2016.05

PySpark (Python + Spark) Practice on Project WordCount

**Quick tutorial for pySpark practice**

**Raymond XIE**

PySpark (Python + Spark) Practice on Project WordCount

Contents

[1. Creating a base RDD and pair RDDs 3](#_Toc460836326)

[1.1 Create a base RDD 3](#_Toc460836327)

[1.2 Pluralize and test 3](#_Toc460836328)

[1.2.1 Create a udf to make the word plural 3](#_Toc460836329)

[1.2.2 Test 3](#_Toc460836330)

[1.3 Apply the udf to the base RDD 3](#_Toc460836331)

[1.4 Using a lambda function with map 4](#_Toc460836332)

[1.5 Calculate the length of each word 4](#_Toc460836333)

[1.6 Generate a list from RDD with map/lambda 4](#_Toc460836334)

[2. Counting with pair RDDs 5](#_Toc460836335)

[2.1 Generate pairs (groupByKey or map/lambda) 5](#_Toc460836336)

[2.1.1 approach 1: groupByKey, no map/lambda 5](#_Toc460836337)

[2.1.2 approach 2: with map/lambda 5](#_Toc460836338)

[2.2 Get the counts (with import add or with lambda) 5](#_Toc460836339)

[2.2.1 approach 1: using reduceByKey with import add 5](#_Toc460836340)

[2.2.2 approach 2: no import, using reduceByKey with lambda 6](#_Toc460836341)

[3. Finding unique words and a mean value 7](#_Toc460836342)

[3.1 Finding unique words 7](#_Toc460836343)

[3.2 Get mean using reduce 7](#_Toc460836344)

[4. Apply word count to a file 8](#_Toc460836345)

[4.1 Create a udf wordCount function 8](#_Toc460836346)

[4.2 RDD text element transformation 8](#_Toc460836347)

[4.2.1 Capitalization and punctuation 8](#_Toc460836348)

[4.3 Loading a text file 9](#_Toc460836349)

[4.4 words from lines 10](#_Toc460836350)

[4.4.1 Approach 1: flatMap 10](#_Toc460836351)

[4.4.2 Approach 2: 10](#_Toc460836352)

[4.5 Data Cleansing 11](#_Toc460836353)

[4.6 Count the words 11](#_Toc460836354)

# 1. Creating a base RDD and pair RDDs

## 1.1 Create a base RDD

*wordsList = ['cat', 'elephant', 'rat', 'rat', 'cat']*

*wordsRDD = sc.parallelize(wordsList, 4)*

*# Print out the type of wordsRDD*

*print type(wordsRDD)*

### 

## 1.2 Pluralize and test

### 1.2.1 Create a udf to make the word plural

*def makePlural(word):*

*"""Adds an 's' to `word`.*

*Note:*

*This is a simple function that only adds an 's'. No attempt is made to follow proper*

*pluralization rules.*

*Args:*

*word (str): A string.*

*Returns:*

*str: A string with 's' added to it.*

*"""*

*return word + 's'*

*print makePlural('cat')*

### 1.2.2 Test

*def makePlural(word):*

*return word + 's'*

*print makePlural('cat')*

### 

## 1.3 Apply the udf to the base RDD

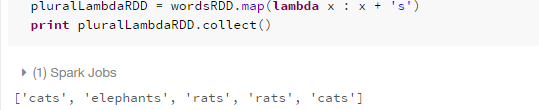
*pluralRDD = wordsRDD.map(lambda x : makePlural(x))*

*print pluralRDD.collect()*

#### 

## 1.4 Using a lambda function with map

*pluralRDD = wordsRDD.map(lambda x : x + 's')*



## 1.5 Calculate the length of each word

*pluralLengths = (pluralRDD*

*.map(lambda x : len(x))*

*.collect())*

*print pluralLengths*

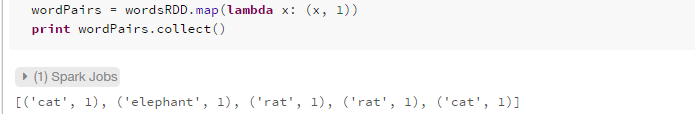
#### 

## 1.6 Generate a list from RDD with map/lambda

Create a pair tuple (k, v) for each word in the RDD

*wordPairs = wordsRDD.map(lambda x: (x, 1))*

*print wordPairs.collect()*



# 2. Counting with pair RDDs

With the pair list generated, we can group the (k, v) that has the same k, and then aggregate the v for each k, thus complete the map-reduce process

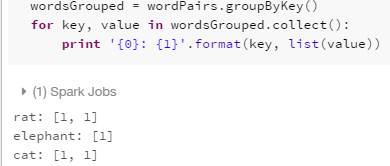
## 2.1 Generate pairs (groupByKey or map/lambda)

### 2.1.1 approach 1: groupByKey, no map/lambda

*wordsGrouped = wordPairs.groupByKey()*

*for key, value in wordsGrouped.collect():*

*print '{0}: {1}'.format(key, list(value)) wordsGrouped = wordPairs.groupByKey()*

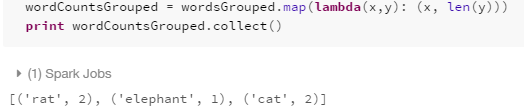


Note: groupByKey needs no parameter

### 2.1.2 approach 2: with map/lambda

*wordCountsGrouped = wordsGrouped.map(lambda(x,y): (x, len(y)))*

*print wordCountsGrouped.collect()*



## 2.2 Get the counts (with import add or with lambda)

### 2.2.1 approach 1: using reduceByKey with import add

*from operator import add*

*wordCounts = wordPairs.reduceByKey(add)*

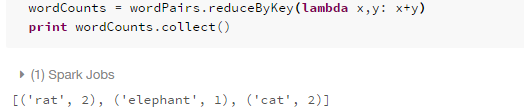
*print wordCounts.collect()*

### 

### 2.2.2 approach 2: no import, using reduceByKey with lambda

*wordCounts = wordPairs.reduceByKey(lambda x,y: x+y)*

*print wordCounts.collect()*



# 3. Finding unique words and a mean value

## 3.1 Finding unique words

*uniqueWords = len(wordCountsCollected)*

*#uniqueWords = wordsRDD.distinct().count()*

*print uniqueWords*

# 

## 3.2 Get mean using reduce

*from operator import add*

*totalCount = (wordCounts*

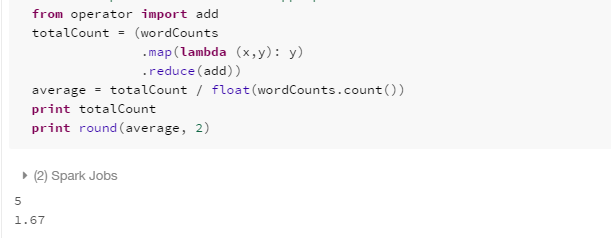
*.map(lambda (x,y): y)*

*.reduce(add))*

*average = totalCount / float(wordCounts.count())*

*print totalCount*

*print round(average, 2)*



# 4. Apply word count to a file

## 4.1 Create a udf wordCount function

*def wordCount(wordListRDD):*

*"""Creates a pair RDD with word counts from an RDD of words.*

*Args:*

*wordListRDD (RDD of str): An RDD consisting of words.*

*Returns:*

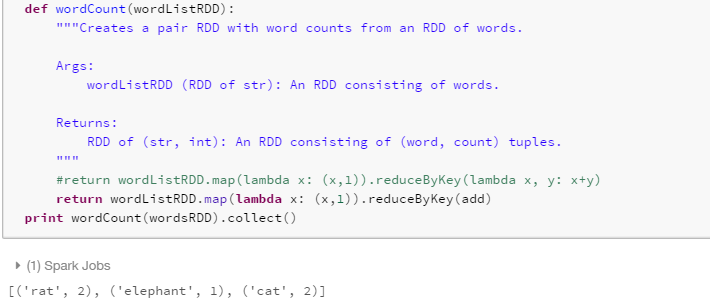
*RDD of (str, int): An RDD consisting of (word, count) tuples.*

*"""*

*#return wordListRDD.map(lambda x: (x,1)).reduceByKey(lambda x, y: x+y)*

*return wordListRDD.map(lambda x: (x,1)).reduceByKey(add)*

*print wordCount(wordsRDD).collect()*



#### Note: there are two approaches when using reduceByKey: 1. using lambda; 2. using built-in function add

## 4.2 RDD text element transformation

### 4.2.1 Capitalization and punctuation

#### 4.2.1.1 Use the Python [re](https://docs.python.org/2/library/re.html) module to remove any text that is not a letter, number, or space

*import re*

*help(re.sub)*

#### 

*import re*

*from pyspark.sql.functions import regexp\_replace*

*def removePunctuation(text):*

*"""Removes punctuation, changes to lower case, and strips leading and trailing spaces.*

*Note:*

*Only spaces, letters, and numbers should be retained. Other characters should should be*

*eliminated (e.g. it's becomes its). Leading and trailing spaces should be removed after*

*punctuation is removed.*

*Args:*

*text (str): A string.*

*Returns:*

*str: The cleaned up string.*

*"""*

*str = re.sub('[^a-z| |0-9]', '', text.strip().lower()).strip()*

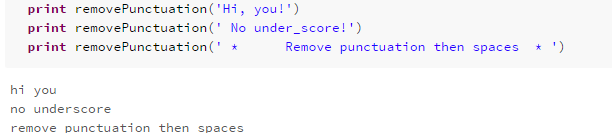
*#str = regexp\_replace(str, ' ', '')*

*return str*

*print removePunctuation('Hi, you!')*

*print removePunctuation(' No under\_score!')*

*print removePunctuation(' \* Remove punctuation then spaces \* ')*



Note: 1. punctuation like , \*, \_, ! are removed with the regex '[^a-z| |0-9]'

strip() is used to remove the leading and trailing spaces

## 4.3 Loading a text file

*import os.path*

*fileName = "dbfs:/" + os.path.join('databricks-datasets', 'cs100', 'lab1', 'data-001', 'shakespeare.txt')*

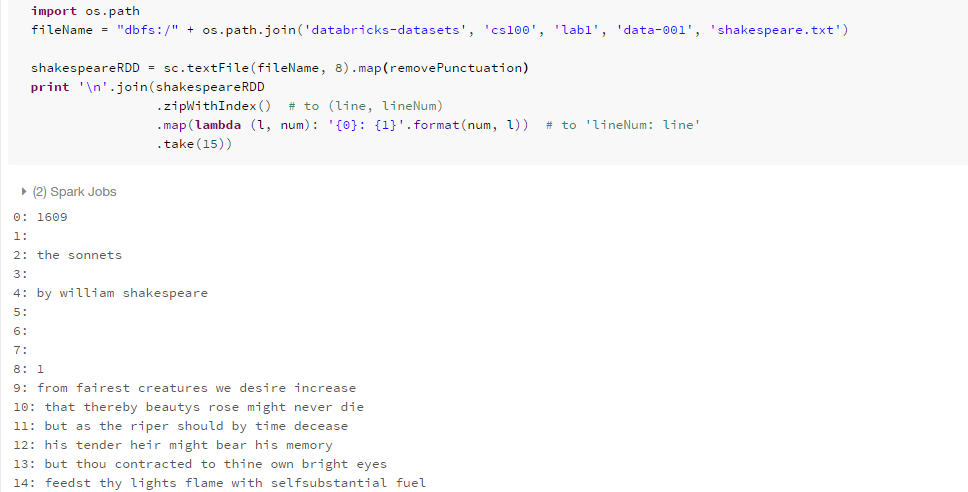
*shakespeareRDD = sc.textFile(fileName, 8).map(removePunctuation)*

*print '\n'.join(shakespeareRDD*

*.zipWithIndex() # to (line, lineNum)*

*.map(lambda (l, num): '{0}: {1}'.format(num, l)) # to 'lineNum: line'*

*.take(15))*



## 4.4 words from lines

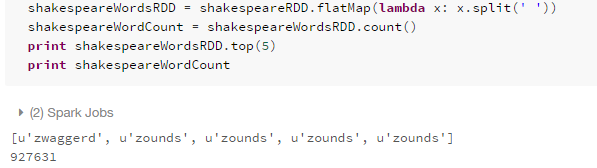
### 4.4.1 Approach 1: flatMap

*shakespeareWordsRDD = shakespeareRDD.flatMap(lambda x: x.split(' '))*

*shakespeareWordCount = shakespeareWordsRDD.count()*

*print shakespeareWordsRDD.top(5)*

*print shakespeareWordCount*



### 4.4.2 Approach 2:

*shakespeareWordsRDD = shakespeareRDD.flatMap(lambda x: removePunctuation(x).split(' '))*

*shakespeareWordCount = shakespeareWordsRDD.count()*

*print shakespeareWordsRDD.top(5)*

*print shakespeareWordCount*

#### 

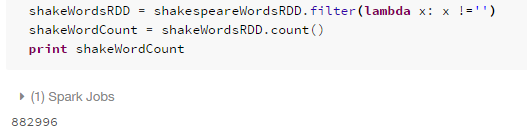
## 4.5 Data Cleansing

#### Now we need to remove all empty lines, i.e. remove all entries with word is ''

*shakeWordsRDD = shakespeareWordsRDD.filter(lambda x: x !='')*

*shakeWordCount = shakeWordsRDD.count()*

*print shakeWordCount*



## 4.6 Count the words

So now we have an RDD with only words (non empty), we are to apply wordCount() udf to produce a list of word counts. Use takeOrdered() to obtain the top n occurrences from the result

*top15WordsAndCounts = wordCount(shakeWordsRDD).takeOrdered(15, key=lambda (a,b): -b)*

*print '\n'.join(map(lambda (w, c): '{0}: {1}'.format(w, c), top15WordsAndCounts))*

#### 