejora las posibilidades de utilizar con éxito el aprendizaje automático en producción.

Fase de desarrollo

En la fase de desarrollo, utilizamos una tienda de funciones para rastrear y desarrollar funciones. Un almacén de funciones ahorra tiempo que se habría dedicado a crear las mismas funciones utilizadas en experimentos anteriores. Utilizamos el seguimiento de experimentos para automatizar el seguimiento del desarrollo del aprendizaje automático. Esto también ayuda a evaluar los modelos y alinearlos con las métricas clave establecidas en la fase de diseño. El seguimiento del experimento también garantiza que el proceso de desarrollo sea reproducible. Podemos encontrar qué configuraciones se utilizaron y cuáles fueron los resultados.

Fase de implementación

En la fase de implementación, podemos utilizar la contenedorización para mitigar diferentes entornos de ejecución. En términos de escalabilidad, tener aplicaciones en contenedores facilita el inicio de múltiples versiones de la misma aplicación cuando llegan más solicitudes. Por ejemplo, cuando la empresa crece y necesitamos predecir la pérdida de clientes para muchos clientes al mismo tiempo. Se utiliza una canalización de CI/CD para permitir cambios incrementales rápidos durante el desarrollo mediante el uso de la automatización. Esto permite que varios desarrolladores trabajen en el mismo código y ayuda a automatizar el proceso de desarrollo e implementación.

Una arquitectura de microservicios puede ser de gran ayuda a la hora de escalar el aprendizaje automático. Cada nuevo servicio se puede desarrollar e integrar de forma independiente sin afectar a otros servicios.

1.3. Mantenimiento del aprendizaje automático en producción

1.3.1. Monitoreo de modelos de aprendizaje automático

Seguimiento y reciclaje

El seguimiento y el reentrenamiento de los modelos de aprendizaje automático es la última parte de la fase de implementación. Primero examinaremos el seguimiento.

Monitoreo

Cuando un modelo de aprendizaje automático se implementa en producción, todavía no hemos terminado. En producción, el modelo de aprendizaje automático comenzará a hacer predicciones basadas en datos nuevos e invisibles. Para asegurarnos de que el modelo funcione como se esperaba, debemos monitorearlo.

Tipos de seguimiento

Podemos monitorear el modelo observando los datos de entrada y la salida del modelo, sus predicciones. A esto se le llama seguimiento estadístico. Por ejemplo, **podríamos monitorear la probabilidad prevista de que un cliente abandone.**

También podemos analizar métricas más técnicas del modelo. A esto se le llama monitoreo computacional. **Podría ser la cantidad de solicitudes entrantes que se realizan, el uso de**

la red del modelo o la cantidad de recursos que utiliza un servidor para mantener el modelo en ejecución.

Seguimiento estadístico y computacional Podemos verlo de esta manera. Podemos controlar la cocina en la que estamos cocinando de dos formas. En primer lugar, podemos controlar si todos los electrodomésticos siguen funcionando, si el gas y la electricidad están encendidos y si hay gente trabajando en la cocina. Todo lo que no tenga que ver con la comida. Esto sería un seguimiento computacional. En segundo lugar, podemos monitorizar las entradas y salidas de la cocina en cuanto a lo que se trata, la comida. Cuál es la calidad de los ingredientes que entran en la cocina y si el sabor de los platos que salen de la cocina está bien. A esto se le llama seguimiento estadístico.

Bucle de retroalimentación

Con el tiempo, descubriremos si ese cliente realmente ha abandonado. El resultado real también se conoce como verdad fundamental. Utilizando la verdad básica, podemos averiguar si el modelo funciona como se esperaba o si la calidad del modelo se deterioró con el tiempo. Este ciclo en el que comparamos el resultado del modelo con la verdad fundamental se llama ciclo de retroalimentación. El circuito de retroalimentación es una parte crucial para mejorar el modelo de aprendizaje automático. Utilizando el circuito de retroalimentación, podemos descubrir cuándo y por qué el modelo estaba equivocado. Podríamos, por ejemplo, ver que el modelo hace una predicción errónea para grupos de clientes concretos.

Es aconsejable monitorear las métricas tanto estadísticas como computacionales. Esto ayudará a ver dónde podría tener problemas el modelo de aprendizaje automático y nos permitirá mitigar esos problemas, que analizaremos en la siguiente lección.

1.3.2. Entrenando a un modelo de machine learning

Reciclaje después de los cambios

Lo inherente a los datos es que cambian con el tiempo. Es un hecho que el mundo está cambiando y, **dado que nuestro modelo de aprendizaje automático depende de los datos, estos cambios también afectan el modelo. Esta es también la razón por la que un modelo podría necesitar reentrenamiento**. Reentrenamiento significa que utilizamos nuevos datos para desarrollar una nueva versión del modelo de aprendizaje automático, de modo que aprenda y se ajuste a nuevos patrones.

Desviación de los datos

En un problema típico de aprendizaje automático, tenemos datos de entrada y datos de salida, que también se conocen como variable objetivo. Los datos de entrada son las variables utilizadas para predecir la variable objetivo. Si analizamos el caso de predecir si un cliente abandonará, tendremos datos sobre el cliente, que son los datos de entrada. La variable objetivo, en este caso, es si el cliente abandonará, representado por los números cero, no abandonó y uno abandonó. Hay dos cambios principales posibles en este tipo de conjunto de datos:

- Deriva de datos.- un cambio en los datos de entrada.
- Deriva de conceptos.- un cambio en la relación entre los datos de entrada y la variable objetivo. por ejemplo, cuando los mismos datos de entrada hacen que un cliente no abandone en lugar de abandonar. En ese caso, la relación entre los datos de entrada y salida ha cambiado. La deriva del concepto podría hacer que el rendimiento de nuestro modelo se deteriore porque los patrones en los que se entrenó previamente el modelo ya no se mantienen.

¿Con qué frecuencia volver a capacitarse?

La frecuencia con la que se debe volver a entrenar depende de varios factores. El primero es el entorno empresarial. Un entorno empresarial puede estar más sujeto a cambios que otros. Esto también puede ser identificado por un experto en la materia que tenga más conocimiento sobre el medio ambiente, por ejemplo, cuándo podría esperar un cambio. En segundo lugar, la frecuencia con la que se debe volver a capacitar también depende del costo de la misma. Entrenar un modelo requiere recursos. Dependiendo de la complejidad del modelo, el reciclaje requiere más recursos y, por tanto, más dinero. Por último, los requisitos comerciales influyen en la frecuencia con la que se

vuelve a entrenar el modelo. Si se requiere que el modelo tenga siempre una precisión superior al 90 % y un pequeño cambio en los datos hace que la precisión disminuya por debajo de ese umbral, será necesario volver a entrenar el modelo con más frecuencia. **La rapidez con la que disminuye la precisión del modelo también se denomina degradación del modelo.**

Métodos de reciclaje

Cuando volvemos a entrenar, se obtiene un nuevo modelo utilizando nuevos datos. Podríamos usar un modelo que solo use datos nuevos, de modo que haya un modelo separado entrenado con datos antiguos y un modelo entrenado con datos nuevos. También podríamos combinar datos nuevos y antiguos para desarrollar un nuevo modelo. Esto también dependerá del dominio, el costo y el rendimiento del modelo requerido.

Reentrenamiento automático

Dependiendo de la madurez del aprendizaje automático dentro de la empresa, también podríamos aplicar un reentrenamiento automático una vez que se detecte una cierta cantidad de datos o una deriva de concepto. Por ejemplo, cuando detectamos que la edad media de los clientes está cambiando.

1.3.3. Niveles de madurez de MLOps

Madurez de MLOps

Podemos observar el ciclo de vida del aprendizaje automático y determinar qué tan maduras son sus prácticas MLOps. La madurez de MLOps **tiene que ver con la automatización, la colaboración y el monitoreo dentro de los procesos de operaciones y aprendizaje automático en una empresa**. No significa necesariamente que un mayor nivel de madurez de MLOps sea mejor. Sin embargo, **muestra dónde existe potencial de mejora para permitir aún más el uso del aprendizaje automático dentro de la empresa**. Los niveles se aplican principalmente a la fase de desarrollo e implementación. La fase de diseño no se puede automatizar completamente ya que requiere participación humana de múltiples roles diferentes, pero se pueden usar plantillas para que la fase avance más rápida y suavemente.

Podemos distinguir tres niveles. Cada uno con su propio nivel de automatización, colaboración y monitoreo.

1. Procesos manuales.
2. Desarrollo automatizado.
3. Desarrollo e implementación automatizados.

En el nivel 1, no hay ninguna automatización y los equipos de operaciones y aprendizaje automático trabajan de forma aislada. En el nivel 2, hay automatización en el desarrollo de modelos de aprendizaje automático, y los equipos de operaciones y aprendizaje automático colaboran juntos cuando un nuevo modelo está listo para su implementación. En el nivel 3, el ciclo de vida del aprendizaje automático está completamente automatizado durante las fases de desarrollo e implementación.

Nivel 1: Procesos manuales En el nivel más bajo de madurez de MLOps, no existen procesos automatizados. Desde la ingesta de datos hasta la implementación del modelo, todo debe hacerse manualmente. Los equipos o roles que trabajan en el caso de uso lo hacen de forma aislada. Cada fase pasa a la siguiente y hay poca colaboración. Hay poca o ninguna trazabilidad. No se realiza un seguimiento de las funciones utilizadas, los experimentos y el rendimiento del modelo. Una empresa que acaba de empezar a utilizar el aprendizaje automático comenzará en este nivel. Dado que todos los procesos son manuales, el desarrollo y la implementación llevarán más tiempo e implicarán más trabajo, especialmente cuando algo sale mal durante una de las fases.

Nivel 2: Desarrollo automatizado En el segundo nivel de madurez de MLOps, ya no todos los procesos son manuales. Existe automatización en el proceso de desarrollo del modelo de aprendizaje automático. Por lo general, esto se hace mediante el uso de tiendas de funciones y capacitación de modelos automatizada. Existe un proceso de integración continua, pero una vez desarrollados, los modelos aún no se implementan automáticamente. Existe cierta colaboración entre los equipos de operaciones y aprendizaje automático. Sin embargo, la implementación de nuevos modelos

todavía se realiza de forma manual. Existe cierta trazabilidad en este nivel, especialmente durante el proceso de desarrollo. Es fácil reproducir modelos y realizar un seguimiento del rendimiento del modelo durante el desarrollo. Después de la implementación, suele haber una pequeña cantidad de seguimiento.

Nivel 3: Desarrollo e implementación automatizados En el nivel más alto de madurez de MLOps, el desarrollo y la implementación de modelos de aprendizaje automático están automatizados. Existe un proceso completo de CI/CD para desarrollar, probar e implementar nuevos modelos en producción. Existe una estrecha colaboración entre los diferentes roles involucrados en el proceso de aprendizaje automático. Los modelos de aprendizaje automático en producción se monitorean y, en algunos casos, incluso se activan automáticamente para que se vuelvan a entrenar.

1.3.4. Herramientas MLOps

Desde la aparición de MLOps, se han desarrollado muchas herramientas que pueden mejorar la eficiencia y confiabilidad de los procesos de aprendizaje automático. Algunas de estas herramientas son incluso de código abierto. Profundicemos en algunas posibles herramientas que podemos usar por componente analizado a lo largo de este curso. `https://www.datacamp.com/blog/infographic-data-and-machine-learning-tools-landscape`

Tienda de funciones

Para la tienda de funciones, hay varias herramientas disponibles, como Feast y Hopsworks. Feast es una tienda de funciones de código abierto; el nombre es un acrónimo de Feature and Store. Feast es una tienda de funciones autoadministrada, lo que significa que tenemos que administrarla nosotros mismos, lo que requiere más trabajo pero también proporciona más flexibilidad en comparación con otras tiendas de funciones. Hopsworks también es una tienda de funciones de código abierto, parte de la plataforma más grande Hopsworks. Por lo tanto, es más probable que se utilice si el resto de herramientas de Hopsworks ya están en uso.

4. Seguimiento de experimentos 00:58 - 01:23 Para el seguimiento de experimentos, podemos utilizar MLFlow, ClearML y Weights and Biases, entre otros. MLFlow y ClearML ofrecen herramientas para el ciclo de vida del aprendizaje automático, incluido el seguimiento de experimentos. MLflow se especializa en el desarrollo de aprendizaje automático, mientras que ClearML también proporciona herramientas para implementar modelos. Weights and Biases se centra principalmente en rastrear y visualizar los resultados de los experimentos.

Contenedorización

Para la contenedorización, Docker es la herramienta más popular para contener una aplicación. Kubernetes se utiliza para ejecutar la aplicación en contenedores, lo que permite la implementación y escalabilidad automáticas. Además de estas herramientas de código abierto, los proveedores de nube AWS, Azure y Google Cloud también ofrecen sus propias herramientas para ejecutar aplicaciones en contenedores.

Canalización de CI/CD

Para proporcionar canales completos de CI/CD existen herramientas como Jenkins y GitLab. Jenkins es una herramienta de CI/CD de código abierto, mientras que GitLab no lo es. Ambas herramientas permiten a los desarrolladores trabajar juntos en el código mediante un repositorio. Para cada proyecto, suele haber un repositorio independiente, que podemos ver como un directorio que contiene todo el código del proyecto.

Monitoreo

Existe una amplia gama de herramientas para monitorear proyectos de aprendizaje automático. Podemos distinguir herramientas que se centran en el seguimiento del modelo de aprendizaje automático y herramientas que monitorean los datos. Tanto Fiddler como Great Expectations proporcionan herramientas de seguimiento estadístico. Fiddler se centra en el rendimiento del modelo, por ejemplo, qué tan bien están funcionando las predicciones de nuestro modelo. Great Expectations se centra en el seguimiento de datos, por ejemplo, cuántos datos faltan en una determinada columna.

Plataformas MLOps

También hay herramientas disponibles que proporcionan una plataforma completa del ciclo de vida del aprendizaje automático. Cada proveedor de nube, AWS, Azure y Google, tiene uno. Se llaman AWS Sagemaker, Azure Machine Learning y Google Cloud AI Platform. Las herramientas que abarcan todo el ciclo de vida del aprendizaje automático proporcionan herramientas para cada tarea del ciclo de vida. Esta podría ser una herramienta para realizar exploración y procesamiento de datos, pero también un almacén de características y una herramienta de capacitación de modelos.

Capítulo 2

Desarrollo de modelos de aprendizaje automático para la producción

2.1. Pasar de la investigación a la producción

2.1.1. Adoptar una mentalidad MLOps

Empezaremos analizaremos por qué casi el 90 % de los experimentos de aprendizaje automático no llegan a producción. Veremos cómo adoptar una mentalidad MLOps y qué hace que un experimento de ML esté listo para pasar a producción. También discutiremos la deuda técnica del aprendizaje automático.

Operaciones mlop

MLOps significa Operaciones de aprendizaje automático. *MLOps es una práctica que se centra en la colaboración entre científicos de datos y equipos de operaciones para garantizar la implementación y gestión exitosa de modelos de aprendizaje automático en producción.* MLOps adecuados ayudan a garantizar que los experimentos de aprendizaje automático se prueben adecuadamente y estén listos para implementarse y escalarse.

Experimentos de aprendizaje automático

Un aspecto importante de MLOps es la experimentación y prueba continua de diferentes modelos de aprendizaje automático. Estos experimentos implican entrenar y evaluar los modelos en varios conjuntos de datos para determinar cuál produce los resultados más precisos y confiables. En el aprendizaje automático, es fundamental considerar cuidadosamente la selección de modelos, ya que **el rendimiento de un modelo puede afectar en gran medida el éxito general del proyecto**. Por lo tanto, es esencial evaluar cuidadosamente las diferentes opciones y elegir el modelo que funcione mejor con los datos proporcionados. Este proceso de experimentación y selección de modelos puede llevar mucho tiempo, pero es un paso crucial para garantizar el éxito de cualquier proyecto de aprendizaje automático.

De los experimentos a la producción

Un experimento de aprendizaje automático está listo para pasar de la fase de experimentación a la producción cuando se ha documentado, probado y validado exhaustivamente para garantizar su precisión y confiabilidad. Esto puede incluir ejecutar el modelo en varios conjuntos de datos y parámetros y probar el modelo en un entorno del mundo real. Además, **el modelo debe ser monitoreado de cerca para garantizar que funcione como se espera y que cualquier cambio se alinee con los resultados deseados. Finalmente, el modelo debe implementarse en un entorno seguro y escalable para manejar grandes volúmenes de datos y tráfico.**

Por qué fallan la mayoría de los experimentos de aprendizaje automático

Desafortunadamente, la mayoría de los experimentos de aprendizaje automático no logran llegar a producción. Algunas razones comunes incluyen:

- Falta de metas y objetivos claros.
- Mala calidad de los datos.
- Arquitecturas de modelos demasiado complejas.
- Datos de entrenamiento insuficientes.
- Modelos sobreajustados o insuficientes.

Es esencial considerar cuidadosamente estos factores y abordar cualquier problema antes de pasar un experimento a producción para aumentar las posibilidades de éxito.

Deuda técnica

La deuda técnica **se produce cuando el código se apresura y no se prueba o valida exhaustivamente**, lo que genera errores o errores cuya resolución puede resultar costosa y llevar mucho tiempo. **También se puede incurrir en deuda técnica con documentación desactualizada o faltante en cualquier proceso de selección de modelo de codificación/ML. Para evitar incurrir en deuda técnica, es fundamental priorizar la calidad y corrección del código y la documentación desde el principio**. Si seguimos las mejores prácticas y nos tomamos el tiempo para probar y validar el código correctamente, podemos garantizar el éxito y la sostenibilidad a largo plazo de nuestros proyectos de aprendizaje automático.

2.1.2. Escribir código ML mantenible

Discutiremos cómo podemos abordar la deuda técnica con la estructura, el control de versiones y la documentación del proyecto adecuados y cómo el código mantenible conduce a aplicaciones de aprendizaje automático adaptables.

Estructuración del proyecto

El primer paso para escribir código ML mantenible es:

1. Estructurar el proyecto de manera lógica.
2. Agrupar archivos relacionados en carpetas separadas.
3. Asegurarse de que los archivos tengan el nombre y la etiqueta adecuados. Esto facilita la identificación y localización de archivos cuando sea necesario.

Estructura de proyecto de muestra

- README.md
- data
  - raw
  - processed
  - interim
- models
  - model1.py
  - model2.py
  - model3.py
- notebooks
  - exploration.ipynb
  - model_training.ipynb
  - model_evaluation.ipynb
- requirements.txt

Este es un ejemplo de una buena estructura para un repositorio de aprendizaje automático. Mantiene los archivos relevantes organizados y de fácil acceso. El archivo README.md proporciona una descripción general de alto nivel del repositorio, lo que facilita que alguien nuevo comprenda su propósito y cómo usarlo. El directorio de datos contiene todos los datos utilizados para entrenar y evaluar los modelos. Está organizado en subdirectorios para datos sin procesar, procesados y provisionales. Esto facilita ver de dónde provienen los datos, qué se ha hecho y cómo se utilizan. El directorio de modelos contiene todos los scripts para crear y entrenar diferentes modelos, lo que permite ver qué modelos se han probado y qué tan bien funcionan. El directorio de cuadernos ayuda a explorar y visualizar los datos. También es posible que deseemos incluir un directorio "fuente", que contendría todo el código fuente del proyecto. El directorio fuente incluiría código para el preprocesamiento de datos, ingeniería de características, entrenamiento y evaluación de modelos. Esta estructura mantiene todo organizado y fácil de entender, lo cual es importante cuando se trabaja con proyectos complejos de aprendizaje automático.

Versionado del código

Una forma de mantener el código de aprendizaje automático es utilizar un sistema de control de versiones. Esto le permite realizar un seguimiento de los cambios realizados en su código, lo cual es muy útil. Por ejemplo, le permite revertir rápidamente un cambio realizado. Además, si encuentra un error en su código, un sistema de control de versiones puede ayudarlo a identificar el origen del problema comparando la versión actual del código con versiones anteriores para ver qué cambios pueden haberlo causado. Otra ventaja es que permite a los desarrolladores trabajar en una base de código en paralelo, y el sistema se encarga de fusionar los cambios realizados por cada miembro del equipo. Un sistema de control de versiones es una herramienta invaluable para cualquier proyecto de desarrollo de software, no solo para aquellos que involucran aprendizaje automático.

Documentación

También es esencial documentar el código y la estructura del proyecto. Esto incluye explicar el propósito de cada archivo y función, describir cómo usar el código y proporcionar instrucciones sobre cómo implementar el modelo ML. Esto facilita que otros comprendan y utilicen el código.

Adaptabilidad del código

Uno de los beneficios clave del código mantenible es que es más fácil de entender, modificar y actualizar. Si el código está bien estructurado y es fácil de leer, será más fácil para los desarrolladores saber cómo funciona y realizar los cambios necesarios. La mantenibilidad también puede ahorrar tiempo y recursos a largo plazo. Cuando el código está bien estructurado y bien documentado, es más fácil realizar cambios sin introducir nuevos errores. Esto ahorra tiempo y recursos que de otro modo se gastarían en depurar y solucionar problemas. El código mantenible es esencial para crear aplicaciones de aprendizaje automático adaptables. Mantener nuestras bases de código limpias y bien organizadas facilita la integración de nuevas funciones o tecnologías según sea necesario. Esto puede ayudar a que sus aplicaciones de aprendizaje automático se mantengan al día con los requisitos cambiantes y las fuentes de datos, preparándonos para el éxito a largo plazo.

2.1.3. Redacción de documentación de aprendizaje automático eficaz

Entenderemos por qué es necesaria una documentación eficaz, concisa y reutilizable para las aplicaciones de máquina que se implementarán, y podremos explicar las características clave de dicha documentación.

Los componentes de una excelente documentación de ML

Examinaremos seis áreas principales de documentación:

- Fuentes de datos.
- Esquemas de datos.
- Métodos de etiquetado.
- Experimentación con modelos.
- Criterios de selección.

- Entornos de entrenamiento.

Documentar las fuentes de datos Documentar las fuentes de datos nos permite establecer procesos para evaluar la calidad de nuestros datos al proporcionar una base para la comparación e identificar posibles errores o inconsistencias. También nos ayuda a realizar un seguimiento de dónde provienen los datos y si podemos acceder a ellos en el futuro. También nos permite configurar procesos para evaluar la calidad de nuestros datos e iterar sobre la calidad si es necesario.

Esquemas de datos Una vez que hayamos documentado de dónde provienen nuestros datos, la siguiente área a documentar son los esquemas de datos. **Un esquema de datos es una estructura que describe la organización de los datos**. Por ejemplo, un esquema de base de datos relacional especificaría las tablas, los campos de cada tabla y las relaciones entre campos y tablas. Al escribir esquemas en nuestra documentación, podemos proporcionar estructura para datos que de otro modo estarían desorganizados y permitir que otros sepan de qué tipo de datos está aprendiendo nuestro modelo.

Métodos de etiquetado (para clasificación) A menudo nos ocupamos de tareas de clasificación supervisadas. Si ese es el caso, entonces queremos documentar cómo llegamos a las etiquetas finales para la variable de respuesta. Por ejemplo, cuando se trabaja con datos sin procesar no estructurados, como imágenes, es posible que no se hayan anotado ni etiquetado previamente. Comprender exactamente cómo se recopilaron y etiquetaron los datos es vital para la **reproducibilidad**. También podemos usar esto para evaluar la calidad de las etiquetas, lo que afecta la **confiabilidad** de los modelos de aprendizaje automático. El **rendimiento** del modelo también se puede mejorar con acceso a datos mejor etiquetados. Los métodos de etiquetado pueden evolucionar si las etiquetas varían o si se dispone de mejores fuentes de etiquetado.

Pseudocódigo modelo El pseudocódigo del modelo es una representación simplificada de los pasos involucrados en la construcción de su modelo de aprendizaje automático. Esto a menudo incluye escribir los pasos de nuestro trabajo de ingeniería de características, los componentes de un conjunto de tuberías y delinear las entradas y salidas esperadas de un modelo. Esta documentación le permite realizar un seguimiento de estos pasos para futuras referencias y fines de depuración.

Experimentación de modelos + selección.

Una vez que tengamos nuestros datos y sepamos cómo fueron etiquetados, es hora de documentar cómo ejecutamos nuestros experimentos y seleccionamos nuestros modelos de aprendizaje automático. Esto es importante porque permite realizar un seguimiento del desarrollo del modelo y compartirlo para que otros puedan repetir el proceso para mejorarlo. También queremos documentar

- Qué arquitecturas de modelo se consideraron.
- Las métricas utilizadas para decidir qué modelo se consideró "mejor".
- Las combinaciones de hiperparámetros consideradas al entrenar los modelos.

De esta manera podemos obtener una imagen completa de cómo y por qué se tomó la decisión de elegir un modelo particular y una combinación de hiperparámetros y potencialmente repetir esto en el futuro.

Entornos de formación

Además de documentar el proceso de selección del modelo, también debemos documentar cómo era nuestro entorno de capacitación, incluidos los paquetes de terceros (como scikit-learn) o las semillas aleatorias que usamos durante la capacitación. Este paso es vital para ayudar a cualquiera a reproducir los resultados de nuestra capacitación en aprendizaje automático. También puede afectar el rendimiento del algoritmo de aprendizaje automático. Por ejemplo, si los datos se transforman o se entrena un modelo con una semilla aleatoria diferente a aquella en la que se implementará el algoritmo de aprendizaje automático, es posible que el algoritmo no funcione como se esperaba.

2.2. Garantizar la reproducibilidad

2.2.1. Diseño de experimentos reproducibles.

Hablaremos sobre el diseño de experimentos de ML reproducibles.

Experimentos reproducibles

La reproducibilidad en el aprendizaje automático es esencial para generar confianza y garantizar la precisión de los resultados de los modelos de aprendizaje automático. **La reproducibilidad permite la replicación de resultados y mejora la colaboración con otros desarrolladores e investigadores**. **Los experimentos reproducibles ayudan a reducir el riesgo de sesgo y garantizan la integridad del proceso de investigación y los resultados que produce.** En pocas palabras, al adherirnos a los principios de reproducibilidad, podemos tener más confianza en la precisión y confiabilidad de los modelos desarrollados.

MLFlow

MLflow es una plataforma de código abierto desarrollada por Databricks que ayuda a rastrear y administrar experimentos de aprendizaje automático. Permite a los usuarios rastrear y administrar fácilmente dependencias, versiones de código y configuraciones de experimentos, lo que facilita la creación de canales de aprendizaje automático reproducibles. MLflow también es una gran plataforma para la colaboración, ya que varios usuarios pueden acceder a experimentos y ver los resultados. MLflow facilita la reproducción de canales de ML completos de una manera rápida y eficiente.

Ejemplo de uso de Mlflow Veamos un ejemplo del uso de mlflow con scikit-learn. Aquí tenemos algunas importaciones estándar para crear un clasificador RandomForest con dos importaciones adicionales de mlflow. Podemos ver que mlflow tiene soporte scikit-learn incorporado.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Nuevas importaciones de mlflow
import mlflow
import mlflow.sklearn

with miflow.start_run(): #Comenzar a correr un MLflow asumiendo que tenemos unos datos preparados
    # Construir un entrenar el modelo
    rf = RandomForestClassifier()
    rf.fit(X_train, y_train)

    # Registrar parámetros e información del modelo
    mlflow.log_param("n_estimators", rf.n_estimators)
    mlflow.sklearn.log_model(rf, "model")

    y_pred = rf.predict(X_test) # Evaluar el modelo
    accuracy = accuracy_score(y_test, y_pred)
    mlflow.log_metric("accuracy", accuracy) # Registrar la métrica de precisión de la prueba
```

Este código demuestra el uso básico de MLflow para registrar parámetros, información del modelo y métricas de un modelo scikit-learn. Inicia una ejecución de MLflow con $mlflow.start\_run()$, registra parámetros e información del modelo y registra métricas como la precisión. Estos valores registrados se pueden rastrear, almacenar y comparar en la interfaz de usuario de mlflow, lo cual es un paso importante en la reproducibilidad.

Código de seguimiento

MLflow facilita el seguimiento de las versiones y los cambios del código registrándolos, además de comparar diferentes versiones del código. Esto ayuda a garantizar que los experimentos se puedan reproducir exactamente, ya que le permite identificar qué versión del código se utilizó para

producir un conjunto determinado de resultados. **Esto hace que sea mucho más fácil depurar y solucionar problemas de código, así como reproducir experimentos**. El seguimiento del código con MLflow es esencial para crear experimentos reproducibles en aprendizaje automático.

Registros modelo

Los registros de modelos **son depósitos centralizados de modelos y sus metadatos, como**

- Versiones de modelos.
- Métricas de rendimiento.
- Detalles del entorno.

MLflow se puede utilizar para administrar registros de modelos registrando, almacenando y comparando diferentes versiones de modelos, lo que permite la reproducción de canales completos de ML. Esto también permite la comparación de modelos, asegurando la precisión y confiabilidad de los modelos producidos. **Al utilizar MLflow para gestionar registros de modelos, los investigadores pueden estar seguros de que sus experimentos son reproducibles y sus resultados precisos.**

Reproducibilidad del experimento MLflow se puede utilizar para garantizar la reproducibilidad del experimento mediante el seguimiento y el registro de datos de entrada, código y configuraciones. Esto permite validar los hallazgos y replicar los resultados, permitiendo que otros verifiquen y desarrollen el trabajo, y garantiza resultados consistentes en diferentes ejecuciones.

Revisar la documentación

Una buena documentación es esencial para la investigación y el desarrollo del ML reproducible. La documentación adecuada debe incluir documentación clara y detallada de los datos de entrada, el código, la configuración utilizada en un experimento y los resultados del experimento. También es importante hacer que la documentación sea accesible para que otros la vean y mantener un registro actualizado del experimento. Seguir estos principios de buena documentación garantizará que los experimentos sean reproducibles y que otros puedan verificar y desarrollar el trabajo.

2.2.2. Ingeniería de características

Veremos algunos de los métodos comunes que utilizan los ingenieros de ML para agregar y transformar datos a su estado más óptimo para ayudar a nuestros algoritmos de aprendizaje a realizar predicciones mejores y más precisas.

Introducción a la ingeniería de características.

Es el proceso de transformar los datos de entrenamiento para maximizar el rendimiento del proceso de aprendizaje automático y reducir la complejidad de los cálculos. Implica, por ejemplo, agregar datos de múltiples fuentes, construir nuevas características y aplicar transformaciones de características. Al diseñar nuestras funciones de esta manera, los canales de ML se pueden optimizar para mejorar la precisión y eficiencia de los modelos desarrollados.

Agregar datos de múltiples fuentes

Agregar datos de múltiples fuentes es un aspecto importante de la ingeniería de funciones. Podemos combinar datos de diferentes conjuntos de datos e incluir múltiples tipos de datos en nuestro conjunto de entrenamiento. Esto puede mejorar en gran medida la precisión de los modelos y permitir el uso de modelos más complejos utilizando datos más complejos.

Ejemplo de agregación de datos Este es un ejemplo de una clase *DataAggregator* simple que carga datos de tres fuentes separadas usando el método *pd.read_csv*. Combina todos los datos en un solo marco de datos utilizando el método *pd.concat*. Nuestra clase tiene dos métodos: ajustar y transformar. El método de ajuste no hace nada y se incluye por compatibilidad con scikit-learn. El método de transformación es donde ocurre el verdadero trabajo.

```
class DataAggregator:
    def __init__(self):
    pass

    def fit(self, X, y=None):
 return self  # nada que ajustar

    def transform(self, X, y=None):
 # Cargar datos de multiples fuentes
 data1 = pd.read_csv('data1.csv')
 data2 = pd.read_csv('data2.csv')
 data3 = pd.read_csv('data3.csv')

 # Combinar los datos de todas las fuentes (incluyendo X) en un solo marco de datos
 aggregated_data = pd.concat([X, data1, data2, data3], axis=0)

 return aggregated_data  # Retornar los datos agregados
```

Construcción de características

La construcción de características **es el proceso de combinar funciones existentes para crear nuevas características**. Por ejemplo, podemos sumar dos características numéricas para construir una característica. La construcción suele realizarse con la ayuda de expertos en el campo que conocen bien los datos. La construcción puede ayudar a mejorar el rendimiento y la interpretabilidad del modelo al considerar nuevas características más relevantes.

Ejemplo de construcción de características. Al igual que la clase DataAggregator, solo usamos el método transform para hacer nuestro trabajo. El método de transformación construye dos características nuevas restando la media de dos columnas de cada valor en esas columnas. Las nuevas características representan la desviación de cada punto de datos de la media en cada columna. Puede crear nuevas funciones utilizando una amplia gama de operaciones, como calcular diferencias o crear interacciones entre varias funciones.

```
class FeatureConstructor:
    def __init__(self):
    pass

    def fit(self, X, y=None):
 return self  # nada que ajustar

    def transform(self, X, y=None):
 # Calcular la media de cada columna en los datos
 mean_values = X.mean()

 # Crear nuevas características basados en los valores medios
 X['mean_col1'] = X['col1'] - mean_values['col1']
 X['mean_col2'] = X['col2'] - mean_values['col2']

 return X  # Retornar los datos con las nuevas características
```

Transformaciones de características

La transformación de características es el proceso de transformar funciones existentes en funciones más utilizables para ML. Ejemplos de cómo hacer esto incluyen normalizar datos y eliminar valores atípicos. Al igual que con la construcción y el aprendizaje, nuestro objetivo con las funciones transformadoras es mejorar el rendimiento del modelo. Por ejemplo, el transformador $StandardScaler$ en $scikit-learn$ escala las características para que tengan una media de 0 y una desviación estándar de 1 y se usa para obtener datos que viven en la misma escala entre sí, lo que puede ayudar a algunos modelos de ML.

Selección de característica

La selección de características se realiza seleccionando un subconjunto de un conjunto más grande de características. Esto suele hacerse eliminando características redundantes e irrelevantes. **La selección de características ayuda a reducir el sobreajuste, mejorar el rendimiento del modelo y mejorar la interpretabilidad del modelo.**

Echemos un vistazo a un ejemplo de canalización de ingeniería de características. Esta canalización reúne muchos pasos de ingeniería de características en una clase ejecutable. Vemos que aquí se utilizan nuestro agregador de datos, constructor de características y escalador estándar. La canalización utiliza una técnica llamada prueba de Chi-cuadrado para realizar la selección de funciones determinando qué funciones son más relevantes para la tarea. Se seleccionan las 10 mejores características. Una vez que la canalización se ha adaptado a los datos, se puede utilizar para transformar datos futuros de la misma manera.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.pipeline import Pipeline

pipeline = Pipeline([ # Definir la canalización de ingeniería de características
 ('aggregator', DataAggregator()), # Agregar datos de multiples fuentes
 ('constructor', FeatureConstructor()), # Construir nuevas características
 ('scaler', StandardScaler()), # Escalar las características
 ('selector', SelectKBest(chi2, k=10)) # Seleccionar las 10 mejores características
])
X_transformed = pipeline.fit_transform(X) # Ajustar y transformar los datos
```

2.2.3. Versionado de datos y modelos.

En el aprendizaje automático, es importante realizar un seguimiento de las diferentes versiones de los datos y modelos utilizados en los experimentos. Esto se conoce como versionado. Al versionar los datos y los modelos, podemos garantizar la reproducibilidad y trazabilidad de los experimentos. Esto significa que podemos volver fácilmente a una versión anterior de los datos o del modelo si es necesario y ver cómo progresó el experimento a lo largo del tiempo.

Versiones mayores y menores

Hay dos tipos de versiones: mayor y menor. **El control de versiones mayor se utiliza para indicar un gran cambio en los datos o el modelo, como una nueva característica, mientras que el control de versiones menor indica un cambio pequeño, como una corrección de error**. Al utilizar versiones mayores y menores, podemos ver fácilmente cómo los datos o el modelo han cambiado con el tiempo y qué cambios se realizaron. Esto nos permite realizar un seguimiento de las diferentes versiones de datos y modelos utilizados en los experimentos.

Versionado de datos de entrenamiento

Para versionar los datos de entrenamiento, podemos usar etiquetas principales/menores únicas o marcas de tiempo para identificar diferentes versiones de los datos. Al hacerlo, podemos rastrear qué datos se utilizaron para qué experimentos y volver a una versión anterior de los datos si es necesario para ver cómo han cambiado con el tiempo. Por ejemplo, digamos que la versión 1.0 es nuestro conjunto de datos inicial. La versión 1.1 de datos incluyó transformaciones de funciones adicionales como el escalado. La versión 1.2 de datos agregó un método de selección de características como Chi-cuadrado. Finalmente, la versión 2.0 de datos incluye una nueva fuente de datos que marca nuestra primera actualización importante de nuestros datos.

Tiendas de funciones

Un almacén de funciones es un repositorio central para almacenar y administrar diferentes versiones de funciones. Las tiendas de funciones rastrean las versiones de las funciones, lo que mejora la colaboración y reduce la duplicación del trabajo. Al utilizar un almacén de funciones, podemos asegurarnos de que los experimentos utilicen la misma versión de los datos y modelos.

Versionado de modelos ML

Al igual que los datos, también podemos versionar los modelos de ML para garantizar la reproducibilidad y permitir reversiones. A menudo, incrementamos las versiones de nuestro modelo junto con nuestros datos, pero a veces nuestra versión del modelo puede cambiar independientemente de nuestros datos. Por ejemplo, también podemos empezar con un modelo 1.0, nuestro modelo inicial. Cuando incrementamos a las versiones de datos 1.1, podemos ejecutar un ciclo de ajuste fino y encontrar que el mismo modelo RandomForest sigue siendo el ganador, incluso si tiene hiperparámetros diferentes a los de la versión 1.0. Cuando lleguemos a la versión 1.2 de datos de entrenamiento, tal vez el experimento muestre que un modelo XGBoost está funcionando mejor ahora y tengamos un cambio de versión importante con la versión 2.0 afinada en la versión de datos 1.2. Cuando nuestros datos cambian a la versión 2.0 pero nuestro modelo sigue siendo XGBoost, aún sería un cambio menor. Depende de usted decidir qué es o no un cambio mayor o menor; este es solo un ejemplo.

Tiendas de modelos

Al igual que una tienda de funciones, una tienda de modelos es un repositorio para gestionar versiones de modelos. Usando una tienda de modelos, podemos rastrear versiones de los modelos y retroceder a diferentes versiones. Los almacenes de modelos se utilizan a menudo junto con los almacenes de características, lo que proporciona un control total sobre los datos y las versiones de los modelos.

Ejemplo de versionado de modelo con MLflow

Podemos usar mlflow para registrar versiones del modelo. Aquí lo configuramos en "1.0". Luego guardamos el modelo usando la función *log_model* de mlflow. Este ejemplo es bastante simple y, en la práctica, el control de versiones es más complicado que esto, pero podemos tener una idea de cómo funcionaría.

```
import mlflow

# Comenzar corriendo un nuevo mlflow
with mlflow.start_run():
    # Registrar la versión del modelo como parámetro
    mlflow.log_param("model_version", "1.0")

    # Entrenar y guardar el modelo
    model = train_model()
    mlflow.sklearn.log_model(model, "model")
```

2.3. Desarrollo de modelos de aprendizaje automático para producción

2.3.1. ML en entornos de producción

Empaquetado de modelos ML

Profundizaremos en los métodos para preparar modelos de ML para su implementación. Examinaremos la importancia de serializar y contenerizar los modelos y sus entornos.

Por qué es importante el embalaje

Los modelos de aprendizaje automático deben empaquetarse e implementarse en un entorno de producción, para que funcionen mejor, sean compatibles con diferentes sistemas y sean fáciles de implementar. **Los métodos de embalaje incluyen:**

- Serialización.- Es simple y liviana, pero puede no ser adecuada para modelos complejos.
- Empaquetado del entorno.- captura todo el entorno del software, pero puede resultar pesado.
- Contenedorización.- proporciona un entorno portátil y aislado, pero requiere experiencia en tecnologías de contenedores.

Cómo empaquetar modelos ML

- Serialización.- Convierte un modelo de ML a un formato que se puede almacenar y recuperar.
- Empaquetado del entorno.- Crea un entorno coherente y reproducible en el que se pueda ejecutar el modelo de ML.
- Contenedorización.- Empaqueta el modelo, sus dependencias y el entorno en el que se ejecuta en un contenedor que se puede implementar fácilmente.

Serialización de modelos de aprendizaje de ciencia ficción Los modelos serializados se pueden cargar en la memoria y utilizar para predicción o puntuación. Los modelos de Scikit-learn se pueden serializar fácilmente usando la biblioteca pickle, como se muestra a la izquierda y usando el formato HDf5 a la derecha.

```
import pickle

model = .... # Entrenar un modelo

# Serializar el modelo a un archivo
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)

# Cargar el modelo serializado
with open('model.pkl', 'rb') as f:
    model = pickle.load(f)
```

```
import h5py
import numpy as np

model = .... # Entrenar un modelo

# Serializar el modelo a un archivo HDF5
with h5py.File('model.h5', 'w') as f:
    f.create_dataset('model_weights',
      data = joblib.dump(model))

# Cargar el modelo serializado
with h5py.File('model.h5', 'r') as f:
    model = joblib.load(f['model_weights'][:])
```

Serialización de modelos PyTorch y Tensorflow PyTorch y Tensorflow son bibliotecas de Python populares para el aprendizaje profundo y proporcionan una variedad de herramientas para entrenar e implementar modelos de ML. En PyTorch, la serialización se puede realizar mediante el uso de las funciones *torch.save* y *torch.load*, que permiten guardar y cargar modelos de PyTorch hacia y desde archivos. De manera similar, en Tensorflow, la serialización se puede lograr utilizando la API $SavedModel$ de Tensorflow. Esta API permite a los usuarios guardar y cargar modelos en una variedad de formatos, incluido el formato Tensorflow $SavedModel$, que está optimizado para servir modelos en entornos de producción. Cualquiera de las opciones permite almacenar el modelo y utilizarlo para predicciones o entrenamiento adicional en un momento posterior.

Empaquetado del entorno ML con Docker El empaquetado del entorno es un paso importante en la implementación de modelos de aprendizaje automático, ya que ayuda a garantizar que el modelo se ejecutará de manera consistente en cualquier lugar, independientemente de cualquier cambio en el sistema subyacente o las dependencias. Los entornos virtuales, como conda o virtualenv, se pueden utilizar para crear un entorno coherente y reproducible en el que se pueda ejecutar el modelo ML. Los entornos virtuales permiten que diferentes proyectos tengan sus propios entornos aislados, lo que garantiza que los cambios en un proyecto no afecten a otros proyectos. Docker también se puede utilizar para embalaje ambiental. Los contenedores Docker son unidades autónomas que se pueden implementar y ejecutar fácilmente en cualquier entorno. Los contenedores Docker proporcionan un entorno coherente y reproducible para los modelos de aprendizaje automático, lo que garantiza que el modelo se ejecutará de forma coherente dondequiera que lo implementemos.

En este Dockerfile de ejemplo, comenzamos con una imagen de Python 3.8 como imagen base. El archivo de requisitos se copia a la imagen y las dependencias requeridas se instalan usando pip. Luego, el modelo ML y sus dependencias se copian en la imagen y se configura el punto de entrada para ejecutar el modelo mediante el script *run_model.py*. Este Dockerfile se puede integrar en una imagen usando el comando Docker Build y luego la imagen se puede ejecutar como un contenedor usando el comando Docker Run.

```
# Usar una imagen existente como una imagen base
FROM python:3.8-slim
```

```
# Conjunto del directorio de trabajo
WORKDIR /app

# Copiar el archivo de requisitos y las dependencias
COPY requirements.txt .

# Instalar las dependencias
RUN pip install -r requirements.txt

# Copiar el modelo ML y estas dependencias
COPY model/ .

# conjunto de comandos de ejecución
ENTRYPOINT ["python", "run_model.py"]
```

Experimento ->Flujo de trabajo de Docker

A continuación se muestra un flujo de trabajo de ejemplo para experimentar, serializar, contener un modelo de aprendizaje automático y su entorno, e implementarlo mediante Docker. En este flujo de trabajo,

1. Entrenamos un modelo de aprendizaje automático.
2. Serializamos el modelo en un archivo.
3. Serializamos las dependencias del modelo en un archivo, sus dependencias y el entorno se colocan en contenedores mediante Docker.
4. Implementamos la imagen de Docker en un entorno de destino y se inicia un contenedor de Docker desde la imagen.
5. El modelo ML se puede ejecutar desde el contenedor Docker.

2.3.2. Escalabilidad

Aprenderemos sobre estrategias de escalado para modelos de ML. El escalado es importante para garantizar que los modelos de aprendizaje automático puedan manejar conjuntos de datos más grandes y complejos, así como un mayor uso. Consideraremos

- Los costos a largo plazo del servicio.
- El reentrenamiento.
- La velocidad de las implementaciones.

mientras exploramos el concepto de restricciones informáticas y complejidad del modelo y su impacto en la escalabilidad general de una canalización de ML.

Calcular las restricciones de escalabilidad.

Las restricciones informáticas, como los requisitos de CPU, memoria y disco, pueden afectar la escalabilidad de un modelo de aprendizaje automático. Si el modelo requiere más recursos computacionales de los disponibles, puede volverse lento o no responder, lo que dificulta su uso en aplicaciones del mundo real. Las restricciones informáticas se pueden identificar midiendo el uso de CPU, memoria y disco del modelo durante el entrenamiento y haciendo coincidir esos requisitos en la máquina que sirve a los modelos. Es importante identificar estas limitaciones desde el principio, ya que pueden afectar la escalabilidad y el rendimiento del modelo más adelante, cuando esté en producción y atendiendo a las personas en tiempo real.

Complejidad y escalabilidad del modelo.

La complejidad del modelo puede afectar la escalabilidad, ya que los modelos más complejos pueden requerir más recursos computacionales. **Equilibrar la complejidad y la escalabilidad del modelo puede resultar complicado, pero tenemos varias estrategias**. Por ejemplo, podríamos implementar técnicas de selección de características durante el tiempo de entrenamiento

del modelo, como pruebas de Chi-cuadrado o análisis de componentes principales, para reducir el tamaño de nuestros datos y nuestro modelo. También se pueden utilizar técnicas de compresión de modelos más avanzadas, como la poda, para reducir el tamaño del modelo eliminando parámetros redundantes y preservando al mismo tiempo el rendimiento.

Velocidad de implementaciones y escalabilidad

La velocidad de las implementaciones mide la rapidez con la que se implementan y actualizan nuevos modelos y es un factor crítico en la escalabilidad. La rápida implementación e iteración del modelo ayuda a garantizar que los modelos de aprendizaje automático sigan siendo relevantes y precisos en aplicaciones del mundo real, incluso cuando los datos cambian con el tiempo. Las estrategias para gestionar la velocidad de las implementaciones incluyen la integración y la implementación continuas, que cubriremos en nuestro próximo video, o el aprendizaje en línea, que es la capacidad de actualizar un modelo de ML a medida que hay nuevos datos disponibles sin tener que volver a entrenar todo el modelo desde cero.

Estrategias de escalamiento óptimas

En general, escalar los modelos de ML implica equilibrar una serie de compensaciones, incluido el costo de servir el modelo, a menudo medido en cuánto dinero costaría las máquinas de servicio, el costo de volver a entrenar el modelo y la velocidad de las implementaciones. Es importante considerar estas compensaciones al elegir estrategias de escalamiento para garantizar que el modelo sea escalable y rentable. Hay varias estrategias de escalado disponibles, incluido

- Escalado horizontal.- Agregar más máquinas al sistema.
- Escalado vertical.- Aumentar el tamaño de la máquina.
- Escalado automático.- Ajustar dinámicamente la cantidad de máquinas según la carga de trabajo actual.

Considerando tu estrategia de escalamiento óptima El escalado horizontal se puede implementar mediante el uso de técnicas de equilibrio de carga, como round-robin o conexiones mínimas, para distribuir la carga de trabajo entre varias máquinas. También se pueden utilizar técnicas de partición para distribuir los datos entre varias máquinas. El escalado horizontal aumenta la complejidad de la tubería y aumenta los costos, ya que se requieren más máquinas para ejecutar el modelo. El escalado vertical implica aumentar el tamaño de la máquina, lo que puede ayudar a aumentar la escalabilidad del modelo. El escalado vertical se puede implementar aumentando el tamaño de la máquina, como agregando más RAM o una CPU más rápida, o utilizando hardware de alto rendimiento, como GPU. El escalado vertical también genera mayores costos, ya que el hardware más potente y costoso generalmente cuesta más dinero. En un mundo ideal, implementaríamos una política de escalado automático para escalar automáticamente tanto horizontal como verticalmente.

2.3.3. Automatización

Bienvenido al vídeo sobre automatización. La automatización es un componente clave de MLOps que garantiza tanto la confiabilidad como la eficiencia de los procesos de ML. En este video, aprenderemos sobre los principios de

- Integración continua.
- Entrega continua.
- Capacitación continua.
- Monitoreo continuo.
- Cómo se pueden utilizar para automatizar los flujos de trabajo de ML.

Introducción a la automatización del aprendizaje automático

La automatización garantiza la confiabilidad y eficiencia de los canales de ML. La automatización ayuda a reducir el riesgo de error humano. La automatización también puede agilizar el proceso de desarrollo e implementación. Nuestros cuatro principios principales de automatización son

- Integración continua (CI).
- Entrega continua (CD).
- Capacitación continua (CT).
- Monitoreo continuo (CM).

Cuatro principios de automatización La integración continua (CI) es la práctica de integrar periódicamente cambios de código en un repositorio compartido. Esto permite la detección y resolución temprana de conflictos y garantiza que el código esté siempre funcionando. La entrega continua (CD) es la práctica de crear, probar e implementar cambios de código automáticamente. Esto permite una entrega de modelos rápida y consistente. La capacitación continua (CT) es la práctica de volver a capacitar y actualizar continuamente el modelo a medida que hay nuevos datos disponibles. Esto permite que el modelo siga siendo preciso y actualizado, incluso cuando los datos cambian con el tiempo. El Monitoreo Continuo (CM) es la práctica de monitorear el desempeño y la precisión del modelo de manera continua. Esto permite la detección temprana de problemas y puede usarse para activar una nueva capacitación.

Integración y entrega continuas La integración continua puede ayudar a garantizar que el código esté siempre funcionando y al mismo tiempo reducir el riesgo de error humano. También puede ayudar a detectar problemas en las primeras etapas del proceso de desarrollo, reduciendo el tiempo necesario para resolverlos. La entrega continua puede ayudar a garantizar que los modelos se implementen de manera rápida y consistente, reduciendo el tiempo necesario para poner nuevos modelos en producción. Al igual que la integración continua, también puede ayudar a reducir el riesgo de error humano al agilizar el engorroso proceso de implementación de código. Hay varias herramientas y prácticas de CI/CD disponibles, incluidas Git, AWS CodePipeline, Jenkins y Travis CI.

Formación y seguimiento continuo La capacitación continua ayuda a garantizar que el modelo permanezca preciso y actualizado, lo que reduce el riesgo de deterioro del modelo y reduce el tiempo necesario para volver a entrenarlo, ya que el proceso está automatizado y se puede realizar de forma continua. La monitorización continua también reduce el riesgo de deterioro del modelo y mejora la precisión general al permitir el acceso a métricas de rendimiento de ML consistentes y confiables. Al hacerlo, podemos identificar los problemas del modelo de manera temprana y configurar desencadenantes para el reentrenamiento del modelo si el rendimiento disminuye.

Ejemplo de automatización de ML a escala

La implementación de la automatización en los procesos de ML implica integrar los cuatro principios de automatización: CI, CD, CT y CM en nuestro proceso de ML. A continuación se muestra un flujo de trabajo de muestra para un modelo de clasificación simple. Primero, el código del modelo de clasificación se envía a un sistema de control de versiones como Git. Luego, el código confirmado se crea y prueba automáticamente utilizando una herramienta CI/CD como Jenkins. Si la compilación y las pruebas pasan, el código se implementa automáticamente en un entorno de prueba. Luego, el modelo se serializa y sus dependencias se agregan a una imagen de Docker que se implementa en un entorno de destino, como una máquina local o una plataforma en la nube. Una vez que implementamos nuestro modelo, el rendimiento del modelo se monitorea continuamente utilizando herramientas de monitoreo como Prometheus o Grafana. La retroalimentación del proceso de monitoreo se utiliza para informar decisiones sobre el modelo, como volver a entrenar u optimizar el modelo. Gracias a nuestro seguimiento, podemos activar más reentrenamientos que inicien todo el proceso de nuevo.

Pasos de CI/CD

1. La necesidad de volver a entrenar el modelo se identifica mediante un seguimiento.
2. El modelo se vuelve a entrenar utilizando nuevos datos para mejorar su rendimiento.
3. El código del modelo actualizado se envía al repositorio Git.
4. El modelo actualizado se serializa y sus dependencias se incluyen en una imagen de Docker.
5. La imagen Docker que contiene el modelo actualizado se implementa en un entorno de destino.
6. El monotoreo continuo se activa nuevamente y el modelo se vuelve a entrenar periódicamente para mantener su efectividad y precisión.

2.4. Prueba de canalizaciones de aprendizaje automático

2.4.1. Fiabilidad del modelo

En este video, veremos la confiabilidad del modelo en el aprendizaje automático. La forma en que **la gente percibe la confiabilidad de su modelo no tiene que ver solo con el rendimiento. Se trata de los datos y el entorno en el que vive el modelo y también de elementos como la latencia o la velocidad de su modelo**. Veremos cómo los modelos pueden producir resultados precisos y consistentes cuando se exponen a nuevos datos y cómo monitorear la confiabilidad para poder intervenir si es necesario.

Alinear los modelos de aprendizaje automático con las métricas de impacto empresarial

Las empresas otorgan un valor significativo a la confiabilidad del modelo, que va más allá de la obtención de resultados. Los resultados deben ser confiables y estar alineados con los objetivos de la empresa. El uso de métricas de impacto empresarial permite a las empresas evaluar el impacto de los modelos de ML en sus operaciones. Estas métricas deben ser coherentes con el propósito previsto del modelo y utilizarse para determinar si el modelo está cumpliendo sus objetivos. Para establecer las métricas de impacto empresarial adecuadas, las empresas deben identificar el propósito previsto del modelo. Las métricas pueden incluir ingresos, ahorro de costos o satisfacción del cliente, entre otras. Es importante señalar que el propósito previsto del modelo debe guiar la selección de métricas de impacto empresarial.

Rutinas de prueba en canalizaciones de ML

Las rutinas de prueba ayudan a identificar problemas con los datos, el modelo o la canalización y nos informan para realizar mejoras para obtener modelos más precisos y confiables. Tres tipos importantes de pruebas:

- Pruebas unitarias.- Prueban componentes individuales de una canalización de ML. Cómo probar una instancia PCA.
- Pruebas de integración.- Prueban todo el proceso y cómo funcionan todos los componentes juntos.
- Pruebas de humo.- Pruebas rápidas para garantizar que el sistema esté funcionando como se espera. Por ejemplo, probar la latencia de nuestra API y modelo.

Establecer rutinas de prueba y utilizarlas con frecuencia puede ayudar a garantizar que cualquier problema se detecte a tiempo y pueda abordarse antes de que se vuelva más grave.

Ejemplo de prueba unitaria A continuación se muestra un ejemplo de una prueba unitaria para una canalización de aprendizaje automático que implica un paso de preprocesamiento de datos y un paso de entrenamiento del modelo. La prueba unitaria genera datos simulados, ajusta la canalización a los datos de entrenamiento y evalúa la canalización a partir de los datos de prueba. Luego afirmamos que la precisión es superior al 80 %. Esta prueba unitaria se puede ejecutar como parte de un conjunto de pruebas más grande para garantizar que el proceso de aprendizaje automático esté funcionando correctamente.

```
def test_pipeline():
# Generar datos simulados para pruebas
X_train = pd.DataFrame('age': [25, 30, 35, 40], 'income': [50000, 60000, 70000, 80000])
y_train = pd.Series([0, 1, 0, 1])

pipeline = Pipeline([
 ('preprocessor', DataPreprocessor(),
 ('model', LogisticRegression())
])

# Ajustar la canalización a los datos de entrenamiento
pipeline.fit(X_train, y_train)

% GEnerar datos simulados para pruebas
X_test = pd.DataFrame('age': [30, 35, 40, 45], 'income': [55000, 65000, 75000, 85000])
y_test = pdf.Series([0, 0, 1, 1])
y_pred = pipeline.predict(X_test)
accurracy = accuracy_score(y_test, y_pred)

assert accuracy > 0.8, "Error: pipeline accuracy is too low."
```

Monitoreo de la obsolescencia del modelo

Paralelamente a las rutinas de prueba, también debemos verificar si el modelo está obsoleto. El estancamiento ocurre cuando el rendimiento de un modelo disminuye con el tiempo. **Esto puede suceder debido a cambios en los datos o cambios en el entorno en el que se utiliza el modelo. Estos cambios a menudo se denominan deriva de datos o modelos y pueden dar lugar a predicciones inexactas.**

Identificar y abordar la obsolescencia del modelo

La obsolescencia del modelo se puede identificar monitoreando el desempeño del modelo a lo largo del tiempo. Los signos de obsolescencia del modelo incluyen cambios en los datos o cambios en el entorno en el que se utiliza el modelo. Abordar el estancamiento del modelo podría implicar volver a entrenar el modelo con nuevos datos o actualizar la canalización de datos para tener en cuenta los cambios en el entorno. Por ejemplo, si el tiempo que alguien pasa en un sitio web es una característica del modelo y su plataforma de análisis cambia la forma en que se calcula, podría confundir su modelo ya que sus datos de entrenamiento ahora no están sincronizados con los datos de inferencia. Otras técnicas pueden incluir actualizar el proceso de ingeniería de características o cambiar la arquitectura del modelo.

2.4.2. Datos de prueba

Es importante garantizar que los datos utilizados en una canalización de aprendizaje automático sean precisos, consistentes y libres de errores porque los datos inexactos o inconsistentes pueden generar predicciones incorrectas y modelos poco confiables. Este video se centra en las formas en que podemos implementar pruebas de datos para asegurarnos de que nuestra canalización funcione de manera precisa y consistente.

Validación de datos y pruebas de esquema.

Probar datos puede ayudar a identificar problemas como valores atípicos faltantes o anormales, tipos de datos inconsistentes y datos que no son representativos de la población objetivo. **Las pruebas de validación de datos se utilizan para garantizar que los datos que se utilizan en una canalización de ML sean precisos y coherentes**. Las pruebas de esquema se utilizan para garantizar que los datos que se utilizan estén en el formato correcto y cumplan con ciertos criterios. Por ejemplo, si tenemos una función llamada "tiempo de obtención de valor"medida en segundos, queremos asegurarnos de que siempre se mida en segundos y no, por ejemplo, en minutos. La validación de datos y las pruebas de esquemas se pueden automatizar en una canalización de aprendizaje automático utilizando herramientas como *Great Expectations*, que permite realizar pruebas automatizadas de canalizaciones de datos.

Más allá de las simples pruebas

Las pruebas de validación de datos estándar y las pruebas de esquema verifican problemas básicos, como valores faltantes y formatos incorrectos, pero debemos pensar en pruebas más complicadas, como pruebas de expectativas, para verificar problemas más específicos o complejos. Por ejemplo, **es posible que deseemos comprobar si ciertos valores de datos están dentro de los rangos esperados dada una media y una desviación estándar o si ciertos patrones o tendencias están presentes en los datos.**

Pruebas de expectativas

Las pruebas de expectativas **son un tipo de rutina de prueba de validación de datos que se utiliza para probar la calidad de los datos en una canalización de ML**. Comprueban si los datos cumplen ciertos criterios o .[ex]pectativas"definidas por el usuario o el sistema. El objetivo es garantizar que los datos entrantes se ajusten al formato o estructura esperado. Por ejemplo, podríamos esperar que el tiempo en un sitio web sea de unos 4 minutos o que las visitas al consultorio médico de un paciente sean siempre anteriores al día actual.

Pruebas de importancia de características

Las pruebas de importancia de características **prueban la importancia de las características en un modelo de ML y ayudan a identificar qué características son las más importantes al hacer predicciones. Un ejemplo es la importancia de la permutación. La idea es permutar aleatoriamente los valores de las características para ver cuánto cambia el rendimiento del modelo. Este tipo de pruebas prueban constantemente la sensibilidad de un modelo a las características y pueden informar si vale la pena volver a entrenarlo con un conjunto de datos actualizado.**

Ejemplo de importancia de permutación A continuación se muestra un ejemplo de cómo realizar un análisis de importancia de permutación utilizando Python y la biblioteca scikit-learn. Comenzamos entrenando un clasificador de bosque aleatorio en el conjunto de entrenamiento. Luego usamos la función permutation_importance del módulo sklearn.inspection que toma el modelo entrenado, el conjunto de pruebas, los valores objetivo y la cantidad de veces para repetir el proceso de permutación como entrada. El resultado de permutation_importance es un diccionario que contiene la media, la desviación estándar y las puntuaciones de importancia bruta para cada característica. La función permutation_importance se puede utilizar con otros tipos de modelos, incluidos también los modelos de regresión.

```
import numpy as np
from sklearn.ensaemble import RandomForestClassifier
from sklearn.inspection import permutation_importance

# Entrenar un clasificador de bosque aleatorio
model = RandomForestClassifier().fit(X_train, y_train)

# Calcular la importancia de la permutación de las características
results = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42)
# Mostrar las características más importantes
feature_names = ['feature1', 'feature2', 'feature3', ...]
importances = results.importances_mean
fro i in range(len(feature_names):
print(f'{feature_names[i]}: {importances[i]}')
```

Buscando deriva de datos

Una vez que se implementa un modelo, es importante probar la desviación entre las etiquetas y los datos de entrada. La deriva de datos, también conocida como deriva de características, **se refiere a un cambio en la distribución de los datos de entrada del modelo**. Es posible que el modelo no pueda generalizarse a los nuevos datos tan bien como lo hizo con los datos originales y comience a tener un rendimiento inferior. La deriva de etiquetas se refiere a un cambio en la distribución real de las etiquetas. Esto puede suceder cuando la distribución de etiquetas en los datos cambia con el tiempo, como cuando cambia el comportamiento de los usuarios o la población modelada.

2.4.3. Modelos de prueba

Así como probamos los datos, deberíamos probar nuestro modelo para garantizar la confiabilidad. Nos centraremos en los métodos para evaluar el rendimiento y la confiabilidad de los modelos de ML.

Justicia individual versus grupal

De manera similar a cómo probamos nuestros datos en busca de sesgo y equidad, también **podemos probar nuestros modelos de ML para asegurarnos de que no estén sesgados contra ciertos grupos o individuos, manteniéndolos justos y confiables. La justicia individual es la idea de que personas similares deben ser tratadas de manera similar. En otras palabras, el modelo debería hacer predicciones similares para individuos que tienen rasgos o características similares**. Por ejemplo, si dos personas tienen educación, experiencia laboral y habilidades similares, se les deben brindar oportunidades laborales similares. **La justicia grupal**, por otra parte, **es la idea de que diferentes grupos deben ser tratados por igual**. Esto significa que el modelo debería hacer predicciones similares para individuos de diferentes grupos, como razas o géneros. Por ejemplo, si el modelo predice aprobaciones de préstamos, no debería discriminar a personas de determinados grupos raciales o étnicos.

Pruebas de resistencia

La prueba de reserva **es el proceso de probar un modelo en un conjunto de datos separado que no se utilizó durante el entrenamiento**. Esto se utiliza para probar la confiabilidad del modelo. Un ejemplo es realizar pruebas de reserva en el momento del entrenamiento del modelo. El objetivo es evaluar el rendimiento de un modelo en función de datos con los que el modelo no se entrenó. Al probar un modelo en un conjunto de datos reservado, **es posible identificar problemas con el modelo, como sobreajuste o desajuste.**

Buscando la deriva del modelo

Los modelos pueden mostrar signos de deriva o cambios en la forma en que realizan predicciones a lo largo del tiempo. **La deriva conceptual se refiere a un cambio en la relación entre las características y la respuesta**. Esto puede ocurrir cuando el significado o el uso de las funciones cambian con el tiempo. **La deriva de la predicción se refiere a un cambio en la distribución de la predicción del modelo, mientras que la deriva de la etiqueta se refiere a un cambio en la distribución de la etiqueta real**. Ambos tipos de deriva pueden ocurrir cuando los datos subyacentes cambian, pero se miden a nivel de modelo.

Ejemplo de búsqueda de deriva del modelo En este ejemplo de codificación, estamos simulando la deriva de conceptos al introducir la deriva de datos en X. La adición de ruido a los datos de prueba es una forma de deriva de datos, que es un cambio en la distribución de los datos de entrada. Al introducir este cambio, estamos probando la capacidad del modelo para adaptarse a patrones nuevos o inesperados en los datos de entrada, que es uno de los desafíos clave al lidiar con la deriva de conceptos.

```
# Importar las librerias necesarias y cargar los datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Ajustar el modelo
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

% Calcular la precisión en los datos de prueba
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# Probar para la deriva
X_test_drift = X_test + 1.0 # Simula la deriva de datos
y_pred_drift = clf.predict(X_test_drift)
accuracy_drift = accuracy_score(y_test, y_pred_drift)
```

```
    print(f'Accuracy drift: {accuracy_drift}')

    # Umbral de detección de deriva basado en la precisión
    drift_trhesold = accuracy * 0.9

    # Verificar para la perdida de precisión en los datos de deriva
    if accuracy_drift < drift_trhesold:
  print('Modelo ha derivado')
 else:
  print('Modelo no ha derivado')
```

Costo de los modelos complejos frente a los modelos básicos

Los modelos complejos pueden ser más costosos que los modelos de referencia más pequeños en términos de tiempo de entrenamiento y de inferencia, lo que puede afectar la escalabilidad y eficiencia del modelo. **La latencia es la cantidad de tiempo que le toma al modelo procesar una sola entrada y generar una predicción**. Esto puede incluir el tiempo que lleva cargar los datos en la memoria, ejecutar el algoritmo de predicción y devolver el resultado. La latencia es crucial para aplicaciones que requieren procesamiento en tiempo real o casi en tiempo real. El rendimiento se refiere a la cantidad de predicciones que el modelo puede hacer en un período de tiempo determinado. Esto se puede expresar como el número de predicciones por unidad de tiempo. El rendimiento es una métrica importante para aplicaciones que requieren un procesamiento de gran volumen, como el procesamiento de datos a gran escala o el procesamiento por lotes. Al probar la latencia y el rendimiento de un modelo, podemos identificar cuánta complejidad podemos permitirnos agregar al modelo sin dejar de mantener un rendimiento aceptable. Esto puede ayudarnos a optimizar la arquitectura y los parámetros del modelo para lograr el equilibrio deseado entre precisión y eficiencia.

Capítulo 3

Implementación del MLOps y ciclo de vida

3.1. MLOps en pocas palabras

3.1.1. El marco moderno de MLOps

MLOps >DevOps

El aprendizaje automático es un subtipo de desarrollo de software. Pero, mientras que el desarrollo de software clásico comienza con el código fuente puro y finaliza con una aplicación en ejecución, el aprendizaje automático añade una complejidad significativa al agregar datos y la mezcla de modelos.

Alto costo del aprendizaje automático sin operaciones

Por supuesto, la mayoría de las organizaciones comienzan a jugar con ML sin operaciones, ejecutando manualmente todos los flujos de trabajo y monitoreando los modelos solo ad hoc. Desafortunadamente, muchos no evolucionan mucho más allá de eso y pagan un alto precio en el futuro. Esto provoca la acumulación de la llamada deuda técnica.

Deuda Técnica

Que Wikipedia define como: el costo implícito de retrabajo adicional causado por elegir una solución fácil (limitada) ahora en lugar de utilizar un enfoque mejor que llevaría más tiempo. Con cada día y cada nuevo modelo en producción, la cantidad de deuda técnica y el riesgo del modelo aumentarán exponencialmente, haciendo que el proceso sea cada vez más lento, más frustrante y propenso a errores, lo que en última instancia afectará nuestra capacidad de ofrecer valor a través del aprendizaje automático. Este ha sido el tema de un famoso artículo de Google titulado "Machine Learning: The high-interes credit card of Technical Deud".

Flujos de trabajo de aprendizaje automático

Los flujos de trabajo de ML más típicos son la recopilación y preparación de datos, el etiquetado de datos, la selección de modelos, el entrenamiento de modelos, el empaquetado de modelos, la implementación de modelos, el monitoreo y el mantenimiento de modelos. **Cuanto más automatizados e integrados estén estos flujos de trabajo en el marco general de TI, mayor será la madurez de MLOps de una organización.**

MLOps implementados

Por otro lado, la implementación de herramientas y prácticas de MLOps hará que sus procesos sean automatizados, rápidos, reproducibles y explicables, lo que producirá la más alta calidad de servicio y se ganará la confianza de sus clientes.

Centrarse en las operaciones

Nos centraremos en la parte de Operaciones de MLOps. Esa es la parte del proceso de aprendizaje automático que comienza después del entrenamiento del modelo, con la implementación del modelo como un servicio para el usuario final.

No lineal

Estamos saliendo de la fase exploratoria no lineal de selección de modelos y entrando el dominio simplificado y estructurado de ejecutar, monitorear y mantener nuestro modelo en producción.

En el centro de atención

Nuestro modelo está ahora en el centro de atención y cada comportamiento inesperado es inmediatamente visible para el cliente, lo que hace que los márgenes de error sean muy estrechos y da poco tiempo para corregir errores y actualizar modelos, por lo que debemos estar en la cima de nuestro juego.

3.1.2. Etapas del ciclo de vida

Entonces, *¿a qué nos referimos exactamente cuando hablamos del ciclo de vida del modelo?* Para evitar confusiones en el futuro, diferenciemos entre los diferentes ciclos de vida que encontramos en el dominio del aprendizaje automático. Tenemos

- Proyecto ML.
- Aplicación ML.
- Ciclo de vida del modelo ML.

Primero definamos estos tres términos y expliquemos las relaciones entre ellos. El flujo detallado del proyecto de ML y el ciclo de vida no son el enfoque de este capítulo, por lo que lo definiremos en términos generales como el esfuerzo general de resolver un problema empresarial utilizando el aprendizaje automático, que, si tiene éxito da como resultado la construcción de nuestra aplicación y modelos de ML.

A partir de ahora nos centraremos únicamente en el ciclo de vida de la aplicación y el modelo. Estas dos son cosas completamente separadas. Un modelo de ML es simplemente un estimador puro, como un modelo predictivo que produce un pronóstico de ventas diario. Pero una aplicación de ML, aunque tiene modelos de ML en su núcleo, normalmente incluye muchas otras características, como reglas comerciales, como "Si un usuario ha calificado menos de 10 películas, recomiende las películas más populares a nivel mundial; de lo contrario, utilice el modelo de recomendación personalizado".

- Una base de datos, para almacenar características adicionales y registrar resultados del modelo.
- Una interfaz gráfica para usuarios administradores, para configurar y solucionar problemas de la aplicación.
- Una API, a través de la cual la aplicación puede comunicarse con el mundo exterior de manera consistente y segura, etc.

En la práctica, no es raro ver el modelo ML bloqueado en la aplicación ML, formando una unidad monolítica. La mejor práctica, sin embargo, es separarlos para que cada uno pueda seguir su propio camino, que es la filosofía de la llamada .ªrquitectura de microservicios".

Resultando en un ciclo de vida de la aplicación de ML separado y un ciclo de vida del modelo de ML. Piense en la aplicación ML como un automóvil, que puede tener una vida útil de varias décadas, y en los modelos ML como sus neumáticos, que se pueden cambiar de forma independiente numerosas veces durante ese mismo período. Dicho esto, en este capítulo nos centraremos en el ciclo de vida del modelo. Dentro de ese ámbito, debemos ser específicos con una cosa: para nosotros un modelo no es algo abstracto o teórico sino un modelo concreto, entrenado, listo para ser puesto en uso. Un modelo entrenado y otros recursos necesarios para la implementación comprenden un paquete de implementación de modelo. Poner ese modelo en uso es lo que llamamos implementación y lo que marca el comienzo del ciclo de vida de nuestro modelo. Una vez en funcionamiento,

vigilamos atentamente el comportamiento del modelo para asegurarnos, en primer lugar, de que esté funcionando y, en segundo lugar, de que su rendimiento cumpla con las expectativas. A esto lo llamamos monitoreo del modelo posterior a la implementación. Finalmente, llegará un momento en el que querremos reemplazar nuestro modelo actual por una nueva versión. Quizás encontramos un modelo mejor, o mejores características, o el proceso modelado cambió, invalidando el modelo existente. El modelo antiguo será dado de baja y archivado, dando paso al nuevo chico de la cuadra.

Un archivo adecuado es especialmente importante en industrias reguladas, donde los reguladores pueden pedirnos que expliquemos la decisión de nuestro modelo de hace varios años. Para eso, necesitamos poder cargar y ejecutar todas las versiones del modelo anterior, lo cual no es un requisito tan trivial como podría parecer. A esto se le llama reproducibilidad.

3.1.3. Componentes MLOps

Hablaremos sobre los componentes fundamentales de un marco MLOps y su interacción mutua. Cubriremos conceptos generales de desarrollo de software, como flujos de trabajo, canalizaciones y artefactos; y los específicos de ML, como el registro de modelos, el almacén de características y el almacén de metadatos.

MLOps es una extensión de los principios de desarrollo de software DevOps para incluir flujos de trabajo de ML. **Un flujo de trabajo es un término comercial genérico para cualquier secuencia de tareas que, a partir de determinadas entradas, produce determinadas salidas**. Los flujos de trabajo se pueden ejecutar:

- a mano,
- automáticamente,
- semiautomáticamente.

Para automatizar un flujo de trabajo necesitamos escribir un programa, a menudo llamado "script", para cada acción que contiene a este programa lo llamamos **canalización y se ha convertido en un término general para cualquier flujo de trabajo automatizado de un extremo a otro en TI, ya sea en desarrollo, datos o MLOps.**

Las salidas de las tuberías se denominan artefactos, que en realidad es otro término general para cualquier resultado del proceso de desarrollo de software.

Hablemos ahora de las canalizaciones específicas de ML. Dijimos que el ciclo de vida del modelo comienza con la implementación, pero tener algo que desplegar. Primero debemos construirlo. Construir significa transformar código y datos puros en aplicaciones y modelos implementables. Piense en algo así como un instalador de software que descargaría de Internet, pero para aplicaciones de aprendizaje automático. Ése es el objetivo de los llamados oleoductos de construcción. En MLOps tenemos al menos dos flujos de trabajo de compilación separados:

- uno para construir el modelo en sí
- y otro para construir la aplicación ML que servirá a nuestro modelo.

El proceso de creación de la aplicación es la construcción clásica del software DevOps. Toma nuestros scripts de un repositorio central, donde administramos todo nuestro código, realiza algunas transformaciones automatizadas que empaquetan nuestra aplicación para su implementación y almacena ese paquete en un repositorio dedicado. Pero el proceso de construcción de modelos, también conocido como proceso de capacitación de modelos, es algo completamente diferente. Aparte del código que contiene la definición del modelo y el script de entrenamiento, también necesitamos los datos para entrenar. Este único ingrediente adicional añade toda una serie de complejidades.

Nuestro proceso de formación puede empezar desde cero o datos ya procesados, llamamos çaracterísticas.ª las variables procesadas para el entrenamiento de modelos y a una base de datos que las almacena un .ªlmacén de características". Después de que el oleoducto esté ejecutado con éxito, terminamos con el objeto modelo entrenado y datos complementarios necesarios para su implementación y ciclo de vida. A estos datos complementarios los llamamos "metadatos del modelo".

Ahora insertamos el objeto modelo en nuestro registro de modelos, que es un registro especializado para almacenar y versionar modelos de ML y los metadatos en el llamado almacén de metadatos, cuyo propósito es almacenar toda la información importante sobre los diferentes modelos que construimos e implementamos.

Ahora tenemos algo que implementar, pero necesitamos el cómo. Ese es el trabajo de, los canales de implementación. Su trabajo es tomar los artefactos de compilación (en nuestro caso, el paquete de la aplicación ML y el paquete del modelo) y colóquelos en su lugar en la plataforma de servicio de destino. Una vez que hayan terminado su trabajo, estamos bloqueados, cargados y listos para comenzar.

Ahora podemos iniciar nuestro servicio y monitorear si se ejecuta y funciona como se esperaba, y esa es una arquitectura MLOps típica de alto nivel.

3.2. Desarrollar para la implementación

3.2.1. Desarrollo impulsado por la implementación

Para resaltarlo nuevamente: este curso se enfoca en la parte de Operaciones de MLOps, pero eso no significa que MLOps comience allí. Si las operaciones son sólo una ocurrencia tardía, corremos el riesgo de terminar con un modelo inútil que no se puede implementar en absoluto. Para usar una analogía: Al igual que un piloto de carreras ajusta bien su velocidad y dirección antes de llegar a una curva desafiante, necesitamos pensar en la implementación desde las primeras etapas de desarrollo. ¿A qué nos referimos con eso? Imaginemos que debemos implementar y mantener la aplicación y el modelo de aprendizaje automático de otro colega. Inmediatamente nos deben venir a la mente una serie de preocupaciones:

- Compatibilidad de infraestructura.
- Transparencia y reproducibilidad.
- Validación de datos de entrada.
- Monitoreo.
- Depuración.

Preocupación: ¿Despliegue de infraestructura?

En primer lugar, ¿puede esto siquiera ejecutarse en la infraestructura de destino? Es fácil dejarse llevar por la construcción de un modelo que ocupa gigabytes de memoria y se ejecuta en 16 núcleos en paralelo. Sólo para darte cuenta de que debes implementarlo en un teléfono inteligente. Para evitar estos obstáculos, asegúrese de comprender la infraestructura de destino lo antes posible.

Preocupación: ¿Transparencia y reproducibilidad?

Luego nos preguntamos sobre la transparencia y la reproducibilidad. ¿Es obvio quién entrenó este modelo, cuándo y con qué script, datos e hiperparámetros? Implementar modelos sin un origen claro es una gran prohibición. La capacitación de modelos de producción solo debe realizarse utilizando conjuntos de datos versionados, que podemos recuperar sin cambios en cualquier momento, y canalizaciones totalmente transparentes.

Transparencia significa que no hay absolutamente ningún dilema sobre cómo pasamos del código y los datos sin procesar al modelo completamente entrenado; y medios de reproducibilidad Es simple y directo recrear este modelo exacto en cualquier momento posterior. Además de eso, es enormemente beneficioso registrar los experimentos realizados durante la fase de exploración, para que los futuros mantenedores no reinventen la rueda. Para ello podemos utilizar el almacén de metadatos.

Preocupación: ¿Validación de datos de entrada?

Luego, cuando futuros usuarios realicen una solicitud no válida, por ejemplo, proporcionando un valor negativo cuando se espera uno estrictamente positivo, ¿cómo detectaremos esto y les devolveremos un mensaje de error? Para abordar esto, necesitamos referencias de comparación,

a las que llamamos perfiles de datos o expectativas. El mejor enfoque es garantizar que nuestro proceso de creación de modelos los cree y guarde dentro de los metadatos del modelo.

Preocupación: ¿Deterioro del rendimiento?

Entonces, ¿cómo sabremos si el rendimiento predictivo de nuestro modelo, se ha deteriorado con el tiempo? Para permitir dicho **monitoreo**, debemos asegurarnos de que nuestra aplicación registre de alguna forma las entradas que recibe y las predicciones que produce.

Preocupación: depuración

Además, ¿podremos localizar con precisión errores en el código, también conocidos como bugs, cuando ocurran incidentes? La respuesta es: Registro.

No debemos improvisar, sino utilizar herramientas construidas para este fin.

Por último, pero no menos importante: ¿me siento cómodo realizando cambios en este código? Inevitablemente tendremos que corregir algunos errores en nuestra aplicación ML. Será mejor que tengamos una colección elaborada de pruebas ya creadas cuando llegue ese momento. De lo contrario, fácilmente podemos terminar con más errores de los que empezamos.

Tipos de prueba

Podemos escribir numerosas pruebas:

- pruebas unitarias,
- pruebas de integración,
- pruebas de carga,
- pruebas de estrés,
- pruebas de implementación, etc.