DOI 10.37882/2223–2966.2022.06–2.26

МЕТОДИКА МОДУЛЬНО-КОНВЕЙЕРНОЙ ОБРАБОТКИ ДАННЫХ НА ОСНОВЕ SPARK SQL И SPARK MLLIB С ИНТЕГРАЦИЕЙ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

THE METHODOLOGY OF MODULAR PIPELINE DATA PROCESSING BASED ON SPARK SQL AND SPARK MLLIB WITH THE INTEGRATION OF PROGRAMMING LANGUAGES

V. Monastyrev
S. Molodyakov

Summary. A methodology for constructing a data processing architecture based on Spark SQL and Spark MLlib with the possibility of integrating various programming languages is proposed. Thanks to the use of such an architecture, it is possible to modularly build the data processing process, where each step is a separate and independent part that can be added or removed from the processing process. An example of conveyor-modular processing is presented. A processing pipeline is organized using Spark MLlib. Spark SQL is used to organize queries and to process data. The structure of its processing classes is considered in Scala using the Transform and Estimator base classes of the Spark MLlib library. An example of a processing pipeline is given, which begins with data preparation and ends with training a machine learning model. In the Python language, an example of the implementation of the code of the model to which the conversion takes place directly from the pipeline is presented. The possibility of implementing data processing in one language and model training in another is shown.

Keywords: big data, machine learning, Spark, pipeline, Spark SQL, Spark MLlib.

Монастырев Виталий Викторович
Аспирант, Санкт-Петербургский политехнический университет Петра Великого
vit34–95@mail.ru
Молодяков Сергей Александрович
Д.т.н., профессор, Санкт-Петербургский политехнический университет Петра Великого
molodyakov_sa@spbstu.ru

Аннотация. Предлагается методика построения архитектуры обработки данных на основе Spark SQL и Spark MLlib с возможностью интеграции различных языков программирования. Благодаря использованию такой архитектуры можно модульно выстраивать процесс обработки данных, где каждый шаг является отдельной и независимой частью, которую можно добавлять или убирать из процесса обработки. Представлен пример конвейерно-модульной обработки. С использованием Spark MLlib организован конвейер обработки. Spark SQL применен для организации запросов и для обработки данных. Построение собственных классов обработки рассмотрено на языке Scala при помощи базовых классов Transform и Estimator библиотеки Spark MLlib. Приведен пример конвейера обработки, который начинается с подготовки данных и заканчивается обучением модели машинного обучения. На языке Python представлен пример реализации кода модели, к которой происходит обращение напрямую из конвейера. Показана возможность реализации обработки данных на одном языке, а обучение моделей на другом.

Ключевые слова: большие данные, машинное обучение, Spark, конвейер, Spark SQL, Spark MLlib.

Введение

В настоящее время появляется все больше областей, в которых используется обработка больших объемов информации, в частности, можно выделить банковские системы, системы анализа интересов пользователей, системы поиска и распознавания изображений и т.д. [1, 2]. Особенностью архитектуры таких систем является то, что требуется модифицировать отдельные элементы процесса обработки без изменения других. Основной архитектурой таких систем является модульная архитектура. В случае модульной архитектуры каждый этап обработки данных представляет собой отдельную и независимую часть, которая может быть добавлена или удалена из процесса обработки. Для повышения производительности можно организовать конвейер продвижения данных по модулям обработки (рис. 1). Чем больше модулей в конвейере, тем выше скорость обработки. Более того, самый медленный модуль определяет максимальную частоту продвижения данных.

В настоящее время при работе с большими данными Apache Spark является одним из основных фреймворков. Spark предоставляет разработчикам набор библиотек [3, 4]: Spark SQL — библиотека, позволяющая работать с данными с помощью запросов и операций, близких к языку SQL; Spark Machine Learning Library (MLlib) — набор классов и методов, содержащих инструменты для реализации моделей машинного обучения на больших объемах

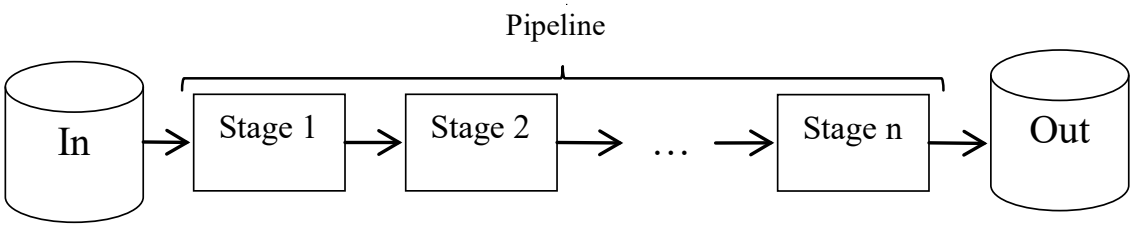

Рис. 1. Схема модульного конвейера обработки данных

данных; Spark GraphX — библиотека, которая позволяет работать с графами, распределенными поверх RDD (Resilient Distributed Dataset); Spark Streaming — это библиотека для работы с распределенным потоковым набором данных (данные из Kafka и т.д.) [5].

При работе с Spark возникают две основные трудности. Во-первых, необходимо создать модульные архитектуры для построения процесса обработки данных. Во-вторых, необходимо интегрировать модели машинного обучения в общую архитектуру. Во втором случае решается вопрос о том, как интегрировать несколько языков программирования. Spark предоставляет свой API для языков Java, Scala, Python и R, но в большинстве случаев Scala используется для обработки данных, поскольку это родной язык для Spark. Spark предоставляет собственную библиотеку машинного обучения Spark MLlib [4], однако она отстает от своих аналогов по количеству реализованных алгоритмов и функциональности, поэтому используется не так часто. Большинство наиболее популярных библиотек машинного обучения реализованы на Python (например, PyTorch [6], TensorFlow и многие другие [3]). В результате наиболее релевантной комбинацией языков является комбинация, когда код для подготовки данных реализован на Scala, а модель машинного обучения реализована на Python [7].

Существующие подходы к построению процессов обработки больших объемов информации

Рассмотрим существующие альтернативные подходы к разработке процессов обработки больших данных, которые могут быть реализованы, в том числе с использованием Apache Spark. Как правило, большинство подходов сводится к созданию собственной архитектуры процесса обработки данных. Есть два основных варианта того, как это может быть реализовано. Во-первых, непрерывный процесс обработки данных с использованием одного скрипта. В этом случае реализуется общий алгоритм обработки данных для приведения данных к требуемому виду. Алгоритм может включать в себя различные операции агрегирования, группировки, фильтрации и т.д. В то же время необходимо понимать, какие данные перетасовываются между узлами кластера [4, 8]. Вызов такого скрипта может происходить вручную, когда разработчик при необходимости сам запускает скрипт, или запуск может происходить автоматически с помощью инструментов CI/CD (например, Jenkins, TeamCity и т.д.). Недостатки такого подхода очевидны — все преобразования данных находятся в одном месте, и логически сложно понять, какая часть кода за что отвечает. Еще одним важным моментом является сложность редактирования и дальнейшей поддержки. Во-вторых, обработка данных разделена на отдельные процессы или модули. Создается архитектура, которая позволяет вам выполнять эти этапы в правильном порядке. В этом случае при обработке данных могут быть использованы различные инструменты оркестровки — например, Apache Airflow. Однако построение этой архитектуры — это довольно сложный и длительный процесс. Ошибки проектирования могут повлиять на дальнейшую работу, но модульность позволяет быстрее вносить коррективы во время дальнейшей разработки.

Другими вариантами интеграции языков и моделей машинного обучения данных являются. Во-первых, обработка данных и модель машинного обучения реализованы на одном и том же языке. В этом случае обучение модели и запуск прогнозирования обычно полностью интегрированы с процессом обработки данных. Это может быть представлено в виде одного скрипта или в виде модулей с использованием оркестраторов. Если говорить о едином языке для обработки данных и обучения моделей, то обычно таким языком является Python, т.е. используется PySpark [9]. Среди недостатков такого подхода стоит отметить, что решения на PySpark для реализации процессов обработки данных работают медленнее, чем решения на Scala [10]. Во-вторых, обработка данных и модель машинного обучения реализованы на разных языках. В этом случае вам необходимо организовать вызов процесса обучения модели после процесса обработки данных. Для этой цели, опять же, можно использовать оркестратор или инструменты CI/CD. Из недостатков такого подхода стоит отметить, что необходимо четко понимать степень готовности данных, был ли пропущен какой-либо шаг обработки и так далее.

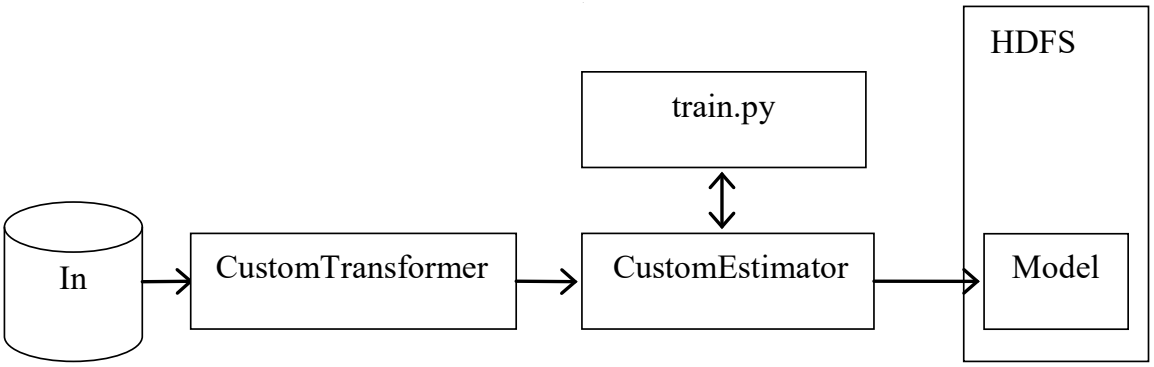

Рис. 2. Конвейер для обучения модели

Таким образом, мы сформируем окончательное представление о том, какой подход был бы желателен в процессе обработки большого объема информации и обучения моделей:

- Процесс обработки разделен на независимые модули;
- Архитектура процесса обработки проста и прозрачна;
- Процесс обработки и обучения моделей может быть реализован на различных языках программирования;
- Работа моделей может быть довольно просто интегрирована в общий процесс обработки без использования сторонних фреймворков.

В статье рассмотрен подход, в котором реализована представленная схема (рис. 1). Конвейер обработки будет построен с использованием базовых классов из библиотеки MLlib. Spark SQL используется для организации запросов и обработки данных. Рассматривается построение пользовательских классов обработки на Scala с использованием базовых классов Transform и Estimator библиотеки Spark MLlib. Благодаря использованию этих классов можно будет использовать такие классы, как Pipeline, которые делают довольно простым и модульным компиляцию процесса подготовки данных и работы моделей машинного обучения.

Пример реализации модульной конвейера обработки с интеграцией различных языков программирования

Рассмотрим пример реализации модульного конвейерного метода обработки данных с использованием Spark SQL и Spark MLlib. Spark SQL можно использовать для обработки структурированных данных. Он основан на логике запросов, основанной на языке SQL, так что пользователь, не имеющий опыта разработки, но имеющий опыт в аналитике, также может заниматься обработкой данных. Spark MLlib позволяет реализовывать конвейер обработки и работать с использованием класса Pipeline. Pipeline позволяет осуществлять обработку данных поэтапно. Если происходит только преобразование данных, то Spark MLlib использует класс Transformer для этой цели. Если модель обучается, то используется базовый класс Estimator. Все этапы обработки передаются в Pipeline в методе setStages.

Реализация проприетарных классов обработки данных на основе Transformer и Estimator будет рассмотрена на языке Scala, однако аналогичным образом возможно реализовать кодовую базу на других языках, обеспечивающих возможность работы со Spark через API: Scala, Java, Python и R.

Упрощенный процесс обработки данных показан на рис. 2. Представлен случай, когда в процессе обработки участвуют один CustomTransformer и один CustomEstimator. Для работы с моделью используется программа на Python (train.py), а файлы распределяются между узлами вычислительного кластера, например, с использованием распределенной файловой системы Hadoop (HDFS).

Последовательно рассмотрим структуру реализованного класса для преобразования данных на основе базового класса Spark MLlib Transformer.

```
class CustomTransformer(override val uid: String)
extends Transformer
    with DefaultParamsWritable
    {def this() = this(Identifiable.randomUID("org.apache.
spark.ml.feature. CustomTransformer"))
    override def transform(dataset: Dataset[_]): DataFrame
= {???}
    override def copy(extra: ParamMap): Transformer =
defaultCopy(extra)
    override def transformSchema(schema: StructType):
StructType = {???}
    }
```

В приведенном выше примере предполагается, что реализованный класс называется CustomTransformer. Этот класс наследуется от двух классов: Transformer и DefaultParamsWritable.

- Transformer — это абстрактный класс библиотеки Spark MLlib, который позволяет реализовывать преобразования данных и интегрировать их в класс Pipeline в качестве этапа обработки.
- DefaultParamsWritable — особенность библиотеки Spark MLlib, которая помогает сделать реализуемый класс доступным для записи в файловое хранилище.

Каждый Transformer в рамках Spark MLlib должен иметь свой собственный уникальный идентификатор — параметр uid. Он генерируется с помощью метода Identified.randomUID.

Метод tranform принимает набор данных в качестве аргумента. Это непосредственно набор данных, переданный в Transformer. Он может быть получен как исходный набор данных или как выходной результат предыдущего этапа обработки в конвейере. Внутри метода преобразования пользователь реализует все необходимые преобразования для требуемых данных. В этом случае предлагается использовать модуль Spark SQL. Например, в качестве шага обработки вам нужно добавить новый столбец, который является суммой двух уже существующих, тогда метод преобразования будет выглядеть следующим образом:

```
override def transform(dataset: Dataset[_]): DataFrame
= {
dataset.withColumn("newFeature", col("col1") +
col("col2"))

}
```

- Метод copy позволяет создать копию экземпляра с тем же UID и некоторыми дополнительными параметрами.
- Метод transformSchema содержит схему для набора данных, которая будет получена в конце преобразования. Это необходимо, потому что Spark работает с отложенными вычислениями и перед началом вычисления конвейера проверяет, что все этапы обработки могут быть выполнены, т.е. содержатся все необходимые столбцы, с которыми они работают. Например, шаг 1 выводит набор данных со столбцами col1 и col2, а шаг 2 в методе преобразования обращается к столбцу col3, который не существует. Тогда Spark выдаст соответствующую ошибку, и вычисления не запустятся.

В дополнение к наборам методов, описанных выше, transformer также может содержать наборы параметров. Если они необходимы, то для них реализуются соответствующие методы get и set.

Наконец, чтобы иметь возможность читать, вам необходимо реализовать пользовательский объект Transformer:

```
object CustomTransformer extends DefaultParamsRead
able[CustomTransformer] {
override def load(path: String): CustomTransformer =
super.load(path)
}
```

Этот объект наследуется от признака DefaultParamsReadable и реализует метод load. Этот метод позволяет вам считывать Transformer в сохраненном Pipeline.

Теперь необходимо рассмотреть структуру реализованного класса, чтобы иметь возможность обучать данные на основе базового класса Spark MLlib Estimator.

```
class CustomEstimator(override val uid: String) extends
Estimator[CustomEstimatorModel] with
DefaultParamsWritable
{def this() = this(Identifiable.
randomUID("CustomEstimator"))
override def fit(dataset: Dataset[_]):
CustomEstimatorModel = {???}
override def copy(extra: ParamMap): CustomEstimator =
defaultCopy(extra)
override def transformSchema(schema: StructType):
StructType = {???}}
```

В этом случае предполагается, что имя реализованного класса — CustomEstimator. В этом случае Estimator должен наследоваться от Estimator[CustomEstimatorModel] и DefaultParamsWritablearamsWritable. Класс Estimator, как и класс Transformer, имеет уникальный идентификатор uid, который генерируется с использованием метода Identified.randomUID. По аналогии с Transformer, методы копирования также реализованы для создания экземпляра объекта и transformSchema для указания выходной схемы набора данных.

Использование Python для реализации алгоритма обработки

При реализации алгоритмов обработки существует несколько возможных вариантов:

- Алгоритм машинного обучения реализован на том же языке, что и код Estimator. Например, если код был реализован на Pyspark, то различные библиотеки Python можно было бы исполь-

зовать прямо внутри оценщика для обучения моделей.

- Алгоритм машинного обучения реализован на другом языке. В этом случае вы можете интегрировать обучение модели на другом языке программирования следующим образом: преобразуйте набор данных, подготовленный для обучения, в RDD, а затем примените метод rdd.pipe(filename), который запустит указанный файл, где указанный RDD будет использоваться в качестве аргумента командной строки. То есть, таким образом, вы можете реализовать код преобразования в Scala и выполнить обучение модели на Python.

Если мы рассмотрим второй вариант, то файл Python может выглядеть следующим образом:

```
rows = [] #here we keep input data to Dataframe constructor
for line in sys.stdin:
# parsing the input dataset and writing to rows
#initialization of pandas dataframe
df = pd.DataFrame(rows)
# initialization of the property and result columns
feature_columns = [“feature1”, “feature2”]
label_column = “label”
# model training, we can use any machine learning algorithm we need
model = MachineLearningAlgorithm ()
model.fit(df[feature_columns], df[label_column])
model_string = base64.b64encode(pickle.dumps(model)).decode(‘utf-8’)
# returning the result
print(model_string)
```

В приведенном выше примере входной RDD анализируется, модель обучается, и результат (обученная модель) возвращается в строковом формате. Это строка, которую вернет метод rdd.pipe.

По аналогии с Transformer, Estimator также должен реализовать объект CustomEstimator, чтобы иметь возможность считывать данные с файловой системы:

```
object CustomEstimator extends DefaultParamsReadable[CustomEstimator] {
override def load(path: String): CustomEstimator = super.load(path)
}
```

Заключение

В этой статье представлена методология модульно-конвейерной обработки данных с интеграцией различных языков программирования. В качестве примера реализации был рассмотрен алгоритм построения процесса обработки данных и обучения модели. Полученный алгоритм позволяет разделить процесс обработки на независимые модули. При необходимости их можно легко редактировать, добавлять и удалять из конвейера.

Благодаря возможностям RDD, передаваемым в качестве входных строковых аргументов, разработчик может реализовать обработку данных на одном языке и обучение модели на другом. Это особенно полезно, когда команды разделены на команды Data Engineers и Data Scientists, т.е. инженеры по обработке данных могут реализовать обработку данных на удобном для них языке с помощью Spark, а команда по обучению данным может использовать стандартные и знакомые библиотеки Python.

Кроме того, благодаря построению процессов с использованием Spark MLlib вам не нужно реализовывать какие-либо дополнительные классы обработчиков, использовать посторонние фреймворки и т.д., поскольку все, что вам нужно, уже содержится в стандартной библиотеке.

ЛИТЕРАТУРА

1. Monastyrev V.V., Drobintsev P.D. Recommendation system based on user actions in the social network // Proceedings of the Institute for System Programming of the RAS. — 2020. — V. 32. — N3. — P. 101–108 doi: 10.15514/ISPRAS-2020–32(3)-9.
2. Voinov N.V., Voroshilov M.K., Molodyakov S.A., Drobintsev P.D., Prokofiev O.V. Zajtsev I.V. Predicting RTS Index Futures Using Machine Learning // Proceedings 24th International Conference on Soft Computing and Measurements. — 2021. — P. 193–196 doi: 10.1109/SCM52931.2021.9507184.
3. Tang S., He B., Yu C., Li Y., Li K. A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications // IEEE Transactions on Knowledge and Data Engineering. — 2022. — V. 34. — N1. — P. 71–91 doi: 10.1109/TKDE.2020.2975652.
4. Luu H. Beginning Apache Spark 2: with resilient distributed datasets, Spark SQL, structured streaming and Spark Machine Learning library. NY: Apress, 2018.
5. Nabi Z. Pro Spark streaming: the zen of real-time analytics using Apache Spark, Berkeley. NY: Apress, 2016.
6. Ketkar N., Jojo M. Deep learning with Python: learn best practices of deep learning models with PyTorch. Berkeley, CA: Apress, 2021.
7. Nguyen G., Dlugolinsky S., Bobák M., et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey // Artif Intell Rev. — 2019. — V. 52. — P. 77–124 doi: 10.1007/s10462–018–09679-z
8. Popov M., Drobintsev P.D. Data Shuffling Minimizing Approach for Apache Spark Programs // Lecture Notes in Networks and Systems. — 2020. — V. 95. — P. 131–139.

9. Rao T.R., Mitra P., Bhatt R., Goswami A. The big data system, components, tools, and technologies: a survey // Knowledge and Information Systems.— 2018.-V. 60.— N3.-P. 1165–1245 doi: 10.1007/s10115–018–1248–0
10. García S., Ramírez-Gallego S., Lueng J., Benítez J.M., Herrera F. Big data preprocessing: methods and prospects // Big Data Analytics.— 2016.— N. 1, (9) doi: 10.1186/s41044–016–0014–0

Журнал «Современная наука: актуальные проблемы теории и практики»

Санкт-Петербургский политехнический университет Петра Великого