МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ОБНИНСКИЙ ИНСТИТУТ АТОМНОЙ ЭНЕРГЕТИКИ - филиал
Федерального государственного автономного образовательного учреждения
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»
(ИАТЭ НИЯУ МИФИ)

Кафедра «Информационных систем»

ОТЧЕТ

По лабораторной работе №4

«Детальный анализ текста: выделение сущностей из литературного произведения на Apache Spark»

Выполнил: студент группы ИС-М16
Рябов П. В.

Проверил: Грицюк С. В.
Бобков И.А

Обнинск – 2016 г.

В данной лабораторной работе разрабатывалась программа в интегрированной среде разработки Scala IDE, в соответствии с вариантом.

Описание задания (Вариант 9: Лев Николаевич Толстой – Воскресение)

Программа должна предоставлять следующий функционал:.

1) Выделение сущностей из текста произведения

Функционал приложения:

Данное приложение было реализовано на основе самописных правил, основанных на регулярных выражениях(regEx-based подход) и алгоритме выделения слов как часть речи.(**Part Of Speech tagging**).(rule-based подход)

1) Приложение использует WordCount и функции получения топа и антитопа из RDD данных на основе исходного кода лабораторной работы №3.

2) Отбор части RDD данных, которые удовлетворяет "Условию общих признаков" . Данная операция производится функцикй-оберткой:

```
def markWord(word: String): Boolean = {
    val first: Char =  word.charAt(0)
    val rus_lang = """[а-яА-Я0-9]{3,10}""".r
    val roman = """(?i)(?:I{1,3}|IV|VI{0,3}|I?X)""".r
    val general: Boolean = rus_lang.pattern.matcher(word).matches
    val rome: Boolean = roman.pattern.matcher(word).matches
    return ((first.isUpper).&(general)).&(rome.equals(false))
  }
```

Встроены следующие общие признаки:

- Должно начинаться с большой буквы
- Должно не содержать римских выражений (Оптимизация под конкретный текст из варианта)
- Должно быть с набором русских символов не менее 3 символов(в общем случае набор алфавита можноо изменить - данное правило является также конфигурацией для функции - обертки)

3) функция catchEntities на основе "маркировщика" markWord путем создания отдельной колонки в RDD данных логического типа отбирает данные, которые удовлетворяют условию маркировки:

```
def catchEntities(wc: RDD[(String, Int)]): RDD[(String, Int)] = {
    //val names = wc.map(x => (x._1))
    //val freq = wc.map(x => (x._2))
    val markTable = wc.map(x => (x._1, markWord(x._1) , x._2))
    val markRecords = markTable.filter(x => (x._2.equals(true)))
    val res = markRecords.map(x => (x._1, x._3))
    return res
  }
```

4) "общий набор RDD" попадает на вход к функции которая выполняет более детальный разбор слов по типу частей речи. Используется java Tagger в отдельном файле `RuleBasedPosTagger.java` и в главном scala файле создается объект, который уже через свой собственный метод сопоставляет каждому слову его часть речи:

```
object Main {
  …
  val tagger = new RuleBasedPosTagger()
  …
def detailParse(general: RDD[(String, Int)]): RDD[(String, Int)] = {
    val posData = general.map(x => (x._1, x._2, tagger.posTag(x._1)))
    println("\n=== Tagging Processing... ===")
    posData.take(50).foreach(println)
    println("\n=== Detail Processing... ===")
    val detail =  posData.filter(x => (x._3.equals(PosTag.NOUN)))
    val res = detail.map(x => (x._1, x._2))
```

```
    return res
  }
```

5) Из полученного RDD на 4 шаге отбираются записи по признаку NOUN - существительное так как это самое популярное для сущности.

Part Of Speech Tagger:

Tagger анализирует слово из RDD набора и присваивает один из следующих типов частей речи, основываясь на встроенных самописных правилах, основанных на подборе окончаний слов и соответствия определенным частям речи определенных окончаний.

Поддерживаются следующие типы:

```
public static enum PosTag {

	ADJECTIVE("JJ"),

	PARTICIPLE("VBG"),

	VERB("VB"),

	NOUN("NN"),

	ADVERB("RB"),

	NUMERAL("NUM"),

	CONJUCTION("CC"),

	PREPOSITION("IN"),

	QUOTES_OPEN("``"),

	QUOTES_CLOSE("''"),

	COMMA(","),

	DASH("--"),

	END_OF_SENTENCE("."),

	OTHER("X");
```

Для каждой части речи выполняет соответствующий метод применения правил

Кроме того Маркировщик и Part Of Speech Tagger написаны так, чтобы легко было в дальнейшем модифицировать и расширить их функционал. Таким образом программа адаптирована для дальнейшей доработки в целях улучшения базвых алгоритмов распознавания сущностей или изменения языковой основы. Также все преобразования работают через RDD, что может быть применено на вычислениях с использованием кластера из большого количества ПК.

Пример конечного результата для Топ 50 сущностей:

```
(Нехлюдов,1009)
(Нехлюдова,294)
(Маслова,185)
(Нехлюдову,176)
(Масловой,117)
(Маслову,63)
(Катюша,62)
(Марья,51)
(Мисси,49)
(Смелькова,33)
(Катюшу,32)
(Кораблева,30)
(Игнатий,28)
(Бочкова,27)
(Крыльцов,26)
(Смотритель,24)
(Дмитрий,24)
(Катюшей,23)
(Вера,21)
(Федосья,20)
(Наталья,20)
```

…

Пример определения части речи для RDD

=== Tagging Processing... ===

(Вспомните,1,VERB)

(Высокие,1,ADJECTIVE)

(Бочкова,27,NOUN)