SIEMENS

Отчёт по заданию на стажировку. Градиентный спуск. Java+Spark.

Выполнил Ощепков Артём
semitro_8@mail.ru
github.com/semitro/GradientDescent

Санкт-Петербург 2018

Задание

Реализовать распределенную версию градиентного спуска в Apache Spark на Java.

Описание программы

Градиентный спуск реализован дважды: сначала в последовательной, а затем и в параллельной версиях. В обоих количество коэффициентов θ произвольно.
На первом шаге алгоритма всем θ присваивается значение 0. Далее θ_i изменяются сильнее или слабее в зависимости от величины частной производной.
Ошибка на каждой следующей итерации оценивается как средняя разность между θ_j и θ_{j+1}. Алгоритм останавливается тогда, когда ошибка становится меньше указанной точности или если она увеличилась в сравнении с предыдущей.
Отмечу, что функция ошибок растёт с увеличением размера набора данных и не может быть минимизирована меньше определённого порога, так как модель линейной регрессии не может точно аппроксимировать произвольный набор данных.

Запуск

1. Собрать jar

git clone https://github.com/semitro/GradientDescent
cd GradientDescent
mvn clean compile assembly:single

2. Положить поближе к нему датасеты

cp ./src/main/resources/*csv ./target

3. Запустить

cd ./target

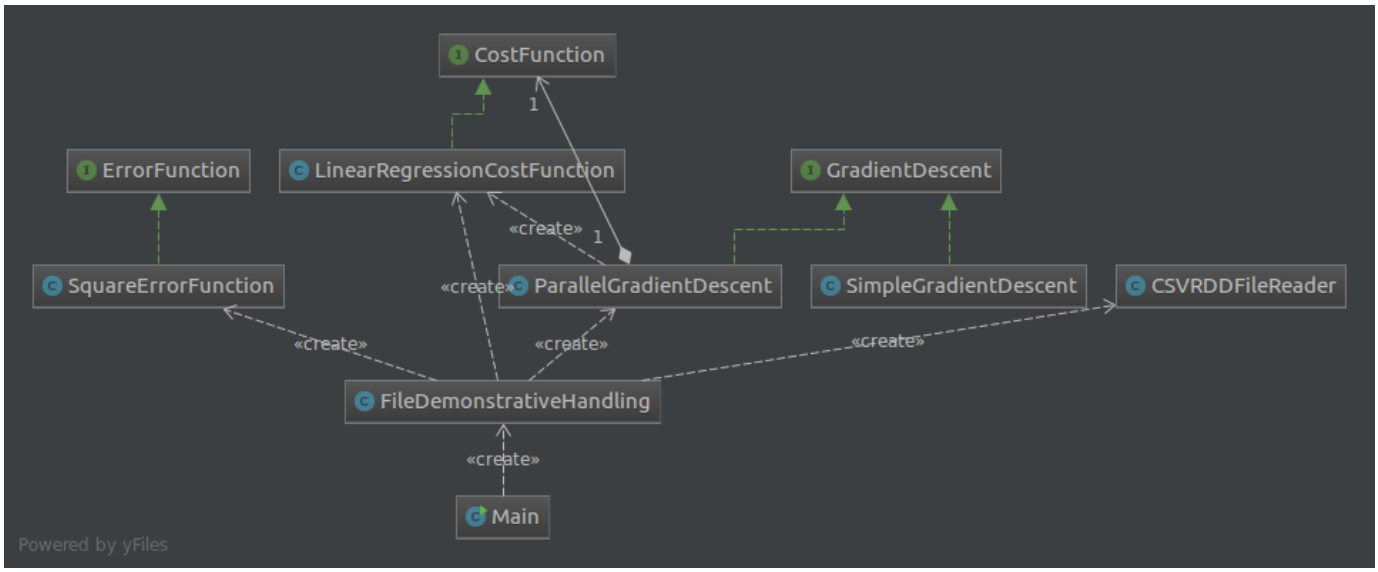
java -jar ./GradientDescent*.jar dataset.csv epsilon speed

java -jar ./GradientDescent.jar dataset1.csv 0.1 0.0005*

4. Для запуска юнит-тестов

mvn test

Диаграмма классов



Тестирование

Написано 3 категории Unit-тестов:

- Подбор коэффициентов, хранящихся в памяти для последовательной и параллельной реализации.

К примеру, для теста { {2.0, 2.0, 4.0}, {4.0, 4.0, 8.0} результат должен быть примерно { 1.0, 1.0, 0.0 }, т.к. 4 = 1*2 + 1*2 + 0

- Работа с csv-файлами на для параллельного спуска:
  dataset1.csv (284 K, 2 поля) при epsilon = 1. и step=0.0005:
  61.907, 256.400
  33.513, 118.679
  70.349, 300.218
  …
  Значение функции ошибки J с начальными коэффициентами: 6.18E8.
  Средняя разность между предсказанием и действительным значением снизилась с 7219.14 на первой итерации до 4.28 на последней.
  Для полученных коэффициентов [4.28, -0.045] J приняла значение 3561802, т.е. уменьшилась.
  По приведённой выборке данных видно, что полученные коэффициенты удовлетворяют набору данных:
  70.35*4.28 - 0.045 = 301.05 ≈ 300.22

Алгоритм отработал за 7679 ms на Celeron N2830 @ 2.16GHz × 2
На датасете размера 3мб и 10 полей при epsilon = 5.0, speed = 5.0E-14 средняя погрешность предсказания снизилась с 1.18E7 до 802 за 81838 ms.
J при этом снизилась с 1.84930974E8 до 43898538,31.

О кластерах

Была предпринята попытка запустить программу на кластере Spark. Удалось объединить в кластер 3 ноутбука, получив суммарно 10 ядер.

Запуск программы через spark-submit дал следующие результаты:

В режиме cluster - программа отрабатывала, но только на одном ядре.

В режиме client — gui информировал, что задействованы все 10 ядер, однако вычисления заступорились на первом же вызове метода класса JavaRDD.