Система выполнения моделей машинного обучения на потоке событий

А.Е. Стариков, Д.Е. Намиот

***Аннотация* — В работе рассматриваются существующие системы потоковой обработки событий, разбираются характеристики этих систем, анализируются их преимущества и недостатки. Потоковая обработка значительно сокращает время от момента получения данных до реакции на них по сравнению с пакетной обработкой. Используя такой подход, компании могут своевременно реагировать на поступающую информацию, принимая меры в тот момент, когда они необходимы. В работе приведен анализ систем, использующихся в крупных компаниях для разработки моделей машинного обучения, начиная от сбора данных, заканчивая выводом модели в эксплуатацию. В работе рассматриваются вопросы вывода моделей машинного обучения в продуктивную среду для приложений потоковой обработки, в том числе форматы представления моделей, сравниваются преимущества и недостатки различных подходов. Приведена архитектура и практическая и разработана практическая реализация приложения, позволяющего использовать каждый из рассмотренных форматов.**

***Ключевые слова* — потоковая обработка событий, системы разработки моделей машинного обучения.**

I. Введение

В основание этой статьи положена выпускная квалификационная работа, выполненная одним из авторов на факультете ВМК МГУ имени М.В. Ломоносова.

Рост вычислительных мощностей и удешевления оборудования позволил компаниям собирать и обрабатывать большие массивы данных. Конкурентное преимущество получили компании, которые научились извлекать пользу из имеющихся данных. Информация имеет свойство устаревать, и скорость реакции на ее изменение важна. Для обработки и аналитики данных, генерирующихся на постоянной основе, используют инструменты потоковой обработки.

Другой важной тенденцией в аналитике данных стало использование моделей машинного обучения. Алгоритмы машинного обучения используют для решения многих задач: прогнозирование спроса и оттока пользователей, улучшения качества ранжирования и рекомендаций, предсказание поломок оборудования, улучшения качества инструментов, использующихся в финансах, здравоохранении, работы систем компьютерного зрения и самоуправляемых автомобилей и др. Крупные компании используют в своих продуктах не одну, а десятки и сотни моделей машинного обучения одновременно. Обучение, разработка и внедрение большого количества моделей, над которыми одновременно работаю команды исследователей и инженеров - нетривиальная задача. Для ее решения компании разрабатывают системы для разработки моделей машинного обучения.

Ввиду увеличения популярности потоковой обработки и использования моделей машинного обучения, важным практическим вопросом представляется изучение вопроса применения моделей машинного обучения на потоке событий.

В задачи данной работы входит рассмотрение инструментов потоковой обработки, рассмотрение подходов к разработке систем машинного обучения, использующихся в крупных компаниях, рассмотрение способов использования моделей машинного обучения на потоке событий.

В рамках работы разработана и доведена до практической реализации система, позволяющая использовать существующие подходы к использованию моделей машинного обучения на потоке событий.

Оставшаяся часть статьи структурирована следующим образом. В разделе II дана характеристика потоковой обработки событий, проанализированы преимущества и недостатки существующих систем потоковой обработки. В разделе III рассмотрены системы, использующиеся для работы с моделями машинного обучения в крупных компаниях. Раздел IV содержит описание форматов представления моделей машинного обучения. В разделе V рассмотрены способы использования моделей машинного обучения на потоке событий. В завершающем VI разделе приводится архитектура и описывается разработанное приложения, которое позволяет использовать подходы, описанные в разделе IV.

II. Потоковая обработка событий

A. Общая характеристика потоковой обработки

Потоковая обработка — это обработка данных в момент их создания или поступления в систему, которая работает с бесконечным потоком событий.

Многие типы данных представляют из себя бесконечный поток событий, например данные датчиков и сенсоров приборов, активность пользователей сети интернет или мобильных приложений, банковские транзакции и финансовые операции и др.

Статья получена 29 мая 2020.
А.Е. Стариков - выпускник ВМК МГУ им. М.В. Ломоносова (email: starikovaleksei1992@mail.ru).
Д.Е. Намиот - МГУ им. М.В. Ломоносова (e-mail: dnamiot@gmail.com)

Ранее стандартным подходом для обработки информации была так называемая пакетная обработка [1]. Пакетная обработка — это подход, при котором данные хранятся в хранилище — базе данных или в файловой системе, а затем извлекаются из него в нужном виде в определенный момент времени в форме запроса. Основной недостаток такого подхода — это задержка между созданием данных и их использованием для анализа и принятия решений.

Потоковая обработка меняет эту парадигму. Данные поступают в систему и обрабатываются в реальном времени. Приложение, которое обрабатывает поток данных может отреагировать на пришедшее событие или «запомнить» его для последующей агрегации. Приложение также может обрабатывать несколько потоков событий и порождать новые потоки на их основе.

Потоковая обработка значительно сокращает время от момента получения данных до реакции на них по сравнению с пакетной обработкой. Используя такой подход, компании могут своевременно реагировать на поступающую информацию, принимая меры в тот момент, когда они необходимы.

Парадигма потоковой обработки информации элегантно решает проблемы, с которыми сегодня сталкиваются разработчики систем аналитики, которые должны работать в реальном времени. В данном подходе задержка между получением очередного события, принятием решения и соответствующей реакцией должна быть минимальна.

Потоковая обработка легко моделирует непрерывную природу получения новой информации. По мере получения новых данных мы обновляем состояние системы, а не проводим все вычисления вновь, как происходит в случае с пакетными запросами.

Архитектура систем потоковой обработки позволяет децентрализовать и разъединить инфраструктуру, что уменьшает потребность в больших и дорогих базах данных. Каждое приложение может работать обособленно, поддерживая свое собственное состояние. Такой подход хорошо ложиться на популярную сегодня микро сервисную архитектуру [2].

B. Потоковая обработка с сохранением состояния

Как уже говорилось выше, система потоковой обработки должна уметь сохранять свое текущее состояние. Это необходимо, так как предшествующие события могут влиять на способ обработки последующих событий. Например, приложения для предотвращения мошенничества (fraud detection) должны помнить последние транзакции по кредитной карте для вынесения решения по подозрительной операции. Приложения онлайн рекомендаций хранят параметры, которые описывают пользовательские предпочтения, сформированные на основе предыдущей активности пользователя. Каждое действие, предпринимаемое пользователем, генерирует событие, которое обновляет эти параметры. Приложение, которое отвечает за корзину покупок получает событие после каждого добавления пользователем очередной вещи.

Для получения результатов в реальном времени или почти в реальном времени система должна постоянно вычислять и обновлять результат работы с использованием каждой новой записи или события.

Подход с обновлением состояния хорошо ложиться на многие приложения. Это упрощает инфраструктуру работы с данными, так как большинство приложений могут быть построены с использованием похожих методов. Используя подход с сохранением состояния системы, разработчики могут создавать приложения, которые немедленно реагируют на происходящие события. Например:

- классификация транзакций по кредитным картам на основе аналитической модели, а затем автоматическая блокировка транзакции в случае фрода;
- отправка push уведомлений пользователям на основе действия, которые они выполняют;
- автоматическая настройка параметров автомобиля на основе данных его датчиков.

C. Характеристики систем потоковой обработки

Системы потоковой обработки становятся популярны в настоящие время. Уже существует не одна открытая реализация такого рода систем. Самые известные из них это: Apache Spark (Spark Streaming), Apache Flink, Apache Kafka (Kafka-Streams), Apache Storm, Apache Samza.

Все перечисленные выше системы потоковой обработки являются распределенными, т.е. поддерживают работоспособность не только на одной машине (сервере), но и на нескольких в режиме кластера.

Важные характеристики распределенных систем потоковой обработки данных:

Гарантия доставки сообщений:
Части распределенных систем периодически выходят из строя. Это нормальное состояние системы и при разработки распределенных систем разработчики закладываются на это. Вопрос в том, какую гарантию дает система на доставку сообщений. По этому критерию системы делят на системы, которые гарантируют доставку сообщений:

не более одного раза (atmost-once)
Сообщение будет доставлено 0 или 1 раз. Используется в случаях, когда повторное получение сообщение недопустимо, а потеря сообщения — приемлема.

хотя бы один раз (atleast-once)
Сообщение будет получено 1 или более раз. Используется в случаях, когда нужно обязательно получить сообщение, но повторное получение сообщения не влияет на работу системы или не обрабатывается.

точно один раз (exactly-once)
Сообщение будет получено строго один раз. В общем случае это предпочтительное поведение для большинства систем, однако более сложно реализуемое по сравнению с двумя предыдущими.

Отказоустойчивость:
В случае выхода из строя нодов (машин) кластера или сети система должна уметь восстанавливать свое

состояние и продолжать работу с точки последнего сохранения состояния. Обычно, в системах потоковой обработки этот механизм реализован через механизм сохранения состояний системы (чекпоинтов).

Предоставление доступа к состоянию: Система должна предоставлять доступ к своему состоянию для просмотра или изменения.

Производительность:
Когда говорят о производительности системы потоковой обработки данных, обычно имеют ввиду задержку обработки сообщений (как долго обрабатывается сообщение) и пропускную способность системы (сколько сообщений может быть обработано в единицу времени).

Продвинутые возможности: Здесь имеются ввиду такие возможности системы потоковой обработки как обработка по времени поступления, оконные функции, триггер и другие [3].

Зрелость системы: Прошла ли система или фреймворк проверку временем. Есть ли успешные результаты внедрения в крупных компаниях. Используется и поддерживается ли система сообществом.

Системы потоковой обработки также делятся по реализации обработки сообщений на *строго потоковую* и на *микро-пакетную*. Под строго потоковой системой мы понимаем систему, которая обрабатывает сообщения одно за другим, рассматривая поток, как отдельные события. Под микро-пакетной системой будем понимать систему, которая обрабатывает сообщения в потоке, объединяя их в небольшие группы (пакеты или батчи) по несколько штук. Каждый подход имеет свои преимущества и недостатки. Строго потоковый подход кажется более естественным и обладает меньшей задержкой при обработке. Однако добиться отказоустойчивости, не проиграв в пропускной способности, при таком подходе сложнее, так как чекпоинтинг (сохранение состояния) должен быть произведен после каждого события. Также при строго потоковом подходе доступ к управлению состоянием реализуется проще. Микро-пакетный подход напротив — увеличивает задержку, но мы получаем отказоустойчивость меньшими усилиями, так как чекпоинт создается при получении очередной порции данных. При микро-пакетном подходе также становится сложнее поддерживать управление состоянием.

D. Обзор основных систем потоковой обработки

1) Apache Storm

Одна из самых надежных и проверенных временем систем потоковой обработки данных — это Apache Storm [4]. Это система является строго потоковой и используется для обработки событий одно за другим.
Преимущества:

высокая пропускная способность
низкая задержка
зрелая система (проверена временем)
отлично подходит для несложных стриминговых систем

Недостатки:

отсутствует управление состоянием
отсутствие продвинутых возможностей
гарантия доставки сообщений — хотя бы один раз

2) Spark Streaming

Apache Spark — фреймворк, отлично зарекомендовавший себя для пакетной обработки больших объемов данных [5]. Позднее в фреймворк внедрили и потоковую обработку [6]. Apache Spark стал первым фреймворком полностью поддерживающий лямбда архитектуру [7]. В лямбда архитектуре используется и пакетная и потоковая обработка данных. Пакетная используется для точных расчетов над данными, потоковая — для быстрых расчетов. Для обработки потока событий в Spark Streaming используется микро-пакетный подход.
Преимущества:

- полная имплементация лямбда архитектуры внутри одного фреймворка.
- высокая пропускная способность
- высокая отказоустойчивость за счет использования микро-пакетного подхода
- простой интерфейс для использования фреймворка
- активная поддержка сообществом и частые релизы с улучшениями и исправлением багов.
- гарантия доставки сообщений — точно один раз

Недостатки:

- не подходит для имплементации систем, требующих низкой задержки, ввиду использования не строго потоковой обработки
- большое количество параметров для настройки системы, иногда становится сложно подобрать оптимальные параметры
- отсутствие сохранения состояния
- отстает в реализации продвинутых возможностей в отличии от конкурентов, например, от Apache Flink

3) Apache Flink

Apache Flink развился из академического проекта в университете Berlin TU [8]. Поддержка лямбда архитектуры внедрена в фреймворк. Аналогично Apache Spark поддерживается и пакетная и потоковая обработка. Но существует отличие в подходе к потоковой обработке — здесь она строго потоковая. Интерфейс использования похож на Spark, а внутренняя имплементация функций ближе в Storm. Если Apache Spark приобрел известность за счет пакетной обработке данных, Flink — популярен из-за своей потоковой библиотеки.
Преимущества:

- лидер открытого программного обеспечения в потоковой обработке — все передовые технологии потоковой обработки первыми появляются в этом фреймворке
- первый строго потоковый фреймворк с реализованными продвинутыми функциями
- низкая задержка и высокая пропускная способность
- гарантия доставки сообщений — точно один раз

- используется в крупных известных компаниях для потоковой обработки, таких Uber и Alibaba

Недостатки:

- поддержка сообщества не настолько большая, как у Apache Spark
- пакетная библиотека фреймворка практически не используется — используется только стриминговая часть
- не зрелый — молодой игрок по сравнению с конкурентами

4) Kafka-Streams

Если предыдущие варианты — это полносоставные фреймворки, то Kafka-Streams [9] — легковесная библиотека, предназначенная для потоковой обработки событий, поступающих из Apache Kafka [10]. Эта библиотека считывает событие из Kafka, производит обработку события, и записывает обработанное событие обратно в Kafka. Этот сценарий стандартен для работы строго потоковых систем, поэтому мы можем рассматривать эту библиотеку наряду с полновесными фреймворками.

Преимущества:

- легковесность библиотеки делает ее отличным выбором для использования в микро сервисной архитектуре и для систем, использующихся для интернета вещей
- не требует отдельного кластера для развертывания
- поддерживает некоторые продвинутые возможности потоковых фреймворков
- гарантия доставки сообщений — точно один раз

Недостатки:

- работает только с Apache Kafka — не может использоваться с другим брокером сообщений
- незрелая система, мало применяется в крупных компаниях
- не подходит для больших нагрузок, в отличии от крупных фреймворков

5) Apache Samza

Apache Samza — это фреймворк для потоковой обработки данных [11]. С точки зрения внутренней реализации фреймворк очень похож на Kafka-Streams. Kafka-Streams в компании Confluence [12] фактически разрабатывалась теми же разработчиками, которые ранее реализовывали Apache Samza в компании LinkedIn [13]. Можно рассматривать Apache Samza как масштабируемую версию Kafka-Streams. Если Kafka-Streams — легковесная библиотека, использующееся для микро сервисной архитектуры, то Apache Samza — полноценный фреймворк для потоковой обработки, который может запускаться на кластере под управлением Yarn [14].

Преимущества:

- подходит для обработки большого количества данных.
- отказоустойчивость и высокая производительность за счет работы с Apache Kafka
- зрелая система
- высокая пропускная способность и низкая задержка

Недостатки:

- используется только совместно с Apache Kafka и Yarn
- гарантия доставки сообщений — хотя бы один раз
- реализованы не все продвинутые возможности потоковой обработки

III. Системы разработки моделей машинного обучения

A. Системы крупных компаний

В настоящее время рост вычислительных мощностей и удешевления оборудования позволил компаниям собирать и обрабатывать большие массивы данных. В следствии чего стало возможно с успехом применять разработанные ранее алгоритмы машинного обучения в бизнесе, что послужило мощным толчком к развитию данного направления в мире. Компании стали разрабатывать и с успехом применять различные модели машинного обучения.

Под моделью машинного обучения будем понимать систему, которая обучена на исторических размеченных данных и способна делать предсказания для новых неразмеченных данных в случае обучения с учителем. В случае обучения без учителя моделью будем называть систему, которая может разделять объекты по группам или детектировать аномалии используя, заложенный алгоритм.

Использование моделей машинного обучения на больших данных требует от компаний разрабатывать и поддерживать инфраструктуру, а также инструменты для построения и использования этих моделей. На рынке существуют готовые решения для разработки и внедрения моделей машинного обучения. В основном, это разработки крупных корпораций, таких как SAS [16], Google [17], Microsoft [18].

Одновременно с развитием проприетарных решений развивались свободное программное обеспечение, призванное решать схожие задачи. Поэтому сегодня многие компании идут по пути использования свободных программных продуктов и построения на их основе систем для обработки данных и построения моделей машинного обучения.

Такие системы включают в себя:

- хранилища данных, которые могут состоять из распределенных файловых систем (hdfs), NoSql хранилищ (Cassandra, Mongo, Redis) и реляционных баз данных (MySql, Postgres);
- системы обработки данных: пакетные (Apache Spark, Google BigQuery) и потоковые (Apache Flink, Apache Spark, Apache Storm);
- брокеры сообщений (Apache Kafka, Rabbit MQ);
- фреймворки и библиотеки для построения моделей машинного обучения (MLLib, XGBoost, Tensorflow, scikit-learn);

- инструменты мониторинга и визуализации (Grafana, Zeppelin)

Системы, построенные на базе таких инструментов, позволяют реализовывать процессы работы с данными и использование моделей машинного обучения на этих данных.

Системы разработки машинного, которые разрабатываются сейчас крупными компаниями, призваны упростить, и унифицировать процесс разработки моделей машинного обучения. Из разработанных систем на данный наибольшую известность получили: FBLearner от Facebook, TFX от Google и Michelangelo от Uber. Сервисы этих компаний активно используют модели машинного обучения для решения своих задач и соответствуют вышеописанным процессам. Большое количество специалистов работают над их развитием. Не все из них имеют обширный опыт в качестве разработчиков погромного обеспечения, но должны иметь возможность развивать сервисы, работающие на основе моделей машинного обучения. С целью упрощения разработки моделей машинного обучения внутри компании разрабатываются специальные фреймворки.

1) FBLearner

Facebook — крупная социальная сеть с аудиторией в 2,5 миллиарда активных пользователей в месяц [19]. Компания стремиться предоставить пользователю максимально релевантный для пользователя контент с момента его первого посещения сайта. Модели машинного обучения используются в компании для персонализации новостной ленты, фильтрации не релевантного для пользователя контента, предоставление пользователю актуальных новостей, индексации поиска, машинного перевода, классификации фото и видео в реальном времени [20]. Для обслуживания столь большой аудитории компания разрабатывает инфраструктуру, в том числе вычислительные устройства, жесткие диски и собственные центры по обработке данных, а также программное обеспечение [21].

С целью упрощения написания и использования моделей машинного обучения Facebook создал фреймворк, который предоставляет машинное обучение в качестве сервиса (ML-as- a-service) (Рис.1).

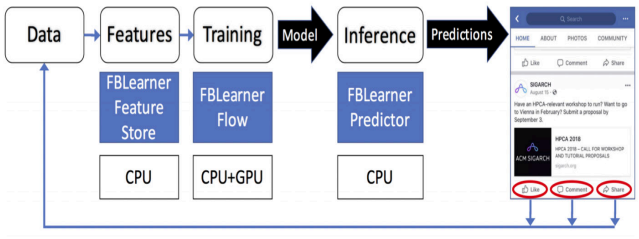

Рис. 1. Схема работы FBLearner

Фреймворк позволяет создавать данные для обучения моделей, обучать модели, осуществлять предсказания с использованием обученных моделей. Он призван сделать инженеров, которые создают модели машинного обучения, более продуктивными и высвободить их время на создание и улучшение новых моделей и алгоритмов.

Фреймворк FBLearner состоит из 3-х модулей: модуль хранение данных для обучения (FBLearner Feature Store), модуль построения и обучения моделей (FBLearner Flow) [22], модуль применения моделей (FBLearner Predictor).

Модуль хранения данных для обучения моделей Модуль представляет из себя хранилище параметров для обучения моделей. Эти параметры могут использоваться также при использовании готовых моделей. В компании используется большое количество источников информации, на основе которых создаются эти параметры. Создав параметры один раз, их можно поместить в хранилище для дальнейшего использования. С момента сохранения параметров в системе они могут быть использованы другими командами для разработки своих моделей.

Модуль построения и обучения моделей Модуль используется для описания процесса обучения и применения моделей (Рис. 2). В нем также описываются ресурсы, которые требуются для этого процесса. Процесс описывается с помощью последовательности блоков или операторов и представляет из себя блок-схему. Каждый оператор имеет вход и выход, с помощью которых описывается взаимодействие между блоками. С помощью модуля можно управлять расписанием запуска процессов. Пользователи модуля работают с ним через интерфейс, в котором они могут строить описанные процессы с помощью схем. Используя интерфейс с богатыми возможностями настройки, специалисты могут подготавливать нужные для модели данные и проверять эти данные без написания кода. Такой подход упрощает процесс обучения и применения моделей.

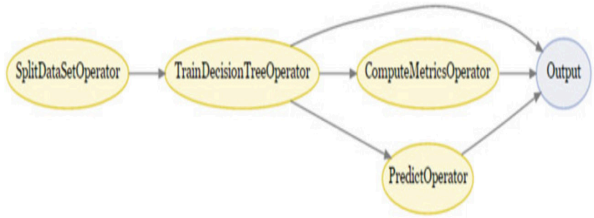

Рис. 2. Пример блок-схемы описания процесса из модуля построения и обучения моделей

Модуль применения моделей Модели, обученные в предыдущем модуле, применяются в модуле применения моделей. Под применением моделей имеется ввиду получение предсказания, которое производит модель. Модуль позволяет производить предсказания в реальном времени. Его также используют для проведения экспериментов при подборе моделей с различными параметрами. Изменяя применяемые алгоритмы и параметры модели, инженеры смотрят на результаты модели и имеют возможность выбрать лучшую из имплементаций. Все проведенные эксперименты индексируются в системе поиска (Elasticsearch [23]).

Написав запрос к системе, инженер может найти модель с нужными настройками, которую он использовал ранее. Модуль позволяет осуществлять версионный контроль моделей.

Система поддерживает большое количество алгоритмов уже готовых к использованию (классические модели: логистическую регрессию, метод опорных векторов, градиентный бустинг, основанный на деревьях решений, и нейронные сети: полносвязные, сверточные, рекуррентные). Новые модели добавляются разработчиками компании по мере разработки. Эти модели сразу становятся готовы для переиспользования в новом процессе.

2) TFX

Google — международная компания знаменитая не только своим поисковым движком, но и сервисами, разработанными на основе искусственного интеллекта и машинного обучения. Для создания и обслуживания моделей машинного обучения в продуктивной среде Google разработал систему, состоящую из следующих компонент:

- модуль для создания и обучения моделей
- модули для создания и проверки данных и моделей
- инфраструктуру для запуска и поддержания работоспособности моделей в продуктивной среде

Результатом работы стала платформа для машинного обучения общего назначения – TFX (TensorFlow Extended) [24]. При разработке Google полагался на следующие принципы:

одна платформа для различных задач машинного обучениях

В качестве основной средства для разработки моделей используется Tensorflow — открытая платформа, содержащая инструменты и библиотеки для создания моделей машинного обучения [25]. Tensorflow предназначена для работы с нейросетями, но в TFX также используются другие библиотеки и инструменты, позволяющие работать с широким спектром моделей, это открытые инструменты, позволяющие
работать с моделями машинного обучения на больших данных [26].

непрерывное обучение моделей

Процесс обучения моделей обычно представлен как направленный ациклический граф (directed acyclic graph, DAG) [27, 28], последовательность его вершин задает различные операции в этом процессе — подготовку данных, обучение модели, выполнение модели. Если требуется обучить модель на новых данных граф будет перезапущен целиком, но во многих случаях модель должна дообучаться на постоянной основе. С этой целью в TFX применяются алгоритмы динамической подачи данных для обучения модели, а также использование параметров модели, полученных на предыдущих этапах обучения (разогретый старт).

легкая настройка системы

Пользователи TFX используют единую настройку системы, а отдельные внутренние модули системы используют эту общую конфигурацию.

отказоустойчивость и масштабируемость

Написание алгоритма модели только маленькая часть системы машинного обучения [29]. Если платформа способна скрыть сложность вывода моделей в продуктивную среду от разработчиков моделей машинного обучения, высвободившееся время они могут потратить на усовершенствование алгоритмов. Также не всегда очевидно, как будет вести себя модель на новых данных [30], поэтому в системе должна быть предусмотрена проверка качества работы моделей. Наряду с этим требуется также проверка качества данных для обучения моделей. Предварительная проверка данных и моделей необходимо для поддержания отказоустойчивости системы. Для реализации масштабируемости системы в TFX используется открытый фреймворк для обработки больших объемов данных, разработанный Google — Apache Beam [31].

TFX состоит из следующих модулей: анализа данных, преобразования данных, проверки данных, обучения моделей, проверки моделей и модуля использования моделей (Рис. 3).

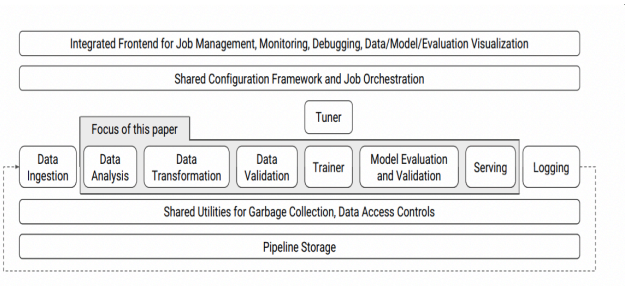

Рис. 3. Общая схема платформы машинного обучения TFX

Модуль анализа данных

В модуле происходит анализ данных, имеющихся в системе и подсчет статистик и показателей по ним (количество параметров, распределения параметров, средние значения, квантили, стандартные отклонения и др.). Используя эти статистики, пользователи составляют представление об имеющихся данных в целом. У пользователей также имеется возможность расширять модули расчета показателей новыми статистиками.

Модуль преобразования данных

Для корректной и эффективной работы моделей данные для них нужно преобразовывать к требуемому формату. В некоторых случаях количественные признаки требуется перевести в категориальные, некоторые признаки требуют масштабирования и др. операции. Такие преобразования не только позволяют сделать работу модели эффективнее, но и более оптимально использовать вычислительные ресурсы и оптимизировать хранение информации.

Модуль проверки данных

После завершения анализа данных требуется проверить, что с данными все в порядке, нет ли аномалий в данных, о которых стоит сообщить пользователю. Для проверки данных в TFX используется схема данных. Для отслеживания изменений в схеме можно использовать версионный контроль. В схеме используется следующая

информация: наименование переменной, тип, возможные значения переменной и др.

Модуль обучения моделей
Модуль поддерживает офлайн и онлайн обучение моделей. Под офлайн обучением понимают классическое обучение на размеченных данных с последующим использованием модели. Под онлайн обучением понимают дообучение модели в процессе эксплуатации. Модуль поддерживает смену алгоритма обучения без изменения всего процесса. Также имеется механизм предварительного обучения моделей, позволяющий командам использовать модель с предварительно обученными весами и не тратить большую часть времени на обучение каждой модели. Такой эффект достигается за счет предварительного обучения базовой нейросети на обобщенных данных [32]. Затем дополнительные слои доучиваются уже на тренировочной выборке [33]. Такой подход в машинном обучении называется transfer learning.

Модуль проверки моделей
Модели машинного обучения зачастую являются частями сложных систем, которые содержат большое количество источников данных и других связных компонент. В таких сложных системах могут возникать непредвиденные ситуации и баги. Например, разные модули системы могут принимать на вход модели в разных форматах сериализации. Человеку бывает сложно отслеживать такого рода ошибки, поэтому хорошо иметь автоматизированный модуль, который будет проверять модель на корректность при ее развертывании в продуктивную среду. Наряду с очевидными ошибками, такими как ошибки компиляции, выдача результата в неверном формате, чрезмерное использование процессора и оперативной памяти, имеет смысл проверять в автоматизированном режиме корректность работы модели с точки зрения здравого смысла. Установив разумные пределы результатов работы модели, можно проверять его не только на корректный формат, но и на правильность работы.

Модуль использования моделей
Получить эффект от моделей машинного обучения возможно только при эффективном использовании этих моделей. Для использования моделей в TFX используется фреймворк Tensorflow Serving [34]. Его разработали в Google как фреймворк, который позволяет гибко использовать модели, поддерживать различные алгоритмы и эксперименты. Как и любой модуль, модуль использования моделей должен быть масштабируемым, отказоустойчивым и иметь низкую задержку.

3) Michelangelo

Michelangelo — это платформа для машинного обучения, разработанная в Uber [35]. Uber — технологическая компания, которая предоставляет сервис такси клиентам по всему миру. В своих продуктах они также использует модели машинного обучения. Michelangelo позволяет инженерам разрабатывать, запускать в продуктивной среде и использовать модели машинного обучения. Система разработана для поддержания процесса построения и использования моделей: подготовка данных, обучение модели, проверка модели, вывод модели в продуктивную среду, использование модели, мониторинг работы модели. Michelangelo поддерживает классические алгоритмы машинного обучения, алгоритмы на временных рядах и глубокое обучение. Десятки команд в Uber используют Michelangelo для исследований и разработки моделей. Система работает на нескольких центрах данных компании.

Предпосылки создания системы
До появления системы Michelangelo в компании разработчики моделей машинного обучения использовали различные открытые инструменты для создания моделей (R, scikit-learn, собственные алгоритмы), а разрозненные команды использовали собственные инструменты для вывода моделей в продуктивную среду. В компании не было единого репозитория для хранения результатов экспериментов, а обучение моделей производилось на персональных компьютерах аналитиков.

Систему Michelangelo разработали с целью стандартизировать процесс разработки моделей внутри компании, стандартизировать инструменты, которые применяются для разработки моделей, дать возможность разработчикам масштабировать модели в продуктивной среде.

Для вывода моделей в продуктивную среду в системе используются средства контейнеризации и виртуализации, которые будут рассмотрены в главе 3.

Устройство системы
В первую очередь, в системы используются открытые библиотеки и программные продукты (HDFS, Spark, Samza, Cassandra, MLLib, XGBoost, Tensorflow). Мы уже упоминали об этих системах ранее. Некоторые из этих продуктов расширялись и улучшались разработчиками Uber специально для системы Michelangelo.

Процесс разработки и использования моделей
Michelangelo позволяет строить процесс разработки и использования моделей, состоящий из следующих стадий: подготовка данных, обучение модели, проверка модели, вывод модели в продуктивную среду, получения предсказания с использованием модели, мониторинг работы моделей.

Разберем этапы процесса в отдельности:

Подготовка данных. Определение характеристик для дальнейшего обучения из сырых данных, зачастую является сложнейшей частью процесса создания модели машинного обучения, создания инструментов для работы с данными — самой дорогостоящей частью процесса. В Michelangelo модуль для работы с данными делится на онлайн и офлайн составляющие. Офлайн используется для доставки пакетов данных для обучения моделей офлайн. Онлайн используется для потоковой обработки данных. В системе имеется модуль, который позволяет сохранять различные характеристики из данных в едином хранилище и использовать часто встречаемые характеристики в разных моделях различных команд.
Иногда некоторые характеристики используется в разных моделях в различных форматах. Так, например время и дата могут быть представлены в виде timestamp, где представлено полная дата и время в миллисекундах,

а могут в виде только дня недели. Для выборки характеристик из хранилища в нужном формате в Michelangelo используется специальный DSL (Domain specific language), написанный на языке программирования Scala.

Обучение моделей. Модуль обучения моделей позволяет тренировать в распределенном режиме модели решающих деревьев, линейные и логистические модели, модели, работающие на временных рядах, модели обучения без учителя (k-means), нейронные сети. В качестве параметров обучения модели используются: тип модели, гипер- параметры модели (параметры настройки модели, которые зависят от типа модели, например количество скрытых слоев в нейросети или количество соседей в алгоритме k-means), ссылки на источники данных и DSL выражения для получения характеристик из данных, ресурсы, требуемые для обучения модели (количество машин, количество памяти, требуются ли видеокарты и др.). Модуль позволяет конфигурировать обучение моделей через пользовательский интерфейс или с использованием API через Jupyter Notebook [36]. Jupyter Notebook широко используется для создания моделей машинного обучения.

Проверка модели. Обучение модели являются частью итеративного процесса создания модели, которая подходит для решения конкретной бизнес задачи. На каждой итерации обучается модель с немного измененными параметрами (характеристики данных, гипер-параметры модели, тип алгоритма). Проведя несколько итераций, можно подобрать оптимальные параметры для решения данной задачи. Обучения модели — довольно долгий процесс, и аналитики проводят обучения десятки тысяч моделей. Michelangelo позволяет сохранять результаты обучения моделей (кем и когда тренировалась модель, на каких данных, с использованием каких параметров и др.), проверять их и сравнивать между собой. Также система имеет удобный функционал для проверки моделей: построение отчета о точности модели (в том числе в графическом виде), визуализация решающих деревьев, важность характеристик данных для результата работы модели и др.

Вывод модели в продуктивную среду. В Michelangelo для вывода модели в продуктивную среду возможно использовать графический интерфейс или API. Поддерживается вывод моделей офлайн, онлайн и в режиме библиотеки. *Офлайн вывод модели в продуктивную среду*
Модель выводится в продуктивную среду в виде контейнера, в котором запускается Spark приложение. Приложение работает с пакетом данных и запускается либо единожды, либо по расписанию. *Онлайн вывод модели в продуктивную среду*
Модель выводится на кластер из нескольких машин (кластер может содержать сотни машин). Используя вызов удаленных процедур (RPC) пользователи посылают запросы на получение предсказания. Вывод модели в продуктивную среду в режиме библиотеки
Модель используется в отдельном сервисе в виде библиотеки. Сервис выводится в продуктивную среду в виде контейнера, на отдельные машины кластера. Контейнер содержит модель в виде архива в формате zip. Обращаться к сервису можно по Java API. Так как модели машинного обучения не имеют состояния (stateless), то можно легко их масштабировать. В случае онлайн вывода, пользователи добавляют новые контейнеры в кластер, а распределитель нагрузки выполнит оставшуюся работу. В случае офлайн вывода, Spark запустит больше параллельных приложений.

Получение предсказания. После вывода контейнера с моделью в продуктивную среду, характеристики данных, преобразованные с помощью DSL, преобразуются в вектор, который поступает в модель. Для данного вектора осуществляется предсказание модели. Если модель используется в онлайн режиме, то результат работы модели возвращается сервису, который запрашивал предсказание. В случае офлайн режима — результат работы записывается в хранилище (Hive). Далее этот результат доступен для других приложений, а также пользователи могут писать запросы к результатам, используя SQL-подобный синтаксис. В системе одновременно работают тысячи моделей. Пользователи могу запускать различные модели для одного сервиса и, сравнивая результаты, проводить A/B тестирование.

Мониторинг работы моделей и системы. Под мониторингом работы моделей понимается отслеживание изменение точности модели с течением времени. Все модели обучаются на исторических данных и со временем их точность может ухудшаться, поэтому есть смысл следить за метриками качества моделей. Например, для регрессионных моделей можно следить за изменением среднеквадратичной ошибки. В системе ведется лог точности предсказания модели с течением времени и в случае ухудшения точности, подается сигнал в систему мониторинга. Системы разработки и внедрения моделей машинного обучения зачастую состоят из множества взаимосвязанных компонент и выход из строя отдельных ее частей может повлиять на работоспособность системы в целом. Поэтому есть смысл также отслеживать работоспособностью всей системы.

На рисунке 4 представлена общая архитектура системы Michelangelo.

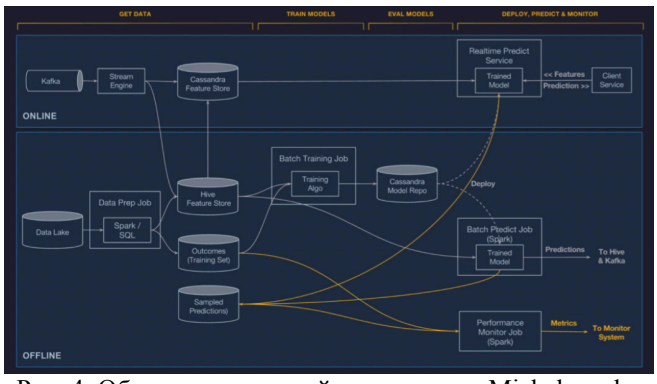

Рис. 4. Общая схема устройства системы Michelangelo.

4) MLFlow

Все три системы (FBLearner, TFX, Michelangelo) разработаны в крупных компаниях для решения в том числе и для проблем версионирования,

воспроизводимости и вывода моделей машинного обучения в продуктивную среду. Эти системы разработаны для внутреннего использования и используются только внутри компании разработки. Однако потребность в работе с моделями машинного обучения возникают и в компаниях, где недостаточно средств и ресурсов для разработки собственной системы по работе с моделями машинного обучения.

Компания Databricks [37] начала разработку открытого фреймворка для построения процессов работы с моделями машинного обучения - MLFlow [38]. MLFlow на данный момент состоит из 3 модулей: модуль логирования метрик (tracking), модуль проекта моделей машинного обучения (projects), модуль пакетирования и вывода моделей в продуктивную среду (models).

модуль логирования метрик
Модуль служит для логирования параметров запуска и результатов работы моделей машинного обучения. Используя API библиотеки, пользователи могут логировать требуемые параметры в коде. Логирование может осуществляться локально в файл или централизованного на сервере с дальнейшим сравнением результатов между моделями. К сохраненным параметром предоставляется доступ через пользовательский интерфейс. Интерфейс позволяет наглядно сравнивать параметры запусков и результаты работы моделей, а также группировать запуски по экспериментам. Пользовательский интерфейс вдохновлен другими средствами визуализации для моделей машинного обучения, такими как Sacred [39], ModelDB [40], TensorBoard [41].

модуль проекта моделей машинного обучения
Модуль проекта моделей машинного обучения указывает на директорию или git репозиторий [42] с кодом модели и пользовательский файл (на языке yaml), где описаны зависимости и способ запуска модели. Используя модуль проекта модели MLFlow, вы можете запускать модель, код которой находится в репозитории на GitHub [43], а также соединять их в последовательные блоки для запуска.

модуль пакетирования и вывода моделей в продуктивную среду
Модуль позволят пакетировать модели в собственный формат MLFlow (flavors) и содержит инструменты для вывода моделей в формате flavors в продуктивную среду. Модель машинного обучения сериализуется в формат pickle — стандартный формат сериализации языка python. В файле конфигурации (yaml) описано каким образом нужно запускать модель, которая сериализована в pickle. Модель может быть функцией на языке python (python_function) или написана с помощью одной из библиотек для разработки моделей машинного обучения, например sklearn [44].

Как мы видим, процесс разработки моделей машинного обучения — важный вопрос в современной разработке. Крупные компании разрабатывают собственные системы для упрощения процесса разработки моделей. Сообщество также работает над открытыми системами такого рода.

B. Разработка моделей машинного обучения.

Разработка моделей машинного обучения отличается от стандартной разработки программного обеспечения. При разработке ПО существует определенный набор бизнес требований, которые необходимо реализовать. Разработка моделей машинного обучения — больше исследовательская задача. Разработчики моделей машинного обучения (мл- инженеры) экспериментируют с новыми наборами данных, алгоритмами, библиотеками, подбирают параметры и оптимизируют метрики, такие как среднеквадратическое отклонение. В разработке моделей машинного обучения важен такой параметр как воспроизводимость. Скорость обучения модели сильно зависит от входных данных и применяемого алгоритма. После обучения модели и подбора мета признаков необходимо вывести получившуюся модель в промышленную эксплуатацию. Это может быть модель, к которой вы обращаетесь по API или же обновляется на регулярной основе приложение, работающее с определенным набором данных. Вывод моделей может представлять особую сложность, когда требует взаимодействия людей из различных команд, представители которых не являются экспертами в машинном обучении. Для упрощения использования моделей были разработаны универсальные форматы для представления моделей машинного обучения.

IV. Форматы представления моделей машинного обучения

Чтобы абстрагировать работу системы машинного обучения от реализации конкретной модели можно представлять модель в каком-либо универсальном формате. Например, можно использовать один из видов сериализованного представления моделей. Преимущество использования сериализованного формата, в том, что его можно использовать в системе не привязываюсь к технологиям, на которых была разработана модель. Например, вы можете разработать модель машинного обучения, использую стандартные python библиотеки, сохранить в унифицированный формат и использовать модель уже в другой среде, реализованной, например, на Java.

A. Predictive Model Markup Language

Predictive Model Markup Language (PMML) — формат сериализации моделей машинного обучения, основанный на xml [45]. Формат разработан Data Mining Group (DMG) [46]. Структура модели сохраняется в виде xml документа [47] и может быть десериализована с использованием библиотек. Разработаны библиотеки для сериализации моделей в PMML формат из многих популярных средств разработки моделей, таких как, scikit-learn, Keras, XGBoost, LightGBM, Tensorflow, R. Для многих систем, которые могут работать с моделями машинного обучения разработаны библиотеки десериализации моделей из PMML формата [48]. В Apache Spark имеется библиотека jpmml-sparkml для десериализации модели из формата PMML в модель из

spark.mlib библиотеки [49]. Для Apache Flink — это библиотека flink-jpmml [50]. Также имеется Java API для работы с моделями PMML в Java приложении [51].

B. Tensorflow SavedModel

SavedModel — универсальны формат представления моделей, созданных и обученных с помощью Tensorflow [52]. Модель, сохраненная с помощью формата SavedModel, представляет из себя не единственный файл, как в случае с PMML, а структуру директорий, в которых содержится информация о модели, в том числе сериализованный в формате protobuf [53] граф модели.

C. Open Neural Network Exchange

Open Neural Network Exchange (ONNX) — формат представления моделей глубокого обучения [54]. Модели глубокого обучения — модели использующие глубокие нейронные сети. Формат изначально разработан компаниями Microsoft и Facebook с целью помочь разработчикам унифицировано работать с моделями машинного обучения, разработанными в различных системах. ONNX позволяет тренировать модель в одной системе, а затем использовать ее в другой. Формат сериализует модели в protobuf. Для сериализации модели ONNX представляет модель в виде графа. Формат разработан и направлен на работу с моделями глубокого обучения. ONNX поддерживает работу с следующими фреймворками и библиотеками: Caffe, Caffe2, CoreML, Keras, LibSVM, LightGBM, MATLAB, ML.NET, MXNet, PyTorch, SciKit-Learn, SINGA, TensorFlow [55].

В формате ONNX представлена библиотека предобученных моделей ONNX Model Zoo [56]. В библиотеке имеются нейронные сети для классификации изображений, детектирования объектов, анализа мимики и жестов, обработки аудио и голоса.

D. MLeap

MLeap — открытый формат для представления моделей машинного обучения [57], но не является стандартом в отличии, например от PMML. Он позволяет сериализовать модели в формат json или protobaf. Компоненты написаны на Scala. MLeap подразумевает работы с ограниченным количеством фреймворков — это scikit-learn, Spark и Tensorflow. Эти фреймворки самые используемые в настоящее время. Mleap может сериализовать не только саму модель, но и трансформации данных, которые происходят до и после применения модели, например подготовка данных и трансформация результатов модели. Последовательность трансформация данных, модель, трансформация результатов работы модели, называют пайплайном модели. Именно этот пайплайн сериализуется в формат mleap и называется mleap Bundle. С помощью бибилиотек mleap Runtime, mleap Bundle затем может использоваться в Java приложении, которое не будет содержать исходных зависимостей модели от Spark или Tensorflow.

E. Portable Format for Analytics (PFA)

Portable Format for Analytics - второй после PMML формат сериализации моделей машинного обучения, разработанный Data Mining Group (DMG) [58]. PFA – json-подобный язык, который позволяет описывать и представлять в сериализованном виде трансформации данных для работы с моделью и саму модель, а также математические и статистические функции [59]. Разработчики языка рассматривают его не только как формат сериализации моделей машинного обучения, но и как формат работы с аналитическими функциями. В отличие от PMML имеет модули, описывающие преобразования данных до и после работы аналитического модуля.

При работе на больших наборах данных PFA предусматривает работу с данными в формате Avro. Avro – формат сериализации данных, при котором схема данных хранится в формате json, а сами данные в бинарном формате. Avro используется в системах, которые работают с Apache Kafke и Hadoop для передачи и хранения данных [60]. Десериализация языка доступна в Java и Python приложениях с использованием открытых библиотек [61] и может использоваться в фреймворках Hadoop, Spark, Storm.

V. Способы вывода моделей машинного обучения в продуктивную среду

Разработка модели машинного обучения - достаточно сложная задача, но, чтобы получить выгоду от использования модели нужно решить не менее сложную задачу по выводу вашей модели в продуктивную среду. Способ вывода модели зависит от многих факторов:

- требования к задержке ответа модели - модель должна отвечать с минимальной задержкой или возможна задержка в несколько секунд или даже минут
- как часто планируется обновление модели
- как часто к модели поступают запросы
- с каким объемом данных вы работаете
- какие типы моделей вы планируете использовать
- требуется ли мониторинг работы модели и др.

В зависимости от конкретной ситуации вы можете использовать различные подходы к использованию моделей машинного обучения в продуктивной среде. Рассмотрим общий подход к выводу моделей, а также варианты специфические для потоковой обработки.

A. Вывод модели в качестве сервиса

1) Rest web service

Широко известным подходом к продуктивизации моделей машинного обучения считается использование модели в качестве сервиса и обращению к сервису по REST API. Часто для создания REST сервиса предлагают использовать фреймворк Flask и реализовывать приложение с помощью языка программирования Python [62]. При таком подходе вам потребуется предобучить модель, сохранить ее в стандартный формат сериализации языка Python –

pickle. Формат pickle позволяет сериализовывать объекты языка Python на диск, а затем восстанавливать их с диска [63]. Также вы можете использовать любой из рассмотренных в главе 3 форматов сериализации моделей, совместимых с языком Python. После обучения вы сможете использовать десериализованную версию модели в Flask приложении и по REST API возвращать результат работы модели пользователю. Для удобства использования вашего приложения вы можете использовать Docker [64] в качестве инструмента контейнеризации, это позволит вам использовать ваше приложение в различных окружениях. В качестве веб сервера вашего приложения вы можете использовать nginx [65], который отличается лучшей масштабируемостью и большей надежностью по сравнению со встроенным веб сервером Flask [66]. Пример реализации и запуска веб приложения с использованием модели машинного обучения с помощью Flask, Docker и Nginx можно посмотреть здесь [66].

При разработке сервиса для выполнения модели машинного обучения вы не ограничены рассмотренным набором инструментов. Вы также можете использовать язык программирования Java, с характерными для него инструментами разработки веб приложений, такими как Spring Framwork [67] и Apache Tomcat [68, 69]. При таком подходе вам придется использовать форматы сериализации машинного обучения, совместимые с языком программирования Java.

Способ использования модели в качестве rest сервиса обычно используется для вывода в продуктивную среду отдельной модели. Если вам нужно использовать несколько моделей различных типов, возможно вам подойдет один из сервисов для вывода моделей машинного обучения, рассмотренный в следующем пункте.

2) TensorFlow Serving

Для использования модели машинного обучения в качестве сервиса вы также можете использовать TensorFlow Serving [70]. TensorFlow Serving - открытая система использования моделей машинного обучения, разработанная в Google [71]. TensorFlow Serving используется в качестве модуля запуска моделей в системе TFX, рассмотренной в главе 2. TensorFlow Serving позволяет вам использовать модели машинного обучения, разработанные с использованием фреймворка Tensoflow и сохраненные в формат Tensorflow SavedModel, рассмотренном в главе 3. Вы также можете использовать ONNX формат предварительно преобразовав его в Tensorflow SavedModel [72]. Такая возможность была добавлена в систему в следствие потребности использовать в TensorFlow Serving модели, разработанные в библиотеке машинного обучения PyTorch [73]. Для взаимодействия с моделью после вывода вам нужно использовать gRPC или REST протокол. gRPC - открытый фреймворк для взаимодействия с использованием RPC (Remote Procedure Call) [74, 75], также разработанный в Google.

Основным недостатком использования TensorFlow Serving по сравнению с предыдущим подходом является привязка к фреймворку TensorFlow и возможность использовать модели только в формате Tensorflow SavedModel.

Однако, по сравнению с выводом модели в качестве REST сервиса, TensorFlow Serving имеет и ряд преимуществ:

- Не требуется разрабатывать API взаимодействия
- Поддерживает версионирование моделей
- Поддерживает работу с пакетным предсказанием, т.е. способен кешировать несколько запросов и выдавать предсказание для пакета. Такой подход может быть применим при более долгом предсказании модели и большом количестве клиентов, от которых поступают запросы.
- Поддерживать работу с тяжеловесными моделями (> 2 Gb)
- Дает возможность проводить A/B тестирование моделей

Для масштабирования и управления сервисами, рассмотренных в последних двух пунктах, применяются открытые платформы для запуска моделей машинного обучения такие как KubeFlow [76], Seldon [77], Hydrosphere.io [78]. Эти платформы позволяют вам использовать отдельные модели в качестве микросервисов. Для реализации этой возможности внутри этих систем используется Kubernetes – открытая платформа для работы с микросервисами и распределением нагрузки между ними [79].

На рисунке 5 приведена схема потокового приложения с использованием модели машинного обучения, запущенной в качестве сервиса.

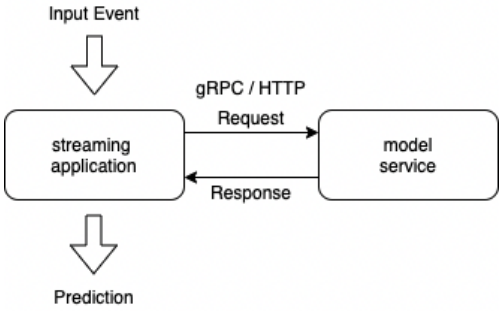

Рис. 5. Использование модели машинного обучения в качестве сервиса на потоке событий

Подходы, рассмотренные в последних двух пунктах, позволяют выполнять вызовы к моделям для получения результата работы. Однако, выполнять gRPC и REST вызовы из приложения потокового вреймворка не всегда приемлемо по требованиям пропускной способности и задержки. Вызов стороннего сервиса осуществляется с использованием сети.

Подход использования моделей машинного обучения в качестве сервиса на потоке событий позволяет использовать общий подход к выводу моделей машинного обучения, который используется не только на потоке событий, а также использовать встроенные инструменты версионирования моделей и проведения A/B тестирования. Основным недостатком такого

подхода является вызов сторонних сервисов из потока, что является антипаттерном потоковой обработки [80].

Рассмотрим способы использования моделей машинного обучения на поток событий, которые не предусматривают вызовы сторонних сервисов.

B. Вывод модели в приложении, обрабатывающем поток событий

1) Отдельное потоковое приложение для каждой модели

Для потоковой обработки в настоящее время наиболее популярны Spark Streaming и Apache Flink, описанные в главе II. Если вы хотите использовать модель машинного обучения на потоке событий, вы можете использовать модель, обученную в соответствующей системе. Для Spark Streaming вы можете обучить модель с использованием фреймворка Apache Spark, используя библиотеку spark.mlib [49]. Библиотека поддерживает широкий набор алгоритмов машинного обучения, позволяет создавать пайплайны (spark.ml.Pipeline), включающие в себя стадии преобразования данных для модели, саму модель и преобразование результатов работы модели. После тренировки пайплайна вы можете сохранить обученную модель, а затем использовать ее в Spark приложении, в том числе и в Spark Streaming приложении. Этот подход будет реализован в системе, описанной в главе 5. Библиотека spark.mlib также дает вам возможность сериализовать модель в формат mleap и для некоторых типов моделей поддерживается формат PMML [81].

Аналогично фреймворку Spark и библиотеки spark.mlib в фреймворке Flink присутствует библиотека flink.ml [82], которая позволяет обучать и применять модели машинного обучения в рамках фреймворка Flink. Количество типов поддерживаемых моделей небольшое по сравнению с фреймворком Spark. Небольшое количество поддерживаемых моделей объясняется тем, что разработчики не позиционируют Flink как инструменты для машинного обучения, а больше, как фреймворк для потоковой обработки. Начиная с версии 1.9 библиотека flink.ml удалена из фреймворка [83]. Дальнейшее развитие библиотека получит в рамках нового API Flink – TableAPI, которое придет на смену существующему DataSet API [84].

Данный подход является самым простым с точки зрения реализации, а также пропускная способность приложения остается высокой, ввиду использования модели прямо в приложении потоковой обработки.

Однако, такое решения не является гибким и требует реализации отдельного приложения потоковой обработки для каждой модели.

2) Потоковое приложение, работающее с сериализованным форматом модели

Форматы представления моделей машинного обучения, рассмотренные в главе IV, позволяют использовать их в Java и Python приложениях. Приложения потоковых фреймворков представляют из себя Java и Python программы, что позволяет использовать в них модели, сериализованные в рассмотренных форматах. На рисунке 6 приведена схема использования модели машинного обучения в приложении, обрабатывающем поток событий.

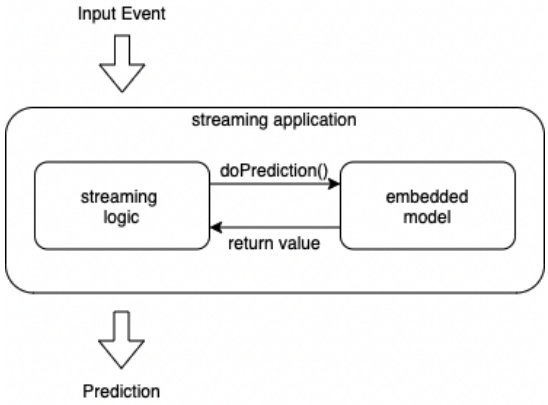

Рис. 6. Использование модели машинного обучения в приложении, обрабатывающем поток событий

Использование модели машинного обучения внутри приложения потоковой обработки позволяет сократить задержки, возникающие при подходе с сервисами. Однако, такой подход требует реализации версионирования моделей и возможности проведения A/B тестирования.

В [80] рассмотрена архитектура потокового приложения, позволяющая реализовать версионирование моделей в рамках одного приложения потоковой обработки. На рисунке 7 приведена архитектура такого приложения.

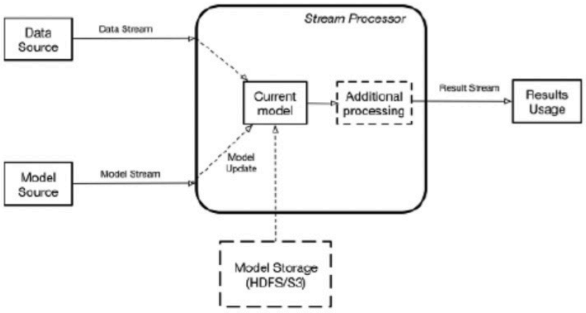

Рис. 7. Архитектура приложения потоковой обработки с возможностью версионирования моделей

Рассмотренные в текущей главе подходы к выводу моделей машинного обучения на поток событий имеют описанные преимущества и недостатки. В главе VI предложена архитектура системы, которая позволяет использовать все рассмотренные подходы. Таким образом, пользователь имеет возможность выбора способа использования модели в зависимости от ситуации.

VI. Система выполнения моделей машинного обучения на потоке событий

A. Архитектура системы

В главе V были рассмотрены различные подходы к выводу моделей машинного обучения на поток событий. Каждый из подходов имеет преимущества и недостатки.

На рисунке 8 изображена архитектура системы выполнения моделей машинного обучения на потоке событий. Предложенная архитектура предоставляет возможность реализовывать и использовать различные способы вывода моделей в рамках одной системы потоковой обработки. Таким образом, имеется возможность использовать сильные стороны подходов для разных моделей.

Пользователь взаимодействует с внешней Кафкой (External Kafka), посылая в нее события и получая из нее предсказания. Приложение состоит из 3-х модулей: input-adapter, модуль вывода моделей машинного обучения, output-adapter. Взаимодействие между модулями происходит с использованием внутренней Кафки (Internal Kafka).

В качестве фреймворка потоковой обработки для модулей input-adapter, output- adapter выбран Apache Flink, так как событийная модель лучше подходит для данной задачи. Инструменты для реализации модуля вывода моделей зависят от способа их использования. Используемые инструменты (Apache Kafka, Apache Flink) позволяют обрабатывать потоки событий с низкой задержкой, обеспечивая высокую пропускную способность. Масштабирование системы осуществляется с использованием возможности масштабирования используемых технологий.

Для демонстрации рассмотрены 2 модели машинного обучения. Первая позволяет определять эмоциональную окраску твита (короткого сообщения Twitter [85]), вторая категорирует цветки ириса (категоризация цветков ириса - классическая учебная задача машинного обучения [86]).

Рассмотрим подробнее отдельные модули системы.

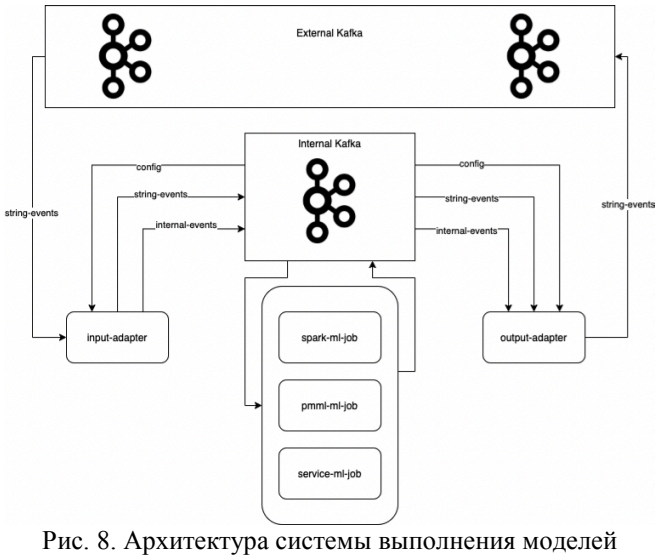

Рис. 8. Архитектура системы выполнения моделей машинного обучения на потоке событий

B. Модуль input-adapter

Модуль input-adapter реализован с помощью фреймворка Apache Flink и позволяет распределять входящий поток событий по моделям, которые работают в системе, для дальнейшей их обработки в модуле вывода моделей. Модуль также позволяет осуществлять трансформации событий, если они требуются для работы с моделью. Можно, например, производить подготовку данных в этом модуле, если требуемая функция уже доступна в приложении. Управление распределением и трансформацией событий осуществляется с помощью конфигурации. Конфигурация передается в модуль через топики Кафки, т.е. имеется возможность обновления конфигурации во время работы приложения. Это позволяет вам добавлять новые модели или перераспределять траффик между работающими моделями, например, для проведения A/B тестирования. Текущая конфигурация является состоянием приложения, таким образом, input-adapter — это приложение потоковой обработки, которое имеет состояние (statefull).

На рисунке 9 изображена схема модуля input-adapter.

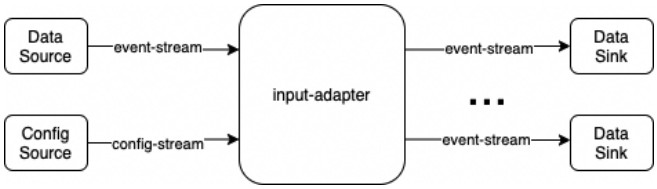

Рис. 9. Схема модуля input-adapter

C. Модуль вывода моделей машинного обучения

Модуль вывода моделей машинного обучения состоит из трех вспомогательных модулей. Каждый вспомогательный модуль реализует один из способов вывода моделей, разобранных в главе V.

1) Модуль spark-ml-job

Вспомогательный модуль spark-ml-job представляет из себя потоковое приложение, реализованное с использованием Spark Streaming. Приложение использует модель машинного обучения, обученную и используемую с помощью библиотеки spark.mlib. Таким образом, модуль реализует способ запуска моделей - отдельное приложение для каждой модели.

В качестве модели используется логистическая регрессия. Модель позволяет определять эмоциональную окраску твита (позитивная или негативная).

2) Модуль pmml-ml-job

Вспомогательный модуль pmml-ml-job представляет из себя потоковое приложение, реализованное с использованием фреймворка Apache Flink. Приложение использует библиотеку flink-jpmml [50]. Библиотека позволяет использовать модели, сериализованные в формате PMML. Таким образом, модуль реализует способ запуска моделей - потоковое приложение, работающее с сериализованым форматом.

При разработке модуля использовалась, архитектура, представленная на рисунке 7. Соответственно, в приложении реализована возможность обновления существующих и добавления новых моделей без прерывания работы приложения.

Для демонстрации работы используется модель, реализующая алгоритм k-средних для кластеризации цветков Ириса.

3) Модуль service-ml-job

Вспомогательный модуль service-ml-job представляет из себя потоковое приложение, реализованное с использованием библиотеки Kafka-Streams. Для запуска моделей машинного обучения используется TensorFlow Serving. Приложение реализовывает подход, вывода моделей машинного обучения в качестве сервиса.

D. Модуль output-adapter

Модуль output-adapter предназначен для обработки результатов работы моделей и направления исходящих событий в требуемые топики Кафки. Модуль реализован с помощью фреймворка Apache Flink. Работа с конфигурацией модуля осуществляется аналогично работе в модуле input-adapter.

На рисунке 10 изображена схема модуля output-adapter.

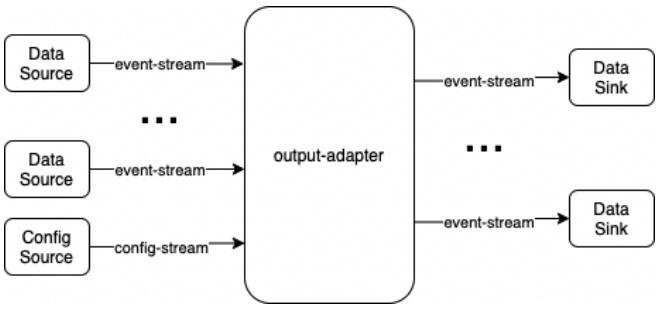

Рис. 10. Схема модуля input-adapter

E. Мониторинг системы

Мониторинг работы системы осуществляется с помощью связки Grafana – Prometheus. Prometheus используется в качестве инструмента хранения метрик [87]. Grafana используется в качестве инструмента визуализации - с помощью этого инструмента мы можем строить диаграммы и графики работы системы [88].

Мониторинг работы приложений Flink осуществляется средствами фреймворка для мониторинга кластера.

F. Дальнейшее развитие системы

В системе реализованы каждый из рассмотренных способов вывода моделей машинного обучения на поток событий. В качестве направления дальнейшего развития можно рассмотреть возможность разработки универсальных приложений, поддерживающих другие форматы использования моделей, рассмотренные в главе IV, а также разработки универсального приложения для работы с моделями в качестве сервиса. Для этой задачи интересно использовать библиотеку Asynchronous I/O Apache Flink [89], которая позволяет выполнять асинхронные вызовы из Flink приложения, что частично решает проблему задержки при вызове сторонних сервисов.

Управление работой системы осуществляется с использованием конфигурации, а мониторинг - сторонними инструментами. Следующим этапом развития системы возможна разработка веб интерфейса для управления и мониторинга системы.

Интересным направлением исследования и дальнейшего развития системы является способы дообучения моделей машинного обучения в режиме работы (онлайн обучение).

Исходный код системы опубликован на GitHub [90].

VII. Заключение

В рамках работы рассмотрена потоковая обработка событий, приведен разбор основных инструментов, использующихся в потоковой обработке, проанализированы характеристики этих инструментов, выделены основные их преимущества и недостатки.

В работе исследован процесс создания моделей машинного обучения в крупных компаниях, проанализированы системы, использующиеся для работы с данными, обучения моделей и использования моделей в продуктивной среде в условиях одновременной работы многих команд.

В работе рассматриваются вопросы вывода моделей машинного обучения в продуктивную среду для приложений потоковой обработки, в том числе форматы представления моделей, сравниваются преимущества и недостатки различных подходов.

В качестве результата разработана архитектура и приведена практическая реализация системы, позволяющей использовать каждый из рассмотренных подходов.

Библиография

[1] Shahrivari S, Jalili S. Beyond batch processing: towards real-time and streaming big data. Computers. 2014; 3(4):117–29.

[2] Thones, J. Microservices. IEEE Software 32, 1, 2015, 116 – 116.

[3] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. PVLDB, 2015.

[4] Iqbal, M.H., Soomro, T.R. Big data analysis: Apache storm perspective. Int. J. Comput. Trends Technol, 2015; 9–14.

[5] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, et al. “Apache Spark: A Unified Engine for Big Data Processing”. In: Commun. ACM 59.11, 2016, pp. 56–65.

[6] Nair, L. R., Shetty, S. D., & Shetty, S. D. Applying spark based machine learning model on streaming big data for health status prediction. Computers & Electrical Engineering, 2017

[7] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja. Lambda architecture for cost-effective batch and speed big data processing. In IEEE Intl Conf. on Big Data, pages 2785– 2792. IEEE, 2015. 14

[8] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, page 28, 2015.

[9] Bejeck, William P, Kafka Streams in action: real-time apps and microservices with the Kafka Streams APIShelter Island, NY: Manning Publications, 2018.

[10] Shree R. et. al., KAFKA: The modern platform for data management and analysis in big data domain, in Proceedings of the 2nd International Conference on Telecommunication and Networks (TEL-NET), 2017.

[11] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell. Samza: Stateful Scalable Stream Processing at LinkedIn. Proc. VLDB Endow., 10(12):1634–1645, Aug. 2017.

[12] Confluence. Retrieved: April 2020 https://www.atlassian.com/ru/software/confluence

[13] LinkedIn. Retrieved: April 2020 https://www.linkedin.com

[14] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas 45 Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, Eric Baldeschwieler: Apache Hadoop YARN: yet another resource negotiator. SoCC 2013:5

[15] Jordan MI, Mitchell TM Machine learning: Trends, perspectives, and prospects, Science 349(6245), 2015, pp. 255–260.

[16] SAS. Retrieved: April 2020 https://www.sas.com

[17] Google. Retrieved: April 2020 https://about.google

[18] Microsoft. Retrieved: April 2020 https://www.microsoft.com

[19] Facebook. Информация о компании. Retrieved: June 2019. https://newsroom.fb.com/company-info

[20] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2018.

[21] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo∗ , Peizhao Zhang, Machine Learning at Facebook: Understanding Inference at the Edge, Facebook, Inc., 2019

[22] Introducing FBLearner Flow: Facebook's AI backbone. Retrieved: June 2019. https://code.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone

[23] Open Source Search & Analytics · Elasticsearch | Elastic. Retrieved: June 2019. https://www.elastic.co

[24] TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines. Retrieved: May 2020. https://www.tensorflow.org/tfx

[25] TensorFlow - An end-to-end open source machine learning platform. Retrieved: May 2020. https://www.tensorflow.org/

[26] Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data 2, 1 (2015), 24

[27] Jimmy J. Lin and Alek Kolcz. 2012. Large-scale machine learning at twitter. In SIGMOD. 793–804.

[28] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2016. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. CoRR abs/1610.09451 (2016).

[29] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, JeanFran¸cois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In NIPS. 2503–2511.

[30] Yann Dauphin, Razvan Pascanu, C¸ aglar G¨ul¸cehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. Identifying and attacking the saddle point problem in high- dimensional non-convex optimization. CoRR abs/1406.2572 (2014).

[31] Apache Beam: An Advanced Unified Programming Model. Retrieved: June 2019. https://beam.apache.org

[32] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. IEEE Trans. on Knowl. and Data Eng. 22, 10 (Oct. 2010), 1345–1359.

[33] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In NIPS. 3320–3328.

[34] Running your models in production with TensorFlow Serving. Retrieved: June 2019 https://ai.googleblog.com/2016/02/running-your-models-in-production-with.html

[35] Jeremy Hermann and Mike Del Balso. Meet Michelangelo: Uber's machine learning platform. https://eng.uber.com/michelangelo, 2017. [Online; accessed 14-April-2019].

[36] Jupyter Notebook. Retrieved: June 2019 https://jupyter.org

[37] Databricks Inc. Retrieved: June 2019 https://databricks.com

[38] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with MLflow. IEEE Data Engineering Bulletin, 41(4), 2018.

[39] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jurgen Schmidhuber. The Sacred Infrastructure for Computational Research. In Katy Huff, David Lippa, Dillon Niederhut, and M. Pacer, editors, Proceedings of the 16th Python in Science Conference, pages 49 – 56, 2017.

[40] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. In Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA '16, pages 14:1–14:3, New York, NY, USA, 2016. ACM.

[41] Google. Tensorboard: Visualizing learning. Retrieved: June 2019 https://www.tensorflow.org/guide/summaries_and_tensorboard.

[42] Git. Retrieved: June 2019. https://git-scm.com 47

[43] GitHub. Retrieved: June 2019. https://github.com

[44] Sklearn. Retrieved: June 2019. https://scikit-learn.org/stable

[45] Zhang YY, Jiao YQ. Design and Implementation of Predictive Model Markup Language Interpretation Engine. 2015 International Conference on Network and Information Systems for Computers (ICNISC). 2015:527–31. 10.1109/Icnisc., 2015.105.

[46] Data Mining Group. Retrieved: June 2019. http://dmg.org

[47] Pmml examples. Retrieved: May 2020. http://dmg.org/pmml/pmml_examples/index.html,

[48] Library for serialization and deserialization PMML models. Retrieved: May 2020. https://github.com/nyoka-pmml/nyoka

[49] Spark.mlib library. Retrieved: May 2020. https://spark.apache.org/mllib/

[50] Flink-jpmml library. Retrieved: May 2020. https://github.com/FlinkML/flink-jpmml

[51] Java PMML API. Retrieved: May 2020. https://github.com/jpmml

[52] Tensorflow, saved model. Retrieved: June 2019. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/python/saved_model

[53] Protobuf. Retrieved: June 2019. https://developers.google.com/protocol-buffers/?hl=ru

[54] ONNX. Retrieved: June 2019. https://onnx.ai

[55] ONNX libraries. Retrieved: May 2020. https://github.com/onnx/tutorials

[56] ONNX Model Zoo. Retrieved: May 2020. https://github.com/onnx/models

[57] MLeap. http://mleap-docs.combust.ml, Retrieved: June 2019.

[58] J. Pivarski, C. Bennett and RL. Grossman, "Deploying Analytics with the Portable Format for Analytics (PFA)", Proceedings of the International Conference of Knowledge Discovery and Data Mining, (2016)

[59] PFA description. Retrieved: May 2020. http://dmg.org/pfa/docs/motivation/

[60] Plase D., Niedrite L., Taranovs R. Comparison of HDFS compact data formats: Avro Versus Parquet // Mokslas-Lietuvos ateitis. 2017. No. 9. P. 267-276.

[61] Titus 2 - Portable Format for Analytics (PFA) implementation for Python 3.5+. Retrieved: May 2020. https://github.com/animator/titus2

[62] How to Easily Deploy Machine Learning Models Using Flask. Retrieved: May 2020. https://towardsdatascience.com/how-to-easily-deploy-machine-learning-models-using-flask-b95af8fe34d4

[63] Python serialization format – pickle. Retrieved: May 2020. https://docs.python.org/3/library/pickle.html

[64] Docker. Retrieved: May 2020. https://www.docker.com/

[65] Nginx. Retrieved: May 2020. https://nginx.org/ru/

[66] Apache Tomcat. Retrieved: May 2020. http://tomcat.apache.org/

[67] Implement RESTful Web Service using Java. Retrieved: May 2020. https://habr.com/ru/post/150034/

[68] TensorFlow - Serving Models. Retrieved: May 2020. https://www.tensorflow.org/tfx/guide/serving

[69] Implement RESTful Web Service using Java. Retrieved: May 2020. https://habr.com/ru/post/150034/

[70] TensorFlow - Serving Models. Retrieved: May 2020. https://www.tensorflow.org/tfx/guide/serving

[71] Relise of TensorFlow Serving as an open source tool for serving machine learning model in production. Retrieved: May 2020. https://ai.googleblog.com/2016/02/running-your- models-in-production-with.html,

[72] ONNX to TensorFlow SavedModel. Retrieved: May 2020. https://github.com/onnx/onnx- tensorflow/issues/490

[73] PyTorch. Retrieved: May 2020. https://pytorch.org

[74] GRPC. Retrieved: May 2020. https://grpc.io/

[75] Remote call procedure, wikipedia. Retrieved: May 2020. https://en.wikipedia.org/wiki/Remote_procedure_call

[76] Kubeflow. Retrieved: May 2020. https://www.kubeflow.org/
[77] Seldon.io. Retrieved: May 2020. https://www.seldon.io/tech/
[78] Hydrosphere.io. Retrieved: May 2020. https://hydrosphere.io/
[79] Kubernates. Retrieved: May 2020. https://kubernetes.io/
[80] Boris Lublinsky. Serving Machine Learning Models. O'Reilly Media, Inc. 2017.
[81] PMML model export - RDD-based API. Retrieved: May https://spark.apache.org/docs/latest/mllib-pmml-model-export.html, Retrieved: 2020.
[82] Looking under the hood of pipelines. Retrieved: May https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/libs/ml/pipelines.html
[83] Release Notes - Flink 1.9. Retrieved: May https://github.com/apache/flink/blob/master/docs/release-notes/flink-1.9.md
[84] LIP-39 Flink ML pipeline and ML libs. Retrieved: May 2020. https://cwiki.apache.org/confluence/display/FLINK/FLIP-39+Flink+ML+pipeline+and+ML+libs
[85] Twiter. Retrieved: May 2020. https://twitter.com/
[86] Ирисы Фишера. Википедия. Retrieved: May 2020. https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%B8%D1%81%D1%8B_%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0
[87] Prometheus.io. Retrieved: May 2020. https://prometheus.io/
[88] Grafana.io. Retrieved: May 2020. https://grafana.com/
[89] Asynchronous I/O Apache Flink. Retrieved: May 2020. https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/asyncio.html
[90] Machine learning model serving system for event streams. Retrieved: May 2020. https://github.com/axreldable/msu-diploma-thesis

Machine learning model serving system for event streams

Aleksei Starikov, Dmitry Namiot

***Abstract* — The paper considers the existing systems of streaming event processing, analyzes the characteristics of these systems, analyzes their advantages and disadvantages. Stream processing significantly reduces the time from the moment data is received to the reaction to it compared to batch processing. Using this approach, companies can respond to incoming information in a timely manner, taking action when they are needed. The paper presents an analysis of the systems used in large companies to develop machine learning models, from data collection to putting the model into operation. The paper discusses the issues of introducing machine learning models into a productive environment for streaming processing applications, including the presentation formats of models, and compares the advantages and disadvantages of various approaches. The architecture and practical are given and the practical implementation of the application is developed, which allows using each of the considered formats.**

***Keywords* — streaming processing, machine learning models serving, machine learning DevOps.**

REFERENCES

[1] Shahrivari S, Jalili S. Beyond batch processing: towards real-time and streaming big data. Computers. 2014; 3(4):117–29.

[2] Thones, J. Microservices. IEEE Software 32, 1, 2015, 116 – 116.

[3] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. PVLDB, 2015.

[4] Iqbal, M.H., Soomro, T.R. Big data analysis: Apache storm perspective. Int. J. Comput. Trends Technol, 2015; 9–14.

[5] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, et al. “Apache Spark: A Unified Engine for Big Data Processing”. In: Commun. ACM 59.11, 2016, pp. 56–65.

[6] Nair, L. R., Shetty, S. D., & Shetty, S. D. Applying spark based machine learning model on streaming big data for health status prediction. Computers & Electrical Engineering, 2017

[7] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja. Lambda architecture for cost-effective batch and speed big data processing. In IEEE Intl Conf. on Big Data, pages 2785– 2792. IEEE, 2015. 14

[8] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, page 28, 2015.

[9] Bejeck, William P, Kafka Streams in action: real-time apps and microservices with the Kafka Streams APIShelter Island, NY: Manning Publications, 2018.

[10] Shree R. et. al., KAFKA: The modern platform for data management and analysis in big data domain, in Proceedings of the 2nd International Conference on Telecommunication and Networks (TEL-NET), 2017.

[11] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell. Samza: Stateful Scalable Stream Processing at LinkedIn. Proc. VLDB Endow., 10(12):1634–1645, Aug. 2017.

[12] Confluence. Retrieved: April 2020 https://www.atlassian.com/ru/software/confluence

[13] LinkedIn. Retrieved: April 2020 https://www.linkedin.com

[14] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas 45 Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, Eric Baldeschwieler: Apache Hadoop YARN: yet another resource negotiator. SoCC 2013:5

[15] Jordan MI, Mitchell TM Machine learning: Trends, perspectives, and prospects, Science 349(6245), 2015, pp. 255–260.

[16] SAS. Retrieved: April 2020 https://www.sas.com

[17] Google. Retrieved: April 2020 https://about.google

[18] Microsoft. Retrieved: April 2020 https://www.microsoft.com

[19] Facebook. Информация о компании. Retrieved: June 2019. https://newsroom.fb.com/company-info

[20] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, “Applied machine learning at facebook: A datacenter infrastructure perspective,” in Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2018.

[21] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo∗ , Peizhao Zhang, Machine Learning at Facebook: Understanding Inference at the Edge, Facebook, Inc., 2019

[22] Introducing FBLearner Flow: Facebook’s AI backbone. Retrieved: June 2019. https://code.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone

[23] Open Source Search & Analytics · Elasticsearch | Elastic. Retrieved: June 2019. https://www.elastic.co

[24] TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines. Retrieved: May 2020. https://www.tensorflow.org/tfx

[25] TensorFlow - An end-to-end open source machine learning platform. Retrieved: May 2020. https://www.tensorflow.org/

[26] Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. Journal of Big Data 2, 1 (2015), 24

[27] Jimmy J. Lin and Alek Kolcz. 2012. Large-scale machine learning at twitter. In SIGMOD. 793–804.

[28] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2016. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. CoRR abs/1610.09451 (2016).

[29] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, JeanFran¸cois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In NIPS. 2503–2511.
[30] Yann Dauphin, Razvan Pascanu, C¸ aglar G¨ul¸cehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. Identifying and attacking the saddle point problem in high- dimensional non-convex optimization. CoRR abs/1406.2572 (2014).
[31] Apache Beam: An Advanced Unified Programming Model. Retrieved: June 2019. https://beam.apache.org
[32] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. IEEE Trans. on Knowl. and Data Eng. 22, 10 (Oct. 2010), 1345–1359.
[33] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In NIPS. 3320–3328.
[34] Running your models in production with TensorFlow Serving. Retrieved: June 2019 https://ai.googleblog.com/2016/02/running-your-models-in-production-with.html
[35] Jeremy Hermann and Mike Del Balso. Meet Michelangelo: Uber's machine learning platform. https://eng.uber.com/michelangelo, 2017. [Online; accessed 14-April-2019].
[36] Jupyter Notebook. Retrieved: June 2019 https://jupyter.org
[37] Databricks Inc. Retrieved: June 2019 https://databricks.com
[38] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with MLflow. IEEE Data Engineering Bulletin, 41(4), 2018.
[39] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jurgen Schmidhuber. The Sacred Infrastructure for Computational Research. In Katy Huff, David Lippa, Dillon Niederhut, and M. Pacer, editors, Proceedings of the 16th Python in Science Conference, pages 49 – 56, 2017.
[40] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. In Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA '16, pages 14:1–14:3, New York, NY, USA, 2016. ACM.
[41] Google. Tensorboard: Visualizing learning. Retrieved: June 2019 https://www.tensorflow.org/guide/summaries_and_tensorboard.
[42] Git. Retrieved: June 2019. https://git-scm.com 47
[43] GitHub. Retrieved: June 2019. https://github.com
[44] Sklearn. Retrieved: June 2019. https://scikit-learn.org/stable
[45] Zhang YY, Jiao YQ. Design and Implementation of Predictive Model Markup Language Interpretation Engine. 2015 International Conference on Network and Information Systems for Computers (ICNISC). 2015:527–31. 10.1109/Icnisc., 2015.105.
[46] Data Mining Group. Retrieved: June 2019. http://dmg.org
[47] Pmml examples. Retrieved: May 2020. http://dmg.org/pmml/pmml_examples/index.html,
[48] Library for serialization and deserialization PMML models. Retrieved: May 2020. https://github.com/nyoka-pmml/nyoka
[49] Spark.mlib library. Retrieved: May 2020. https://spark.apache.org/mllib/
[50] Flink-jpmml library. Retrieved: May 2020. https://github.com/FlinkML/flink-jpmml
[51] Java PMML API. Retrieved: May 2020. https://github.com/jpmml
[52] Tensorflow, saved model. Retrieved: June 2019. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/python/saved_model
[53] Protobuf. Retrieved: June 2019. https://developers.google.com/protocol-buffers/?hl=ru
[54] ONNX. Retrieved: June 2019. https://onnx.ai
[55] ONNX libraries. Retrieved: May 2020. https://github.com/onnx/tutorials
[56] ONNX Model Zoo. Retrieved: May 2020. https://github.com/onnx/models
[57] MLeap. http://mleap-docs.combust.ml, Retrieved: June 2019.
[58] J. Pivarski, C. Bennett and RL. Grossman, "Deploying Analytics with the Portable Format for Analytics (PFA)", Proceedings of the International Conference of Knowledge Discovery and Data Mining, (2016)
[59] PFA description. Retrieved: May 2020. http://dmg.org/pfa/docs/motivation/
[60] Plase D., Niedrite L., Taranovs R. Comparison of HDFS compact data formats: Avro Versus Parquet // Mokslas-Lietuvos ateitis. 2017. No. 9. P. 267-276.
[61] Titus 2 - Portable Format for Analytics (PFA) implementation for Python 3.5+. Retrieved: May 2020. https://github.com/animator/titus2
[62] How to Easily Deploy Machine Learning Models Using Flask. Retrieved: May 2020. https://towardsdatascience.com/how-to-easily-deploy-machine-learning-models-using-flask-b95af8fe34d4
[63] Python serialization format – pickle. Retrieved: May 2020. https://docs.python.org/3/library/pickle.html
[64] Docker. Retrieved: May 2020. https://www.docker.com/
[65] Nginx. Retrieved: May 2020. https://nginx.org/ru/
[66] Apache Tomcat. Retrieved: May 2020. http://tomcat.apache.org/
[67] Implement RESTful Web Service using Java. Retrieved: May 2020. https://habr.com/ru/post/150034/
[68] TensorFlow - Serving Models. Retrieved: May 2020. https://www.tensorflow.org/tfx/guide/serving
[69] Implement RESTful Web Service using Java. Retrieved: May 2020. https://habr.com/ru/post/150034/
[70] TensorFlow - Serving Models. Retrieved: May 2020. https://www.tensorflow.org/tfx/guide/serving
[71] Relise of TensorFlow Serving as an open source tool for serving machine learning model in production. Retrieved: May 2020. https://ai.googleblog.com/2016/02/running-your- models-in-production-with.html,
[72] ONNX to TensorFlow SavedModel. Retrieved: May 2020. https://github.com/onnx/onnx- tensorflow/issues/490
[73] PyTorch. Retrieved: May 2020. https://pytorch.org
[74] GRPC. Retrieved: May 2020. https://grpc.io/
[75] Remote call procedure, wikipedia. Retrieved: May 2020. https://en.wikipedia.org/wiki/Remote_procedure_call
[76] Kubeflow. Retrieved: May 2020. https://www.kubeflow.org/
[77] Seldon.io. Retrieved: May 2020. https://www.seldon.io/tech/
[78] Hydrosphere.io. Retrieved: May 2020. https://hydrosphere.io/
[79] Kubernates. Retrieved: May 2020. https://kubernetes.io/
[80] Boris Lublinsky. Serving Machine Learning Models. O'Reilly Media, Inc. 2017.
[81] PMML model export - RDD-based API. Retrieved: May https://spark.apache.org/docs/latest/mllib-pmml-model-export.html, Retrieved: 2020.
[82] Looking under the hood of pipelines. Retrieved: May https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/libs/ml/pipelines.html
[83] Release Notes - Flink 1.9. Retrieved: May https://github.com/apache/flink/blob/master/docs/release-notes/flink-1.9.md
[84] LIP-39 Flink ML pipeline and ML libs. Retrieved: May 2020. https://cwiki.apache.org/confluence/display/FLINK/FLIP-39+Flink+ML+pipeline+and+ML+libs
[85] Twiter. Retrieved: May 2020. https://twitter.com/
[86] Ирисы Фишера. Википедия. Retrieved: May 2020. https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%B8%D1%81%D1%8B_%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0
[87] Prometheus.io. Retrieved: May 2020. https://prometheus.io/
[88] Grafana.io. Retrieved: May 2020. https://grafana.com/

[89] Asynchronous I/O Apache Flink. Retrieved: May 2020. https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/asyncio.html

[90] Machine learning model serving system for event streams. Retrieved: May 2020. https://github.com/axreldable/msu-diploma-thesis