УДК 004.042

МЕТОД РАСПРЕДЕЛЕННОГО ОБНАРУЖЕНИЯ ИЗМЕНЕНИЯ КОНЦЕПЦИИ

А.А. Волков, Л. Бюх, А. Андреяк

Представлен метод распределенного обнаружения изменения концепции для алгоритмов интеллектуального анализа данных. Под изменением концепции понимается любое непредсказуемое изменение входных данных алгоритма. Предложена реализация метода с использованием технологии распределенных вычислений MapReduce. Разработанный алгоритм предназначен для обнаружения изменения концепции в потоке входных данных в режиме реального времени. С целью обеспечения итеративного поведения фаз Map и Reduce разработан специальный MapReduce-фреймворк и осуществлена его программная реализация. Использование алгоритма позволит автоматически обнаруживать изменение входных данных, требующее изменение параметров используемой модели и переключение на использование новой модели в режиме реального времени.

Ключевые слова: изменение концепции, интеллектуальный анализ данных, распределенные вычисления, итеративный MapReduce.

Введение

Благодаря современному развитию технологий мы можем накапливать огромные объемы данных в любых областях науки. Для анализа этих данных используются различные модели и алгоритмы. Однако реальные данные редко имеют постоянное распределение, и, следовательно, не могут описываться одной моделью на протяжении длительного периода времени. В связи с этим возникает необходимость каким-либо образом обнаруживать скачкообразные изменения во входных данных с целью своевременного изменения параметров модели.

В результате изменения входных данных использование текущей модели приводит к неверным результатам и становится неэффективным, что приводит к необходимости изменить параметры используемой модели. Такое изменение входных данных называют изменением концепции (concept drift) [1, 2, 5, 6]. При этом под концепцией понимается любая сущность реального мира, которая подвергается интеллектуальному анализу.

Возможность автоматического обнаружения изменения концепции во время анализа данных позволит существенно ускорить процесс получения результата и значительно снизит ошибки анализа данных.

В работах [2, 5, 6] предложены последовательные алгоритмы обнаружения изменения концепции, авторы статьи [1] предлагают распределенный и параллельный подход. Основным средством для отслеживания возможного изменения концепции, авторы перечисленных работ используют изменение значения ошибки. Способ получения значений ошибки зависит от конкретной области интеллектуального анализа данных.

Во всех рассмотренных подходах для получения новой модели используются элементы входного потока данных, начиная с элемента, при котором впервые было зафиксировано состояние возможного изменения концепции (warning level). Переход на новую модель осуществляется в точке изменения концепции, когда достигнут заданный уро-

вень ошибки (drift level). Отличием [1] является то, что настройка новой модели осуществляется параллельно с использованием основной модели.

В данной работе предлагается усовершенствованный вариант алгоритма распределенного обнаружения изменения концепции, описанного в [1]. В качестве инструмента, позволяющего реализовать параллельную, распределенную и устойчивую к сбоям обработку больших массивов данных, использована парадигма программирования MapReduce с использованием расширений, предложенных в [3], обеспечивающих возможность обработки потоковых данных в инкрементном режиме.

Предложенный алгоритм относится к классу итеративных MapReduce алгоритмов. Для учета специфики алгоритма была разработана специальная программная структура (фреймворк).

В разделе 1 описана реализация предложенного усовершенствованного алгоритма распределенного обнаружения изменения концепции. В разделе 2 представлено описание MapReduce-фреймворка, обеспечивающего исполнение разработанного алгоритма. В заключении изложены возможные пути дальнейшего развития предложенного алгоритма.

1. Алгоритм распределенного обнаружения изменения концепции

1.1. Причины необходимости доработки предыдущей реализации

В статье [1] подробно описывается алгоритм распределенного обнаружения изменения концепции, реализованный с использованием парадигмы программирования MapReduce. Однако практические испытания данного алгоритма указали на ряд неучтенных факторов, снижающие его эффективность.

Во-первых, Map-элементы получают сообщения об изменении концепции (warning или drift) со значительным запозданием. То есть Map-элемент может обработать и отправить Reduce-элементу сотни значений после того момента времени, когда произошло изменение концепции. Чтобы не потерять точность вычислений, Map-элементу необходимо повторно обработать элементы входного потока, начиная с элемента, соответствующего последнему сообщению об изменении концепции.

Данная проблема в [1] решалась путем синхронизации Map и Reduce элементов, однако синхронизация, как известно, приводит к снижению эффективности распределенных алгоритмов.

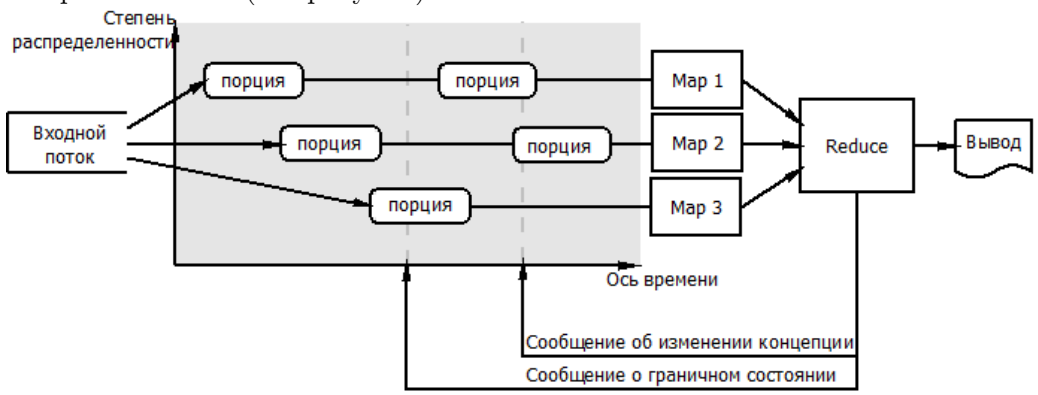
Во-вторых, в случае если Map-элемент начинает повторно обрабатывать элементы входного потока, но с использованием новой модели, Reduce-элемент должен иметь возможность фильтровать значения, полученные с использованием разных моделей, и выбирать только актуальные результаты.

Кроме того, для повышения наглядности и прозрачности алгоритма, был введен ряд абстрактных структур, которые более подробно рассмотрены далее.

1.2. Общая схема алгоритма

Входящий поток данных делится на порции и обрабатывается Map-элементами. Пары <ключ, значение>, произведенные всеми Map-элементами, группируются по ключу и направляются в виде пары <ключ, массив значений> в адрес Reduce-элементов. Reduce-элементы, независимо друг от друга, обрабатывают эти пары и выдают окончательный результат. Вместе с тем, Reduce-элементы могут посылать Map-элементам со-

общения о нормальном, граничном состоянии, или состоянии изменения концепции посредством обратной связи (см. рисунок).



Общая схема алгоритма обнаружения изменения концепции

Таким образом, процесс обучения модели включает следующие стадии:

1. Разделение входного потока на «порции», которые раздаются Map-элементам по очереди. Размер порции является одним из параметров алгоритма. Он не должен быть маленьким, чтобы снизить расходы на выделение и освобождение памяти, но и не должен быть большим, чтобы издержки на передачу данных отдельным рабочим узлам не превышали время обработки этих данных рабочим узлом.
2. Каждый Map-элемент строит свою собственную модель, основываясь только на своих входных данных.
3. Reduce-элемент обобщает результаты, поступившие от всех Map-элементов, и посылает одновременно всем Map-элементам одно из трех сообщений о состоянии текущей модели (нормальное, граничное, изменение концепции). Механизм обратной связи предоставляется фреймворком.

Важной особенностью данного алгоритма является то, что обработка Map-элементом входных данных осуществляется независимо от работы Reduce-элемента, которая включает в себя принятие решения об изменении концепции. То есть, в отличие от распространенного подхода, при котором каждая итерация алгоритма представляет собой последовательность фаз Map и Reduce, в предложенном алгоритме итерации происходят в Map и Reduce фазах параллельно.

Решение об изменении концепции Reduce-элемент может принять, лишь обработав последовательность исторических данных. То есть Map-элемент получит сообщение о том, что изменение концепции имело место в определенный момент в прошлом. В связи с этим, для получения корректного результата, Map-элементу необходимо обработать данные повторно, начиная с того момента, когда изменение концепции имело место.

Поскольку может возникнуть ситуация, что Reduce-элемент получит результаты обработки одного и того же входного значения, но с использованием разных моделей, независимо друг от друга Reduce и Map элементы хранят номер текущей, актуальной модели. После рассылки сообщения об изменении концепции Reduce-элемент игнорирует входные значения, обработанные предыдущими моделями.

1.3. Map-элемент

Каждый Map-элемент имеет кэш, в который он записывает входные данные для того, чтобы обеспечить возможность их повторной обработки.

Для получения сообщений от Reduce-элемента о нормальном, граничном состоянии или состоянии изменения концепции Map-элемент имеет почтовый ящик. Проверка сообщений и обработка входных данных осуществляются параллельно.

Если Map-элемент получает сообщение о граничном состоянии после сообщения о нормальном состоянии, он продолжает использовать текущую модель, но должен начать обучение новой модели в фоновом режиме, то есть одновременно с использованием текущей модели. Для обучения он использует кэшированные значения входного потока, начиная с позиции, в которой Reduce-элементом зафиксировано граничное состояние. При этом кэш старше последнего сообщения о граничном состоянии может быть очищен. В случае получения сообщения о нормальном состоянии после сообщения о граничном состоянии, модель, обучаемая в фоновом режиме, сбрасывается.

При поступлении сообщения об изменении концепции, Map-элемент сохраняет действующую модель и приступает к использованию модели, обучаемой в фоновом режиме с позиции, в которой Reduce-элемент зафиксировал изменение концепции. При этом порядковый номер действующего классификатора (модели) увеличивается на единицу.

Выходные значения Map-элемента имеют обязательную пометку с номером модели, используемой для его вычисления, а также привязку ко временной оси.

Позиции нужны для того, чтобы Reduce-элементом обрабатывал данные, поступающих от всех Map-элементов, в хронологическом порядке. В противном случае результаты обнаружения изменения концепции будут некорректными.

Номер классификатора используется между Map- и Reduce-элементами как средство идентификации того, что они используют наиболее актуальную модель. Поскольку сообщения от Map- к Reduce-элементам поступают асинхронно, часто возникает ситуация, когда часть Map-элементов системы продолжают отправлять значения, полученные устаревшей моделью. Такие сообщения должны быть проигнорированы Reduce-элементами.

1.4. Reduce-элемент

Поскольку обнаружение явления изменения концепции зависит от момента времени, необходимо обеспечить, чтобы Reduce-элемент обрабатывал поступившие к нему от Map-элементов значения не в порядке их поступления, а в соответствии с их привязкой к временной оси.

Для этого Reduce-элемент использует специальную структуру – скользящее окно. Все значения, поступающие от Map-элементов, имеют привязку ко времени. В соответствии с этой привязкой, входные значения Reduce-элемента помещаются в скользящее окно. Поскольку значения от Map-элементов поступают асинхронно, скользящее окно может содержать пустые места (пропуски). Обработка полученных значений может производиться только над непрерывной временной последовательностью значений с начала (левой границы) скользящего окна. Обработанные значения удаляются, начало скользящего окна смещается до временной метки последнего обработанного значения.

Для исключения возможности попадания в скользящее окно неактуальных значений, полученных с помощью устаревших моделей, Reduce-элемент производит проверку всех поступающих значений. Для этого все входные данные имеют пометку Map-элемента с номером модели, используемой для получения данного результата. Reduce-элемент ведет собственный учет текущей модели, и в случае несовпадения этих номеров – игнорирует входное значение.

В случае обнаружения Reduce-элементом состояния изменения концепции происходит полная очистка скользящего окна.

Reduce-элемент, по результатам обработки последовательности значений из скользящего окна, отправляет Map-элементам сообщение о нормальном, граничном состоянии или состоянии изменения концепции посредством механизма обратной связи. Данное сообщение рассылается фремворком в ящики входящих сообщений всех Map-элементов.

В случае завершения входного потока, либо достижения Reduce-элементом условия останова, Reduce-элемент генерирует сообщение об останове работы программы и рассылает его Map-элементам.

2. Фреймворк для итеративного MapReduce

Большинство программных структур для алгоритмов MapReduce не предназначено для реализации итеративных алгоритмов [4]. Проблемы, в первую очередь, связаны с необходимостью чтения и записи данных в начале и окончании каждой итерации соответственно.

Предложенный алгоритм имеет иной принцип организации итеративного поведения: итерации осуществляются внутри Map- и Reduce-элементов независимо друг от друга. Для обеспечения такого поведения был разработан специальный MapReduce-фреймворк.

Разработанный фреймворк обеспечивает Hadoop-совместимый интерфейс программирования приложений (API), а также предоставляет дополнительные функциональные возможности для реализации итеративного поведения MapReduce алгоритмов.

Основными особенностями разработанного фреймворка являются:

1. Возможность обратной связи между Reduce-элементами и Map-элементами;
2. Возможность получения предварительного результата;
3. Реализация итеративного поведения алгоритмов MapReduce;
4. Инкрементная обработка больших объемов данных [3].

Рассмотрим данные особенности более подробно.

2.1. Наличие обратной связи

Предлагаемый фреймворк предусматривает два типа обратной связи:

– Broadcast;

– Restream.

Обратная связь Broadcast – это сигнал, поступающий от Reduce-элемента всем Map-элементам по типу рассылки электронной почты. У каждого Map-элемента есть свой «ящик», который он периодически проверяет на наличие новых сообщений. Данный тип обратной связи предназначен для посылки различных служебных сообщений: изменение глобальных параметров, необходимость очистки памяти (кэша) входных данных, либо преждевременный останов программы.

При обратной связи Restream пересылаемый сигнал представляет собой обычную пару <ключ, значение>, которая попадает во входной поток и обрабатывается как обычный вход. Данную пару получит только один Map-элемент, причем конкретный номер Map-элемента, который будет ее обрабатывать, зависит от диспетчера Map-задач. Данный тип обратной связи применяется в случае, когда нужно повторно обработать какие-то входные данные (если предварительный результат не удовлетворяет определенным условиям).

2.2. Использование предварительных результатов

Разработанный фреймворк позволяет получить предварительные результаты вычислений [3]. Использование предварительных результатов позволяет оценить качество модели без необходимости обрабатывать все входные данные полностью. Если качество предварительного результата не удовлетворяет заданным требованиям, фреймворк позволяет осуществить преждевременный останов программы.

Эти возможности позволяют уменьшить время до получения решения. Пользователь может получить знание о том, способно ли некоторое изменение параметров задачи привести к улучшению окончательного решения без необходимости решать задачу до выполнения условия останова.

2.3. Реализация итеративного поведения

Каждый Map-элемент существует постоянно на протяжении всего времени работы алгоритма, и все данные, необходимые для работы Map-элемента, хранятся в оперативной памяти. В результате этого не происходит лишних операций чтения и записи при каждой итерации. Чтение входных данных и их обработка осуществляются параллельно.

Таким образом, представленный фреймворк избавлен от накладных расходов, связанных с чтением и записью данных на каждой итерации.

Каждый Map-элемент частями отправляет результаты своих вычислений во входной буфер Reduce-элемента, и продолжает вычисления над другими данными. После того, как Reduce-элемент выполнит необходимые действия над данными, находящимися в его входном буфере – он выдает предварительный результат и, при необходимости, посредством обратных связей, посылает корректирующие сообщения Map-элементам.

Заключение

В сообщении описана реализация алгоритма распределенного обнаружения изменения концепции с использованием технологии программирования MapReduce. Приведено описание разработанной программной структуры, предоставляющей набор классов и интерфейсов для реализации итеративных алгоритмов MapReduce.

Для обнаружения изменения концепции используется изменение значения ошибки алгоритма анализа данных. Алгоритм обладает высокой степенью масштабирования, и может применяться для обработки очень больших данных

В дальнейшем работа будет продолжена с целью расширения возможных областей применения данного алгоритма, а также унификации формата входных данных.

Исследование выполнено в рамках реализации программы развития ПНР-5 ФГБОУ ВПО «ЮУрГУ» (НИУ) на 2009–2019 гг.

Литература

1. Andrzejak, A. Parallel Concept Drift Detection with Online Map-Reduce / A. Andrzejak, J.B. Gomes // International Workshop on Knowledge Discovery (KDCloud-2012). Dec. 2012. — P. 402–407.

2. Baena-Garcıa, M. Early drift detection method / M. Baena-Garcıa, J. Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno // The 4th International Workshop on Knowledge Discovery from Data Streams. Sep. 2006. — P. 77–86.
3. Bose, J.-H. Beyond online aggregation: Parallel and incremental data mining with online mapreduce / J.-H. Bose, A. Andrzejak, M. Hogqvist // ACM Workshop on Massive Data Analytics over the Cloud (MDAC 2010). Apr. 2010.
4. Doulkeridis, C. A survey of large-scale analytical query processing in MapReduce / C. Doulkeridis, K. Nørvåg // The VLDB Journal. 2013. — P. 1–26.
5. Gama, J. Learning with drift detection / J. Gama, P. Medas, G. Castillo, P. Rodrigues // Advances in Artificial Intelligence. Nov. 2004. — Vol. 3171. — P. 286–295.
6. Sobhani, P. New Drift Detection Method for Data Streams / P. Sobhani, H. Beigy // Adaptive and Intelligent Systems. Sep. 2011. — Vol. 6943. — P. 88–97.

Волков Антон Александрович, магистрант факультета Вычислительной математики и информатики, Южно-Уральский государственный университет (Челябинск, Российская Федерация), notnavol@gmail.com.

Люц Бюх, Институт информатики, Гейдельбергский университет (Германия), lutz.buech@informatik.uni-heidelberg.de.

Артур Андреяк, доктор., профессор Института информатики, Гейдельбергский университет (Германия), artur.andrzejak@informatik.uni-heidelberg.de.

Поступила в редакцию 2 октября 2013 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 1, pp. 113–120

A METHOD FOR DISTRIBUTED CONCEPT DRIFT DETECTION

A.A. Volkov, South Ural State University (Chelyabinsk, Russian Federation),
L. Büch, Heidelberg University (Heidelberg, Germany)
A. Andrzejak, Heidelberg University (Heidelberg, Germany)

The paper introduces a method for distributed concept drift detection for data mining algorithms. Concept drift is understood as any unpredictable alteration in input data. There is an algorithm implementation proposed, based on MapReduce distributed computing technology. Proposed algorithm meant for concept drift detection in streaming data in online fashion. In order to provide iterative Map and Reduce phases a MapReduce framework is introduced. The algorithm is able to automatically detect input data alteration, which demands model parameters change and switching a new model online.

Keywords: concept drift, data mining, distributed computations, iterative MapRedice.

References

1. Andrzejak A., Gomes J.B. Parallel Concept Drift Detection with Online Map-Reduce // International Workshop on Knowledge Discovery (KDCloud-2012). Dec. 2012. P. 402–407.
2. Baena-Garcıa M., Campo-Avila J., Fidalgo R., Bifet A., Gavalda R., Morales-Bueno R. Early drift detection method // The 4th Int. Workshop on Knowledge Discovery from Data Streams. Sep. 2006. P. 77–86.
3. Bose J.-H., Andrzejak A., Hogqvist M. Beyond online aggregation: Parallel and incremental data mining with online mapreduce // ACM Workshop on Massive Data Analytics over the Cloud (MDAC-2010). Apr. 2010.
4. Doulkeridis C., Nørvåg K. A survey of large-scale analytical query processing in MapReduce // The VLDB Journal. 2013. P. 1–26.
5. Gama J., Medas P., Castillo G., Rodrigues P. Learning with drift detection // Advances in Artificial Intelligence. Nov. 2004. Vol. 3171. P. 286–295.
6. Sobhani P., Beigy H. New Drift Detection Method for Data Streams // Adaptive and Intelligent Systems. Sep. 2011. Vol. 6943. P. 88–97.

Received 2 October 2013