МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Обнинский институт атомной энергетики –
филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»
(ИАТЭ НИЯУ МИФИ)

Отделение интеллектуальных кибернетических систем

ЛАБОРАТОРНАЯ РАБОТА №2
«Изучение возможностей Scala»
по дисциплине
«Анализ литературной работы»

Выполнил студент 1 курса
группы ИВТ-М20
Лискунов Р. Г.

Проверил:
кандидат технических наук
Грицюк С. В.

Обнинск, 2020

Цель работы

Разработать приложение для Apache Spark в автономном режиме. Проанализировать литературное произведение (Федор Михайлович Достоевский – Преступление и наказание) с помощью Spark RDD.

Краткая теория

Apache Spark — фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию MapReduce с хранением промежуточных данных на накопителях, Spark работает в парадигме резидентных вычислений — обрабатывает данные в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы для некоторых классов задач, в частности, возможность многократного доступа к загруженным в память пользовательским данным делает библиотеку привлекательной для алгоритмов машинного обучения.

Структуры данных RDD

RDD – это распределенная коллекция данных, расположенных по нескольким узлам кластера, набор объектов Java или Scala, представляющих данные. RDD работает со структурированными и с неструктурированные данными. Также, как DataFrame и DataSet, RDD не выводит схему загруженных данных и требует от пользователя ее указания.

RDD-коллекция сериализуется каждый раз, когда Spark требуется распределить данные внутри кластера или записать информацию на диск. Затраты на сериализацию отдельных объектов Java и Scala являются дорогостоящими, т.к. выполняется отправка данных и структур между узлами.

Ход работы

В начале работы происходит инициализация конфигурации состояния Spark, что позволяет в дальнейшем подключить контекст работы для прочтения файла с литературным произведением и дальнейшим взаимодействием с ним.

Для анализа я подготавливаю исходный текст с помощью устранения пунктуационных знаков: точка, запятая и так далее. В дополнение, слова данного произведения переводятся в нижний регистр для увеличения числа показателей значений.

В процессе обработки текста участвует специальный список стоп-слов, хранящийся в файле stop.txt. При сравнении исходного текста произведения со словами, которые представлены в этом файле, происходит устранение союзов, предлогов, глаголов и других лишних слов.

Затем происходит поиск 50 самых популярных и 50 самых редко используемых слов. Так, исследование показывает, что имя фамилия главного героя – Раскольников встречается 563 раз, что показывает нам близость автора и основного героя романа.

В процессе лабораторной работы был произведён поиск для 50 самых популярных и редко встречающихся слов, приведу пример выполняемого кода для пяти:

Top5 most common words:
(раскольников,564)
(соня,264)
(разумихин,244)
(петрович,208)
(ивановна,174)

Top5 least common words:
(сменялись,1)
(отворяль,1)
(вырезай,1)
(прибудут-с,1)
(публики,1)

Для семантического поиска однокоренных слов используется stemmer, который позволяет выделять общей корень у родственных (однокоренных) слов. Его использование характеризовано вызов метода mapPartitions, который позволяет преобразовать каждый раздел исходного RDD в результат из нескольких элементов.

В процессе лабораторной работы был произведён поиск для 50 самых популярных и редко встречающихся слов с использованием стемминга, приведу пример выполняемого кода для пяти:

Top5 most common stems:
((пуст,List(пустил, пустейшем, пустите, пустила, пустят, пустую, пустая, пустили, пустой, пустейших, пустился, пусть, пустилась, пустить, пуст, пусто, пусти, пустились, пустим, пустых, пустом, пустит, пустые)),23)
((сдела,List(сделал, сделано, сделала, сделаешь, сделанная, сделался, сделан, сделает, сделали, сделается, сделай, сделайте, сделав, сделать, сделалось, сделаете, сделаны, сделают, сделаться, сделались, сделаю, сделалась)),22)
((начина,List(начинала, начинавший, начинало, начинающего, начинает, начинаем, начинавшаяся, начинал, начинается, начинали, начинай, начинаете, начинались, начинайте, начинавшее, начиналась, начинавшейся, начинают, начинаю, начинающий, начиналось)),21)
((испуга,List(испугалась, испуганно, испугать, испугаться, испуганы, испуганные, испуганный, испугались, испуганном, испуган, испугается, испугали, испуганное, испуганным, испуганного, испугавшись, испуганной, испугался, испугал, испуганная)),20)
((выход,List(выходившему, выход, выходит, выходов, выхода, выходить, выходи, выходом, выходишь, выходя, выходил, выходят, выходивших, выходе, выходили, выходу, выходившая, выходило, выходила, выходите)),20)

Top5 least common stems:
((поцелу,List(поцелуев)),1)
((чер-р-рт,List(чер-р-рт)),1)
((наклепа,List(наклепал)),1)
((прибудут-с,List(прибудут-с)),1)
((примыка,List(примыкавшей)),1)

Листинг кода

```
package LabTwo

import org.apache.log4j.Level.WARN
import org.apache.log4j.LogManager
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.tartarus.snowball.ext.russianStemmer

object LabTwo {
  val PATH: String = "src/main/data"
  val NODES: Int = 3

  def main(args: Array[String]): Unit = {
    val conf: SparkConf = new SparkConf().setAppName("Lab2").setMaster(s"local[$NODES]")
    val sc: SparkContext = new SparkContext(conf)
    LogManager.getRootLogger.setLevel(WARN)

    val book: RDD[String] = sc.textFile(s"$PATH/var.txt")
    val stop: Array[String] = sc.textFile(s"$PATH/stop.txt").collect()
    val text: RDD[(String, Int)] = parse(book = book, stop = stop)

    println("\n Top50 most common words: ")
    val most: Array[(String, Int)] = popular(text = text, ascending = false)
    most.foreach(println)

    println("\n Top50 least common words: ")
    val least: Array[(String, Int)] = popular(text = text, ascending = true)
    least.foreach(println)

    val stemmed: RDD[((String, Iterable[String]), Int)] = text
      .mapPartitions(stemming)
      .groupBy(_._2)
      .map(w => (w._1, w._2.map(_._1._1)) -> w._2.size)

    println("\n Top50 most common stems: ")
    val mostStemmed: Array[((String, Iterable[String]), Int)] = stemmed
      .sortBy(_._2, ascending = false)
      .take(5)
    mostStemmed.foreach(println)

    println("\n Top50 least common stems: ")
    val leastStemmed: Array[((String, Iterable[String]), Int)] = stemmed
      .sortBy(_._2, ascending = true)
      .take(5)
    leastStemmed.foreach(println)
  }

  private def stemming(iter: Iterator[(String, Int)]): Iterator[((String, Int), String)] = {
    val stemmer: russianStemmer = new russianStemmer
    iter.map(w => (w._1, 1) -> {
```

```
    stemmer.setCurrent(w._1)
    stemmer.stem
    stemmer.getCurrent
   })
  }

  private def popular(text: RDD[(String, Int)], ascending: Boolean): Array[(String, Int)] = {
   text.sortBy(_._2, ascending = ascending).take(num = 5)
  }

  private def parse(book: RDD[String], stop: Array[String]): RDD[(String, Int)] = book
   .flatMap(_.toLowerCase.split(" ")
    .map(_.replaceAll("[;:,.!?\"«» “–]", ""))
    .filter(word => word.length > 1 && !stop.contains(word)))
   .map(word => (word, 1))
   .reduceByKey(_ + _)
}
```

Вывод

В ходе лабораторной работы я настроил сборку проекта с помощью Maven для версии языка Scala 2.11.0. Подключил основную библиотеку для фреймворка Spark и написал исходный код программы для анализа романа Ф. М. Достоевского «Преступление и наказание». В процессе были выявлены 50 самых популярных и 50 самых редко используемых слов. Полученные результаты были дополнены процессом стемминга, что позволило глубже изучить семантический анализ текста. Благодаря расширяемому списку стоп-слов проект может быть расширен в рамках более детального рассмотрения.