Управление признаками сущностей в Apache Kafka

Введение

Во время работы над задачами машинного обучения с онлайн-данными есть необходимость собирать различные сущности в одну для дальнейшего анализа и оценки. Процесс сбора должен быть удобным и быстрым. А также часто должен предусматривать бесшовный переход от процесса разработки к промышленному использованию без дополнительных усилий и рутинной работы. Для решения этой проблемы можно воспользоваться подходом с использованием Feature Store. Этот подход со многими деталями описан вот здесь: [Meet Michelangelo: Uber’s Machine Learning Platform](<https://eng.uber.com/michelangelo-machine-learning-platform/>). В этой статье описывается интерпретация указанного решения для управления признаками в виде прототипа.

Feature Store для онлайн потоков

Feature Store можно рассматривать как сервис, который должен выполнять свои функции строго по его спецификации. Прежде чем определить эту спецификацию, следует разобрать простой пример.

Пример

Пусть даны следующие сущности.

Фильм, который обладает идентификатором и заголовком.

Рейтинг фильма, у которого так же есть собственный идентификатор, идентификатор фильма, а также значение рейтинга. Рейтинг меняется во времени.

Источник рейтинга, который так же имеет собственный рейтинг. И меняется во времени.

И нужно эти сущности объединить в одну.

Вот что получается.

|  |
| --- |
|  |
| Диаграмма сущностей. |

Как можно понять, объединение происходит по ключам сущностей. Т.е. к фильму ищутся все рейтинги фильма, а к рейтингу фильма все рейтинги источника.

Обобщение примера

Теперь можно обобщить пример и масштабировать его на все сущности, которые могут быть связаны по ключам.

Есть kafka-потоки, которые определяют собой сущности: A, B… NN.

Нужно объединять эти потоки для создания новых потоков: AB, BCD... NM.

Этим процессом должен управлять сервис: Feature Stream Engine.

Feature Stream Engine умеет объединять сущности в  kafka-потоках, используя хранилище метаданных Feature Stream Store и Feature Stream Center, как единую точку входа по управлению объединением.

|  |
| --- |
|  |
| Обобщенная диаграмма сущностей и Feature Stream Engine. |

Feature Stream Store

Хранилище метаданных представляет из себя сервис по хранению данных о потоках, сущностях и их связях.

Основная единица хранилища – это признак (feature).

Признак имеет свой идентификатор, ссылку на источник, наименование и тип.

Источник группирует признаки и привязывается к определенному потоку.

Feature Stream Center

Центр управления позволяет создавать новые потоки, а также взаимодействовать со службами доставки и развертывания для поддержки работы новых потоков в различных средах исполнения, в том числе и промышленной среде.

Feature Stream Engine

Feature Stream Engine обеспечивает работу с потоками, а так же взаимодействие с внешними сервисами и конечными пользователями.

|  |
| --- |
|  |
| Компоненты Feature Stream Engine. |

Архитектура Feature Stream Engine

Feature Stream Engine представляет из себя конструктор, который позволяет собирать признаки из различных потоков и доставлять этот функционал на различные среды.

Feature Stream Engine должен реализовывать следующие функции.

Описывать источники данных.

Привязывать источники данных к потокам kafka.

Описывать признаки и привязывать их к источникам данных.

Создавать новые источники данных на основе имеющихся путем объединения по ключам (особым признакам).

Развертывать функционал работы потоков в различных средах, включая промышленную среду.

|  |
| --- |
|  |
| Архитектура Feature Stream Engine. |

Прототип

Для реализации идеи необходимо упросить функциональность.

Будут объединяться несколько потоков по ключам и записываться в один поток.

Предположим, что метаданные описываются файлами со свойствами (configration .properties).

Эти данные реализуют следующие модель.

Источники данных в виде имен topic-ов для kafka. Перечисляются через “,”.

Ключи в этих источниках данных. Перечисляются через “,”.

Имя результирующего topic-а.

Конвертация входных параметров в структуру, которая описывает объединение потоков.

**public static** FeaturesDescriptor createFromProperties(Properties properties) {  
 String sources = properties.getProperty(***FEATURES\_DESCRIPTOR\_FEATURE\_DESCRIPTORS\_SOURCES***);  
 String keys = properties.getProperty(***FEATURES\_DESCRIPTOR\_FEATURE\_DESCRIPTORS\_KEYS***);  
 String sinkSource = properties.getProperty(***FEATURES\_DESCRIPTOR\_SINK\_SOURCE***);  
 String[] sourcesArray = sources.split(**","**);  
 String[] keysArray = keys.split(**","**);  
 List<FeatureDescriptor> featureDescriptors = **new** ArrayList<>();  
 **for** (**int** i = 0; i < sourcesArray.**length**; i++) {  
 FeatureDescriptor featureDescriptor =  
 **new** FeatureDescriptor(sourcesArray[i], keysArray[i]);  
 featureDescriptors.add(featureDescriptor);  
 }  
 **return new** FeaturesDescriptor(featureDescriptors, sinkSource);  
}

**public static class** FeatureDescriptor {  
 **public final** String **source**;  
 **public final** String **key**;  
  
 **public** FeatureDescriptor(String source, String key) {  
 **this**.**source** = source;  
 **this**.**key** = key;  
 }  
}

**public static class** FeaturesDescriptor {  
 **public final** List<FeatureDescriptor> **featureDescriptors**;  
 **public final** String **sinkSource**;  
  
 **public** FeaturesDescriptor(List<FeatureDescriptor> featureDescriptors, String sinkSource) {  
 **this**.**featureDescriptors** = featureDescriptors;  
 **this**.**sinkSource** = sinkSource;  
 }  
}

Основной метод по объединению.

**void** buildStreams(StreamsBuilder builder)

Создаются topic-и, в которых ключам являются, те ключи, по которым нужно объединять.

Serde<String> stringSerde = Serdes.*String*();  
  
List<KStream<String, String>> streams = **new** ArrayList<>();  
  
**for** (FeatureDescriptor featureDescriptor : **featuresDescriptor**.**featureDescriptors**) {  
 KStream<String, String> stream =  
 builder.stream(featureDescriptor.**source**, Consumed.*with*(stringSerde, stringSerde))  
 .map(**new** KeyValueMapperSimple(featureDescriptor.**key**));  
 streams.add(stream);  
}

Само объединение.

KStream<String, String> pref = streams.get(0);  
**for** (**int** i = 1; i < streams.size(); i++) {  
 KStream<String, String> cur = streams.get(i);  
 pref = pref.leftJoin(cur,  
 **new** ValueJoinerSimple(),  
 JoinWindows.*of*(Duration.*ofSeconds*(1)),  
 StreamJoined.*with*(  
 Serdes.*String*(),  
 Serdes.*String*(),  
 Serdes.*String*())  
 );  
}

Отправка в конечный topic.

pref.to(**featuresDescriptor**.**sinkSource**);

Все вместе:

**public void** buildStreams(StreamsBuilder builder) {  
 Serde<String> stringSerde = Serdes.*String*();  
  
 List<KStream<String, String>> streams = **new** ArrayList<>();  
  
 **for** (FeatureDescriptor featureDescriptor : **featuresDescriptor**.**featureDescriptors**) {  
 KStream<String, String> stream =  
 builder.stream(featureDescriptor.**source**, Consumed.*with*(stringSerde, stringSerde))  
 .map(**new** KeyValueMapperSimple(featureDescriptor.**key**));  
 streams.add(stream);  
 }  
  
 **if** (streams.size() > 0) {  
 **if** (streams.size() == 1) {  
 KStream<String, String> stream = streams.get(0);  
 stream.to(**featuresDescriptor**.**sinkSource**);  
 } **else** {  
 KStream<String, String> pref = streams.get(0);  
 **for** (**int** i = 1; i < streams.size(); i++) {  
 KStream<String, String> cur = streams.get(i);  
 pref = pref.leftJoin(cur,  
 **new** ValueJoinerSimple(),  
 JoinWindows.*of*(Duration.*ofSeconds*(1)),  
 StreamJoined.*with*(  
 Serdes.*String*(),  
 Serdes.*String*(),  
 Serdes.*String*())  
 );  
 }  
 pref.to(**featuresDescriptor**.**sinkSource**);  
 }  
 }  
}

Запуск.

**void** run(Properties config)

Конструируется объект объединения потоков (основной объект):

FeaturesStream featuresStream = **new** FeaturesStream(config);

Создается обвязка для kafka.

StreamsBuilder builder = **new** StreamsBuilder();

featuresStream.buildStreams(builder);  
Topology topology = builder.build();  
KafkaStreams streams = **new** KafkaStreams(topology, featuresStream.buildStreamsProperties());

Осуществляется запуск.

streams.start();

Все вместе:

**public static void** run(Properties config) {  
 StreamsBuilder builder = **new** StreamsBuilder();  
 FeaturesStream featuresStream = **new** FeaturesStream(config);  
 featuresStream.buildStreams(builder);  
 Topology topology = builder.build();  
 KafkaStreams streams = **new** KafkaStreams(topology, featuresStream.buildStreamsProperties());  
 CountDownLatch latch = **new** CountDownLatch(1);  
 Runtime.*getRuntime*().addShutdownHook(**new** Thread(() -> {  
 streams.close();  
 latch.countDown();  
 }));  
 **try** {  
 streams.start();  
 latch.await();  
 } **catch** (Throwable e) {  
 System.*exit*(1);  
 }  
 System.*exit*(0);  
}

Пример запуска приложения.

java -jar features-stream-1.0.0.jar -c plain.properties

Язык: Java 1.8.

Библиотеки: kafka 2.6.0, jsoup 1.13.1.

Заключение

Изложенное решение имеет ряд ограничений и не реализует полный функционал. Но имеет и несколько преимуществ.

Во-первых: позволяет быстро конструировать объединение topic-в.

Во-вторых: позволяет быстро запускать объединение в различных средах.

Так же стоит отметить, что решение налагает ограничение на структуру входных данных. А именно topic-и должны иметь табличную структуру. Для преодоления этого ограничения можно ввести дополнительный слой, который будет позволять сводить различные типы сущностей к табличной.

Для промышленной реализации полной функциональности стоит обратить внимание на очень мощный и, самое главное, гибкий функционал: [KSQL]( https://www.confluent.io/blog/ksql-streaming-sql-for-apache-kafka/).

Ссылки и ресурсы

[Исходный код]( https://github.com/kartzum/intro\_to\_apache\_kafka/tree/main/story/resources/k-plain-features)

[Meet Michelangelo: Uber’s Machine Learning Platform](<https://eng.uber.com/michelangelo-machine-learning-platform/>).