DML

Introducción

DML (*Distributed Machine Learning*): es un sistema que permite a los usuarios entrenar modelos de machine learning de forma distribuida y realizar predicciones utilizando modelos previamente entrenados.

El sistema expone una **API REST** que ofrece los servicios de entrenamiento y predicción, y además incluye una aplicación de consola que encapsula toda la lógica necesaria para que el funcionamiento del sistema sea completamente transparente para el usuario.

Alcance del Sistema

1. El sistema ofrece diversos modelos de machine learning para tareas de ***regresión*** y ***clasificación.***
2. Permite a los usuarios subir archivos ****.csv**** con los datasets de entrenamiento y predicción.
3. Los usuarios pueden crear procesos de entrenamiento indicando:
    - el tipo de entrenamiento (regresión o clasificación).
    - el dataset que se usará.
    - los modelos a entrenar.
4. El sistema permite consultar el estado de los entrenamientos en curso.
5. Los usuarios pueden ejecutar predicciones utilizando modelos previamente entrenados.
6. Se pueden consultar los resultados generados por las predicciones.
7. Los usuarios pueden descargar los modelos ya entrenados.

Arquitectura y Características Técnicas

1. El sistema utiliza una arquitectura modular y desacoplada basada en dos tipos de nodos: **Workers** y **DataBase**.
2. Es tolerante a fallas de hasta dos nodos, independientemente de su tipo.
3. Ofrece escalado horizontal mediante la adición de más workers.
4. Garantiza consistencia eventual en los datos distribuidos.
5. Tolera divisiones y reconexiones de la red sin afectar su operación general.

Nodos de Base de Datos (DataBase Nodes)

Los nodos de base de datos actúan como servicios independientes que exponen una API REST para administrar sus datos locales y responder a las solicitudes que reciben desde los workers.

Cada nodo mantiene su propio almacenamiento, procesa consultas de lectura y escritura, y participa en el mecanismo de replicación distribuida para lograr la consistencia del sistema.

Almacenamiento Local

- Los nodos no utilizan bases de datos SQL, sino que toda la información se almacena en archivos CSV, lo que simplifica la replicación y el manejo distribuido.
- Los datasets no se guardan como un único archivo, sino divididos en batches de 64 muestras. Cada batch se almacena como un CSV independiente.
- Las relaciones internas del sistema se manejan con dos archivos adicionales:
  - Un CSV que vincula entrenamientos → datasets, y almacena metdatos de los entrenamientos.
  - Un CSV que vincula modelos → entrenamientos, y almacena metdatos de los modelos.

Control de Versiones y Consistencia

- Cada nodo mantiene una variable de versión asociada a sus datos locales. Esta versión permite detectar si un nodo está actualizado o si necesita replicar cambios desde otro nodo.
- **Escrituras**:
  - Cuando un nodo recibe una solicitud de escritura, registra el cambio localmente y luego reenvía la operación a nodos adicionales.
  - Este proceso se repite hasta lograr la consistencia en toda la red.
- **Lecturas**:

  Una solicitud de lectura se reenvía a varios nodos, y:
  - El sistema devuelve el dato con la versión más reciente.
  - Si un nodo devuelve una versión atrasada o no posee el dato requerido, se le ordena actualizarse antes de continuar operando.

Tolerancia a Fallos, Particiones y Reconexiones

El diseño distribuido permite que la red siga operativa incluso si algunos nodos fallan o se desconectan temporalmente. Cuando un nodo vuelve a conectarse, compara su versión local y sincroniza automáticamente los datos pendientes.

Tolerancia a Particiones de Red

El sistema soporta particiones de red sin detener el funcionamiento:

- Cada partición puede seguir procesando lecturas y escrituras de manera independiente.
- No existe un nodo central cuya caída detenga el sistema.

Además, debido a la forma en que se realizan los entrenamientos:

- Las iteraciones del entrenamiento avanzan secuencialmente sobre los batches.
- Si ocurre una partición, cada subred sigue entrenando el modelo por su cuenta, incrementando su versión en cada iteración.
- Al reconectarse la red, los nodos comparan versiones:
  - La versión más alta corresponde al modelo que más iteraciones realizó,
  - y se asume como mejor, por lo que el resto de nodos se actualizan a esa versión.

Este mecanismo garantiza que después de una partición y su posterior reconexión, el sistema siempre converge hacia el modelo más avanzado sin generar conflictos.

Reconexión y Consistencia Eventual

Cuando la red vuelve a unificarse:

- Los nodos intercambian sus versiones actuales.
- Los nodos desactualizados recuperan los datos de la versión más reciente mediante el proceso de replicación.
- El sistema converge automáticamente hacia un estado consistente.

Con esta estrategia, el sistema asegura consistencia eventual, incluso bajo fallos, divisiones temporales de la red y reconexiones posteriores.

Nodos Worker

Los nodos Worker son responsables de procesar las peticiones de los usuarios y ejecutar las tareas de entrenamiento y predicción de los modelos.

Cada worker funciona de forma totalmente independiente, sin necesidad de comunicarse ni coordinarse con otros workers. Su única interacción es con los nodos DataBase, que actúan como fuente de datos y como mecanismo de control del trabajo distribuido.

Funciones Principales:

1. Recepción de solicitudes del usuario:

Cada worker expone una API REST que le permite a los clientes:

- Subir datasets.
- Crear entrenamientos.
- Consultar estado y resultados de entrenamientos.
- Crear predicciones.
- Consultar estado y resultados de predicciones.
- Descargar modelos entrenados.

2. Orquestación de tareas:

Los workers coordinan cada operación enviando las solicitudes necesarias a los nodos DataBase y ejecutando la lógica interna del entrenamiento o la predicción.

3. Entrenamiento y predicción de modelos:

Los workers realizan:

- Descarga de los batches de datos desde los nodos DB.
- Entrenamiento secuencial del modelo batch por batch.
- Actualización continua del estado y progreso en los nodos DB.
- Guardado del modelo entrenado al finalizar.
- Ejecución de predicciones utilizando modelos ya entrenados.

Independencia entre Workers

Los workers no conocen ni necesitan conocer la existencia de otros workers. No comparten estado ni intercambian mensajes.

Esta independencia permite escalar horizontalmente simplemente agregando nuevos nodos.

Mecanismo de Control y Asignación de Trabajo (Campo health)

Para evitar que dos workers entrenen o predigan sobre el mismo modelo al mismo tiempo, el sistema utiliza un mecanismo de heartbeat basado en un campo llamado ***health***, almacenado en los nodos DataBase.

Su funcionamiento es el siguiente:

1. **Selección del modelo disponible**:
    - Los nodos DataBase llevan un registro del health de cada modelo.
    - Un worker puede solicitar al DB un modelo cuyo health sea de hace más de 20 segundos (Ese modelo se considera libre o abandonado).
2. **Heartbeat del worker**:

- Una vez que un worker toma un modelo, envía cada 10 segundos una señal al DB informando que sigue activo.
- El DB actualiza (health = instante_actual)

3. **Detección de fallo o abandono**:
   - Si no se recibe un heartbeat en más de 20 segundos, el DB considera que:
     - El worker falló, o
     - El worker se desconectó, o
     - El worker finalizó sin reportarlo correctamente.
   - Ese modelo vuelve a estar disponible para que otro worker lo tome.
4. **Reasignación automática**:
   - Cualquier worker que pida un modelo libre podrá tomarlo inmediatamente.
   - No se necesitan bloqueos distribuidos ni coordinación entre workers.

Ventajas del Mecanismo:

- Evita conflictos: ningún modelo será entrenado por dos workers simultáneamente.
- Alta disponibilidad: si un worker se cae, el trabajo se reasigna automáticamente.
- Escalabilidad simple: más workers = más capacidad de entrenamiento sin cambios estructurales.
- Desac acoplamiento total: la lógica distribuida se delega a los nodos DB mediante el campo health.

Descubrimiento de Nodos

Para que cada nodo del sistema pueda localizar a otros nodos activos dentro de la red, se utilizan dos mecanismos de descubrimiento complementarios. Esto garantiza que, si uno falla, el otro pueda actuar como respaldo, manteniendo la comunicación y la operatividad del sistema.

Domain Name Service (DNS)

El sistema se ejecuta sobre una red de Docker utilizando Docker Swarm, que proporciona un DNS interno capaz de resolver automáticamente los nombres de servicio de cada nodo.

Gracias a este DNS:

- Los nodos no necesitan conocer direcciones IP específicas.
- Un worker puede comunicarse con los nodos DataBase simplemente usando su dominio o nombre de servicio.
- Docker gestiona la resolución y el balanceo entre instancias activas.

- Los fallos de nodos individuales quedan aislados del resto del sistema.
- Este mecanismo es la forma principal de descubrimiento dentro de la red.

IP Cache

Como mecanismo de respaldo, cada nodo mantiene una caché local de direcciones IP correspondientes a otros nodos conocidos. Esta técnica funciona incluso si la resolución por DNS falla.

El funcionamiento es el siguiente:

IPs fijas al iniciar el nodo:

Cuando un nodo se levanta, carga en su caché un conjunto de IPs fijas preconfiguradas. Esto le permite tener acceso inmediato a nodos potenciales, incluso si el DNS presenta fallos desde el inicio.

Actualización dinámica:

A medida que el nodo interactúa con otros, añade nuevas IPs a la caché cuando detecta nodos que responden correctamente.

Uso como respaldo:

Si el DNS no funciona o la red presenta inconsistencias, el nodo reutiliza las IPs guardadas en la caché para intentar conectarse a otros nodos activos.

Depuración automática:

Si una IP en la caché deja de ser válida, se descarta al detectar fallos consecutivos.

Gracias a esta combinación, el sistema puede arrancar, operar y recuperarse incluso en momentos donde la infraestructura de red esté parcialmente degradada.

Comunicación entre Nodos

La comunicación entre todos los componentes del sistema (workers, nodos DataBase y clientes) se realiza exclusivamente mediante peticiones HTTP.

Cada nodo expone una API REST que permite intercambiar datos y coordinar las operaciones distribuidas sin necesidad de mantener conexiones persistentes.

Gracias a este enfoque, los nodos se mantienen desacoplados, pueden operar de forma independiente y es posible reemplazarlos, escalarlos o reiniciarlos sin afectar el funcionamiento global del sistema.