Результаты работы

Реализован алгоритм поиска коэффициентов линейной регрессии с помощью градиентного спуска. Имеются две реализации:

1) однопоточная обработка данных,

2) распределённая обработка данных с помощью фреймворка Spark.

Написаны unit тесты, проверяющие корректность поведения алгоритма.

Описание распределённого алгоритма

1. Сохраняем выборку точек из файла в RDD.
2. Инициализируем вектор параметров регрессии как единичный вектор.
3. Инициализируем значение функции стоимости максимально большим.
4. Пока количество итераций не превысит пороговое значение
   - 4.1. Сохраняем вектор параметров как Broadcast переменную.
   - 4.2. Для каждого элемента вектора
     - 4.2.1. Запускаем подсчёт производной функции стоимости распределённо, аккумулируя прибавочное значение.
     - 4.2.2. Прибавляем к элементу вектора полученное значение.
   - 4.3. Сохраняем новое значение вектора в память рабочих машин.
   - 4.4. Распределённо рассчитываем значение функции стоимости.
   - 4.5. Сравниваем полученное значение стоимости с полученным на предыдущем шаге (или инициализированным, если шаг первый).
     - 4.5.1. Если модуль разности меньше допустимого, результат принимается, возвращается вектор коэффициентов
     - 4.5.2. Иначе сохраняем значение стоимости и идём к пункту 4.1.

Запуск программы

1) Сохраняем файл с выборкой точек в формате: в каждой строке по одной точке, в первых колонках записываются значения аргуметов (x_1 x_2 x_3 ... x_m), в последней - зависимой переменной (y).

x_1	x_2	...	x_n	y
x_1	x_2	...	x_n	y
x_1	x_2	...	x_n	y

Смотрите пример в data/samples/sample_1.txt.

2) Сохраняем данные о выборке в конфигурационный файл: в первой строке размер выборки (количество точек), во второй строке размерность вектора независимой переменной.

3) Запускаем программу со следующими аргументами: путь к файлу с выборкой, путь к конфигурационному файлу, Master URLs, скорость поиска минимума (learning rate), точность результата (convergence criteria), максимальное количество итераций. Пример:

"data/samples/sample_1.txt" "data/samples/config_1.txt" " local[*]" 1.0 0.0001 1000

Тестирование

Для каждого теста генерируется случайная выборка на основе случайной линейной функции. К каждой функции прибавляется случайный шумовой коэффициент.

Первый тест проверяет, что добавление в выборку точных результатов приведёт к более точному результату: вектор параметров, полученный с помощью алгоритма для более точной выборки, будет меньше отличаться от искомого, по которому строилась выборка.

1. Строится зашумлённая выборка.
2. По выборке строится вектор коэффициентов.
3. Добавляются 100 точных значений.
4. По новой выборке строится новый вектор коэффициентов регрессии.
5. Находится среднеквадратичное отклонение вектора из пункта 2 от искомого вектора.
6. Находится среднеквадратичное отклонение вектора из пункта 4 от искомого вектора.
7. Проверяется, что значение из пункта 6 меньше значения из пункта 5.

Второй тест проверяет, что реализация с распределёнными вычислениями и однопоточная, даёт схожие результаты.

1. Строится зашумлённая выборка.
2. По выборке строится вектор коэффициентов с помощью Spark реализации.
3. По выборке строится вектор коэффициентов с помощью однопоточной реализации.
4. Находится среднеквадратичное отклонение одного вектора от другого.
5. Задаётся порог расхождения двух векторов.
6. Проверяется, что отклонение меньше порога.

Bencmark

Результаты замеров средней скорости работы с помощью JMH Benchmark для выборки из 1000 элементов с размерностью вектора зависимых переменных 3.

local[1]

Benchmark	Mode	Cnt	Score	Error	Units
Scalability.measureBatchGradientDescend	avgt	5	1373.675	± 684.800	ms/op

local[2]

Benchmark	Mode	Cnt	Score	Error	Units
Scalability.measureBatchGradientDescend	avgt	5	1456.066	± 532.153	ms/op

local[4]

Benchmark	Mode	Cnt	Score	Error	Units
Scalability.measureBatchGradientDescend	avgt	5	2041.167	± 1592.933	ms/op

local[8]

Benchmark	Mode	Cnt	Score	Error	Units
Scalability.measureBatchGradientDescend	avgt	5	3165.457	± 3207.076	ms/op

(Смотри графическое представление в приложении 1)

Измерения показывают, что увеличение количества потоков не даёт прирост по производительности. Это указывает на недостаток реализации: во время работы происходит избыточное общение клиента и рабочих машин. Можно предположить, что увеличение размера выборки увеличит эффект от масштабируемости основной задачи (подсчёт функции стоимости). Во-первых, увеличится загрузка рабочих машин, во-вторых, нагрузка на клиенте не изменится, количество пересылаемых данных тоже останется прежним.

Для сокращения количества взаимодействий между клиентом и рабочей машиной можно использовать следующее. Вместо итеративного изменения вектора коэффициентов на клиенте, можно аккумулировать значение всего вектора целиком на рабочих машинах. Для этого необходимо реализовать векторный класс-аккумулятор, который будет реализовывать интерфейс AcumulatorV2.

Комментарии к задаче

Реализованный алгоритм является горизонтально масштабируемым на двух участках: подсчёт частной производной функции стоимости, подсчёт самой функции стоимости. Эти две операции — агрегация данных выборки. Результат накапливается на каждом вычислительном узле отдельно, затем складывается. Spark предоставляет такую возможность и делает распределение вычислений и данных автоматически.

Метод градиентного спуска неуниверсальный. На системы уравнений накладываются некоторые ограничения, а именно на собственные числа матрицы и число обусловленности. Работа «неудобными» матрицами может приводить к ситуации, когда алгоритм начинает удаляться от минимума с экспоненциальной скоростью, либо скорость сходимости к минимуму очень мала.

Для решения таких проблем можно использовать другие алгоритмы для уточнения начального приближения вектора коэффициентов регрессии. Либо изменять значение коэффициента, регулирующего величину шага приближения к минимуму, как это делается, например, в алгоритмах барьерных и штрафных функций поиска условного минимума.

Приложения

Приложение 1. График зависимости среднего времени работы алгоритма от количества используемых потоков. Размер выборки: 1000, размерности: 3.