Машинное обучение и высоконагруженные системы

Москва, осень 2023

MLOps. Начало

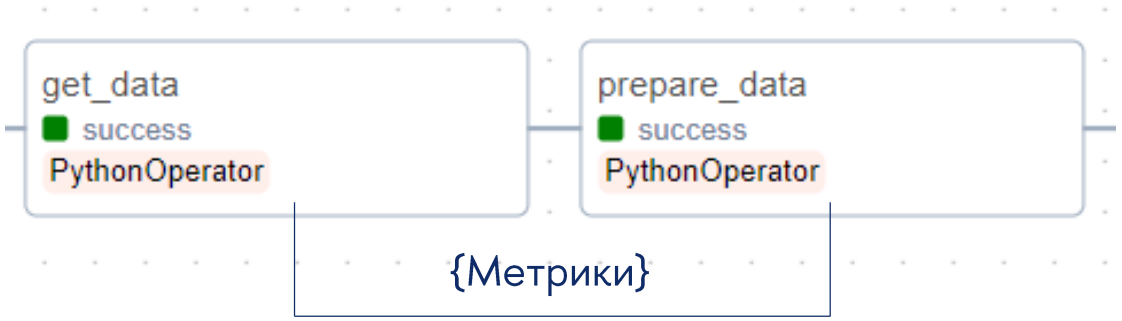
Гаврилова Елизавета,
Senior ML-engineer

Неделя 4. Что будем обсуждать сегодня?

- Концепция XCom
- Сбор метрик
- Хранение результатов на S3

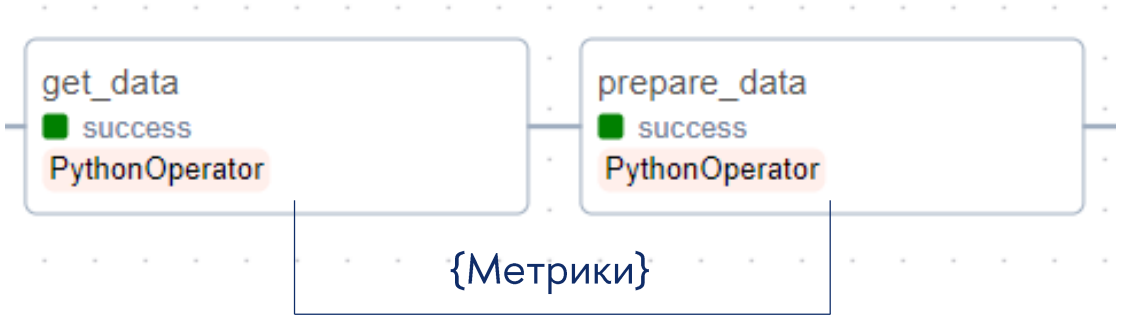
Взаимодействие операторов AirFlow

Мы хотим передавать информацию сквозь операторы DAG'а. Как это сделать?



Xcom (Cross-Comunication) AirFlow

XCom – это инструмент передачи параметров между операторами.



Это таблица в БД, в которой хранятся:
- **key** – ключ
- **value** – значение
- **task_id** – id опреатора
- **dag_id** – id DAG'a
- **execution_date** – дата выполнения DAG
- **timestamp** – время записи в БД

Документация

Xcom (Cross-Comunication) AirFlow

Почему бы не передавать между операторами сразу датасеты?

Перед укладкой данных в таблицу они сериализуются*, а при чтении оператором, соответственно, десериализуются, что занимает время.

Тип BYTEA в Postgres может хранить до 1ГБ данных, но сам же Postgres не рекомендует использовать его для больших объёмов:

While a column of type bytea can hold up to 1 GB of binary data, it would require a huge amount of memory to process such a large value. The Large Object method for storing binary data is better suited to storing very large values, but it has its own limitations.

**Сериализация - это процесс сохранения объекта в виде последовательности байт, чтобы в будущем по этой последовательности можно было бы восстановить исходный объект. В частном случае - это может быть сохранение в текстовую строку определенного формата, например JSON.*

Документация

Xcom (Cross-Comunication) AirFlow

Почему нельзя передавать аргументы, как в python-функциях?

Операторы могут исполняться на разных адресах или физических машинах. В этом случае нужен способ передачи сообщений от одной машины к другой. Для этого и есть база мета-данных в AirFlow.

Примеры работы с XCom

```python
def init() -> Dict[str, Any]:

    metrics = {}
    metrics["start_time"] = datetime.now().strftime("%Y%m%d %H:%M")

    return metrics


def get_data(**kwargs) -> Dict[str, Any]:

    ti = kwargs['ti']
    metrics = ti.xcom_pull(task_ids='init')


task_init = PythonOperator(task_id="init",
                           python_callable=init,
                           dag=dag)

task_get_data = PythonOperator(task_id="get_data",
                               python_callable=get_data,
                               dag=dag,
                               provide_context=True)
```

Что нового?

- у функции появился **return**
- у второй функции появились **аргументы на вход**
- какой-то **ti**
- какой-то **context**

Примеры работы с XCom

```python
def init() -> Dict[str, Any]:

    metrics = {}
    metrics["start_time"] = datetime.now().strftime("%Y%m%d %H:%M")

    return metrics


def get_data(**kwargs) -> Dict[str, Any]:

    ti = kwargs['ti']
    metrics = ti.xcom_pull(task_ids='init')


task_init = PythonOperator(task_id="init",
                           python_callable=init,
                           dag=dag)

task_get_data = PythonOperator(task_id="get_data",
                               python_callable=get_data,
                               dag=dag,
                               provide_context=True)
```

По умолчанию если оператор возвращает значение, это значение автоматически попадает в **XCom**.

Это поведение можно изменить, передав в оператор аргумент **do_xcom_push=False**.

Примеры работы с XCom

```python
def init() -> Dict[str, Any]:

    metrics = {}
    metrics["start_time"] = datetime.now().strftime("%Y%m%d %H:%M")

    return metrics


def get_data(**kwargs) -> Dict[str, Any]:

    ti = kwargs['ti']
    metrics = ti.xcom_pull(task_ids='init')


task_init = PythonOperator(task_id="init",
                           python_callable=init,
                           dag=dag)

task_get_data = PythonOperator(task_id="get_data",
                               python_callable=get_data,
                               dag=dag,
                               provide_context=True)
```

Context – это аргумент метода execute любого оператора.

Это словарь, содержащий в себе информацию о запуске DAG'а и среде AirFlow.

В числе прочих:

- **dag**
- **run_id**
- **next_execution_date**
- **prev_start_date_success**
- **…**
- **ti** – task instance – запуск задания

Документация

Примеры работы с XCom

```python
def init() -> Dict[str, Any]:

    metrics = {}
    metrics["start_time"] = datetime.now().strftime("%Y%m%d %H:%M")

    return metrics


def get_data(**kwargs) -> Dict[str, Any]:

    ti = kwargs['ti']
    metrics = ti.xcom_pull(task_ids='init')


task_init = PythonOperator(task_id="init",
                           python_callable=init,
                           dag=dag)

task_get_data = PythonOperator(task_id="get_data",
                               python_callable=get_data,
                               dag=dag,
                               provide_context=True)
```

task_instance(ti) – запуск задания – это ключ в словаре контекста.

У этого класса есть метод **xcom_pull**, который и позволяет принимать контекст от другого оператора.

Документация

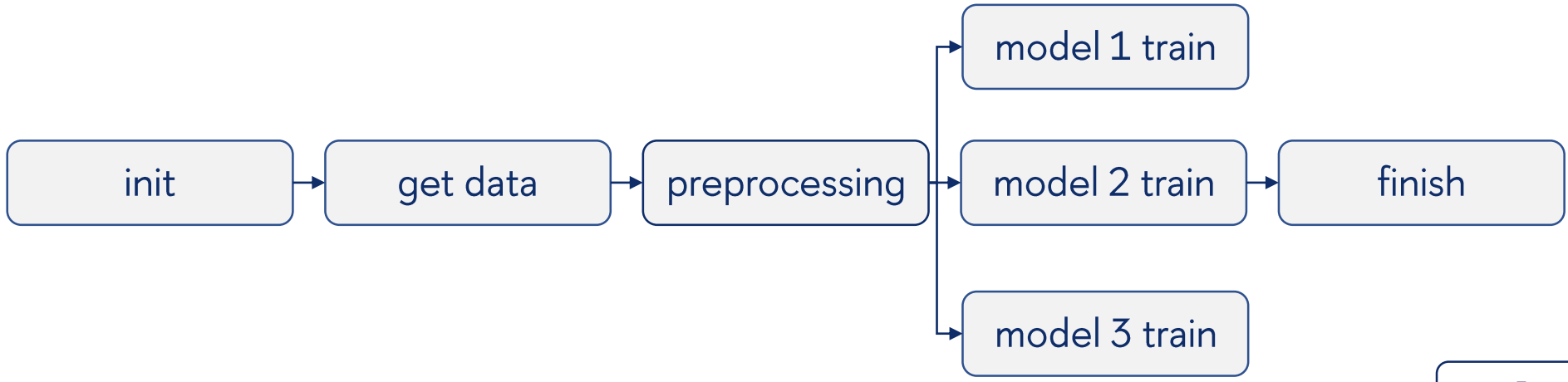
Как обучать несколько моделей?

Несколько DAG'ов, каждый из которых соответствует 1 модели

Один DAG с разветвлением

Что должно получиться?

Вариант 1



Вариант 2

А теперь real-time coding...

Давайте поможем Jupyter Notebook стать автоматически обучаемой моделью

Разобрали сегодня

- Что такое Xcom
- Что такое Context
- Как передавать данные между операторами
- Несколько моделей – несколько DAG'ов

ДЗ

- Несколько моделей – несколько DAG'ов
- Несколько моделей – один DAG