1 Анализ больших данных

Сэнди Риза

[Информационные приложения]
подобны колбасе. Лучше не видеть,
как делают и то и другое.

Отто фон Бисмарк

- Построение модели для выявления мошенничества с кредитными картами на основании тысяч признаков и миллиардов транзакций.
- Обдуманные рекомендации миллионов товаров миллионам пользователей.
- Оценка финансовых рисков с помощью имитационного моделирования портфелей ценных бумаг, включающего миллионы инструментов.
- Легкая обработка данных тысяч человеческих геномов для обнаружения генетической связи заболеваний.

Таковы задачи, выполнение которых десять или даже пять лет назад было просто невозможным. Когда говорят, что мы живем в эпоху больших данных, то подразумевают, что у нас появились инструменты для сбора, хранения и обработки информации в неслыханных доселе масштабах. Основой для этих возможностей является экосистема программного обеспечения с открытым исходным кодом, умеющая использовать кластеры серийных компьютеров для обработки огромных объемов данных. Распределенные системы, такие как Apache Hadoop, сумели стать частью мейнстрима и широко внедряются на предприятиях практически в каждой отрасли.

Но так же, как резец и глыба камня сами не превратятся в статую, так и от доступа к подобным инструментам и данным очень далеко до извлечения из них какой-либо пользы. Именно тут приходит на помощь наука о данных. Как скульптура — это навык преобразования инструментов и сырья во что-то значимое для обычных людей, так и наука о данных — навык превращения инструментов и первичных данных в нечто потенциально значимое для кого-то, кроме исследователей данных.

Зачастую под извлечением пользы подразумевается построение на основе этих данных схемы и применение SQL для ответа на вопросы вроде «Сколько из несметного количества пользователей, дошедших до третьего шага в нашем

процессе регистрации, старше 25 лет?» Структурирование хранилищ данных и организация информации для облегчения получения ответов на подобные вопросы — благодатная тема для обсуждения, но в этой книге мы не будем ее подробно рассматривать.

Иногда для извлечения пользы требуется большее. SQL может по-прежнему лежать в основе подхода, но, чтобы избежать проблем с особенностями структуры данных или выполнить сложный анализ, нам может понадобиться более гибкая и низкоуровневая парадигма программирования, обладающая богатой функциональностью в таких областях, как машинное обучение и статистика. Именно такие виды анализа мы будем обсуждать в книге.

Уже давно такие фреймворки с открытым исходным кодом, как R, стек PyData, а также Octave, делают возможными быстрый анализ и построение моделей на основе небольших наборов данных. С помощью не более чем десяти строк кода мы можем создать модель машинного обучения на основе половины набора данных и использовать ее для прогнозирования второй половины. Приложив еще немного усилий, можно определить значения недостающих данных, опробовать несколько моделей для нахождения наилучшей или использовать результаты одной модели в качестве входных данных для другой. Как же должен выглядеть процесс, который мог бы использовать кластеры компьютеров для достижения аналогичных результатов на больших наборах данных?

Правильным подходом могло бы быть простое расширение этих фреймворков для запуска на нескольких машинах с сохранением их моделей программирования и переписыванием важных составляющих для корректной работы в распределенной среде. Однако проблемы распределенных вычислений требуют от нас пересмотра многих базовых допущений, принимаемых для состоящих из одного узла систем. Например, поскольку данные должны быть распределены по множеству узлов в кластере, алгоритмы с обширными зависимостями данных будут страдать из-за того, что скорость передачи по сети на порядки ниже, чем скорость доступа к памяти. По мере роста количества машин, работающих над конкретной задачей, вероятность сбоя тоже растет. Эти факты требуют парадигмы программирования, которая будет учитывать характеристики базовой системы, препятствовать плохим решениям и облегчать написание кода с высокой степенью параллелизма.

Безусловно, рассчитанные на одну машину инструменты вроде PyData и R, в последнее время ставшие популярными в сообществе разработчиков, — не единственные, используемые для анализа данных. Такие сферы науки, как геномика, имеющие дело с большими наборами данных, уже десятилетиями применяют фреймворки параллельных вычислений. Большинство людей, обрабатывающих данные в этих отраслях, хорошо знакомы со средой кластерных вычислений HPC (High-Performance Computing — высокопроизводительные вычисления). Если проблема с PyData и R заключается в их неспособности к масштабированию, основные проблемы HPC — его относительно низкий уровень абстракции и сложность в использовании. Например, для параллельной обработки большого файла, содержащего результаты секвенирования ДНК, нам придется вручную разбить его на файлы меньшего размера и отправить задание для каждого из них планировщику кластера. Если какие-то из этих заданий завершатся неудачей, пользователю при-

дется выявить место сбоя и вручную отправить их заново. Если анализ требует выполнения затрагивающих все данные операций, таких как сортировка, придется пропустить огромный набор данных через один узел или же прибегнуть к использованию низкоуровневых фреймворков, например MPI, программировать которые нелегко без глубоких знаний языка программирования C и распределенных/сетевых систем. Написанные для сред HPC инструменты часто не могут разделить в оперативной памяти модели данных и низкоуровневые модели хранения. Например, многие инструменты умеют только читать данные из файловой системы POSIX одним потоком, что затрудняет их естественное распараллеливание, или использовать другие серверные хранилища, например базы данных. Новейшие системы из экосистемы Hadoop предоставляют абстракции, позволяющие пользователям работать с кластером практически как с отдельным компьютером: автоматически разбивать файлы и распределять хранилище данных на много машин, автоматически разделять задачу на задания поменьше и выполнять их распределенным образом, а также автоматически восстанавливаться после сбоев. Экосистема Hadoop может автоматизировать множество проблемных задач при работе с большими наборами данных, причем требует гораздо меньших затрат, чем HPC.

Основные проблемы науки о данных

С некоторыми горькими фактами так часто сталкиваешься в практике науки о данных, что их разъяснение стало одной из важных задач команды исследователей данных из компании Cloudera. Системы, стремящиеся сделать возможным системный анализ больших массивов данных, должны учитывать эти факты или по крайней мере не противоречить им.

В-первых, бóльшая часть работы, выполняемой при проведении успешного анализа, заключается в предварительной обработке данных. Данные беспорядочны, и их очистка, изменение, слияние, перемешивание и многие другие действия необходимы для извлечения из них пользы. В частности, большие наборы данных из-за их недоступности для непосредственного анализа людьми могут требовать вычислительных методов, просто чтобы выяснить, какие шаги предварительной обработки необходимы. Даже когда речь идет об оптимизации быстродействия модели, для типичного конвейера данных потребуется потратить гораздо больше времени на проектирование и выбор признаков, чем на подбор и написание алгоритмов.

Например, при построении модели для выявления мошеннических покупок на веб-сайте исследователю данных придется выбирать из широкого множества потенциальных признаков, таких как: любые поля, которые необходимо заполнять пользователю, информация об IP, даты и время входа на сайт, журналы нажатий клавиш во время навигации пользователя по сайту. У них всех есть свои сложности, возникающие при преобразовании в векторы, подходящие для алгоритмов машинного обучения. Системе понадобится поддержка более гибких преобразований, чем простое превращение двухмерного массива чисел с двойной точностью в математическую модель.

Во-вторых, основополагающим понятием науки о данных является итерация. Моделирование и анализ обычно требуют множественных проходов по одним и тем же данным. Одна из причин этого заключается в самой сущности алгоритмов машинного обучения и статистических процедур. Популярные процедуры оптимизации, такие как стохастический градиентный спуск или метод максимизации математического ожидания, включают повторные проходы по входным данным до достижения сходимости. Итерации также имеют значение, когда мы говорим о последовательности действий исследователей данных. Когда исследователи предварительно изучают данные и пытаются их *прочувствовать*, результаты запроса обычно служат источником для следующего запроса. При построении моделей исследователи данных не стараются создать их правильно с первого раза. Выбор правильных признаков, подбор нужных алгоритмов, выполнение правильных проверок по критериям значимости и нахождение правильных гиперпараметров — все это требует экспериментирования. Фреймворк, которому необходимо каждый раз читать один и тот же набор данных с диска при каждом обращении к нему, приводит к задержке, замедляющей процесс исследования, и ограничивает количество вещей, которые мы можем попробовать.

В-третьих, выполнение задачи не заканчивается построением хорошо функционирующей модели. Если смысл науки о данных в том, чтобы сделать данные полезными для кого-то еще, кроме их исследователей, то модель, хранящаяся в виде регрессионных коэффициентов в текстовом файле на компьютере исследователя данных, не очень-то хорошо выполняет эту задачу. Использование механизмов рекомендаций по данным и систем обнаружения мошенничества в режиме реального времени достигает наивысшей точки в информационных приложениях. В них модели становятся частью сервиса, работающего в режиме эксплуатации, и требуют периодической (иногда даже в реальном времени) перестройки.

В таких случаях полезно делать различие между аналитикой в *лабораторных условиях* и в *условиях эксплуатации*. В лабораторных условиях исследователи данных занимаются исследовательской аналитикой. Они пытаются понять природу данных, с которыми имеют дело, визуализируют их и проверяют нелепые теории. Они экспериментируют с различными классами признаков и вспомогательными источниками, которые только можно использовать для дополнения данных. Они закидывают широкие сети различных алгоритмов в надежде, что один или два из них сработают. В условиях эксплуатации при создании реального приложения исследователи данных занимаются операционной аналитикой. Они оформляют свои модели в виде сервисов, которые могут служить источником практических решений. Они постоянно отслеживают производительность своих моделей и все время думают, как бы еще чуть-чуть улучшить модель, чтобы выжать из нее лишний процент точности. Они волнуются по поводу SLA (Service Level Agreement — соглашение об уровне услуг) и времени доступности сервиса. Ранее исследовательская аналитика обычно выполнялась на таких языках, как R, а когда наступало время создавать приложения для промышленной эксплуатации, конвейеры данных полностью переписывались на C++ или Java.

Конечно, немало времени может быть сэкономлено, если исходный код моделирования может быть использован в приложении, для которого он пишется, но языки программирования вроде R отличаются медлительностью и им недостает

интеграции с большей частью элементов стека эксплуатационной инфраструктуры, а языки вроде C++ или Java плохо подходят для исследовательской аналитики. Им не хватает среды REPL (Read — Evaluate — Print Loop, цикл «чтение — оценка — вывод») для интерактивной игры с данными, и они требуют значительного количества кода для реализации простейших преобразований. Фреймворк, дающий возможность удобного моделирования и хорошо подходящий для разработки промышленных систем, — колоссальный шаг вперед.

Знакомство с Apache Spark

Встречайте Apache Spark — фреймворк с открытым исходным кодом, сочетающий в себе механизм распределения программ по кластеру машин с изящной моделью для написания программ поверх него. Spark, созданный на факультете AMPLab Калифорнийского университета в Беркли, а затем отданный фонду Apache Software Foundation, — вероятно, первое программное обеспечение с открытым исходным кодом, по-настоящему дающее исследователям данных возможность использовать распределенное программирование.

Лучше понять Spark можно, рассмотрев его с точки зрения преимуществ перед его предшественником, MapReduce. MapReduce произвел революцию в обработке огромных наборов данных, предложив простую модель написания программ, которые могут выполняться параллельно на сотнях или тысячах машин. Движок MapReduce дает практически линейную масштабируемость: при увеличении объема данных мы можем взять для их обработки больше компьютеров и в результате задание будет выполнено за то же количество времени. Кроме того, он отказоустойчив в свете того факта, что нечастые на отдельной машине сбои происходят постоянно на кластерах из тысяч машин. Он разбивает задачу на небольшие *задания* и умеет изящно обрабатывать сбойные ситуации без ущерба для задания, к которому они относятся.

Spark сохраняет линейную масштабируемость и отказоустойчивость MapReduce, вдобавок расширяя их в трех важных направлениях. Во-первых, вместо того, чтобы полагаться на жесткий формат отображения и свертки, его движок может выполнять более универсальный ориентированный ациклический граф (Directed Acyclic Graph (DAG)) операторов. Это значит, что в тех случаях, когда MapReduce приходится записывать промежуточные результаты в распределенную файловую систему, Spark может передать их непосредственно следующему шагу конвейера. В этом он напоминает Dryad (http://research.microsoft.com/en-us/projects/dryad/) — созданный в исследовательском подразделении компании Microsoft потомок MapReduce. Во-вторых, он дополняет указанную возможность богатым набором преобразований, позволяющих пользователям задавать вычисления более естественным образом. Он ориентирован прежде всего на разработчиков и обладает потоковым API, способным задавать сложные конвейеры в нескольких строках кода.

В-третьих, Spark расширяет возможности своих предшественников с помощью обработки в оперативной памяти. Его абстракция RDD (Resilient Distributed Dataset — устойчивый распределенный набор данных) позволяет разработчикам

материализовать любое место в обрабатываемом конвейере в памяти машин кластера, благодаря чему последующим шагам, желающим работать с теми же данными, не нужно будет заново их вычислять или считывать с диска. Эта возможность открывает путь к сценариям использования, бывшим ранее недоступными для механизмов распределенной обработки. Spark отлично приспособлен к высокоитеративным алгоритмам, требующим множественных проходов по набору данных, равно как и к работающим по запросу приложениям, быстро отвечающим на запросы пользователя благодаря просмотру больших наборов данных в памяти.

Но что важнее всего, Spark хорошо согласуется с упомянутыми ранее горькими фактами науки о данных, признавая, что наиболее узким местом при создании информационных приложений является не процессор, диск или сеть, а производительность аналитика. Вероятно, невозможно преувеличить, насколько слияние всего конвейера, от предварительной обработки до оценки модели, в единую среду программирования может ускорить разработку. Объединение выразительной модели программирования с набором аналитических библиотек в REPL позволяет избежать переключений на IDE и обратно, неизбежных во фреймворках типа MapReduce и трудностей прореживания и перемещения данных из HDFS и в нее, требуемых такими фреймворками, как R. Чем быстрее аналитики могут выполнять эксперименты с их данными, тем больше вероятность, что они извлекут из этих данных какую-то пользу.

Что касается уместности внесения изменений в файлы и ETL[1], Spark ближе к Python для больших данных, чем Matlab для больших данных. Будучи универсальным вычислительным механизмом, базовый API предоставляет прочную базу для преобразований данных, независимую от какой-либо функциональности для статистики, машинного обучения и матричной алгебры. Его API для Scala и Python позволяют программировать на выразительных универсальных языках программирования, а также обращаться к существующим библиотекам.

Способность Spark выполнять кэширование в оперативной памяти делает его идеальным для итерации как на микро-, так и на макроуровне. Алгоритмы машинного обучения, выполняющие несколько проходов по обучающей последовательности, могут кэшировать ее в памяти. При исследовании набора данных и попытке их «прочувствовать» исследователи могут держать его в оперативной памяти во время выполнения запросов и легко кэшировать преобразованную версию, опять же без отправки на диск.

Наконец, Spark заполняет брешь между системами, предназначенными для исследовательской и операционной аналитики. Часто цитируется высказывание, что исследователь данных — лучший инженер, чем большинство статистиков, и лучший статистик, чем большинство инженеров. По меньшей мере, Spark — лучшая система для эксплуатации, чем большинство исследовательских систем, и лучше подходит для исследования данных, чем технологии, обычно используемые в действующих системах. Он с самого начала построен в расчете на производительность и надежность. Работая поверх JVM, он способен использовать многие операционные и отладочные инструменты, созданные для стека Java.

---

[1] ETL (от англ. Extract, Transform, Load — «извлечение, преобразование, загрузка») — процесс, широко используемый при управлении хранилищами данных.

Spark может похвастаться отличной интеграцией с множеством инструментов из экосистемы Hadoop. Он умеет читать и записывать данные во всех форматах, поддерживаемых MapReduce, позволяет взаимодействовать с форматами, часто используемыми для хранения данных в Hadoop, например Avro и Parquet (а также старым добрым CSV). Он может считывать и записывать данные в NoSQL-базы данных, такие как HBase и Cassandra. Его библиотека потоковой обработки Spark Streaming может непрерывно получать данные из таких систем, как Flume и Kafka. Его библиотека SparkSQL может взаимодействовать с хранилищем метаданных Hive (Hive Metastore), и существует проект, на момент написания этой книги находящийся в процессе разработки и призванный дать Spark возможность быть использованным в качестве базового механизма выполнения для Hive в качестве альтернативы MapReduce. Он может запускаться из YARN — планировщика и менеджера ресурсов Hadoop, делая возможным динамическое разделение ресурсов кластера, а также управление с помощью тех же правил, которых придерживаются и другие механизмы обработки, такие как MapReduce и Impala.

Разумеется, Spark отнюдь не идеален. Хотя его базовый движок улучшается прямо на глазах, он все еще молод по сравнению с MapReduce и пока не превосходит его в качестве рабочей лошадки пакетной обработки. Его специализированные подкомпоненты для потоковой обработки, SQL, машинного обучения и работы с графами находятся в разной степени зрелости и подвержены значительным изменениям API. Например, модель преобразователей API и конвейеров MLlib во время написания этой книги находилась в разработке. Его статистическая и моделирующая функциональность очень далека от таковой в предназначенных для одной машины языках, таких как R. Его SQL-функциональность довольно велика, но все еще намного отстает от аналогичной функциональности Hive.

Об этой книге

Остальные главы книги не будут посвящены достоинствам и недостаткам Spark. Издание познакомит вас с моделью программирования Spark и основами Scala, но не будет даже пытаться выдать себя за справочник по Spark или всеобъемлющее руководство по всем его закоулкам. Книга также не будет пытаться стать справочником по машинному обучению, статистике или линейной алгебре, хотя многие ее главы будут предоставлять некоторые предварительные знания по этим темам.

Вместо этого книга попытается помочь читателю *почувствовать*, на что похоже использование Spark для системной аналитики на больших наборах данных. Она будет охватывать весь конвейер: не только создание и оценку моделей, но и очистку, предварительную обработку и исследование данных, уделяя внимание превращению результатов в приложения для промышленной эксплуатации. Мы верим, что наилучший способ научить этому — рассмотрение примеров, так что после короткой главы с описанием Spark и его экосистемы последующие главы будут представлять собой самостоятельные законченные поясняющие примеры

того, что такое использование Spark для анализа данных из различных предметных областей.

Когда только возможно, мы будем стараться не просто предоставлять решение, а показывать полный технологический процесс науки о данных со всеми его итерациями, тупиками и повторными запусками. Эта книга окажется полезна для освоения Scala, Spark и машинного обучения и анализа данных. Однако все это делается во имя более глобальной цели и мы надеемся, что прежде всего книга научит вас, как обращаться с задачами, подобными описанным в начале первой главы. Каждая глава будет стараться максимально приблизиться к тому, чтобы продемонстрировать, как создать один из таких компонентов информационных приложений.