Danmarks Tekniske Universitet

01415
Computational Tools for Big Data

Week 89 - Exercise

Anonymous

November 2016

1 Week 8

Exercise 8.1

Assignment :
Write a Spark job to count the occurrences of each word in a text file. Document that it works with a small example.
Solution :

Since we have been trying to learn as many features of Spark as we could, we have implemented this task in several ways. We have solved the problem both using external functions for the problem and using inline **lambda** functions. We have also resolved the final step of counting the words in two different ways, first using the function **combineByKey()** and then using **reduceByKey()**. We will explain now the steps followed and the different functions used.

The first step is to load the text file and convert it to a RDD parallelizable datastructe. Lucky enough, pySpark comes with the function **sc.textFile(file_dir)** that reads and transforms the given file into an RDD. Each line of the document will be one element of the RDD. We will be applying transformations to this initial RDD to get the job done.

```
[u'1609', u'', u'THE SONNETS', u'', u'by William Shakespeare', u'', u'', u'', u'
              1', u'  From fairest creatures we desire increase,', u"  That thereby bea
uty's rose might never die,", u'  But as the riper should by time decease,', u'  His te
nder heir might bear his memory:', u'  But thou contracted to thine own bright eyes,',
 u"  Feed'st thy light's flame with self-substantial fuel,", u'  Making a famine where
 abundance lies,', u'  Thy self thy foe, to thy sweet self too cruel:', u"  Thou that a
rt now the world's fresh ornament,", u'  And only herald to the gaudy spring,', u'  Wit
hin thine own bud buriest thy content,', u"  And tender churl mak'st waste in niggardin
```

Figure 1: Loaded RDD

Secondly, we use the mapping function **flatMap()** to process each line, dividing it into separate words, but not creating the (key,value) pairs. This function expects to output a list, where every element of the list will be an element of the new RDD.

```
[u'1609', u'the', u'sonnets', u'by', u'william', u'shakespeare', u'1', u'from', u'faires
t', u'creatures', u'we', u'desire', u'increase,', u'that', u'thereby', u"beauty's", u'ros
e', u'might', u'never', u'die,']
```

Figure 2: RDD after flatmap()

Then we use the function **map()** to create the (key,value) pairs from the previous RDD, where every key is the word and every value is 1. Since we are now going to solve the problem in 2 different ways, thus transforming this RDD in two different times, we will use the function **persist()** so that pySpark do not delete the RDD from RAM after the first use. If we do not indicate this, pySpark would need to recompute all the RDD the second time we use it.

```
[(u'1609', 1), (u'the', 1), (u'sonnets', 1), (u'by', 1), (u'william', 1), (u'shakespear
e', 1), (u'1', 1), (u'from', 1), (u'fairest', 1), (u'creatures', 1), (u'we', 1), (u'desir
e', 1), (u'increase,', 1), (u'that', 1), (u'thereby', 1), (u"beauty's", 1), (u'rose', 1),
 (u'might', 1), (u'never', 1), (u'die,', 1)]
```

Figure 3: RDD after map()

Then we use the function **combineByKey()** to solve the problem. This is a generic function to combine the elements for each key using a custom set of aggregation functions. This function also performs the reduction. It takes 3 functions as input:

- createCombiner(v): Initial transformation of the value from each single (key,value) pair.
- combinerMergeValue(v1, v2): For two initial key pairs with the same key, this is the operation between their values.

- mergeCombiners(v1, v2): For two (key,value) with the same key that are output of two combiners, this is the operation to perform. It is like the reducer operation, but if there is only one combiner, it will not be executed.

The other form of solving the problem we implemented is using the **reduceByKey()** function, that merges the values for each key using an associative and commutative reduce function. It will also perform the merging locally on each mapper before sending results to a reducer, similarly to a "combiner" in MapReduce. This function performs the combiner transparently. It only needs one function. In this case, suming up the values of the elements with the same key.

Finally, we use the function **collect()** to get the final RDD into our local machine. We also use the function **sorted()** to sort the words in descending order of ocurrence. Also, we used the function **saveAsTextFile()** to store the data into a local file.

The next images shows the first elements of the sorted RDD.

```
[(u'the', 27267), (u'and', 25340), (u'i', 19540), (u'to', 18656), (u'of', 17301), (u'a',
 14365), (u'my', 12456), (u'in', 10660), (u'you', 10597), (u'that', 10256), (u'is', 868
1), (u'with', 7706), (u'not', 7416), (u'for', 7297), (u'his', 6749), (u'your', 6644),
 (u'be', 6467), (u'he', 5884), (u'but', 5881), (u'this', 5859)]
[(u'the', 27267), (u'and', 25340), (u'i', 19540), (u'to', 18656), (u'of', 17301), (u'a',
 14365), (u'my', 12456), (u'in', 10660), (u'you', 10597), (u'that', 10256), (u'is', 868
1), (u'with', 7706), (u'not', 7416), (u'for', 7297), (u'his', 6749), (u'your', 6644),
 (u'be', 6467), (u'he', 5884), (u'but', 5881), (u'this', 5859)]
```

Figure 4: RDD after map()

1.1 Spark_WC.py

```
def flatMapper_WC(line):
# flatMap applies a function which takes each input value and returns a list.
# Each value of the list becomes a new, separate value in the output RDD

    words = line.split()   # Separate line into words
    for i in range(len(words)):
        words[i] = words[i].lower() # Process the words
    return words       # Return  list of words to be flattered.

def mapper_WC(word):
    return (word, 1)

#### EXTERNAL FUNCTIONS FOR combineByKey()
#combineByKey() Generic function to combine the elements for each key using a custom
#set of aggregation functions.
#So... this also performs the reducer :).

def createCombiner_WC(v):  #
    # Initial transformation of the value from a pair (k,v)
    return v

def combinerMergeValue_WC(v1, v2):  #
    # For two initial key pairs with the same key, this is the operation
    # between their values.
    return v1 + v2

def mergeCombiners_WC(v1, v2):  #
    # For two key pairs that are output of two combiners, this is the operation
    # to perform. It is like the reducer operation
    return v1 + v2

#### EXTERNAL FUNCTIONS FOR reduceByKey()
#Merge the values for each key using an associative and commutative reduce function.
```

```
#This will also perform the merging locally
#on each mapper before sending results to a reducer,
#similarly to a \combiner" in MapReduce.
#So... this also performs the combiner transparently.

def reducer_WC(v1, v2):  #
    # For two key pairs that are output of two combiners, this is the operation
    # to perform. It is like the reducer operation
    return v1 + v2

## Load the text into a parallelizable RDD
file_dir = "./data/cs100/lab1/shakespeare.txt"
text_file = sc.textFile(file_dir)
#print text_file.collect()[0:20]

#mapped_WC = text_file.flatMap(lambda line: line.split()) # Inline version
mapped_WC = text_file.flatMap(flatMapper_WC)           # Ext. func. version
#print mapped_WC.collect()[0:20]           # Printing of the results

#mapped_WC = mapped_WC.map(lambda word: (word, 1))  # Inline version
mapped_WC = mapped_WC.map(mapper_WC)           # Ext. func. version
#print mapped_WC.collect()[0:20]

mapped_WC.persist()  # We make it persist so that we can use it for several steps.
                     # We will use it for combineByKey() and reduceByKey()

####### SOLVE USING combineByKey()  ###############
#combined_WC = mapped_WC.combineByKey(int,
#                                     lambda v1,v2: v1 + v2,
#                                     lambda v1,v2: v1 + v2)

combined_WC = mapped_WC.combineByKey(createCombiner_WC,
                                     combinerMergeValue_WC,
                                     mergeCombiners_WC)
data_WC = combined_WC.collect() # Get the RDD localy.
# Sort by ocurrence
data_WC = sorted(data_WC,key = lambda x: x[1], reverse = True)
print data_WC[0:20]

####### SOLVE USING reduceByKey()  ###############

#reduced_WC = mapped_WC.reduceByKey(lambda x, y: x + y)
reduced_WC = mapped_WC.reduceByKey(reducer_WC)

####### SOLVE USING reduceByKey()  ###############
data_WC = reduced_WC.collect()  # Get the RDD localy.

# Sort by ocurrence
data_WC = sorted(data_WC,key = lambda x: x[1], reverse = True)
print data_WC[0:20]

# reduced_WC.saveAsTextFile("./WC.txt")
```

Exercise 8.2

Assignment :
Write a Spark job that determines if a graph has an Euler tour (all vertices have even degree) where you can assume that the graph you get is connected.

This file https://www.dropbox.com/s/usdi0wpsqm3jb7f/eulerGraphs.txt?dl=0 has 5 graphs – for each graph, the first line tells the number of nodes N and the number of edges E. The next E lines tells which

two nodes are connected by an edge. Two nodes can be connected by multiple edges.

It is fine if you split the file into 5 different files. You do not need to keep the node and edge counts in the top of the file.
Document that it works using a small example.
Solution :

In this case, we will not go as deep in the explanations as we have done in the previous exercise and we will focus on the implementation itself.
We have separated the files into 5 different ones to make the code more modulable. Our program first reads the text file using the function **sc.textFile(file_dir)** as we did before. Then we use **flatMap()** to generate the individual nodes involved in every edge and then we use **map()** to generate (key,value) pairs where every key is the node and every value is 1. The next image shows the first 5 elements of the RDD after each stage for a simple problem.

```
[u'0 1', u'0 4', u'1 2', u'1 2', u'1 5']
[u'0', u'1', u'0', u'4', u'1']
[(u'0', 1), (u'1', 1), (u'0', 1), (u'4', 1), (u'1', 1)]
```

Figure 5: First 5 elements of the RDD during the first 3 stages

Then we use **combineByKey()** to sum the number of edges of every node and compute if the node is has an odd or even degree. The next image shows the number of edges of every node

```
[(u'10', 4), (u'1', 4), (u'2', 4), (u'5', 4), (u'4', 4)]
```

Figure 6: Number of edges of every node

Now we need to count the number of edges that have an odd degree, if the number is either 0 or 2, then we have an Euler path. To do so, we perform another map reduce work in which, we map every (node, 0or1) key value into (0or1, 1) elements, so that we can count by their oddity . We just do it in an inline fashion:

```
combined_EP.map(lambda (k,v): (v,1)).reduceByKey(lambda v1,v2: v1+v2)
```

```
[(u'10', 4), (u'1', 4), (u'2', 4), (u'5', 4), (u'4', 4)]
```

Figure 7: Output of the Euler Path program

This way we know the number of odd degree nodes, it will be the value assigned to the "1" key in the RDD. So we collect the RDD, check the value and output if there is Euler Path.

```
There is an Euler Path
```

Figure 8: Output of the Euler Path program

1.2 Spark_EU.py

```
1  def flatMapper_EP(line):
2      # flatMap applies a function which takes each input value and returns a list.
3      # Each value of the list becomes a new, separate value in the output RDD
4
5      nodes_vertex = line.split()
6      if (len(nodes_vertex) == 2):
7          return nodes_vertex
8
9  def mapper_EP(node):
```

```
    return (node, 1)


#### EXTERNAL FUNCTIONS FOR combineByKey()
def createCombiner_EP(v):  #
    # Initial transformation of the value from a pair (k,v)
    return v

def combinerMergeValue_EP(v1, v2):  #
    # For two initial key pairs with the same key, this is the operation
    # between their values.
    suma = (v1 + v2)%2
    return suma

def mergeCombiners_EP(v1, v2):  #
    # For two key pairs that are output of two combiners, this is the operation
    # to perform. It is like the reducer operation
    suma = (v1 + v2)%2
    return suma


#### EXTERNAL FUNCTIONS FOR reduceByKey()

def reducer_EP(v1,v2):  #
    suma_odd = v1 + v2
    return suma_odd

## Load the text into a parallelizable RDD
file_dir = "./eulerGraphs2.txt"
text_file = sc.textFile(file_dir)
#print text_file.collect()[0:5]

mapped_WC = text_file.flatMap(flatMapper_EP)           # Ext. func. version
#print mapped_WC.collect()[0:5]           # Printing of the results

mapped_WC = mapped_WC.map(mapper_EP)           # Ext. func. version
#print mapped_WC.collect()[0:5]

######## SOLVE USING combineByKey()  ###############

combined_EP = mapped_WC.combineByKey(createCombiner_EP,
                                     combinerMergeValue_EP,
                                     mergeCombiners_EP)
#print combined_EP.collect()
#print len(combined_EP.collect())

# Put the values as keys and check whether or not there is a key = 1 with
# a number other than 0 or 1.
new_mapped = combined_EP.map(lambda (k,v): (v,1)).reduceByKey(lambda v1,v2: v1+v2)
data_EP = new_mapped.collect()
#print data_EP

## Create a dictionary with the keys and check the number of odd degree nodes.
data_dict = dict(data_EP)
if 1 in data_dict.keys():  # If there are odd degree nodes
    Nodd = data_dict[1]
    print "Number of odd vertex nodes " + str(Nodd)
    if (Nodd == 0 or Nodd == 2):
        print  "There is an Euler Path"
    else:
        print "There is no Euler Path"
```

```
else:
     print  "There is an Euler Path"
```

Exercise 8.3

Assignment :
You are given a couple of hours of raw WiFi data from my phone:
https://www.dropbox.com/s/964gq5o5bkzg7q3/wifi.data?dl=0
Compute the following things using Spark:
1. What are the 10 networks I observed the most, and how many times were they observed? Note: the bssid is unique for every network, the name (ssid) of the network is not necessarily unique.
2. What are the 10 most common wifi names? (ssid)
3. What are the 10 longest wifi names? (again, ssid)
Solution :

The way we have done these exercises is reading the Json file as a normal text file and extract the desired key value (bssid or ssid) using the string split() function finding the common pattern to identify where the information is:
Solution for 1) :

In this case, the flatMap() function looks for the string "u'bssids': u'" in every line and obtains the following 16 characters. Then it maps() every bssid as a key with value 1 and performs a counting, the same way that it was done in the previous exercise. The next image shows the obtained and mapped bssids:

```
[(u'00:24:b2:98:39:d2', 1), (u'34:21:09:12:6c:18', 1), (u'34:21:09:12:6c:1a', 1), (u'44:94:fc:56:08:fb', 1), (u'00:17:
3f:82:37:14', 1), (u'00:22:b0:b3:f2:ea', 1), (u'e8:08:8b:c9:c1:79', 1), (u'00:0b:6b:23:4f:11', 1), (u'44:94:fc:56:ce:5
e', 1), (u'5c:d9:98:63:bf:e2', 1), (u'84:1b:5e:6f:ff:b9', 1), (u'00:24:b2:98:39:d2', 1), (u'34:21:09:12:6c:18', 1),
 (u'34:21:09:12:6c:1a', 1), (u'44:94:fc:56:08:fb', 1), (u'00:17:3f:82:37:14', 1), (u'00:22:b0:b3:f2:ea', 1), (u'e8:08:
8b:c9:c1:79', 1), (u'44:94:fc:56:ce:5e', 1), (u'5c:d9:98:63:bf:e2', 1)]
```

Figure 9: Mapped bssids

The next images shows the sorted bssids in decreasing order of ocurrence for the first 20 most common networks.

```
[(u'34:21:09:12:6c:1a', 347), (u'00:24:b2:98:39:d2', 338), (u'34:21:09:12:6c:18', 324), (u'e8:08:8b:c9:c1:79', 318),
 (u'44:94:fc:56:08:fb', 315), (u'00:22:b0:b3:f2:ea', 314), (u'2c:b0:5d:ef:08:2b', 272), (u'44:94:fc:56:ce:5e', 240),
 (u'28:cf:e9:84:a1:c3', 211), (u'bc:ee:7b:55:1a:43', 210), (u'f8:1e:df:ff:a3:a8', 195), (u'bc:ee:7b:55:1a:42', 192),
 (u'28:cf:e9:84:a1:c2', 191), (u'b8:a3:86:50:cb:0c', 186), (u'e0:3f:49:ed:fc:e0', 176), (u'00:25:9c:3b:b1:72', 172),
 (u'84:1b:5e:df:5c:58', 161), (u'f8:1e:df:ff:a3:a7', 159), (u'c0:a0:bb:e8:da:c1', 139), (u'90:94:e4:83:ff:d6', 139)]
```

Figure 10: Most common bssids

1.3 Wifi1.py

```
# Check that Spark is working

def flatMapper_Wifi_getBSSID(line):
    # flatMap applies a function which takes each input value and returns a list.
    # Each value of the list becomes a new, separate value in the output RDD

    bssids = line.split("u'bssid': u'")[1][0:17]

    bssids = [bssids]
    return bssids

## Load the text into a parallelizable RDD
file_dir = "./wifi.data"
text_file = sc.textFile(file_dir)

```

```
data = text_file.collect()

mapped_Wifi = text_file.flatMap(flatMapper_Wifi_getBSSID)           # Ext. func. version
#print mapped_Wifi.collect()[0:20]           # Printing of the results

#mapped_WC = mapped_WC.map(lambda word: (word, 1))  # Inline version
mapped_Wifi = mapped_Wifi.map(lambda k: (k,1))           # Ext. func. version
print mapped_Wifi.collect()[0:20]

reduced_Wifi = mapped_Wifi.reduceByKey(lambda x, y: x + y)

data_Wifi = reduced_Wifi.collect()
data_Wifi = sorted(data_Wifi,key = lambda x: x[1], reverse = True)  # Sort by ocurrence
print data_Wifi[0:20]
```

Solution for 2) :

In this case, the flatMap() function looks for the string inbetween the strings "u'ssid': u'" and "', u'bssid':" so find the ssid. Then it maps() every ssid as a key with value 1 and performs a counting, the same way that it was done in the previous exercise. The next image shows the obtained and mapped ssid:

```
[(u'NETGEAR_1', 1), (u'AirLink126C18', 1), (u'AirLink5GHz126C18', 1), (u'Lausten_5GHz', 1), (u'frede', 1), (u'Bronx',
 1), (u'Housing People', 1), (u'AIObserver', 1), (u'Lausten', 1), (u'Bismarck', 1), (u'NETGEAR90', 1), (u'NETGEAR_1',
 1), (u'AirLink126C18', 1), (u'AirLink5GHz126C18', 1), (u'Lausten_5GHz', 1), (u'frede', 1), (u'Bronx', 1), (u'Housing
 People', 1), (u'Lausten', 1), (u'Bismarck', 1)]
```

Figure 11: Mapped ssids

The next images shows the sorted ssid in decreasing order of ocurrence for the first 20 most common networks.

```
[(u'Internet4realz', 402), (u'Kaspers Wi-Fi-netv\\xe6rk', 402), (u'AirLink5GHz126C18', 347), (u'NETGEAR_1', 338), (u'A
irLink126C18', 324), (u'Housing People', 318), (u'Lausten_5GHz', 315), (u'Bronx', 314), (u'Playhouse', 272), (u'Lauste
n', 240), (u'VoresInternet3', 198), (u'Time Capsule 5 GHz', 195), (u'FOO-NET', 186), (u'belkin54g', 173), (u'Rosenber
g', 172), (u'NETGEAR84', 161), (u'Giver kode for sex', 159), (u'THOM274V', 139), (u'Vejensnet', 139), (u'The Dark Sid
e', 127)]
```

Figure 12: Most common ssids

1.4 Wifi2.py

```
def flatMapper_Wifi_getSSID(line):
    # flatMap applies a function which takes each input value and returns a list.
    # Each value of the list becomes a new, separate value in the output RDD
    ssids =  line.split("u'ssid': u'")[1]
    ssids = ssids.split("', u'bssid':")[0]

    ssids = [ssids]
    return ssids

## Load the text into a parallelizable RDD
file_dir = "./wifi.data"
text_file = sc.textFile(file_dir)

data = text_file.collect()

mapped_Wifi = text_file.flatMap(flatMapper_Wifi_getSSID)           # Ext. func. version
#print mapped_Wifi.collect()[0:20]           # Printing of the results

#mapped_WC = mapped_WC.map(lambda word: (word, 1))  # Inline version
mapped_Wifi = mapped_Wifi.map(lambda k: (k,1))           # Ext. func. version
print mapped_Wifi.collect()[0:20]
```

```
22
23 reduced_Wifi = mapped_Wifi.reduceByKey(lambda x, y: x + y)
24
25 data_Wifi = reduced_Wifi.collect()
26 data_Wifi = sorted(data_Wifi,key = lambda x: x[1], reverse = True)  # Sort by ocurrence
27 print data_Wifi[0:20]
```

Solution for 3) :

In this exercise we obtain the ssid the same way we did before. And then, we reduce the RDD again to get rid of all duplicates, so that we end up with an RDD with unique keys. Then we do a mapping where the new (key,value) pairs will have the length of the ssid as key and the ssid as value. After that we collect them, sort them and print them.

```
[(u'NETGEAR_1', 1), (u'AirLink126C18', 1), (u'AirLink5GHz126C18', 1), (u'Lausten_5GHz', 1), (u'frede', 1), (u'Bronx',
 1), (u'Housing People', 1), (u'AIObserver', 1), (u'Lausten', 1), (u'Bismarck', 1), (u'NETGEAR90', 1), (u'NETGEAR_1',
 1), (u'AirLink126C18', 1), (u'AirLink5GHz126C18', 1), (u'Lausten_5GHz', 1), (u'frede', 1), (u'Bronx', 1), (u'Housing
 People', 1), (u'Lausten', 1), (u'Bismarck', 1)]
```

Figure 13: Mapped ssids

The next images shows the sorted ssid in decreasing order of name lenght.

```
[(31, u'HP-Print-43-Deskjet 3520 series'), (30, u'Charlotte R.s Wi-Fi-netv\\xe6rk'), (29, u'TeliaGatewayA4-B1-E9-2C-9E
-CA'), (29, u'TeliaGateway08-76-FF-84-FF-8C'), (29, u'Emil M\\xf8rkebergs Netv\\xe6rk'), (29, u'TeliaGateway9C-97-26-5
7-15-F9'), (29, u'TeliaGateway08-76-FF-46-3E-36'), (29, u'TeliaGateway9C-97-26-57-15-99'), (29, u'TeliaGateway08-76-FF
-8A-EE-32'), (29, u'TeliaGateway08-76-FF-85-04-2F'), (29, u'TeliaGateway08-76-FF-9C-E0-82'), (27, u'Charlottes Wi-Fi-n
etv\\xe6rk'), (24, u'Kaspers Wi-Fi-netv\\xe6rk'), (24, u'Pia Lynnerups Netv\\xe6rk'), (22, u'Peterh\\xe4nsel (5 GH
z)'), (22, u'DIRECT-roku-562-F4B7AB'), (21, u'UnderKirkeMinisteriet'), (21, u'FC Midtjylland (5ghz)'), (20, u'Apple Ne
twork 5e6d79'), (20, u'TLG18-Kongen-af-2100')]
```

Figure 14: Longest ssids

1.5 Wifi3.py

```
1  def flatMapper_Wifi_getSSID(line):
2      # flatMap applies a function which takes each input value and returns a list.
3      # Each value of the list becomes a new, separate value in the output RDD
4      ssids =  line.split("u'ssid': u'")[1]
5      ssids = ssids.split("', u'bssid':")[0]
6
7      ssids = [ssids]
8      return ssids
9
10 ## Load the text into a parallelizable RDD
11 file_dir = "./wifi.data"
12 text_file = sc.textFile(file_dir)
13
14 data = text_file.collect()
15
16 mapped_Wifi = text_file.flatMap(flatMapper_Wifi_getSSID)           # Ext. func. version
17 #print mapped_Wifi.collect()[0:20]           # Printing of the results
18
19 #mapped_WC = mapped_WC.map(lambda word: (word, 1))  # Inline version
20
21 mapped_Wifi = mapped_Wifi.map(lambda k: (k,1))           # Ext. func. version
22 print mapped_Wifi.collect()[0:20]
23 reduced_Wifi = mapped_Wifi.reduceByKey(lambda x, y: 1)  # Get rid of duplicates
24 reduced_Wifi = reduced_Wifi.map(lambda (k,v): (len(k),k))  # Change the key !
25
26 data_Wifi = reduced_Wifi.collect()
27 data_Wifi = sorted(data_Wifi,key = lambda x: x[0], reverse = True)  # Sort by ocurrence
28 print data_Wifi[0:20]
```

2 Week 9

Exercise 9.1

Assignment :
Make a bag-of-words encoding of the body of the articles. Train a random forest classifier to predict if an article has the topic 'earn' or not from the body-text (encoded using bag-of-words). Use 80% of the data for training data and 20% for test data. Use 50 trees ($n_{estimators}$) in your classifier.

Solution :

We included all necessary libraries.

```
import pandas as pd
import os
import json
import codecs
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction import FeatureHasher
import time
import random
```

We started with loading all json files and joining them together. We also removed the articles without body or topic and converted all text in the body to lower case. As you can see, the dimensions of the created dataframe are 10377x7.

```
#get list of files in directory data
listOfData = os.listdir("data")
completeData = list();

# load every file in data
for dataFile in listOfData:
    data = pd.read_json(codecs.open('data/' + dataFile, 'r', 'utf-8'))
    #remove articles without body or topics
    data = data[pd.notnull(data.body)]
    data = data[pd.notnull(data.topics)]
    completeData.append(data)

#join data together
completeData = pd.concat(completeData)

#convert article body to lower case
completeData.body = completeData.body.str.lower()

print('Dimension of loaded data are ' + str(completeData.shape))
```

Output:

```
Dimension of loaded data are (10377, 7)
```

Then we prepared the function for the training random forest on the data-set and the function meanScore, which is training the random forest multiple times and returning the average score for the classifier.

```
#function preparing training set and fiting randomForest
def trainRandomForest(X,Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20,\
                                                        random_state=random.randint(0, 5546540))
    clf = RandomForestClassifier(n_estimators=50)
    clf = clf.fit(X_train, Y_train)
    return (clf.score(X_test,Y_test))

#calling train method n times on different test/train set and returing mean score
def meanScore(method,X,Y,n):
    results = list()
    for _ in range(n):
        results.append(method(X,Y))
    return np.mean(results)
```

We created a bag of words representation using sklearn CountVectorizer and feature hashing using sklearn FeatureHasher. Our bag of words have smaller dimension than the one mentioned in assignment because sklearn is removing punctuation.
Difference in classifier score between bag of words and feature hashing is approximately 3%, but with feature hashing is model 2.4 times faster to train. Therefore the small loss of accuracy leads to faster training, this can be used if we need to process large data-sets. Also ,using a smarter hash function could improve overall accuracy of the model.

```
#get topics with earn as true for training
Y = completeData.topics.str.join(' ').str.contains('earn')

#bag-of-words
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(completeData.body)
print('Dimensions of bag-of-words are ' + str(X.shape))

t = time.time()
print('Bag-of-words encoding RandomForest score ' + str(meanScore(trainRandomForest,X,Y,10)) + '\n
bagTime = time.time() - t


#feature hashing
h = FeatureHasher(input_type='string',n_features=1000,non_negative=True)
X = h.transform(completeData.body)
print('Dimensions of Feature hashing  are ' + str(X.shape))

t = time.time()
print('Feature hashing  RandomForest score ' + str(meanScore(trainRandomForest,X,Y,10))  + '\n')
hashingTime = time.time() - t

print('Feature hashing is ' + str(bagTime/hashingTime) + ' faster than bag-of-words for Random for
```

Output:

```
Dimensions of bag-of-words are (10377, 28437)
Bag-of-words encoding RandomForest score 0.95243256262

Dimensions of Feature hashing  are (10377, 1000)
Feature hashing  RandomForest score 0.917100192678

Feature hashing is 2.4091595244137767 faster than bag-of-words for Random forest fitting
--------------------------------------------------------------------------------
```

Exercise 9.2

Assignment :
Implement your own MinHash algorithm.
Using the same dataset as before, hash the body of some of the articles (encoded using bag of words) using MinHash – to get the code to run faster, work with just 100 articles to begin with.
Try with different number of hash functions/permutations (for example 3, 5, 10).

Solution :

We implemented the method findSimilarArticles which is using MinHash to get indexes of similar articles.

```
#select N articles and transpose matrix
def getNumberOfArticles(X,N):
    return X[0:N-1].transpose()

#do n permutation on articles
def doPermutation(articles,n):
    for i in range(n):
        articles = articles[np.random.permutation(articles.shape[0]),:]
    return articles

#doMinHash is returning indexes of first nonzero
#element in every row after permutations
def doMinHash(X,numOfArticles,nunOfPermutation):
    articles = getNumberOfArticles(X,numOfArticles)
    articles = doPermutation(articles,nunOfPermutation)
    firstNonZeroIndex = list()
    for i in range(0,numOfArticles-1):
        row = articles[:,i].toarray()
        index = (np.where(row > 0))
        firstNonZeroIndex.append(index[0][0])
    return firstNonZeroIndex

#findSimilarArticles return array of indexes.
#Every array element contains indexes of similar articles
#Array is ordered based on number of elements
def findSimilarArticles(X,numOfArticles,nunOfPermutation):
    minHashResult = doMinHash(X,numOfArticles,nunOfPermutation)
    #join articles index with the same row index
    index_sets = [np.argwhere(i==minHashResult) for i in np.unique(minHashResult)]
    #select only similar articles , more than 1 element in array
    index_sets = list(filter(lambda x: len(x) > 1, index_sets))
    #sort based on number of elements
    index_sets.sort(key = lambda s: len(s), reverse=True)
    return index_sets

#print all similar articles on position n
def printSimilarArticles(completeData,similarArticles,n):
    articlesText = pd.Series(completeData.body).reset_index(drop=True)
    for i in range(similarArticles[n].size):
        print(articlesText[similarArticles[n][i][0]])


#print number of  similar articles on position till n
def printNumberOfArticles(similarArticles,n,nunOfPermutation):
    print('Bins containing biggest number of similar articles for ' + \
          str(nunOfPermutation) + ' permutations')
    for i in range(n):
        print(str(i) + ': ' + str(similarArticles[i].size))
```

In the following codes we are printing similar articles, for better readability we are showing only a small part of articles texts. It is easy to observe that there exists relations between texts.

We also tested different numbers of permutation for MinHash.

```
#Testing how is MinHash behaving for different number of permutations
articles = findSimilarArticles(X,10377,3)
printNumberOfArticles(articles,5,3)
articles = findSimilarArticles(X,10377,5)
printNumberOfArticles(articles,5,5)
articles = findSimilarArticles(X,10377,7)
printNumberOfArticles(articles,5,7)
```

Output:

```
---------------
Bins containing biggest number of similar articles for 3 permutations
0: 935
1: 906
2: 751
3: 599
4: 557
Bins containing biggest number of similar articles for 5 permutations
0: 1081
1: 699
2: 661
3: 617
4: 599
Bins containing biggest number of similar articles for 7 permutations
0: 2629
1: 851
2: 367
3: 338
4: 338
Bins containing biggest number of similar articles for 9 permutations
0: 3210
1: 562
2: 520
3: 455
4: 370
--------------------------------------------------------------------------------
```

```
#Find similar articles with 3 permutations
articles = find similar articles(X,100,3)
printSimilarArticles(completeData,articles,7)
```

Output:

```
---------------
 ARTICLE 0

shr loss one ct vs loss one ct
    net loss 483,518 vs loss 220,582

---------------
 ARTICLE 1

shr loss nine cts vs profit 20 cts
    net loss 257,157 vs profit 414,890
```

```
---------------
 ARTICLE 2

shr loss five cts vs profit 36 cts
    net loss 784,000 vs profit 4,793,000

-------------------------------------------------------------------------------
```

```
#Find similar articles with 7 permutations
articles = findSimilarArticles(X,10377,7)
printSimilarArticles(completeData,articles,200)
```

Output:

```
---------------
 ARTICLE 0

state-owned bank of china has bought a
three to five pct share of baii holding sa, a financial
institution registered in luxembourg, the china daily business
weekly said.
    it said the institution is 50 pct owned by arab interests
and has set up a wholly owned commercial banking branch in hong
kong but gave no more details.
 reuter

---------------
 ARTICLE 1

bank of china has taken a stake in
luxembourg-based finance company baii holdings sa, a spokesman
for baii said.
    the stake was between three and five pct but no further
details of the deal, which was announced simultaneously in
paris, london and hong kong, were immediately available.
    baii, which is 50 pct arab owned, is looking to expand its
activities in the far east and recently established a
wholly-owned merchant banking subsidiary in hong kong, the
spokesman said.
    the group had earnings of 15.4 mln dlrs in 1985.
 reuter

-------------------------------------------------------------------------------
```