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.csv. 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 полное имя главного героя – Родин встречается 89 раз, в то время как уменьшительно-ласкательно – 102 раза, что показывает нам близость автора и основного героя романа.

На первом рисунке приведён скриншот с примером выполнения исходного кода.

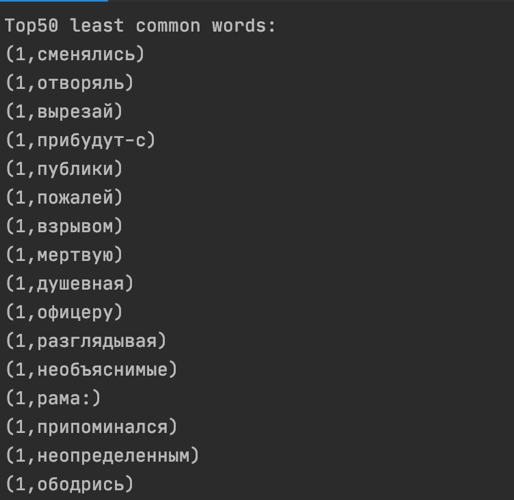


Рисунок 1. Топ 50 мало встречающихся слов

# Листинг кода

package LabTwo  
  
import org.apache.log4j.Level.WARN  
import org.apache.log4j.LogManager  
import org.apache.spark.broadcast.Broadcast  
import org.apache.spark.rdd.RDD  
import org.apache.spark.{SparkConf, SparkContext}  
  
object LabTwo {  
 val PATH: String = "src/main/scala/LabTwo"  
 val NODES: Int = 3  
  
 def main(args: Array[String]): Unit = {  
 val conf: SparkConf = new SparkConf().setAppName("Lab2").setMaster(s"local[$NODES]")  
 val context: SparkContext = new SparkContext(conf)  
 LogManager.getRootLogger.setLevel(WARN)  
  
 val input: RDD[String] = parse(context)  
 val stopWordsInput: RDD[String] = context.textFile(s"$PATH/stop.csv")  
 val text: RDD[String] = stop(context, input, stopWordsInput)  
  
 val most: RDD[(Int, String)] = common(context, text, ascending = false)  
 val least: RDD[(Int, String)] = common(context, text, ascending = true)  
  
 println("Top50 most common words: ")  
 most.foreach(println)  
 println("\nTop50 least common words: ")  
 least.foreach(println)  
 }  
  
 private def parse(sc: SparkContext): RDD[String] = {  
 sc.textFile(s"$PATH/var.txt")  
 .map(\_.toLowerCase())  
 .map(\_.replaceAll(",", ""))  
 .map(\_.replaceAll("\\.", ""))  
 .map(\_.replaceAll("!", ""))  
 .map(\_.replaceAll("\\?", ""))  
 .map(\_.replaceAll(";", ""))  
 .map(\_.replaceAll("\"", ""))  
 }  
  
 private def stop(sc: SparkContext, input: RDD[String], stopWordsInput: RDD[String]): RDD[String] = {  
 // Flatten, collect, and broadcast.  
 val stopWords: RDD[String] = stopWordsInput.flatMap(x => x.split("\n")).map(\_.trim)  
 val broadcastStopWords: Broadcast[Set[String]] = sc.broadcast(stopWords.collect.toSet)  
  
 // Split using a regular expression that extracts words  
 val wordsWithStopWords: RDD[String] = input.flatMap(x => x.split(" "))  
 wordsWithStopWords.filter(!broadcastStopWords.value.contains(\_))  
 }  
  
 private def common(sc: SparkContext, rdd: RDD[String], ascending: Boolean): RDD[(Int, String)] = {  
 // word count  
 val wc: RDD[(String, Int)] = rdd.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)  
 // swap k,v to v,k to sort by word frequency  
 val wc\_swap: RDD[(Int, String)] = wc.map(\_.swap)  
 // sort keys by ascending=false (descending)  
 val words: RDD[(Int, String)] = wc\_swap.sortByKey(ascending = ascending, numPartitions = 1)  
 // get an array of top 50 frequent words  
 val top50: Array[(Int, String)] = words.take(num = 50)  
 // convert array to RDD  
 sc.parallelize(top50)  
 }  
}

# Вывод

В ходе лабораторной работы я настроил сборку проекта с помощью sbt 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. Благодаря список стоп-слов проект может быть расширен в рамках более детального анализа.