Day1am lab1
http://cmj4.web.rice.edu/LDADictionaryBased.html

Data Science Boot Camp Python Activity One: Implementing LDA

In this activity, you will be writing a couple of lines of mathematical/statistical Python code. The first thing that you need to do is to fire up Anaconda. On Windows, from the Start menu, click the Anaconda Navigator desktop app. Or, on Mac, open Launchpad, then click the Anaconda Navigator icon. One Anaconda Navigator fires up, its home screen will list a number of applications that you can run. You will want to run Spyder. Launch Spyder by clicking Spyder's Launch button.

Once Spyder is running, consider the following almost-complete Python code:

```
import numpy as np
# this returns a number whose probability of occurence is p
def sampleValue (p):
	return np.flatnonzero (np.random.multinomial (1, p, 1))[0]
# there are 2000 words in the corpus
alpha = np.full (2000, .1)
# there are 100 topics
beta = np.full (100, .1)
# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the number of each word in the document
wordsInCorpus = {}
# generate each doc
for doc in range (0, 50):
	#
	# no words in this doc yet
	wordsInDoc = {}
	#
	# get the topic probabilities for this doc
	topicsInDoc = np.random.dirichlet (beta)
	#
	# generate each of the 1000 words in this document
	for word in range (0, 1000):
		#
		# select the topic and the word
		whichTopic = ???????????????
		whichWord = ???????????????
		#
		# and record the word
		wordsInDoc [whichWord] = wordsInDoc.get (whichWord, 0) + 1
		#
	# now, remember this document
	wordsInCorpus [doc] = ???????????????
```

This program implements the LDA generative process, and uses that process to create 50 documents, that have 1000 words each, where the words in each document are stored as a map from wordID to the number of times that the word appears in the corpus. Thus, wordsInCorpus[i][j] is either the number of times that the jth dictionary word appears in document i, or else it is undefined (if word j does not appear in document i.

Your task is to now to replace those question marks with appropriate code, so that you can execute the LDA generative process. To do this, copy and paste this code into the pane labeled temp.py, then replace the question marks with code, and press the "play" button (the sideways triangle that is the seventh icon from the left in the toolbar).

Note that there may be some issues with copying and pasting the above code (extra lines and formatting problems may be introduced). If you have a problem, click this link to get a text-based version of the code, and copy and paste that one instead.

One possible answer to this activity can be accessed here. If you want to know if you are getting reasonable results, at the Python prompt (in the iPython console) type wordsInCorpus[4] to display the set of words found in the 4th document in the corpus. You should get something like {0: 3, 7: 1, 9: 2, 10: 2, 12: 2,...} though the actual numbers that you see may differ. This means that word zero appears three times in document four, word seven eapprs once, and so on.

–
Recall our dictionary is (0, bad) (1, I) (2, can’t) (3, stand) (4,
COMP101), (5, to) (6, leave) (7, love) (8, beer) (9, humanities)
(10, classes)

–

```
import numpy as np

# this returns a number whose probability of occurence is p
def sampleValue (p):
        return np.flatnonzero (np.random.multinomial (1, p, 1))[0]

# there are 2000 words in the corpus
alpha = np.full (2000, .1)

# there are 100 topics
beta = np.full (100, .1)

# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the number of each word in the document
wordsInCorpus = {}

# generate each doc
for doc in range (0, 50):
        #
        # no words in this doc yet
        wordsInDoc = {}
        #
        # get the topic probabilities for this doc
        topicsInDoc = np.random.dirichlet (beta)
        #
        # generate each of the 1000 words in this document
        for word in range (0, 1000):
                #
                # select the topic and the word
                whichTopic = sampleValue (topicsInDoc)
                whichWord = sampleValue (wordsInTopic[whichTopic])
                #
                # and record the word
                wordsInDoc [whichWord] = wordsInDoc.get (whichWord, 0) + 1
                #
        # now, remember this document
        wordsInCorpus [doc] = wordsInDoc
```

day1am lab2

http://cmj4.web.rice.edu/LDAArrays.html

Data Science Boot Camp Python Activity Two: Implementing LDA With NumPy Arrays

Python is a scripting language, and it is almost always better to use arrays to implement statistical and numerical computations, in order to keep the number of loops to a minimum. Here is an alternative implementation, that makes better use of NumPy arrays:

```
import numpy as np
# there are 2000 words in the corpus
alpha = np.full (2000, .1)
# there are 100 topics
beta = np.full (100, .1)
# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the vector of words in document i
wordsInCorpus = np.zeros ((50, 2000))
# generate each doc
for doc in range (0, 50):
    #
    # get the topic probabilities for this doc
    topicsInDoc = np.random.dirichlet (beta)
    #
    # assign each of the 1000 words in this doc to a topic
    wordsToTopic = ??????????????????????
    #
    # and generate each of the 1000 words
    for topic in range (0, 100):
        wordsFromCurrentTopic = ??????????????????????
        wordsInCorpus[doc] = ??????????????????????
```

Your task is to complete the missing code. Note that in case you have copy and paste problems with the above code, you can find a text file more amenable to copy and paste here. The first missing line assigns each of the document's 1000 words to a topic. The result should be an array where entry i in the array is the number of words assigned to topic i. Now consider the last two missing lines. These lines should first use the current topic to produce a set of words, and then take the words produced by the current topic, and add them into the current document..

If you are curious as to whether your code is correct, you can type "wordsInCorpus[2,1:10]". This will print out the number of occurrences of words 1 through 9 (inclusive) in document 2. I got something like "array([1., 1., 5., 0., 1., 0., 0., 0., 0.])". This means that word one appeared once, word three five times, and so on.

–

```
import numpy as np
# there are 2000 words in the corpus
alpha = np.full (2000, .1)
# there are 100 topics
beta = np.full (100, .1)
# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the vector of words in document i
wordsInCorpus = np.zeros ((50, 2000))
# generate each doc
for doc in range (0, 50):
    #
    # get the topic probabilities for this doc
    topicsInDoc = np.random.dirichlet (beta)
    #
    # assign each of the 1000 words in this doc to a topic
    wordsToTopic = np.random.multinomial (1000, topicsInDoc)
    #
    # and generate each of the 1000 words
    for topic in range (0, 100):
        wordsFromCurrentTopic = np.random.multinomial (wordsToTopic[topic], wordsInTopic[topic])
        wordsInCorpus[doc] = np.add (wordsInCorpus[doc], wordsFromCurrentTopic)
```

–

alt. solution

```
wordsFromCurrentTopic = wordsToTopic[topic]
wordsInCorpus[doc] = np.add (wordsInCorpus[doc], np.random.multinomial (wordsToTopic[topic], wordsInTopic[topic]))
```

day1am lab3

http://cmj4.web.rice.edu/Subarrays.html

Data Science Boot Camp Python Activity Three: Subarray Operators

Consider the following version of the LDA generative process, that records not only the words in each document, but which topic produced which word:

```
import numpy as np

# there are 2000 words in the corpus

alpha = np.full (2000, .1)

# there are 100 topics

beta = np.full (100, .1)

# this gets us the probabilty of each word happening in each of the 100 topics

wordsInTopic = np.random.dirichlet (alpha, 100)

# produced [doc, topic, word] gives us the number of times that the given word was

# produced by the given topic in the given doc

produced = np.zeros ((50, 100, 2000))

# generate each doc

for doc in range (0, 50):
    #
    # get the topic probabilities for this doc
    topicsInDoc = np.random.dirichlet (beta)
    #
    # assign each of the 1000 words in this doc to a topic
    wordsToTopic = np.random.multinomial (1000, topicsInDoc)
    #
    # and generate each of the 1000 words
    for topic in range (0, 100):
        produced[doc, topic] = np.random.multinomial (wordsToTopic[topic], wordsInTopic[topic])
```

As described in the comments, produced [doc, topic, word] gives the number of times that the given word was produced by the given topic in the given doc.

Given this, here are a number of mini-tasks:

Write a line of code that computes the number of words produced by topic 17 in document 18.

Write a line of code that computes the number of words produced by topic 17 thru 45 in document 18.

Write a line of code that computes the number of words in the entire corpus.

Write a line of code that computes the number of words in the entire corpus produced by topic 17 .

Write a line of code that computes the number of words in the entire corpus produced by topic 17 or topic 23.

Write a line of code that computes the number of words in the entire corpus produced by even numbered topics.

Write a line of code that computes the number of each word produced by topic 15.

Write a line of code that computes the topic responsible for the most instances of each word in the corpus.

Write a line of code that for each topic, computes the max number of occurrences (summed over all documents) of any word that it was responsible for.

Scroll down for some possible answers.

produced[18,17,:].sum () # or produced[18,17].sum ()

produced[18,17:46].sum ()

produced.sum () # or produced[:,:,:].sum ()

produced[:,17,:].sum () # or produced[:,17].sum ()

produced[:,np.array([17,23]),:].sum ()

produced[:,np.arange(0,100,2),:].sum ()

produced[:,15,:].sum (0) # or produced.sum (0)[15]

produced.sum (0).argmax (0)

produced[:,np.arange(0,100,1),produced.sum (0).argmax (1)].sum(0) This works, though it is possible to come up with a much better solution!

Day1pm lab 4

http://cmj4.web.rice.edu/CoOccur.html

Data Science Boot Camp Python Activity Three: Computing the Number of Word Co-Occurences

Computing the number of documents in which a pair of words co-occurs is an important problem in text analytics. Here we'll examine the cost/runtime of doing this several different ways in Python.

First, run this code, which will compute the document corpus and store it using Python dictionaries.

```
# initialize data
import numpy as np
# this returns a number whose probability of occurence is p
def sampleValue (p):
    return np.flatnonzero (np.random.multinomial (1, p, 1))[0]
# there are 2000 words in the corpus
alpha = np.full (2000, .1)
# there are 100 topics
beta = np.full (100, .1)
# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the number of each word in the document
wordsInCorpus = {}
# generate each doc
for doc in range (0, 50):
    #
    # no words in this doc yet
    wordsInDoc = {}
    #
    # get the topic probabilities for this doc
    topicsInDoc = np.random.dirichlet (beta)
    #
    # generate each of the 1000 words in this document
    for word in range (0, 1000):
        #
```

```
            # select the topci and the word
            whichTopic = sampleValue (topicsInDoc)
            whichWord = sampleValue (wordsInTopic[whichTopic])
            #
            # and record the word
            wordsInDoc [whichWord] = wordsInDoc.get (whichWord, 0) + 1
            #
        # now, remember this document
        wordsInCorpus [doc] = wordsInDoc
```

Once you have done this, your goal is to complete the following code, which loops through all of the documents, and then for each document, it counts the number of times that each pair of words co-occurs. Note that when you run this code, it will print out the total execution time. Hint: when you call .get (index, default) to access an entry in a dictionary, if that index in the dictionary is not occupied, the default value is returned instead.

```
import time
start = time.time()
coOccurrences = {}
for doc in range (0, 50):
    for ?????????????:
        for ?????????????:
            coOccurrences [(wordOne, wordTwo)] = ?????????????
end = time.time()
end - start
```

Note the running time. The answer can be found here.

Day 1 AM Lab 4 - attempt 1 - loops through all of the documents, and then for each document, it counts the number of times that each pair of words co-occurs. Note that when you run this code, it will print out the total execution time. Hint: when you call .get (index, default) to access an entry in a dictionary, if that index in the dictionary is not occupied, the default value is returned instead.

```
import time
start = time.time()
coOccurrences = {}
for doc in range (0, 50):
    for wordOne in wordsInCorpus[doc]:
        for wordTwo in wordsInCorpus[doc]:
            coOccurrences [(wordOne, wordTwo)] = coOccurrences.get ((wordOne, wordTwo), 0) +
1
end = time.time()
end - start
```

Next, your task is to write a similar code, this time using the NumPy array-based implementation. First, run the array-based code, which will compute the document corpus, and store it using NumPy arrays.

```
import numpy as np
# there are 2000 words in the corpus
alpha = np.full (2000, .1)
# there are 100 topics
beta = np.full (100, .1)
# this gets us the probabilty of each word happening in each of the 100 topics
wordsInTopic = np.random.dirichlet (alpha, 100)
# wordsInCorpus[i] will give us the vector of words in document i
wordsInCorpus = np.zeros ((50, 2000))
# generate each doc
for doc in range (0, 50):
        #
        # get the topic probabilities for this doc
        topicsInDoc = np.random.dirichlet (beta)
        #
        # assign each of the 1000 words in this doc to a topic
        wordsToTopic = np.random.multinomial (1000, topicsInDoc)
        #
        # and generate each of the 1000 words
        for topic in range (0, 100):
                wordsFromCurrentTopic = np.random.multinomial (wordsToTopic[topic],
wordsInTopic[topic])
                wordsInCorpus[doc] = np.add (wordsInCorpus[doc],
wordsFromCurrentTopic)
```

We will now implement the co-occurrence analysis by looping through all of the documents, taking the outer product of each document with itself, and summing. It is important when you do this that you cap any counts at one, since if wordA appears twice in a document, and wordB appears three times, the number of documents where a co-occurrence occurred is still one, not six. You can cap the value in a NumPy array using np.clip (array, minToAllow, maxToAllow)). Here is a skeleton of the code:

```
import time
start = time.time()
coOccurrences = np.zeros ((2000, 2000))
for doc in range (0, 50):
    coOccurInThisDoc = ?????????????
    coOccurrences = ?????????????
end = time.time()
end - start
```

Note the running time. The answer can be found here.

Day 1 AM Lab 4 - attempt 2 - using the NumPy array-based implementation. First, run the array-based code, which will compute the document corpus, and store it using NumPy arrays. We will now implement the co-occurrence analysis by looping through all of the documents, taking the outer product of each document with itself, and summing. It is important when you do this that you cap any counts at one, since if wordA appears twice in a document, and wordB appears three times, the number of documents where a co-occurrence occurred is still one, not six. You can cap the value in a NumPy array using np.clip (array, minToAllow, maxToAllow)).

```
import time
start = time.time()
coOccurrences = np.zeros ((2000, 2000))
for doc in range (0, 50):
    coOccurInThisDoc = np.outer (wordsInCorpus[doc], wordsInCorpus[doc])
    coOccurrences = np.add (coOccurrences, np.clip (coOccurInThisDoc, 0, 1))
end = time.time()
end - start
```

Finally, your task is to write a one-line code that uses a matrix-multiply to compute the answer. Here is the skeleton of the code:

```
import time
start = time.time()
res = ?????????????
end = time.time()
end - start
```

Note the running time once again. Which implementation is fastest? The answer can be found here.

```
# Day 1 AM Lab 4 - attempt 3 - one-line code that uses a matrix-multiply to compute the answer.

import time
start = time.time()
res = np.dot (np.transpose (np.clip(wordsInCorpus, 0, 1)), np.clip (wordsInCorpus, 0 , 1))
end = time.time()
end - start
```

day1pm = BIG DATA listed as day2am

http://cmj4.web.rice.edu/DSDay2/GettingStarted.html

Data Science Boot Camp: Creating a Spark Cluster and Running Word Count

1) Go to the designated sign-in page to log onto Amazon's AWS website.

2) Sign in with the user name and password provided.

https://simsql.signin.aws.amazon.com/console

iam username: DataScience2019

password: DSIsTtotallySoCool

choose Region = N Virginia

3) Next, you will need to create a "key pair" that will allow you to connect securely to the cluster that you create. To do this, under "All services" and then "Compute", click EC2.

4) Click "key pairs".

5) Click "Create Key Pair".

6) Pick a key pair name that is likely unique to you (such as the name of your eldest child, or your last name, so that it is unlikely that any other people here will choose it). Type it in, and click "Create".

Statistics404

stat405

7) This should create a ".pem" file that you can download. You will subsequently use this .pem file to connect securely to a machine over the internet.

8) Now it is time to create your Spark cluster. Again click "Services" at the upper left of the dashboard and under "Analytics" click "EMR".

9) Click "Create cluster" then click "Go to advanced options".

10) Unclick all applications except for Spark 2.4.3 and Hadoop 2.8.5, which is what we'll be using. Click "Next".

11) Under "instance type", choose one "m3.xlarge" instance for your Master, and two "m3.xlarge" instances for your Core (this should be the deault). This will give you three machines in the cloud of the type "m3.xlarge" to perform your computations. If you are interested, you can find a list of all instance types at https://aws.amazon.com/ec2/instance-types/. Each m3.xlarge machine has 8 CPU cores and 15GB of RAM, along with two SSD drives. Click "Next".

12) Click "Next". Now you are on the "General Options" screen; choose a cluster name that you will remember and that is not generic. Click "Next".

stat405

13) Now you are on "Security Options"; choose the KeyPair that you created. This is important: if you do not do this, you won't be able to access your cluster.

stat405

14) Click “Create Cluster”. Now your machines are being provisioned in the cloud! You will be taken to a page that will display your cluster status. It will take a bit of time for your cluster to come online. You can check the status of your cluster by clicking the little circular update arrow at the top right.

15) One your cluster is up and running, you will want to connect to the master node so that you run Spark jobs on it. Under “Hardware”, you will see a list of “Instance Groups”… you will have two types of instance groups in your cluster… a “core” group (the workers) and the “master” group, which contains the machine that you will interact with. Click on the ID associated with the “master” group, and you will see a clickable link under “EC2 Instance Id”. This is your master machine. Click on this link, and you will be taken to the EC2 dashboard where it will give you all sorts of info about your master node (if you ever want to get back to your cluster, just click on the cube at the upper left of the console, which gets you back to the main menu; click EMR, and then choose your cluster and you will be back to your cluster once again). The thing that we are really interested in is the public IP. This will be a number such as 54.172.82.0.

16) Now that you have created your cluster, and identified the public IP of your master node, it is time to connect to the node and run a Spark job! To connect to your master node:

Mac/Linux. The following assumes that your .pem file is called MyFirstKeyPair.pem and that it is located in your working directory; replace this with the actual name and location of your file, assuming that you called your key pair something else. Type:

chmod 500 MyFirstKeyPair.pem

Now, you can connect to your master machine (replace “54.172.82.0” with the IP address of your own master machine):

ssh -i "MyFirstKeyPair.pem" hadoop@54.172.82.0

This will give you a Linux prompt; you are connected to your master node.

Windows. In Windows, we’ll assume that you are using the PuTTY suite of tools. First fire up PuTTYgen. Click "Load" and then in the file type drop-down menu, choose "all files". Then select "MyFirstKeyPair.pem" (your .pem file will have a different name, depending upon what you called your key pair). Then choose "save" and save your PRIVATE KEY file as "MyFirstKeyPair" in an appropriate directory, where you can find it (again, use the name that you chose above; PuTTYgen will add a .ppk extension to the file you are saving) and "yes" to choose to save the file without paraphrase.

stat405c.ppk

Next, fire up PuTTY. This will allow you to connect to your Amazon machine via SSH. In the left-hand side of the dialog that comes up, click "Connection" then "ssh" then "auth" and then click on "Browse" to select the private key file that you created above using PuTTYgen. Connect ,

hadoop@publicip

hadoop@18.212.99.51

17) Now, whether or not you are using Windows or Mac/Linux, you will have a Linux prompt to your master node. It is time to run a Spark job! At the Linux command prompt, type pyspark. This will open up a Python shell that is connected to your Spark cluster.

18) In the shell, copy and paste the following, simple function, that uses Spark to count the top 200 most frequently occurring words in a text file:

```
def countWords (fileName):
    textfile = sc.textFile(fileName)
    lines = textfile.flatMap(lambda line: line.split(" "))
    counts = lines.map (lambda word: (word, 1))
    aggregatedCounts = counts.reduceByKey (lambda a, b: a + b)
    return aggregatedCounts.top (200, key=lambda p : p[1])
```

Now that you've got that function all set, check out the following four files, which contain a bunch of text data that I've loaded onto Amazon's S3 cloud storage service:

https://s3.amazonaws.com/chrisjermainebucket/text/Holmes.txt

https://s3.amazonaws.com/chrisjermainebucket/text/dictionary.txt

https://s3.amazonaws.com/chrisjermainebucket/text/war.txt

https://s3.amazonaws.com/chrisjermainebucket/text/william.txt

I'll let you guess what these files contain! You can count the top 200 words in any of them by simply typing, at the command prompt, for example:

```
countWords ("s3://chrisjermainebucket/text/Holmes.txt")
```

Or, if you want to count all of them at the same time:

```
countWords ("s3://chrisjermainebucket/text/")
```

19) Type control-d to exit the shell when you are done.

20) Finally, here is a little assignment for you: Change the countWords program so that any words less than length 2 are filtered away (not counted) and so that all words are converted into lower case before counting. Hint: in Python, the length of a word w is computed using len(w), and the function that returns the lower case version of a word is w.lower (). Scroll down to see a possible answer….

```
Using username "hadoop".
Authenticating with public key "imported-openssh-key"
Last login: Mon Aug 12 21:18:08 2019

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
21 package(s) needed for security, out of 30 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2018.03 is available.

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R::::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-10-9-157-215 ~]$
[hadoop@ip-10-9-157-215 ~]$ ls
[hadoop@ip-10-9-157-215 ~]$ pwd
/home/hadoop
[hadoop@ip-10-9-157-215 ~]$ w
 21:21:10 up 18 min,  1 user,  load average: 0.09, 0.19, 0.25
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
hadoop   pts/0    205.251.150.154  21:19    6.00s  0.01s  0.00s w
[hadoop@ip-10-9-157-215 ~]$ pyspark
Python 2.7.12 (default, Nov  2 2017, 19:20:38)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/08/12 21:23:13 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to
uploading libraries under SPARK_HOME.
```

```
19/08/12 21:23:39 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0

19/08/12 21:23:39 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException

19/08/12 21:23:40 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException

Welcome to

      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.1
      /_/


Using Python version 2.7.12 (default, Nov  2 2017 19:20:38)

SparkSession available as 'spark'.

>>> def countWords (fileName):

...     textfile = sc.textFile(fileName)

...     lines = textfile.flatMap(lambda line: line.split(" "))

...     counts = lines.map (lambda word: (word, 1))

...     aggregatedCounts = counts.reduceByKey (lambda a, b: a + b)

...     return aggregatedCounts.top (200, key=lambda p : p[1])

...

>>> countWords ("s3://chrisjermainebucket/text/Holmes.txt")

[Stage 0:>                                                          (0 + 0) / 2]19/08/12 21:24:45 WARN
Errors: The following warnings have been detected: WARNING: The (sub)resource method stageData in
org.apache.spark.status.api.v1.OneStageResource contains empty path annotation.


19/08/12 21:24:45 WARN ServletHandler:

javax.servlet.ServletException: java.util.NoSuchElementException: None.get

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:489)

        at org.glassfish.jersey.servlet.WebComponent.service(WebComponent.java:427)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:388)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:341)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:228)

        at org.spark_project.jetty.servlet.ServletHolder.handle(ServletHolder.java:845)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1689)

        at org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.doFilter(AmIpFilter.java:164)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1676)

        at org.spark_project.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:581)

        at org.spark_project.jetty.server.handler.ContextHandler.doHandle(ContextHandler.java:1180)

        at org.spark_project.jetty.servlet.ServletHandler.doScope(ServletHandler.java:511)

        at org.spark_project.jetty.server.handler.ContextHandler.doScope(ContextHandler.java:1112)

        at org.spark_project.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:141)

        at org.spark_project.jetty.server.handler.gzip.GzipHandler.handle(GzipHandler.java:461)
```

```
        at
org.spark_project.jetty.server.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:21
3)

        at org.spark_project.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)

        at org.spark_project.jetty.server.Server.handle(Server.java:524)

        at org.spark_project.jetty.server.HttpChannel.handle(HttpChannel.java:319)

        at org.spark_project.jetty.server.HttpConnection.onFillable(HttpConnection.java:253)

        at
org.spark_project.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:273)

        at org.spark_project.jetty.io.FillInterest.fillable(FillInterest.java:95)

        at org.spark_project.jetty.io.SelectChannelEndPoint$2.run(SelectChannelEndPoint.java:93)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.executeProduceConsume(ExecuteProduce
Consume.java:303)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.produceConsume(ExecuteProduceConsume
.java:148)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.run(ExecuteProduceConsume.java:136)

        at org.spark_project.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:671)

        at org.spark_project.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:589)

        at java.lang.Thread.run(Thread.java:748)

Caused by: java.util.NoSuchElementException: None.get

        at scala.None$.get(Option.scala:347)

        at scala.None$.get(Option.scala:345)

        at org.apache.spark.status.api.v1.MetricHelper.submetricQuantiles(AllStagesResource.scala:313)

        at org.apache.spark.status.api.v1.AllStagesResource$$anon$1.build(AllStagesResource.scala:178)

        at
org.apache.spark.status.api.v1.AllStagesResource$.taskMetricDistributions(AllStagesResource.scala:181)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:71)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:62)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:130)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.withStage(OneStageResource.scala:97)

        at org.apache.spark.status.api.v1.OneStageResource.withStageAttempt(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.taskSummary(OneStageResource.scala:62)

        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

        at java.lang.reflect.Method.invoke(Method.java:498)

        at
org.glassfish.jersey.server.model.internal.ResourceMethodInvocationHandlerFactory$1.invoke(ResourceMeth
odInvocationHandlerFactory.java:81)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher$1.run(AbstractJavaResou
rceMethodDispatcher.java:144)
```

```
        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.invoke(AbstractJavaReso
urceMethodDispatcher.java:161)

        at
org.glassfish.jersey.server.model.internal.JavaResourceMethodDispatcherProvider$TypeOutInvoker.doDispat
ch(JavaResourceMethodDispatcherProvider.java:205)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.dispatch(AbstractJavaRe
sourceMethodDispatcher.java:99)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:389)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:347)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:102)

        at org.glassfish.jersey.server.ServerRuntime$2.run(ServerRuntime.java:326)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:271)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:267)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:315)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:297)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:267)

        at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:317)

        at org.glassfish.jersey.server.ServerRuntime.process(ServerRuntime.java:305)

        at org.glassfish.jersey.server.ApplicationHandler.handle(ApplicationHandler.java:1154)

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:473)

        ... 28 more

19/08/12 21:24:45 WARN HttpChannel:
//ip-10-9-157-215.ec2.internal:4040/api/v1/applications/application_1565643885901_0001/stages/0/0/
taskSummary?proxyapproved=true

javax.servlet.ServletException: java.util.NoSuchElementException: None.get

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:489)

        at org.glassfish.jersey.servlet.WebComponent.service(WebComponent.java:427)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:388)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:341)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:228)

        at org.spark_project.jetty.servlet.ServletHolder.handle(ServletHolder.java:845)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1689)

        at org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.doFilter(AmIpFilter.java:164)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1676)

        at org.spark_project.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:581)

        at org.spark_project.jetty.server.handler.ContextHandler.doHandle(ContextHandler.java:1180)

        at org.spark_project.jetty.servlet.ServletHandler.doScope(ServletHandler.java:511)

        at org.spark_project.jetty.server.handler.ContextHandler.doScope(ContextHandler.java:1112)

        at org.spark_project.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:141)

        at org.spark_project.jetty.server.handler.gzip.GzipHandler.handle(GzipHandler.java:461)
```

```
        at
org.spark_project.jetty.server.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:21
3)

        at org.spark_project.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)

        at org.spark_project.jetty.server.Server.handle(Server.java:524)

        at org.spark_project.jetty.server.HttpChannel.handle(HttpChannel.java:319)

        at org.spark_project.jetty.server.HttpConnection.onFillable(HttpConnection.java:253)

        at
org.spark_project.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:273)

        at org.spark_project.jetty.io.FillInterest.fillable(FillInterest.java:95)

        at org.spark_project.jetty.io.SelectChannelEndPoint$2.run(SelectChannelEndPoint.java:93)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.executeProduceConsume(ExecuteProduce
Consume.java:303)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.produceConsume(ExecuteProduceConsume
.java:148)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.run(ExecuteProduceConsume.java:136)

        at org.spark_project.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:671)

        at org.spark_project.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:589)

        at java.lang.Thread.run(Thread.java:748)

Caused by: java.util.NoSuchElementException: None.get

        at scala.None$.get(Option.scala:347)

        at scala.None$.get(Option.scala:345)

        at org.apache.spark.status.api.v1.MetricHelper.submetricQuantiles(AllStagesResource.scala:313)

        at org.apache.spark.status.api.v1.AllStagesResource$$anon$1.build(AllStagesResource.scala:178)

        at
org.apache.spark.status.api.v1.AllStagesResource$.taskMetricDistributions(AllStagesResource.scala:181)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:71)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:62)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:130)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.withStage(OneStageResource.scala:97)

        at org.apache.spark.status.api.v1.OneStageResource.withStageAttempt(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.taskSummary(OneStageResource.scala:62)

        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

        at java.lang.reflect.Method.invoke(Method.java:498)

        at
org.glassfish.jersey.server.model.internal.ResourceMethodInvocationHandlerFactory$1.invoke(ResourceMeth
odInvocationHandlerFactory.java:81)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher$1.run(AbstractJavaResou
rceMethodDispatcher.java:144)
```

```
        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.invoke(AbstractJavaReso
urceMethodDispatcher.java:161)

        at
org.glassfish.jersey.server.model.internal.JavaResourceMethodDispatcherProvider$TypeOutInvoker.doDispat
ch(JavaResourceMethodDispatcherProvider.java:205)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.dispatch(AbstractJavaRe
sourceMethodDispatcher.java:99)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:389)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:347)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:102)

        at org.glassfish.jersey.server.ServerRuntime$2.run(ServerRuntime.java:326)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:271)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:267)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:315)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:297)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:267)

        at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:317)

        at org.glassfish.jersey.server.ServerRuntime.process(ServerRuntime.java:305)

        at org.glassfish.jersey.server.ApplicationHandler.handle(ApplicationHandler.java:1154)

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:473)

        ... 28 more

19/08/12 21:24:45 WARN ServletHandler:

javax.servlet.ServletException: java.util.NoSuchElementException: None.get

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:489)

        at org.glassfish.jersey.servlet.WebComponent.service(WebComponent.java:427)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:388)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:341)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:228)

        at org.spark_project.jetty.servlet.ServletHolder.handle(ServletHolder.java:845)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1689)

        at org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.doFilter(AmIpFilter.java:164)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1676)

        at org.spark_project.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:581)

        at org.spark_project.jetty.server.handler.ContextHandler.doHandle(ContextHandler.java:1180)

        at org.spark_project.jetty.servlet.ServletHandler.doScope(ServletHandler.java:511)

        at org.spark_project.jetty.server.handler.ContextHandler.doScope(ContextHandler.java:1112)

        at org.spark_project.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:141)

        at org.spark_project.jetty.server.handler.gzip.GzipHandler.handle(GzipHandler.java:461)

        at
org.spark_project.jetty.server.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:21
3)
```

```
        at org.spark_project.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)

        at org.spark_project.jetty.server.Server.handle(Server.java:524)

        at org.spark_project.jetty.server.HttpChannel.handle(HttpChannel.java:319)

        at org.spark_project.jetty.server.HttpConnection.onFillable(HttpConnection.java:253)

        at
org.spark_project.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:273)

        at org.spark_project.jetty.io.FillInterest.fillable(FillInterest.java:95)

        at org.spark_project.jetty.io.SelectChannelEndPoint$2.run(SelectChannelEndPoint.java:93)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.executeProduceConsume(ExecuteProduce
Consume.java:303)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.produceConsume(ExecuteProduceConsume
.java:148)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.run(ExecuteProduceConsume.java:136)

        at org.spark_project.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:671)

        at org.spark_project.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:589)

        at java.lang.Thread.run(Thread.java:748)

Caused by: java.util.NoSuchElementException: None.get

        at scala.None$.get(Option.scala:347)

        at scala.None$.get(Option.scala:345)

        at org.apache.spark.status.api.v1.MetricHelper.submetricQuantiles(AllStagesResource.scala:313)

        at org.apache.spark.status.api.v1.AllStagesResource$$anon$1.build(AllStagesResource.scala:178)

        at
org.apache.spark.status.api.v1.AllStagesResource$.taskMetricDistributions(AllStagesResource.scala:181)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:71)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:62)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:130)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.withStage(OneStageResource.scala:97)

        at org.apache.spark.status.api.v1.OneStageResource.withStageAttempt(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.taskSummary(OneStageResource.scala:62)

        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

        at java.lang.reflect.Method.invoke(Method.java:498)

        at
org.glassfish.jersey.server.model.internal.ResourceMethodInvocationHandlerFactory$1.invoke(ResourceMeth
odInvocationHandlerFactory.java:81)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher$1.run(AbstractJavaResou
rceMethodDispatcher.java:144)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.invoke(AbstractJavaReso
urceMethodDispatcher.java:161)
```

```
        at
org.glassfish.jersey.server.model.internal.JavaResourceMethodDispatcherProvider$TypeOutInvoker.doDispat
ch(JavaResourceMethodDispatcherProvider.java:205)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.dispatch(AbstractJavaRe
sourceMethodDispatcher.java:99)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:389)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:347)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:102)

        at org.glassfish.jersey.server.ServerRuntime$2.run(ServerRuntime.java:326)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:271)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:267)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:315)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:297)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:267)

        at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:317)

        at org.glassfish.jersey.server.ServerRuntime.process(ServerRuntime.java:305)

        at org.glassfish.jersey.server.ApplicationHandler.handle(ApplicationHandler.java:1154)

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:473)

        ... 28 more

19/08/12 21:24:45 WARN HttpChannel:
//ip-10-9-157-215.ec2.internal:4040/api/v1/applications/application_1565643885901_0001/stages/1/0/
taskSummary?proxyapproved=true

javax.servlet.ServletException: java.util.NoSuchElementException: None.get

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:489)

        at org.glassfish.jersey.servlet.WebComponent.service(WebComponent.java:427)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:388)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:341)

        at org.glassfish.jersey.servlet.ServletContainer.service(ServletContainer.java:228)

        at org.spark_project.jetty.servlet.ServletHolder.handle(ServletHolder.java:845)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1689)

        at org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.doFilter(AmIpFilter.java:164)

        at
org.spark_project.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1676)

        at org.spark_project.jetty.servlet.ServletHandler.doHandle(ServletHandler.java:581)

        at org.spark_project.jetty.server.handler.ContextHandler.doHandle(ContextHandler.java:1180)

        at org.spark_project.jetty.servlet.ServletHandler.doScope(ServletHandler.java:511)

        at org.spark_project.jetty.server.handler.ContextHandler.doScope(ContextHandler.java:1112)

        at org.spark_project.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:141)

        at org.spark_project.jetty.server.handler.gzip.GzipHandler.handle(GzipHandler.java:461)

        at
org.spark_project.jetty.server.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:21
3)

        at org.spark_project.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:134)
```

```
        at org.spark_project.jetty.server.Server.handle(Server.java:524)

        at org.spark_project.jetty.server.HttpChannel.handle(HttpChannel.java:319)

        at org.spark_project.jetty.server.HttpConnection.onFillable(HttpConnection.java:253)

        at
org.spark_project.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:273)

        at org.spark_project.jetty.io.FillInterest.fillable(FillInterest.java:95)

        at org.spark_project.jetty.io.SelectChannelEndPoint$2.run(SelectChannelEndPoint.java:93)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.executeProduceConsume(ExecuteProduce
Consume.java:303)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.produceConsume(ExecuteProduceConsume
.java:148)

        at
org.spark_project.jetty.util.thread.strategy.ExecuteProduceConsume.run(ExecuteProduceConsume.java:136)

        at org.spark_project.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:671)

        at org.spark_project.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:589)

        at java.lang.Thread.run(Thread.java:748)

Caused by: java.util.NoSuchElementException: None.get

        at scala.None$.get(Option.scala:347)

        at scala.None$.get(Option.scala:345)

        at org.apache.spark.status.api.v1.MetricHelper.submetricQuantiles(AllStagesResource.scala:313)

        at org.apache.spark.status.api.v1.AllStagesResource$$anon$1.build(AllStagesResource.scala:178)

        at
org.apache.spark.status.api.v1.AllStagesResource$.taskMetricDistributions(AllStagesResource.scala:181)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:71)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$taskSummary$1.apply(OneStageResource.scala:62)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:130)

        at org.apache.spark.status.api.v1.OneStageResource$
$anonfun$withStageAttempt$1.apply(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.withStage(OneStageResource.scala:97)

        at org.apache.spark.status.api.v1.OneStageResource.withStageAttempt(OneStageResource.scala:126)

        at org.apache.spark.status.api.v1.OneStageResource.taskSummary(OneStageResource.scala:62)

        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

        at java.lang.reflect.Method.invoke(Method.java:498)

        at
org.glassfish.jersey.server.model.internal.ResourceMethodInvocationHandlerFactory$1.invoke(ResourceMeth
odInvocationHandlerFactory.java:81)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher$1.run(AbstractJavaResou
rceMethodDispatcher.java:144)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.invoke(AbstractJavaReso
urceMethodDispatcher.java:161)
```

```
        at
org.glassfish.jersey.server.model.internal.JavaResourceMethodDispatcherProvider$TypeOutInvoker.doDispat
ch(JavaResourceMethodDispatcherProvider.java:205)

        at
org.glassfish.jersey.server.model.internal.AbstractJavaResourceMethodDispatcher.dispatch(AbstractJavaRe
sourceMethodDispatcher.java:99)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.invoke(ResourceMethodInvoker.java:389)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:347)

        at
org.glassfish.jersey.server.model.ResourceMethodInvoker.apply(ResourceMethodInvoker.java:102)

        at org.glassfish.jersey.server.ServerRuntime$2.run(ServerRuntime.java:326)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:271)

        at org.glassfish.jersey.internal.Errors$1.call(Errors.java:267)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:315)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:297)

        at org.glassfish.jersey.internal.Errors.process(Errors.java:267)

        at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:317)

        at org.glassfish.jersey.server.ServerRuntime.process(ServerRuntime.java:305)

        at org.glassfish.jersey.server.ApplicationHandler.handle(ApplicationHandler.java:1154)

        at org.glassfish.jersey.servlet.WebComponent.serviceImpl(WebComponent.java:473)

        ... 28 more

[(u'the', 5404), (u'', 3145), (u'and', 2798), (u'of', 2720), (u'to', 2700), (u'a', 2575), (u'I', 2533),
(u'in', 1702), (u'that', 1559), (u'was', 1360), (u'his', 1096), (u'is', 1076), (u'you', 1029), (u'he',
1014), (u'it', 976), (u'my', 901), (u'have', 893), (u'with', 843), (u'had', 806), (u'as', 776),
(u'which', 753), (u'at', 739), (u'for', 697), (u'be', 612), (u'not', 598), (u'from', 485), (u'upon',
460), (u'said', 448), (u'but', 441), (u'me', 414), (u'we', 413), (u'this', 407), (u'been', 385),
(u'very', 371), (u'her', 367), (u'your', 359), (u'"I', 349), (u'were', 336), (u'by', 334), (u'on',
334), (u'an', 329), (u'all', 321), (u'so', 317), (u'are', 316), (u'would', 313), (u'she', 305), (u'It',
290), (u'no', 286), (u'one', 283), (u'could', 280), (u'has', 277), (u'there', 275), (u'The', 273),
(u'into', 272), (u'out', 272), (u'what', 264), (u'He', 264), (u'or', 260), (u'Mr.', 259), (u'little',
257), (u'when', 257), (u'him', 253), (u'who', 253), (u'will', 250), (u'up', 250), (u'some', 227),
(u'do', 217), (u'should', 207), (u'down', 204), (u'may', 201), (u'Holmes', 197), (u'our', 195),
(u'man', 193), (u'if', 189), (u'see', 184), (u'am', 181), (u'shall', 170), (u'must', 168), (u'can',
165), (u'about', 163), (u'over', 161), (u'than', 159), (u'any', 157), (u'only', 155), (u'more', 151),
(u'came', 142), (u'they', 140), (u'other', 140), (u'before', 138), (u'know', 137), (u'You', 136),
(u'think', 132), (u'two', 128), (u'Holmes,', 127), (u'us', 126), (u'did', 126), (u'"It', 124),
(u'There', 121), (u'might', 118), (u'come', 117), (u'"You', 112), (u'just', 110), (u'it.', 110),
(u'such', 110), (u'much', 107), (u'back', 106), (u'heard', 104), (u'time', 102), (u'made', 102),
(u'where', 100), (u'But', 100), (u'found', 100), (u'"And', 99), (u'how', 96), (u'Sherlock', 96),
(u'now', 95), (u'their', 95), (u'it,', 94), (u'own', 94), (u'never', 92), (u'then', 92), (u'after',
90), (u'like', 90), (u'however,', 89), (u'We', 89), (u'quite', 89), (u'most', 86), (u'through', 85),
(u'good', 85), (u'saw', 84), (u'them', 84), (u'away', 84), (u'its', 84), (u'tell', 84), (u'She', 84),
(u'took', 84), (u'And', 83), (u'me,', 83), (u'him.', 82), (u'Project', 80), (u'go', 80), (u'St.', 80),
(u'way', 79), (u'face', 79), (u'without', 79), (u'nothing', 78), (u'Holmes.', 78), (u'Miss', 77),
(u'few', 77), (u'make', 76), (u'left', 76), (u'small', 75), (u'door', 75), (u'every', 75), (u'matter',
75), (u'last', 74), (u'take', 74), (u'you,', 74), (u'me.', 74), (u'find', 74), (u'A', 73), (u'long',
73), (u'"The', 73), (u'until', 73), (u'young', 73), (u'say', 72), (u'case', 72), (u'As', 72), (u'"But',
72), (u'he,', 69), (u'these', 68), (u'Then', 68), (u'put', 67), (u'"Well,', 67), (u'first', 67),
(u'then,', 66), (u'seemed', 65), (u'round', 65), (u'once', 65), (u'him,', 64), (u'while', 64),
(u'even', 64), (u'thought', 64), (u'right', 64), (u'went', 63), (u'If', 63), (u'old', 62), (u'seen',
62), (u'ever', 61), (u'he.', 61), (u'hand', 61), (u'still', 61), (u'himself', 61), (u'three', 61),
(u'those', 60), (u'rather', 59), (u'though', 59), (u'something', 59), (u'eyes', 58), (u'"Yes,', 58)]

>>> countWords ("s3://chrisjermainebucket/text/")

[(u'', 539190), (u'the', 61883), (u'and', 42250), (u'to', 38824), (u'of', 34727), (u'a', 26450), (u'I',
25312), (u'in', 20134), (u'that', 16425), (u'his', 14963), (u'with', 13386), (u'he', 13203), (u'my',
12814), (u'you', 12629), (u'is', 12109), (u'not', 12072), (u'was', 10469), (u'for', 9927), (u'it',
9268), (u'be', 9123), (u'as', 8922), (u'have', 8067), (u'And', 7980), (u'had', 7340), (u'at', 7288),
(u'your', 7094), (u'her', 7075), (u'The', 6891), (u'this', 6889), (u'but', 6816), (u'on', 6276),
(u'him', 6065), (u'me', 5902), (u'from', 5851), (u'by', 5619), (u'all', 5567), (u'will', 5269),
```

(u'_sb._', 5247), (u'see', 5104), (u'so', 4767), (u'S;', 4690), (u'are', 4487), (u'she', 4470), (u'thou', 4270), (u'which', 4082), (u'were', 4021), (u'what', 3924), (u'do', 3847), (u'or', 3818), (u'no', 3761), (u'they', 3712), (u'an', 3658), (u'thy', 3639), (u'we', 3629), (u'would', 3624), (u'our', 3474), (u'their', 3448), (u'shall', 3426), (u'one', 3351), (u'He', 3273), (u'To', 3270), (u'But', 3251), (u'said', 3100), (u'S2;', 3065), (u'if', 3014), (u'That', 2959), (u'did', 2799), (u'more', 2699), (u'am', 2681), (u'who', 2679), (u'_v._', 2535), (u'been', 2503), (u'when', 2494), (u'good', 2421), (u'up', 2345), (u'them', 2333), (u'A', 2310), (u'PP;', 2296), (u'out', 2271), (u'like', 2229), (u'What', 2171), (u'than', 2151), (u'should', 2115), (u'know', 2108), (u'now', 2101), (u'there', 2091), (u'some', 2077), (u'S,', 2011), (u'man', 1989), (u'must', 1962), (u'You', 1936), (u'such', 1928), (u'Enter', 1926), (u'could', 1912), (u'If', 1904), (u'upon', 1903), (u"I'll", 1901), (u'make', 1893), (u'For', 1891), (u'very', 1890), (u'into', 1833), (u'It', 1830), (u'may', 1817), (u'thee', 1784), (u'S2,', 1778), (u'Lat.', 1750), (u'how', 1740), (u'My', 1739), (u'you,', 1709), (u'come', 1682), (u'As', 1679), (u'Prince', 1678), (u'In', 1663), (u'him.', 1660), (u'MD;', 1653), (u'these', 1641), (u'let', 1630), (u'hath', 1619), (u'then', 1612), (u'can', 1577), (u'love', 1573), (u'OF', 1547), (u'say', 1538), (u'only', 1523), (u'go', 1488), (u'us', 1481), (u'him,', 1475), (u'me,', 1470), (u'This', 1419), (u'time', 1419), (u'about', 1402), (u'any', 1390), (u'take', 1383), (u'_adj._', 1379), (u'most', 1373), (u'old', 1363), (u'never', 1339), (u'it.', 1309), (u'much', 1298), (u'has', 1298), (u'give', 1297), (u'made', 1269), (u'yet', 1263), (u'Pierre', 1260), (u'S3;', 1252), (u'before', 1252), (u'How', 1248), (u'where', 1239), (u'tell', 1229), (u'When', 1223), (u'other', 1218), (u'She', 1213), (u'think', 1211), (u'two', 1202), (u'own', 1202), (u'here', 1200), (u'little', 1188), (u'O', 1186), (u'_pl._,', 1156), (u'With', 1145), (u'those', 1130), (u'down', 1129), (u'men', 1118), (u'_pt.', 1115), (u'"I', 1108), (u'well', 1089), (u'came', 1063), (u"th'", 1058), (u'So', 1045), (u'great', 1043), (u'Of', 1039), (u'being', 1028), (u'too', 1025), (u'without', 1019), (u'O,', 1008), (u'sir,', 1002), (u'after', 997), (u'went', 996), (u'still', 987), (u'himself', 984), (u'See', 983), (u'me.', 982), (u'first', 973), (u's._,', 967), (u'it,', 958), (u'you.', 957), (u'thought', 953), (u'We', 941), (u'eyes', 936), (u'KING', 931), (u'look', 931), (u'PP,', 909), (u'Thou', 902), (u'Is', 902), (u'lord,', 898), (u'cannot', 894), (u'They', 886), (u'face', 886), (u'His', 881), (u'speak', 879)]

>>>

Now that you've got that function all set, check out the following four files, which contain a bunch of text data that I've loaded onto Amazon's S3 cloud storage service:

https://s3.amazonaws.com/chrisjermainebucket/text/Holmes.txt

https://s3.amazonaws.com/chrisjermainebucket/text/dictionary.txt

https://s3.amazonaws.com/chrisjermainebucket/text/war.txt

https://s3.amazonaws.com/chrisjermainebucket/text/william.txt

I'll let you guess what these files contain! You can count the top 200 words in any of them by simply typing, at the command prompt, for example:

```
countWords ("s3://chrisjermainebucket/text/Holmes.txt")
```

Or, if you want to count all of them at the same time:

```
countWords ("s3://chrisjermainebucket/text/")
```

19) Type control-d to exit the shell when you are done.

20) Finally, here is a little assignment for you: Change the countWords program so that any words less than length 2 are filtered away (not counted) and so that all words are converted into lower case before counting. Hint: in Python, the length of a word w is computed using len(w), and the function that returns the lower case version of a word is w.lower (). Scroll down to see a possible answer….

--

```
def countWords2 (fileName):
        textfile = sc.textFile(fileName)
        lines = textfile.flatMap(lambda line: line.split(" "))
        lines = lines.filter (lambda word: True if len (word) > 1 else False)
        counts = lines.map (lambda word: (word.lower (), 1))
        aggregatedCounts = counts.reduceByKey (lambda a, b: a + b)
        return aggregatedCounts.top (200, key=lambda p : p[1])
```

countWords2 ("s3://chrisjermainebucket/text/Holmes.txt")

countWords2 ("s3://chrisjermainebucket/text/")

```
>>> def countWords2 (fileName):
...     textfile = sc.textFile(fileName)
...     lines = textfile.flatMap(lambda line: line.split(" "))
...     lines = lines.filter (lambda word: True if len (word) > 1 else False)
...     counts = lines.map (lambda word: (word.lower (), 1))
...     aggregatedCounts = counts.reduceByKey (lambda a, b: a + b)
...     return aggregatedCounts.top (200, key=lambda p : p[1])
...
>>> countWords2 ("s3://chrisjermainebucket/text/Holmes.txt")
[(u'the', 5704), (u'and', 2882), (u'of', 2756), (u'to', 2719), (u'in', 1757), (u'that', 1606), (u'was',
1370), (u'he', 1278), (u'it', 1266), (u'you', 1171), (u'his', 1146), (u'is', 1082), (u'my', 955),
(u'have', 902), (u'with', 869), (u'as', 848), (u'had', 812), (u'at', 770), (u'which', 754), (u'for',
727), (u'be', 615), (u'not', 603), (u'but', 542), (u'we', 502), (u'from', 498), (u'this', 466),
(u'upon', 461), (u'said', 448), (u'me', 414), (u'there', 396), (u'she', 389), (u'been', 385), (u'your',
379), (u'her', 378), (u'very', 377), (u'on', 366), (u'by', 350), (u'"i', 349), (u'all', 339), (u'so',
337), (u'were', 337), (u'an', 335), (u'are', 320), (u'would', 317), (u'what', 312), (u'one', 308),
(u'no', 299), (u'when', 295), (u'could', 284), (u'has', 280), (u'out', 275), (u'into', 272), (u'or',
267), (u'mr.', 262), (u'who', 261), (u'little', 257), (u'will', 253), (u'him', 253), (u'if', 253),
(u'up', 251), (u'some', 238), (u'do', 235), (u'our', 210), (u'should', 209), (u'down', 205), (u'may',
204), (u'holmes', 200), (u'man', 195), (u'see', 190), (u'am', 181), (u'they', 177), (u'shall', 171),
(u'about', 170), (u'can', 169), (u'must', 169), (u'over', 162), (u'any', 162), (u'then', 160),
(u'than', 159), (u'only', 156), (u'more', 152), (u'came', 142), (u'other', 141), (u'before', 139),
(u'did', 137), (u'know', 137), (u'two', 135), (u'"it', 133), (u'think', 133), (u'holmes,', 127),
(u'us', 126), (u'come', 123), (u'might', 123), (u'"you', 120), (u'just', 119), (u'how', 115), (u'now',
115), (u'such', 114), (u'it.', 110), (u'"and', 110), (u'where', 107), (u'much', 107), (u'back', 106),
(u'heard', 104), (u'time', 102), (u'made', 102), (u'sherlock', 101), (u'found', 100), (u'however,',
97), (u'their', 96), (u'never', 95), (u'it,', 94), (u'after', 94), (u'own', 94), (u'its', 92),
(u'like', 90), (u'quite', 89), (u'most', 88), (u'nothing', 87), (u'through', 86), (u'tell', 86),
```

```
(u'good', 86), (u'"but', 85), (u'project', 84), (u'saw', 84), (u'them', 84), (u'away', 84), (u'took',
84), (u'miss', 83), (u'me,', 83), (u'him.', 82), (u'st.', 81), (u'every', 81), (u'go', 80),
(u'without', 80), (u'way', 79), (u'face', 79), (u'holmes.', 78), (u'"the', 78), (u'then,', 77),
(u'left', 77), (u'few', 77), (u'make', 76), (u'take', 76), (u'these', 75), (u'small', 75), (u'door',
75), (u'matter', 75), (u'young', 75), (u'you,', 75), (u'find', 75), (u'last', 74), (u'case', 74),
(u'until', 74), (u'me.', 74), (u'long', 73), (u'here', 72), (u'say', 72), (u'even', 71), (u'he,', 69),
(u'round', 68), (u'first', 68), (u'put', 67), (u'while', 67), (u'"well,', 67), (u'having', 67),
(u'once', 67), (u'yet', 66), (u'those', 65), (u'right', 65), (u'seemed', 65), (u'three', 65), (u'old',
64), (u'him,', 64), (u'now,', 64), (u'thought', 64), (u'let', 63), (u'went', 63), (u'"that', 63),
(u'seen', 62), (u'he.', 61), (u'ever', 61), (u'hand', 61), (u'still', 61), (u'look', 61), (u'himself',
61), (u'rather', 60), (u'give', 60), (u'though', 60), (u'something', 60), (u'get', 59), (u'great', 58),
(u'eyes', 58), (u'"yes,', 58), (u'"oh,', 58), (u'work', 58), (u'room', 58), (u'between', 58),
(u'light', 58), (u'you.', 57)]

>>> countWords2 ("s3://chrisjermainebucket/text/")

[(u'the', 69158), (u'and', 50911), (u'to', 42114), (u'of', 37313), (u'in', 21802), (u'that', 19614),
(u'he', 16476), (u'his', 15844), (u'with', 14757), (u'you', 14620), (u'my', 14553), (u'is', 13457),
(u'not', 12845), (u'for', 12499), (u'it', 11101), (u'as', 10824), (u'was', 10698), (u'but', 10074),
(u'be', 9841), (u'have', 8551), (u'this', 8334), (u'your', 8034), (u'at', 7846), (u'had', 7526),
(u'her', 7349), (u'by', 7154), (u'on', 6806), (u'from', 6366), (u'all', 6181), (u'him', 6101),
(u'what', 6097), (u'see', 6087), (u'so', 6036), (u'me', 5922), (u'will', 5742), (u'she', 5683),
(u'are', 5279), (u'_sb._', 5247), (u'thou', 5173), (u'or', 5153), (u'if', 4925), (u'which', 4903),
(u's;', 4690), (u'they', 4599), (u'we', 4570), (u'do', 4410), (u'no', 4259), (u'were', 4213), (u'thy',
4071), (u'an', 3990), (u'would', 3900), (u'our', 3899), (u'shall', 3888), (u'when', 3717), (u'one',
3706), (u'their', 3618), (u'who', 3273), (u'said', 3108), (u's2;', 3065), (u'did', 3060), (u'how',
2988), (u'good', 2942), (u'more', 2935), (u'there', 2866), (u'am', 2749), (u'now', 2676), (u'than',
2575), (u'_v._', 2535), (u'been', 2517), (u'let', 2497), (u'like', 2435), (u'then', 2406), (u'some',
2394), (u'out', 2380), (u'up', 2367), (u'them', 2333), (u'such', 2314), (u'pp;', 2297), (u'should',
2267), (u'may', 2257), (u'upon', 2232), (u'know', 2170), (u'must', 2106), (u'enter', 2075), (u'make',
2060), (u'prince', 2037), (u'man', 2015), (u'come', 2014), (u's,', 2011), (u'could', 1991), (u'very',
1965), (u'into', 1924), (u'these', 1920), (u'hath', 1907), (u"i'll", 1901), (u'go', 1884), (u'where',
1815), (u'thee', 1801), (u's2,', 1778), (u'king', 1764), (u'you,', 1763), (u'can', 1758), (u'lat.',
1750), (u'only', 1693), (u'time', 1687), (u'say', 1683), (u'him.', 1663), (u'yet', 1659), (u'md;',
1653), (u'love', 1645), (u'any', 1645), (u'here', 1644), (u'first', 1643), (u'give', 1637), (u'take',
1596), (u'most', 1582), (u'before', 1510), (u'old', 1492), (u'him,', 1483), (u'us', 1481), (u'me,',
1474), (u'about', 1464), (u'never', 1442), (u'tell', 1432), (u'it.', 1420), (u'_adj._', 1379),
(u"'tis", 1367), (u'has', 1352), (u'much', 1341), (u'made', 1338), (u'two', 1332), (u'think', 1290),
(u'sir,', 1283), (u'after', 1274), (u'pierre', 1260), (u's3;', 1252), (u'those', 1248), (u'other',
1239), (u'well', 1231), (u'little', 1206), (u'own', 1202), (u'men', 1163), (u'_pl._,', 1156), (u'nor',
1154), (u'down', 1148), (u'look', 1147), (u"th'", 1146), (u'being', 1144), (u'great', 1131),
(u'without', 1130), (u'even', 1119), (u'and,', 1117), (u'_pt.', 1115), (u'"i', 1108), (u'came', 1102),
(u'though', 1098), (u'long', 1089), (u'too', 1088), (u'still', 1070), (u'o,', 1011), (u'went', 1003),
(u'lord,', 999), (u'me.', 996), (u'himself', 994), (u'speak', 994), (u'now,', 984), (u's._,', 967),
(u'why', 963), (u'you.', 959), (u'it,', 958), (u'thought', 957), (u'lord', 949), (u'eyes', 937),
(u'cannot', 937), (u'cp.', 932), (u'come,', 920), (u'every', 919), (u'doth', 912), (u'pp,', 909),
(u'young', 904), (u'mine', 892), (u'hear', 890), (u'face', 887), (u'might', 876), (u'against', 869),
(u'french', 866), (u'princess', 864), (u'nothing', 856), (u'why,', 856), (u'put', 849), (u'same', 844),
(u'exeunt', 841), (u'art', 838), (u's.--as.', 838), (u'day', 832), (u'over', 832), (u'left', 827),
(u'many', 819), (u'heard', 813), (u'away', 808)]

>>>
```

reduce()

```
Using username "hadoop".
Authenticating with public key "imported-openssh-key"
Last login: Tue Aug 13 13:38:25 2019

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
21 package(s) needed for security, out of 30 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2018.03 is available.
[...]
[hadoop@ip-10-9-157-215 ~]$ pyspark
Python 2.7.12 (default, Nov  2 2017, 19:20:38)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/08/13 13:44:19 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to
uploading libraries under SPARK_HOME.
19/08/13 13:44:40 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.1
      /_/

Using Python version 2.7.12 (default, Nov  2 2017 19:20:38)
SparkSession available as 'spark'.
>>> myData = sc.parallelize (range(20000))
>>> myData.reduce (lambda a, b: a + b)
199990000
>>>
```

aggregate ()

```
myRdd = sc.parallelize ([('red', 9), ('blue', 7),('red', 12), ('green', 4)])
myRdd.aggregate ({}, lambda x, y: add (x, y),lambda x, y: combine (x, y))
```

Java ERROR on 2nd line

http://cmj4.web.rice.edu/DSDay2/kNNBasic.html

Data Science Boot Camp Spark Activity: Building a kNN Classifier

1) The task here is to build a kNN classifier for the 20 Newsgroups data set. That is, given a text string, we will compute the k closest documents to the query document, and then use the majority vote of the k best in order to guess which class the query document belongs to. The 20 categories in the 20 newsgroups data set correspond to the 20 different newsgroups that the posts were extracted from. They are:

alt.atheism

comp.graphics

comp.os.ms-windows.misc

comp.sys.ibm.pc.hardware

comp.sys.mac.hardware

comp.windows.x

misc.forsale

rec.autos

rec.motorcycles

rec.sport.baseball

rec.sport.hockey

sci.crypt

sci.electronics

sci.med

sci.space

soc.religion.christian

talk.politics.guns

talk.politics.mideast

talk.politics.misc

talk.religion.misc

Example: given the text string "god jesus allah" we might hope to return alt.atheism or soc.religion.christian or talk.religion.misc.

You can start by taking a look at the data set. Check out https://s3.amazonaws.com/chrisjermainebucket/comp330_A6/20_news_same_line.txt. There are 19997 lines in this file, each corresponding to a different text document.

2) Start by going to http://cmj4.web.rice.edu/DSDay2/Activity2Out.py. Take a look at this code. It is a bit intricate! It is trying to build a dictionary over the corpus through a series of RDD transformations. There is one tiny bit of code missing… now try to fill it in. The answer is in http://cmj4.web.rice.edu/DSDay2/Activity2Answer.py. You might try experimenting a bit with adding a lambda to the top operation that allows you to select the most common few dictionary words.

```
>>> import re

>>> import numpy as np

>>> # load up all of the 19997 documents in the corpus

... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")

>>>

>>> # each entry in validLines will be a line from the text file

... validLines = corpus.filter(lambda x : 'id' in x)

>>>

>>> # now we transform it into a bunch of (docID, text) pairs

... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))

>>>

>>> # now we split the text in each (docID, text) pair into a list of words

... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])

... # we have a bit of fancy regular expression stuff here to make sure that we do not

... # die on some of the documents

... regex = re.compile('[^a-zA-Z]')

>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))

>>>

>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])

... # to ("word1", 1) ("word2", 1)...

... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))

>>>

>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.

... allCounts = allWords.reduceByKey (lambda a, b: a + b)

>>>

>>> # and get the top 20,000 words in a local array

... # each entry is a ("word1", count) pair

... topWords = allCounts.top (20000, lambda x : x[1])

>>>

>>>topWords

[...]

, (u'yogi', 33), (u'enjoyment', 33), (u'stephens', 33), (u'unocal', 33), (u'defective', 33),
(u'inserted', 33), (u'assurance', 33), (u'implicitly', 33), (u'grandmother', 33), (u'frustrating', 33),
(u'dcr', 33), (u'communion', 33), (u'rental', 33), (u'submissions', 33), (u'pipeline', 33),
(u'religous', 33), (u'reliably', 33), (u'lindbergh', 33), (u'capitalist', 33), (u'affecting', 33),
(u'standpoint', 33), (u'clones', 33), (u'impacts', 33), (u'mexican', 33), (u'armstrong', 33),
(u'generous', 33), (u'jerome', 33), (u'housley', 33), (u'shore', 33), (u'rub', 33), (u'tbxn', 33),
(u'ceremonies', 33), (u'originated', 33), (u'pebbles', 33), (u'sorta', 33), (u'bottles', 33), (u'bury',
33), (u'tucson', 33), (u'ventura', 33), (u'skydive', 33), (u'pains', 33), (u'khoros', 33), (u'ceiling',
```

```
33), (u'usu', 33), (u'preferable', 33), (u'adjusting', 33), (u'mindlink', 33), (u'systematic', 33),
(u'retract', 33), (u'louisiana', 33), (u'shine', 33), (u'cyrix', 33), (u'noticeable', 33),
(u'benchmarks', 33), (u'gonzalez', 33), (u'jfc', 33), (u'origen', 33), (u'alright', 33), (u'scs', 33),
(u'override', 33), (u'mahogany', 33), (u'confined', 33), (u'goddard', 33), (u'rusty', 33), (u'tty',
33), (u'wheelie', 33), (u'revealing', 33), (u'prospects', 33), (u'lev', 33), (u'fighter', 33),
(u'beneath', 33), (u'predicts', 33), (u'migraines', 33), (u'bryce', 33), (u'batse', 33), (u'hubble',
33), (u'concussion', 33), (u'dense', 33), (u'radioactive', 33), (u'dash', 33), (u'organizing', 33),
(u'admits', 33), (u'stepped', 33), (u'fnalf', 33), (u'repository', 33), (u'encourages', 33),
(u'liberalizer', 33), (u'contributors', 33), (u'theologians', 33), (u'passer', 33), (u'twisto', 33),
(u'cheaply', 33), (u'erics', 33), (u'flowing', 33), (u'clocks', 33), (u'sodom', 33), (u'budgets', 33),
(u'correlated', 33), (u'rushed', 33), (u'caralv', 33), (u'ebrandt', 33), (u'queens', 33),
(u'credentials', 33), (u'cyl', 33), (u'assets', 33), (u'xhost', 33), (u'emory', 33), (u'insured', 33),
(u'smokers', 33), (u'mud', 33), (u'priesthood', 33), (u'eniac', 33), (u'bernstein', 33), (u'perpetual',
33), (u'attraction', 33), (u'accompanying', 33), (u'imagery', 33), (u'geographic', 33),
(u'inclination', 33), (u'spaceflight', 33), (u'summit', 33), (u'affiliation', 33), (u'prevalence', 33),
(u'smoked', 33), (u'petty', 33), (u'cols', 33), (u'inevitably', 33), (u'striking', 33), (u'inflated',
33), (u'interpolation', 33), (u'exclusion', 33), (u'bee', 33), (u'daniels', 33), (u'mithras', 33),
(u'lasted', 33), (u'curcio', 33), (u'mcl', 33), (u'jester', 33), (u'strategies', 33), (u'mccullough',
33), (u'comprehend', 33), (u'nnr', 33), (u'substitution', 33), (u'hue', 33), (u'sherman', 33),
(u'condone', 33), (u'liverpool', 33), (u'caucasian', 33), (u'promptly', 33), (u'ivy', 33), (u'contour',
33), (u'constraints', 33), (u'cmptrc', 33), (u'cattle', 33), (u'prescribed', 33), (u'absent', 33),
(u'curse', 33), (u'lover', 33), (u'xo', 33), (u'cpus', 33), (u'amino', 33), (u'savage', 33),
(u'auburn', 33), (u'withdrawal', 33), (u'secession', 33), (u'scrolls', 33), (u'asian', 33),
(u'exploding', 33), (u'ware', 33), (u'addr', 33), (u'kars', 33), (u'charon', 33), (u'asteroid', 33),
(u'hay', 33), (u'preview', 33), (u'hiring', 33), (u'toy', 33), (u'countless', 33), (u'bart', 33),
(u'isolation', 33), (u'atc', 33), (u'recieved', 33), (u'schaefer', 33), (u'compdyn', 33),
(u'pragmatic', 33), (u'stake', 33), (u'compensate', 33), (u'clicks', 33), (u'animated', 33), (u'ida',
33), (u'fdz', 33), (u'argues', 33), (u'bayonet', 33), (u'forecast', 33), (u'pinouts', 33),
(u'doctrinal', 33), (u'fantasies', 33), (u'organize', 33), (u'consisted', 33), (u'gballent', 33),
(u'rethinking', 33), (u'uab', 33), (u'fry', 32), (u'modest', 32), (u'hamer', 32), (u'weber', 32),
(u'tcora', 32), (u'roommate', 32), (u'draws', 32), (u'gnd', 32), (u'mic', 32), (u'atonement', 32),
(u'shoots', 32), (u'flammable', 32), (u'organs', 32), (u'unsupported', 32), (u'xtpointer', 32),
(u'zephyr', 32), (u'genetics', 32), (u'luriem', 32), (u'pkp', 32), (u'xxi', 32), (u'megabyte', 32),
(u'presbyterian', 32), (u'toad', 32), (u'winds', 32), (u'crd', 32), (u'shielding', 32), (u'tops', 32),
(u'hemisphere', 32), (u'teens', 32), (u'positioning', 32), (u'coded', 32), (u'uncompressed', 32),
(u'olvwm', 32), (u'asserts', 32), (u'forgiven', 32), (u'coherent', 32), (u'raytracing', 32),
(u'visits', 32), (u'focused', 32), (u'bel', 32), (u'collector', 32), (u'buyback', 32), (u'divinity',
32), (u'iaea', 32), (u'asthma', 32), (u'alignment', 32), (u'hats', 32), (u'beep', 32),
(u'unconditional', 32), (u'freshman', 32), (u'bylaws', 32), (u'calculation', 32), (u'pexlib', 32),
(u'cake', 32), (u'aviv', 32), (u'spartan', 32), (u'mucus', 32), (u'jeh', 32), (u'forgiveness', 32),
(u'jovanovic', 32), (u'luna', 32), (u'johansson', 32), (u'xf', 32), (u'ferraro', 32),
(u'understandable', 32), (u'peripherals', 32), (u'taite', 32), (u'ccsvax', 32), (u'vince', 32),
(u'eau', 32), (u'lungs', 32), (u'guise', 32), (u'unh', 32), (u'stockpiling', 32), (u'diane', 32),
(u'englishman', 32), (u'insertion', 32), (u'stud', 32), (u'joking', 32), (u'parable', 32), (u'cookie',
32), (u'exp', 32), (u'isaackuo', 32), (u'youths', 32), (u'sbradley', 32), (u'discarded', 32),
(u'spiderman', 32), (u'lustig', 32), (u'citing', 32), (u'detailing', 32), (u'malpractice', 32),
(u'defensemen', 32), (u'pls', 32), (u'profits', 32), (u'ehrlich', 32), (u'subliminal', 32),

[…]

(u'indent', 13), (u'museums', 13), (u'revisited', 13), (u'cardiac', 13), (u'kampf', 13), (u'convey',
13), (u'islander', 13), (u'membry', 13), (u'bureaucrats', 13)]

>>>

>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs

... # start by creating an RDD that has the number 0 thru 20000

... # 20000 is the number of words that will be in our dictionary

... twentyK = sc.parallelize(range(20000))

>>>

>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcommon", 2), ...

... # the number will be the spot in the dictionary used to tell us where the word is located

... # HINT: make use of topWords

... dictionary = twentyK.map (lambda x : (topWords[x][0], x))

>>>

>>> # finally, print out some of the dictionary, just for debugging

... dictionary.top (10)
```

```
[(u'zz', 6504), (u'zyxel', 14083), (u'zyeh', 18087), (u'zy', 9031), (u'zx', 4103), (u'zw', 9805),
(u'zvm', 19153), (u'zv', 3581), (u'zurich', 15972), (u'zur', 19745)]

>>>
```

3) Now check out http://cmj4.web.rice.edu/DSDay2/Activity3Out.py. This code adds a bit to the last one. There are four missing transformations. See if you can figure them out. The answer is in http://cmj4.web.rice.edu/DSDay2/Activity3Answer.py. Note that when we print out the few top items in the resulting RDD, it is not very satisfying, since we see the key, but the result is just a pyspark.resultiterable.ResultIterable object which is not very interesting.

...

```
>>> import re

>>> import numpy as np

>>>

>>> # load up all of the 19997 documents in the corpus

... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")

>>>

>>> # each entry in validLines will be a line from the text file

... validLines = corpus.filter(lambda x : 'id' in x)

>>>

>>> # now we transform it into a bunch of (docID, text) pairs

... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))

>>>

>>> # now we split the text in each (docID, text) pair into a list of words

... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])

... # we have a bit of fancy regular expression stuff here to make sure that we do not

... # die on some of the documents

... regex = re.compile('[^a-zA-Z]')

>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))

>>>

>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])

... # to ("word1", 1) ("word2", 1)...

... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))

>>>

>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.

... allCounts = allWords.reduceByKey (lambda a, b: a + b)

>>>

>>> # and get the top 20,000 words in a local array

... topWords = allCounts.top (20000, lambda x : x[1])

>>>

>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs

... # start by creating an RDD that has the number 0 thru 20000

... # 20000 is the number of words that will be in our dictionary

... twentyK = sc.parallelize(range(20000))
```

```
>>>
>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcommon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
>>>
>>> # next, we get a RDD that takes as input keyAndListOfWords.  This RDD has, for each
... # doc, (docID, ["word1", "word2", "word3", ...]).  We want to transform this to
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now link allWords/dictionary to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # and print some, to make sure they make sense
... allDictionaryWordsInEachDoc.top (20)
[('20_newsgroups/talk.religion.misc/84570', <pyspark.resultiterable.ResultIterable object at
0x7f59cc9db3d0>), ('20_newsgroups/talk.religion.misc/84569', <pyspark.resultiterable.ResultIterable
object at 0x7f59cc9dbdd0>), ('20_newsgroups/talk.religion.misc/84568',
<pyspark.resultiterable.ResultIterable object at 0x7f59cc9db8d0>),
('20_newsgroups/talk.religion.misc/84567', <pyspark.resultiterable.ResultIterable object at
0x7f59cc9db410>), ('20_newsgroups/talk.religion.misc/84566', <pyspark.resultiterable.ResultIterable
object at 0x7f59cfdd8d10>), ('20_newsgroups/talk.religion.misc/84565',
<pyspark.resultiterable.ResultIterable object at 0x7f59ce3c1b10>),
('20_newsgroups/talk.religion.misc/84564', <pyspark.resultiterable.ResultIterable object at
0x7f59cc9db910>), ('20_newsgroups/talk.religion.misc/84563', <pyspark.resultiterable.ResultIterable
object at 0x7f59ce3c1990>), ('20_newsgroups/talk.religion.misc/84562',
<pyspark.resultiterable.ResultIterable object at 0x7f59cc9db950>),
('20_newsgroups/talk.religion.misc/84560', <pyspark.resultiterable.ResultIterable object at
0x7f59cfdd8a10>), ('20_newsgroups/talk.religion.misc/84559', <pyspark.resultiterable.ResultIterable
object at 0x7f59ce3c1a10>), ('20_newsgroups/talk.religion.misc/84558',
<pyspark.resultiterable.ResultIterable object at 0x7f59cc9db450>),
('20_newsgroups/talk.religion.misc/84557', <pyspark.resultiterable.ResultIterable object at
0x7f59ce3c1ad0>), ('20_newsgroups/talk.religion.misc/84556', <pyspark.resultiterable.ResultIterable
object at 0x7f59cc9db490>), ('20_newsgroups/talk.religion.misc/84555',
<pyspark.resultiterable.ResultIterable object at 0x7f59cc9db990>),
('20_newsgroups/talk.religion.misc/84554', <pyspark.resultiterable.ResultIterable object at
0x7f59ce3c16d0>), ('20_newsgroups/talk.religion.misc/84553', <pyspark.resultiterable.ResultIterable
object at 0x7f59cc9db9d0>), ('20_newsgroups/talk.religion.misc/84552',
<pyspark.resultiterable.ResultIterable object at 0x7f59ce3c1050>),
('20_newsgroups/talk.religion.misc/84538', <pyspark.resultiterable.ResultIterable object at
0x7f59cc9db4d0>), ('20_newsgroups/talk.religion.misc/84511', <pyspark.resultiterable.ResultIterable
object at 0x7f59cfdd8b10>)]
>>>
```

4) Now do http://cmj4.web.rice.edu/DSDay2/Activity4Out.py. The answer is in http://cmj4.web.rice.edu/DSDay2/Activity4Answer.py. At this point, the entire input corpus is transformed into a bunch of nice NumPy vectors storing the bag-of-words for each document.

```
>>> import re
>>> import numpy as np
>>>
>>> # load up all of the 19997 documents in the corpus
... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")
>>>
>>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
>>>
>>> # now we transform it into a bunch of (docID, text) pairs
... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))
>>>
>>> # now we split the text in each (docID, text) pair into a list of words
... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
... # die on some of the documents
... regex = re.compile('[^a-zA-Z]')
>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
>>>
>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])
... # to ("word1", 1) ("word2", 1)...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
>>>
>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.
... allCounts = allWords.reduceByKey (lambda a, b: a + b)
>>>
>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
>>>
>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
... # start by creating an RDD that has the number 0 thru 20000
... # 20000 is the number of words that will be in our dictionary
... twentyK = sc.parallelize(range(20000))
>>>
>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcommon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
```

```
>>>
>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a bag-of-words array
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2, 1, 1, 1, ...] because 0 appears zero times, 1 appears twice, 2 appears once, etc.
... def buildArray (listOfIndices):
...         returnVal = np.zeros (20000)
...         for index in listOfIndices:
...                 returnVal[index] = returnVal[index] + 1
...         return returnVal
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-word numpy array
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # print a few of the docs
... allDocsAsNumpyArrays.top (10)
[(('20_newsgroups/talk.religion.misc/84570', '/talk.religion.misc/'), array([ 2.,  1.,  1., ...,  0.,
0.,  0.])), (('20_newsgroups/talk.religion.misc/84569', '/talk.religion.misc/'), array([ 20.,   1.,
17., ...,   0.,   0.,   0.])), (('20_newsgroups/talk.religion.misc/84568', '/talk.religion.misc/'),
array([ 12.,   3.,   5., ...,   0.,   0.,   0.])), (('20_newsgroups/talk.religion.misc/84567',
'/talk.religion.misc/'), array([ 6.,  6.,  4., ...,  0.,  0.,  0.])),
```

(('20_newsgroups/talk.religion.misc/84566', '/talk.religion.misc/'), array([2., 3., 3., ..., 0.,
0., 0.])), (('20_newsgroups/talk.religion.misc/84565', '/talk.religion.misc/'), array([11.,   6.,
10., ...,   0.,   0.,   0.])), (('20_newsgroups/talk.religion.misc/84564', '/talk.religion.misc/'),
array([14., 13.,   9., ...,   0.,   0.,   0.])), (('20_newsgroups/talk.religion.misc/84563',
'/talk.religion.misc/'), array([10., 16.,   8., ...,   0.,   0.,   0.])),
(('20_newsgroups/talk.religion.misc/84562', '/talk.religion.misc/'), array([9., 2., 4., ..., 0.,
0., 0.])), (('20_newsgroups/talk.religion.misc/84560', '/talk.religion.misc/'), array([94., 20.,
47., ...,   0.,   0.,   0.]))]

5) Finally, take a look at http://cmj4.web.rice.edu/DSDay2/Activity5.py. There is no code for you to fill in here, but this has an additional function that processes an input string and performs the kNN classification by (in parallel) searching through all of the documents that have previously been processed, and finding the k closest. Look over this new code.

```
>>> import re
>>> import numpy as np
b('>>>
 >>> # load up all of the 19997 documents in the corpus
... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")
>>>
h>>> # each entry in validLines will be a line from the text file
]... validLines = corpus.filter(lambda x : 'id' in x)
 >>>
a>>> # now we transform it into a bunch of (docID, text) pairs
 ... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" rl=')], x[x.index('">') + 2:]))
d>>>
i>>> # now we split the text in each (docID, text) pair into a list of words
n... # after this, we have a data set with (docID, ["word1", "word2", "word3", ..])
 ... # we have a bit of fancy regular expression stuff here to make sure that wedo not
=... # die on some of the documents
 x... regex = re.compile('[^a-zA-Z]')
a>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[]).lower().split()))
s>>>
c>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
u... # to ("word1", 1) ("word2", 1)...
 o... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
>>>
u>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.
  ... allCounts = allWords.reduceByKey (lambda a, b: a + b)
n >>>
n>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
>>>
#>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
... # start by creating an RDD that has the number 0 thru 20000
... # 20000 is the number of words that will be in our dictionary
... twentyK = sc.parallelize(range(20000))
>>>
>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcommon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
```

```
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
>>>
>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a bag-of-words array
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2, 1, 1, 1, ...] because 0 appears zero times, 1 appears twice, 2 appears once, etc.
... def buildArray (listOfIndices):
...         returnVal = np.zeros (20000)
...         for index in listOfIndices:
...                 returnVal[index] = returnVal[index] + 1
...         return returnVal
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-word numpy array
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # and finally, we have a function that returns the prediction for the label of a string, using a
kNN algorithm
... def getPrediction (textInput, k):
...         #
```

```
...         # push the text out into the cloud
...         myDoc = sc.parallelize (('', textInput))
...         #
...         # gives us (word, 1) pair for each word in the doc
...         wordsInThatDoc = myDoc.flatMap (lambda x : ((j, 1) for j in regex.sub(' ',
x).lower().split()))
...         #
...         # this wil give us a bunch of (word, (dictionaryPos, 1)) pairs
...         allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda x: (x[1][1],
x[1][0])).groupByKey ()
...         #
...         # and now, get the bag-of-words array for the input string
...         myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])
...         #
...         # now, we get the distance from the input text string to all database documents, using
cosine similarity
...         distances = allDocsAsNumpyArrays.map (lambda x : (x[0][1], np.dot (x[1], myArray)))
...         #
...         # get the top k distances
...         topK = distances.top (k, lambda x : x[1])
...         #
...         # and transform the top k distances into a set of (newsgroupID, 1) pairs
...         newsgroupsRepresented = sc.parallelize (topK).map (lambda x : (x[0], 1))
...         #
...         # now, for each newsgroupID, get the count of the number of times this newsgroup appeared
in the top k
...         numTimes = newsgroupsRepresented.aggregateByKey (0, lambda x1, x2: x1 + x2, lambda x1, x2:
x1 + x2)
...         #
...         # and return the best!
...         return numTimes.top (1, lambda x: x[1])
>>>
```

Then you can run it. Try getPrediction ("god jesus allah", 30) and also try getPrediction ("how many goals Vancouver score last year?",30). Do the answers make sense? Make up some other queries and try those as well.

```
>>> getPrediction ("god jesus allah", 30)
[('/soc.religion.christian/', 15)]
>>> getPrediction ("how many goals Vancouver score last year?",30)
[('/comp.graphics/', 6)]
>>> getPrediction ("Upgrade your car",30)
[('/comp.windows.x/', 9)]
>>> getPrediction ("Best French restaurant",30)
[('/talk.politics.misc/', 5)]
>>> getPrediction ("How to program in Python",30)
[('/comp.windows.x/', 6)]
>>> getPrediction ("Traffic through the Holland tunnel",30)
[('/talk.politics.misc/', 6)]
>>> getPrediction ("To each his own path must traverse",30)
[('/talk.politics.misc/', 6)]
>>> getPrediction ("Buy snowshoes",30)
[('/talk.politics.guns/', 4)]
>>> getPrediction ("how many goals Vancouver score last year?",100)
[('/talk.politics.misc/', 16)]
>>> getPrediction ("Upgrade your car",100)
[('/rec.autos/', 17)]
>>> getPrediction ("Best French restaurant",100)
[('/talk.politics.misc/', 15)]
>>>
```

http://cmj4.web.rice.edu/DSDay2/kNNTFIDF.html

Data Science Boot Camp Spark Activity: Building a kNN Classifier Using TF * IDF

1) The last classifier we built had some problems—see the poor result on the hockey query.

We will try to fix this my moving to TF-IDF. First, check out http://cmj4.web.rice.edu/DSDay2/Activity6Out.py. Take a look at this code. It is exactly the same as the Activity 5 code, but the buildArray function has had its body removed. Your task is to re-write it so that it builds a TF vector, rather than a count vector.

After you make the change, you can try to classify a few strings using the new code to see if it works better, but before you do so you might try leaving the Spark shell, re-starting it, and then re-executing your code, just to make sure that the new function is properly used. You can see my answer at http://cmj4.web.rice.edu/DSDay2/Activity6Answer.py.

```
>>> import re
>>> import numpy as np
>>>
 >>> # load up all of the 19997 documents in the corpus
... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")
v>>>
e>>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
>>>
T>>> # now we transform it into a bunch of (docID, text) pairs
 ... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" rl=')], x[x.index('">')
+ 2:]))
>>>
l>>> # now we split the text in each (docID, text) pair into a list of words
... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
do... # die on some of the documents

 ... regex = re.compile('[^a-zA-Z]')
>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
di>>>
s>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
z... # to ("word1", 1) ("word2", 1)...
)... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
>>>
>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.
p... allCounts = allWords.reduceByKey (lambda a, b: a + b)
a>>>
>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
```

```
>>>
d>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
... # start by creating an RDD that has the number 0 thru 20000
#... # 20000 is the number of words that will be in our dictionary
... twentyK = sc.parallelize(range(20000))
>>>
>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcommon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
>>>
>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a TF vector
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.
... def buildArray (listOfIndices):
...          returnVal = np.zeros (20000)
...          for index in listOfIndices:
...                   returnVal[index] = returnVal[index] + 1
...          mysum = np.sum (returnVal)
...          returnVal = np.divide (returnVal, mysum)
...          return returnVal
```

```
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-word numpy array
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # and finally, we have a function that returns the prediction for the label of a string, using a
kNN algorithm
... def getPrediction (textInput, k):
...         #
...         # push the text out into the cloud
...         myDoc = sc.parallelize (('', textInput))
...         #
...         # gives us (word, 1) pair for each word in the doc
...         wordsInThatDoc = myDoc.flatMap (lambda x : ((j, 1) for j in regex.sub(' ',
x).lower().split()))
...         #
...         # this wil give us a bunch of (word, (dictionaryPos, 1)) pairs
...         allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda x: (x[1][1],
x[1][0])).groupByKey ()
...         #
...         # and now, get the bag-of-words array for the input string
...         myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])
...         #
...         # now, we get the distance from the input text string to all database documents, using
cosine similarity
...         distances = allDocsAsNumpyArrays.map (lambda x : (x[0][1], np.dot (x[1], myArray)))
...         #
...         # get the top k distances
...         topK = distances.top (k, lambda x : x[1])
...         #
...         # and transform the top k distances into a set of (newsgroupID, 1) pairs
...         newsgroupsRepresented = sc.parallelize (topK).map (lambda x : (x[0], 1))
...         #
...         # now, for each newsgroupID, get the count of the number of times this newsgroup appeared
in the top k
...         numTimes = newsgroupsRepresented.aggregateByKey (0, lambda x1, x2: x1 + x2, lambda x1, x2:
x1 + x2)
...         #
...         # and return the best!
...         return numTimes.top (1, lambda x: x[1])
...
>>> getPrediction ("god jesus all", 30)
[('/soc.religion.christian/', 10)]
>>> getPrediction ("how many goals Vancouver score last year?",30)
```

```
[('/rec.sport.hockey/', 7)]
>>> getPrediction ("Upgrade your car",30)
[('/rec.autos/', 14)]
>>> getPrediction ("Best French restaurant",30)
[('/misc.forsale/', 13)]
>>> getPrediction ("Traffic through the Holland tunnel",30)
[('/talk.religion.misc/', 4)]
>>> getPrediction ("Buy snowshoes",30)
[('/comp.sys.mac.hardware/', 5)]
>>> getPrediction ("Best French restaurant",100)
[('/misc.forsale/', 34)]
>>> getPrediction ("Buy snowshoes",100)
[('/rec.motorcycles/', 16)]
>>>
```

2) A few more changes are needed to move from TF to TF * IDF. Check out http://cmj4.web.rice.edu/DSDay2/Activity7Out.py. This code contains a few lines that you need to change to fully move to TF * IDF. You can find an answer at http://cmj4.web.rice.edu/DSDay2/Activity7Answer.py.

Now you can try out a few queries. You will probably find that the classifier works much better now.

```
>>> import re
>>> import numpy as np
i>>>
n>>> # load up all of the 19997 documents in the corpus
o... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_lin.txt")
>>>
 >>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
h >>>
e>>> # now we transform it into a bunch of (docID, text) pairs
... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))
>>>
^>>> # now we split the text in each (docID, text) pair into a list of words
)... # after this, we have a data set with (docID, ["word1", "word2", "word3", ..])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
u... # die on some of the documents
n... regex = re.compile('[^a-zA-Z]')
>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
>>>
t>>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
l... # to ("word1", 1) ("word2", 1)...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
>>>
s>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.
... allCounts = allWords.reduceByKey (lambda a, b: a + b)
>>>
f>>> # and get the top 20,000 words in a local array
c... topWords = allCounts.top (20000, lambda x : x[1])
>>>
a>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
 ... # start by creating an RDD that has the number 0 thru 20000
m... # 20000 is the number of words that will be in our dictionary
... twentyK = sc.parallelize(range(20000))
>>>
 >>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcmmon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
```

```
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))

>>>

>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),

e... # ("word1", docID), ("word2", docId), ...

... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))

a>>>

n>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID) pairs

... allDictionaryWords = dictionary.join (allWords)


>>>

 >>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) airs

... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))

>>>

a>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos...]) pairs

n... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()

 >>>

>>> # now, extract the newsgrouID, so that on input we have a bunch of

... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we

... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs

... # The newsgroupID is the name of the newsgroup extracted from the docID... for example

... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"

... regex = re.compile('/.*?/')

>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))

>>>

>>> # this function gets a list of dictionaryPos values, and then creates a TF vector

... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the

... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.

... def buildArray (listOfIndices):

...         returnVal = np.zeros (20000)

...         for index in listOfIndices:

...                 returnVal[index] = returnVal[index] + 1

...         mysum = np.sum (returnVal)

...         returnVal = np.divide (returnVal, mysum)

...         return returnVal

...

>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs

... # and converts the dictionary positiions to a bag-of-words numpy array...

... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))

>>>
```

```
>>> # now, crete a version of allDocsAsNumpyArrays where, in the array, every entry is either zero or
... # one.  A zero means that the word does not occur, and a one means that it does.
... zeroOrOne = allDocsAsNumpyArrays.map (lambda x: (x[0], np.clip (np.multiply (x[1], 9e9), 0, 1)))
>>>
>>> # now, add up all of those arrays into a single array, where the i^th entry tells us how many
... # individual documents the i^th word in the dictionary appeared in
... dfArray = zeroOrOne.reduce (lambda x1, x2: (("", np.add (x1[1], x2[1]))))[1]
>>>
>>> # create an array of 20,000 entries, each entry with the value 19997.0
... multiplier = np.full (20000, 19997.0)
>>>
>>> # and get the version of dfArray where the i^th entry is the inverse-document frequency for the
... # i^th word in the corpus
... idfArray = np.log (np.divide (multiplier, dfArray))
>>>
>>> # and finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf vectors
... allDocsAsNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.multiply (x[1], idfArray)))
>>>
>>> # and finally, we have a function that returns the prediction for the label of a string, using a
kNN algorithm
... def getPrediction (textInput, k):
...         #
...         # push the text out into the cloud
...         myDoc = sc.parallelize (('', textInput))
...         #
...         # gives us (word, 1) pair for each word in the doc
...         wordsInThatDoc = myDoc.flatMap (lambda x : ((j, 1) for j in regex.sub(' ',
x).lower().split()))
...         #
...         # this wil give us a bunch of (word, (dictionaryPos, 1)) pairs
...         allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda x: (x[1][1],
x[1][0])).groupByKey ()
...         #
...         # and now, get tf array for the input string
...         myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])
...         #
...         # now, get the tf * idf array for the input string
...         myArray = np.multiply (myArray, idfArray)
...         #
...         # now, we get the distance from the input text string to all database documents, using
cosine similarity
...         distances = allDocsAsNumpyArrays.map (lambda x : (x[0][1], np.dot (x[1], myArray)))
...         #
...         # get the top k distances
```

```
...         topK = distances.top (k, lambda x : x[1])
...         #
...         # and transform the top k distances into a set of (newsgroupID, 1) pairs
...         newsgroupsRepresented = sc.parallelize (topK).map (lambda x : (x[0], 1))
...         #
...         # now, for each newsgroupID, get the count of the number of times this newsgroup appeared
in the top k
...         numTimes = newsgroupsRepresented.aggregateByKey (0, lambda x1, x2: x1 + x2, lambda x1, x2:
x1 + x2)
...         #
...         # and return the best!
...         return numTimes.top (1, lambda x: x[1])
>>> getPrediction ("god jesus all", 30)
[('/soc.religion.christian/', 13)]
>>> getPrediction ("how many goals Vancouver score last year?",30)
[('/rec.sport.hockey/', 23)]
>>> getPrediction ("Upgrade your car",30)
[('/comp.sys.mac.hardware/', 13)]
>>> getPrediction ("Best French restaurant",30)
[('/sci.med/', 8)]
>>> getPrediction ("Traffic through the Holland tunnel",30)
[('/sci.crypt/', 13)]
>>> getPrediction ("Buy snowshoes",30)
[('/comp.sys.mac.hardware/', 5)]
>>> getPrediction ("Best French restaurant",100)
[('/misc.forsale/', 20)]
>>> getPrediction ("Buy snowshoes",100)
[('/rec.motorcycles/', 16)]
>>>
```

http://cmj4.web.rice.edu/DSDay2/LinReg.html

Data Science Boot Camp Spark Activity: Building a Linear Regression Classifier

1) Now we'll use our old code as a basis to build a linear regression-based classifier, that will tell us whether a string is about religion or not. First, check out http://cmj4.web.rice.edu/DSDay2/Activity8Out.py. Take a look at this code. I have removed a bunch of the Activity 7 code, and I've added a couple of lines that attempt to map each 20,000-dimensional TF-IDF vector to a 1000-dimensional vector. The answer is at http://cmj4.web.rice.edu/DSDay2/Activity8Answer.py.

```
>>> import re
>>> import numpy as np
>>>
n>>> # load up all of the 19997 documents in the corpus
... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")
>>>
 >>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
>>>
e>>> # now we transform it into a bunch of (docID, text) pairs
... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))
>>>
^>>> # now we split the text in each (docID, text) pair into a list of words
... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
... # die on some of the documents
... regex = re.compile('[^a-zA-Z]')
>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
>>>
 >>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
)... # to ("word1", 1) ("word2", 1)...
v... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
 >>>
g>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.
... allCounts = allWords.reduceByKey (lambda a, b: a + b)
i>>>
>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
>>>
>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
... # start by creating an RDD that has the number 0 thru 20000
 ... # 20000 is the number of words that will be in our dictionary
```

```
... twentyK = sc.parallelize(range(20000))
a>>>
n>>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcmmon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
 >>>
d>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3",...]),
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a TF vector
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.
... def buildArray (listOfIndices):
...         returnVal = np.zeros (20000)
...         for index in listOfIndices:
...                 returnVal[index] = returnVal[index] + 1
...         mysum = np.sum (returnVal)
...         returnVal = np.divide (returnVal, mysum)
...         return returnVal
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-words numpy array...
```

```
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # now, crete a version of allDocsAsNumpyArrays where, in the array, every entry is either zero or
... # one.  A zero means that the word does not occur, and a one means that it does.
... zeroOrOne = allDocsAsNumpyArrays.map (lambda x: (x[0], np.clip (np.multiply (x[1], 9e9), 0, 1)))
>>>
>>> # now, add up all of those arrays into a single array, where the i^th entry tells us how many
... # individual documents the i^th word in the dictionary appeared in
... dfArray = zeroOrOne.reduce (lambda x1, x2: (("", np.add (x1[1], x2[1]))))[1]
>>>
>>> # create an array of 20,000 entries, each entry with the value 19997.0
... multiplier = np.full (20000, 19997.0)
>>>
>>> # and get the version of dfArray where the i^th entry is the inverse-document frequency for the
... # i^th word in the corpus
... idfArray = np.log (np.divide (multiplier, dfArray))
>>>
>>> # and finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf vectors
... allDocsAsNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.multiply (x[1], idfArray)))
>>>
>>> # create a 20,000 by 1000 matrix where each entry is sampled from a Normal (0, 1) distribution...
... # this will serve to map our 20,000 dimensional vectors down to 1000 dimensions
... mappingMatrix = np.random.randn (20000, 1000)
>>>
>>> # now, map all of our tf * idf vectors down to 1000 dimensions, using a matrix multiply...
... # this will give us an RDD consisteing of ((docID, newsgroupID), numpyArray) pairs, where
... # the array is a tf * idf vector mapped down into a lower-dimensional space
...     allDocsAsLowerDimNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.dot (x[1],
mappingMatrix)))
>>>
>>> # create a 20,000 by 1000 matrix where each entry is sampled from a Normal (0, 1) distribution...
... # this will serve to map our 20,000 dimensional vectors down to 1000 dimensions
... mappingMatrix = np.random.randn (20000, 1000)
>>>
>>> # now, map all of our tf * idf vectors down to 1000 dimensions, using a matrix multiply...
... # this will give us an RDD consisteing of ((docID, newsgroupID), numpyArray) pairs, where
... # the array is a tf * idf vector mapped down into a lower-dimensional space
... allDocsAsLowerDimNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.dot (x[1],
mappingMatrix)))
>>>
>>> allDocsAsLowerDimNumpyArrays.top (20)
[Stage 305:>                                                        (0 + 6) / 6]
[...]
```

```
>>> allDocsAsLowerDimNumpyArrays.top (1)

[(('20_newsgroups/talk.religion.misc/84570', '/talk.religion.misc/'), array([ -2.71288225e-01,
3.52662810e-01,   1.73310632e-01,

         1.22131389e-01,  -4.71705342e-01,  -1.16998458e-02,

        -1.65105088e-01,   1.63127102e-01,   1.15176930e-01,

         3.20628269e-01,  -3.39458151e-01,  -1.52779759e-01,

         2.82292822e-01,  -5.31653800e-02,   4.82525857e-01,

        -4.76218288e-01,  -2.54736856e-01,  -4.12804930e-01,

         1.53020533e-01,  -1.33464789e-01,   6.86488078e-03,

        -6.53731414e-02,   3.63186293e-01,   2.53216082e-01,

        -6.24926897e-02,  -1.70042359e-02,  -1.40334110e-01,

        -4.64434246e-02,  -4.74739722e-01,   2.39401745e-01,

         1.99032914e-01,   1.65611578e-01,  -7.38566884e-02,

         1.38024078e-01,  -1.10127465e-01,  -4.26670828e-01,

        -5.85970880e-01,   6.27243279e-04,  -2.42512911e-01,

        -2.36232728e-01,   4.40831886e-01,  -6.60973756e-02,

         4.35185546e-01,   5.12555872e-01,   8.72962579e-02,

         5.28820740e-01,  -2.72692679e-01,  -4.35091079e-01,

        -1.56216105e-01,   2.15901191e-01,  -5.61424349e-02,

         2.51835446e-01,  -1.87298482e-01,  -1.57213422e-02,

         1.15569936e-01,  -6.36090101e-02,   2.67743064e-02,

        -3.79940399e-01,  -3.65754361e-01,   3.02097718e-01,

         5.82634931e-03,  -2.76311282e-01,  -5.83649139e-02,

         1.55407378e-01,   1.01252573e-01,   1.06143235e-01,

        -3.64328308e-02,  -3.54890247e-01,  -1.48070727e-01,

        -5.59866626e-02,   1.99676273e-01,  -1.95824621e-01,

         3.69914061e-01,  -3.61585091e-01,   1.66447944e-01,

         3.02300489e-01,  -7.27516514e-02,   5.07009950e-02,

        -3.85050404e-01,   2.29441043e-01,   3.57903037e-01,

        -2.93335852e-01,   2.19837481e-01,  -4.43316877e-01,

         1.95603857e-01,  -9.60578848e-02,  -5.04247507e-01,

         1.73292209e-01,   2.39710426e-01,   3.61212348e-01,

         2.38671790e-01,  -7.44786645e-02,  -2.99742806e-01,

         8.52861952e-02,  -3.21292653e-01,  -9.05245639e-02,

        -3.74252002e-01,  -5.51201717e-02,  -1.79781267e-01,

         3.83322487e-01,   8.99050389e-02,  -7.19274813e-01,

        -3.18734612e-01,  -1.75943383e-01,  -7.68419232e-02,

         3.29014125e-01,   3.21328404e-01,  -7.23617601e-01,

         8.94011126e-02,  -2.33245862e-01,  -1.22444889e-01,

         1.01211584e-01,  -1.86766812e-01,   2.54288756e-01,

         2.40877683e-01,  -2.32544603e-02,   4.94488139e-02,

         2.87127374e-01,  -2.49237407e-02,  -2.32989532e-01,

[…]
```

2) Now we add to this code to build the inverse Gram matrix. Check out http://cmj4.web.rice.edu/DSDay2/Activity9Out.py and the answer is at http://cmj4.web.rice.edu/DSDay2/Activity9Answer.py.

```
>>> import re
>>> import numpy as np
i>>>
n>>> # load up all of the 19997 documents in the corpus
o... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_lin.txt")
=>>>
>>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
>>>
e>>> # now we transform it into a bunch of (docID, text) pairs
... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">')
+ 2:]))
>>>
^>>> # now we split the text in each (docID, text) pair into a list of words
... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
u... # die on some of the documents
r... regex = re.compile('[^a-zA-Z]')
>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
>>>
 >>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
... # to ("word1", 1) ("word2", 1)...
v... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
>>>
g>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.
... allCounts = allWords.reduceByKey (lambda a, b: a + b)
i>>>
m>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
>>>
>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
h... # start by creating an RDD that has the number 0 thru 20000
... # 20000 is the number of words that will be in our dictionary
]... twentyK = sc.parallelize(range(20000))
e>>>
 >>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcmmon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
 x1... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
```

```
e>>>
>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),
... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
>>>
>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs
... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
>>>
>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
>>>
>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a TF vector
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.
... def buildArray (listOfIndices):
...         returnVal = np.zeros (20000)
...         for index in listOfIndices:
...                 returnVal[index] = returnVal[index] + 1
...         mysum = np.sum (returnVal)
...         returnVal = np.divide (returnVal, mysum)
...         return returnVal
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-words numpy array...
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # now, crete a version of allDocsAsNumpyArrays where, in the array, every entry is either zero or
... # one.  A zero means that the word does not occur, and a one means that it does.
```

```
... zeroOrOne = allDocsAsNumpyArrays.map (lambda x: (x[0], np.clip (np.multiply (x[1], 9e9), 0, 1)))
>>>
>>> # now, add up all of those arrays into a single array, where the i^th entry tells us how many
... # individual documents the i^th word in the dictionary appeared in
... dfArray = zeroOrOne.reduce (lambda x1, x2: (("", np.add (x1[1], x2[1]))))[1]
>>>
>>> # create an array of 20,000 entries, each entry with the value 19997.0
... multiplier = np.full (20000, 19997.0)
>>>
>>> # and get the version of dfArray where the i^th entry is the inverse-document frequency for the
... # i^th word in the corpus
... idfArray = np.log (np.divide (multiplier, dfArray))
>>>
>>> # and finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf vectors
... allDocsAsNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.multiply (x[1], idfArray)))
>>>
>>> # create a 20,000 by 1000 matrix where each entry is sampled from a Normal (0, 1) distribution...
... # this will serve to map our 20,000 dimensional vectors down to 1000 dimensions
... mappingMatrix = np.random.randn (20000, 1000)
>>>
>>> # now, map all of our tf * idf vectors down to 1000 dimensions, using a matrix multiply...
... # this will give us an RDD consisteing of ((docID, newsgroupID), numpyArray) pairs, where
... # the array is a tf * idf vector mapped down into a lower-dimensional space
... allDocsAsLowerDimNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.dot (x[1],
mappingMatrix)))
>>>
>>> # and now take an outer product of each of those 1000 dimensional vectors with themselves
... allOuters = allDocsAsLowerDimNumpyArrays.map (lambda x: (x[0], np.outer (x[1], x[1])))
>>>
>>> # and aggregate all of those 1000 * 1000 matrices into a single matrix, by adding them all up...
... # this will give us the complete gram matrix
... gramMatrix = allOuters.aggregate (np.zeros ((1000, 1000)), lambda x1, x2: x1 + x2[1], lambda x1,
x2: x1 + x2)
>>>
>>> # take the inverse of the gram matrix
... invGram = np.linalg.inv (gramMatrix)
>>>
>>> invGram
array([[  5.02160836e-04,  -8.78548804e-06,  -1.26386388e-06, ...,
          5.62325147e-06,  -5.66312493e-06,  -5.99867118e-06],
       [ -8.78548804e-06,   4.97490024e-04,  -3.05611396e-06, ...,
         -3.21896904e-06,  -5.02738864e-06,  -5.45098180e-06],
       [ -1.26386388e-06,  -3.05611396e-06,   4.95537113e-04, ...,
```

```
          4.94934745e-06,  -2.22854001e-06,  -6.50050160e-06],
       ...,
       [  5.62325147e-06,  -3.21896904e-06,   4.94934745e-06, ...,
          5.15891320e-04,   5.52829505e-06,   2.26269145e-06],
       [ -5.66312493e-06,  -5.02738864e-06,  -2.22854001e-06, ...,
          5.52829505e-06,   4.83957210e-04,   5.95832377e-06],
       [ -5.99867118e-06,  -5.45098180e-06,  -6.50050160e-06, ...,
          2.26269145e-06,   5.95832377e-06,   4.77682297e-04]])
>>>
```

3) Now we add to this code to build the 1000 regression parameters. Check out http://cmj4.web.rice.edu/DSDay2/Activity10Out.py and the answer is at http://cmj4.web.rice.edu/DSDay2/Activity10Answer.py.

```
>>> import re
>>> import numpy as np
i>>>
n>>> # load up all of the 19997 documents in the corpus
o... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_lin.txt")
>>>
 >>> # each entry in validLines will be a line from the text file
... validLines = corpus.filter(lambda x : 'id' in x)
>>>
>>> # now we transform it into a bunch of (docID, text) pairs
 1... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('"rl=')], x[x.index('">')
+ 2:]))
i>>>
>>> # now we split the text in each (docID, text) pair into a list of words
... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
... # we have a bit of fancy regular expression stuff here to make sure that we do not
u... # die on some of the documents
r... regex = re.compile('[^a-zA-Z]')
he>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x]).lower().split()))
00>>>
 >>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])
)... # to ("word1", 1) ("word2", 1)...
v... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
 >>>
g>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.
... allCounts = allWords.reduceByKey (lambda a, b: a + b)
i>>>
m>>> # and get the top 20,000 words in a local array
... topWords = allCounts.top (20000, lambda x : x[1])
>>>
>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs
... # start by creating an RDD that has the number 0 thru 20000
... # 20000 is the number of words that will be in our dictionary
... twentyK = sc.parallelize(range(20000))
>>>
 >>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcmmon", 2), ...
... # the number will be the spot in the dictionary used to tell us where the word is located
1... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
```

```
>>>
r>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3",...]),
o... # ("word1", docID), ("word2", docId), ...
... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
c>>>
h>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID) pairs
I... allDictionaryWords = dictionary.join (allWords)
>>>
>>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) pairs
... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))
 >>>
r>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos...]) pairs
... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()
f>>>
r>>> # now, extract the newsgrouID, so that on input we have a bunch of
... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we
/... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, ditionaryPos3...]) pairs
... # The newsgroupID is the name of the newsgroup extracted from the docID... for example
 i... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupIDill be
"s/comp.graphics/"
... regex = re.compile('/.*?/')
 >>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))
>>>
>>> # this function gets a list of dictionaryPos values, and then creates a TF vector
... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] we would in the
... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.
... def buildArray (listOfIndices):
...         returnVal = np.zeros (20000)
...         for index in listOfIndices:
...                 returnVal[index] = returnVal[index] + 1
...         mysum = np.sum (returnVal)
...         returnVal = np.divide (returnVal, mysum)
...         return returnVal
...
>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2,
dictionaryPos3...]) pairs
... # and converts the dictionary positiions to a bag-of-words numpy array...
... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))
>>>
>>> # now, crete a version of allDocsAsNumpyArrays where, in the array, every entry is either zero or
... # one.  A zero means that the word does not occur, and a one means that it does.
```

```
... zeroOrOne = allDocsAsNumpyArrays.map (lambda x: (x[0], np.clip (np.multiply (x[1], 9e9), 0, 1)))
>>>
>>> # now, add up all of those arrays into a single array, where the i^th entry tells us how many
... # individual documents the i^th word in the dictionary appeared in
... dfArray = zeroOrOne.reduce (lambda x1, x2: (("", np.add (x1[1], x2[1]))))[1]
>>>
>>> # create an array of 20,000 entries, each entry with the value 19997.0
... multiplier = np.full (20000, 19997.0)
>>>
>>> # and get the version of dfArray where the i^th entry is the inverse-document frequency for the
... # i^th word in the corpus
... idfArray = np.log (np.divide (multiplier, dfArray))
>>>
>>> # and finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf vectors
... allDocsAsNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.multiply (x[1], idfArray)))
>>>
>>> # create a 20,000 by 1000 matrix where each entry is sampled from a Normal (0, 1) distribution...
... # this will serve to map our 20,000 dimensional vectors down to 1000 dimensions
... mappingMatrix = np.random.randn (20000, 1000)
>>>
>>> # now, map all of our tf * idf vectors down to 1000 dimensions, using a matrix multiply...
... # this will give us an RDD consisteing of ((docID, newsgroupID), numpyArray) pairs, where
... # the array is a tf * idf vector mapped down into a lower-dimensional space
... allDocsAsLowerDimNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.dot (x[1],
mappingMatrix)))
>>>
>>> # and now take an outer product of each of those 1000 dimensional vectors with themselves
... allOuters = allDocsAsLowerDimNumpyArrays.map (lambda x: (x[0], np.outer (x[1], x[1])))
>>>
>>> # and aggregate all of those 1000 * 1000 matrices into a single matrix, by adding them all up...
... # this will give us the complete gram matrix
... gramMatrix = allOuters.aggregate (np.zeros ((1000, 1000)), lambda x1, x2: x1 + x2[1], lambda x1,
x2: x1 + x2)
>>>
>>> # take the inverse of the gram matrix
... invGram = np.linalg.inv (gramMatrix)
>>>
>>> # now, go through allDocsAsNumpyArrays and multiply each of those 1000-dimensional vectors by the
... # gram matrix... allRows will have a bunch of ((docID, newsgroupID), numpyArray) pairs, where the
... # array is the mapped TF * IDF vector, multiplied by the Gram matrix
... allRows = allDocsAsLowerDimNumpyArrays.map (lambda x: (x[0], np.dot (invGram, x[1])))
>>>
>>> # and now, multiply each entry allRows by 1 if the document came from a religion-oriented
newsgroup;
```

```
... # that is, '/soc.religion.christian/' or '/alt.atheism/' or '/talk.religion.misc/', and by a -1 if
... # it did not come from a religion-oriented newsgroup
... allRowsMapped =  allRows.map (lambda x: (x[0], x[1] if (x[0][1] == '/soc.religion.christian/' or
x[0][1] == '/alt.atheism/' or x[0][1] == '/talk.religion.misc/') else np.multiply (-1, x[1])))
>>>
>>> # finally, we can compute the vector of regression parameters by simply adding up all of the
vectors
... # that we had in the allRowsMapped RDD
... regressionParams = allRowsMapped.aggregate (np.zeros (1000), lambda x1, x2: x1 + x2[1], lambda x1,
x2: x1 + x2)
>>>
>>> regressionParams
array([  5.06390471e-02,  -2.55848957e-02,   3.82407205e-03,
        -1.22914271e-02,   3.06561919e-02,   1.39391429e-03,
         1.26761553e-02,   7.03045394e-02,  -4.34761317e-02,
         1.01396385e-02,  -2.96390387e-02,   1.58804312e-02,
        -3.01202772e-02,  -2.79757417e-02,  -5.18299087e-02,
        -4.66045513e-02,  -6.72987562e-03,   4.79179433e-02,
         2.18739858e-02,   3.63860268e-02,  -8.89578648e-03,
        -1.69930694e-02,  -6.12008287e-04,   5.41071400e-03,
        -1.92593351e-02,  -3.69508750e-03,   4.02175128e-02,
         2.61456621e-02,   2.86933649e-02,   4.80362962e-02,
        -5.88994077e-02,  -3.31270699e-02,  -6.81237631e-02,
         1.73364927e-02,  -2.67908779e-02,   1.41644952e-02,
         6.29864053e-02,   1.76727920e-02,  -3.20054379e-02,
         3.30436058e-02,   8.14665791e-02,  -5.98680211e-02,
        -6.10652454e-03,   1.77575405e-02,   6.64365940e-02,
        -6.28454470e-03,   4.82383555e-02,  -2.64785185e-03,
         7.47632932e-02,   1.58813665e-02,  -1.38560616e-02,
        -1.71806878e-02,  -3.37002894e-02,  -2.74774254e-02,
         3.64647308e-02,  -2.46267873e-02,   5.54597457e-02,
        -5.74457314e-02,   2.33454195e-02,  -2.95144992e-02,
        -2.58931469e-02,  -8.28497249e-02,   3.87322479e-02,
        -4.91208843e-02,   9.96114351e-03,   6.56673523e-03,
         2.39467428e-02,  -3.50484932e-02,   3.94423036e-02,
         2.91256947e-02,  -1.16363509e-02,   1.69339904e-02,
         5.46094340e-02,  -1.07543307e-01,   4.39294391e-03,
        -3.49416728e-03,   9.34178917e-03,  -5.47588775e-02,
         3.80050332e-03,   3.24406703e-02,   9.10820625e-02,
         1.10140024e-02,  -6.75635836e-02,  -4.91876190e-03,
        -2.51009880e-02,  -5.95144748e-02,   1.54432684e-03,
         1.05142709e-02,  -4.45838545e-03,  -8.00252728e-02,
         2.17341416e-02,  -2.76324477e-02,   3.84513802e-02,
         […]
```

4) Finally, look at http://cmj4.web.rice.edu/DSDay2/Activity11.py. There is no code for you to write here, but this file include the code that will actually use the regression parameters to perform a prediction. Try a few queries. Examples: getPrediction ("god jesus allah") and getPrediction ("I have a fish on the back of my car") and getPrediction ("I have a baby on board bumper sticker on the back of my car").

```
>>> import re

>>> import numpy as np

>>>

n>>> # load up all of the 19997 documents in the corpus

... corpus = sc.textFile ("s3://chrisjermainebucket/comp330_A6/20_news_same_line.txt")

>>>

 >>> # each entry in validLines will be a line from the text file

... validLines = corpus.filter(lambda x : 'id' in x)

>>>

e>>> # now we transform it into a bunch of (docID, text) pairs

1... keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" rl=')], x[x.index('">')
+ 2:]))

>>>

^>>> # now we split the text in each (docID, text) pair into a list of words

... # after this, we have a data set with (docID, ["word1", "word2", "word3", ...])

 ... # we have a bit of fancy regular expression stuff here to make sure that wedo not

... # die on some of the documents

cr... regex = re.compile('[^a-zA-Z]')

>>> keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))

0>>>

 >>> # now get the top 20,000 words... first change (docID, ["word1", "word2", "ord3", ...])

... # to ("word1", 1) ("word2", 1)...

... allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))

 >>>

g>>> # now, count all of the words, giving us ("word1", 1433), ("word2", 3423423, etc.

... allCounts = allWords.reduceByKey (lambda a, b: a + b)

>>>

m>>> # and get the top 20,000 words in a local array

... topWords = allCounts.top (20000, lambda x : x[1])

>>>

[>>> # and we'll create a RDD that has a bunch of (word, dictNum) pairs

... # start by creating an RDD that has the number 0 thru 20000

l... # 20000 is the number of words that will be in our dictionary

... twentyK = sc.parallelize(range(20000))

>>>

 >>> # now, we transform (0), (1), (2), ... to ("mostcommonword", 1) ("nextmostcmmon", 2), ...

he... # the number will be the spot in the dictionary used to tell us where the rd is located

... dictionary = twentyK.map (lambda x : (topWords[x][0], x))
```

```
e>>>

>>> # next, we get a RDD that has, for each (docID, ["word1", "word2", "word3", ...]),

... # ("word1", docID), ("word2", docId), ...

e... allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))

>>>

>>> # and now join/link them, to get a bunch of ("word1", (dictionaryPos, docID)) pairs

 I... allDictionaryWords = dictionary.join (allWords)

>>>

 >>> # and drop the actual word itself to get a bunch of (docID, dictionaryPos) airs

... justDocAndPos = allDictionaryWords.map (lambda x: (x[1][1], x[1][0]))

>>>

r>>> # now get a bunch of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos...]) pairs

 ... allDictionaryWordsInEachDoc = justDocAndPos.groupByKey ()

>>>

r>>> # now, extract the newsgrouID, so that on input we have a bunch of

... # (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs, but on output we

... # have a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs

... # The newsgroupID is the name of the newsgroup extracted from the docID... for example

... # if the docID is "20_newsgroups/comp.graphics/37261" then the newsgroupID will be
"s/comp.graphics/"

... regex = re.compile('/.*?/')

>>> allDictionaryWordsInEachDocWithNewsgroup = allDictionaryWordsInEachDoc.map (lambda x: ((x[0],
regex.search(x[0]).group (0)), x[1]))

>>>

 >>> # this function gets a list of dictionaryPos values, and then creates a TF ector

t... # corresponding to those values... for example, if we get [3, 4, 1, 1, 2] w would in the

... # end have [0, 2/5, 1/5, 1/5, 1/5] because 0 appears zero times, 1 appears twice, 2 appears once,
etc.

... def buildArray (listOfIndices):

ch...          returnVal = np.zeros (20000)

...          for index in listOfIndices:

 ...                  returnVal[index] = returnVal[index] + 1

[...          mysum = np.sum (returnVal)

 ...          returnVal = np.divide (returnVal, mysum)

 ...          return returnVal

...

b>>> # this gets us a bunch of ((docID, newsgroupID) [dictionaryPos1, dictionaryos2,
dictionaryPos3...]) pairs

 t... # and converts the dictionary positiions to a bag-of-words numpy array...

... allDocsAsNumpyArrays = allDictionaryWordsInEachDocWithNewsgroup.map (lambda x: (x[0], buildArray
(x[1])))

p>>>

i>>> # now, crete a version of allDocsAsNumpyArrays where, in the array, every etry is either zero or

 ... # one.  A zero means that the word does not occur, and a one means that it oes.
```

```
t ... zeroOrOne = allDocsAsNumpyArrays.map (lambda x: (x[0], np.clip (np.multipl(x[1], 9e9), 0, 1)))
>>>
#>>> # now, add up all of those arrays into a single array, where the i^th entrytells us how many
... # individual documents the i^th word in the dictionary appeared in
... dfArray = zeroOrOne.reduce (lambda x1, x2: (("", np.add (x1[1], x2[1]))))[1]
>>>
>>> # create an array of 20,000 entries, each entry with the value 19997.0
... multiplier = np.full (20000, 19997.0)
>>>
>>> # and get the version of dfArray where the i^th entry is the inverse-document frequency for the
... # i^th word in the corpus
... idfArray = np.log (np.divide (multiplier, dfArray))
>>>
>>> # and finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf vectors
... allDocsAsNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.multiply (x[1], idfArray)))
>>>
>>> # create a 20,000 by 1000 matrix where each entry is sampled from a Normal (0, 1) distribution...
... # this will serve to map our 20,000 dimensional vectors down to 1000 dimensions
... mappingMatrix = np.random.randn (20000, 1000)
>>>
>>> # now, map all of our tf * idf vectors down to 1000 dimensions, using a matrix multiply...
... # this will give us an RDD consisteing of ((docID, newsgroupID), numpyArray) pairs, where
... # the array is a tf * idf vector mapped down into a lower-dimensional space
... allDocsAsLowerDimNumpyArrays = allDocsAsNumpyArrays.map (lambda x: (x[0], np.dot (x[1],
mappingMatrix)))
>>>
>>> # and now take an outer product of each of those 1000 dimensional vectors with themselves
... allOuters = allDocsAsLowerDimNumpyArrays.map (lambda x: (x[0], np.outer (x[1], x[1])))
>>>
>>> # and aggregate all of those 1000 * 1000 matrices into a single matrix, by adding them all up...
... # this will give us the complete gram matrix
... gramMatrix = allOuters.aggregate (np.zeros ((1000, 1000)), lambda x1, x2: x1 + x2[1], lambda x1,
x2: x1 + x2)
>>>
>>> # take the inverse of the gram matrix
... invGram = np.linalg.inv (gramMatrix)
>>>
>>> # now, go through allDocsAsNumpyArrays and multiply each of those 1000-dimensional vectors by the
... # gram matrix... allRows will have a bunch of ((docID, newsgroupID), numpyArray) pairs, where the
... # array is the mapped TF * IDF vector, multiplied by the Gram matrix
... allRows = allDocsAsLowerDimNumpyArrays.map (lambda x: (x[0], np.dot (invGram, x[1])))
>>>
>>> # and now, multiply each entry allRows by 1 if the document came from a religion-oriented
newsgroup;
```

```
... # that is, '/soc.religion.christian/' or '/alt.atheism/' or '/talk.religion.misc/', and by a -1 if

... # it did not come from a religion-oriented newsgroup

... allRowsMapped =  allRows.map (lambda x: (x[0], x[1] if (x[0][1] == '/soc.religion.christian/' or
x[0][1] == '/alt.atheism/' or x[0][1] == '/talk.religion.misc/') else np.multiply (-1, x[1])))

>>>

>>> # finally, we can compute the vector of regression parameters by simply adding up all of the
vectors

... # that we had in the allRowsMapped RDD

... regressionParams = allRowsMapped.aggregate (np.zeros (1000), lambda x1, x2: x1 + x2[1], lambda x1,
x2: x1 + x2)

>>>

>>> # lastly, we have a function that returns the prediction for the label of a string

... def getPrediction (textInput):

...         #

...         # push the text out into the cloud

...         myDoc = sc.parallelize (('', textInput))

...         #

...         # gives us (word, 1) pair for each word in the doc

...         wordsInThatDoc = myDoc.flatMap (lambda x : ((j, 1) for j in regex.sub(' ',
x).lower().split()))

...         #

...         # this wil give us a bunch of (word, (dictionaryPos, 1)) pairs

...         allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda x: (x[1][1],
x[1][0])).groupByKey ()

...         #

...         # and now, get tf array for the input string

...         myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])

...         #

...         # now, get the tf * idf array for the input string

...         myArray = np.multiply (myArray, idfArray)

...         #

...         # map this down to the 1000-dimensional representation

...         reducedRep = np.dot (myArray, mappingMatrix)

...         #

...         # take the dot product with the array of regression coeficients

...         result = np.dot (reducedRep, regressionParams)

...         #

...         # and get out of here!

...         return "about religion" if result > 0 else "not about religion"

...

>>>

>>> getPrediction ("god jesus allah")

'about religion'

>>> getPrediction ("how many goals Vancouver score last year?")
```

```
'not about religion'
>>> getPrediction ("I am not totally sure that I believe in the afterlife")
'not about religion'
>>> getPrediction ("I am not totally sure that I believe in swimming soon after I finish eating")
'not about religion'
>>> getPrediction ("I have a fish on the back of my car")
'not about religion'
>>> getPrediction ("I have a baby on board bumper sticker on the back of my car")
'not about religion'
>>> getPrediction ("Traffic through the Holland tunnel")
'not about religion'
>>> getPrediction ("Buy snowshoes")
'not about religion'
>>> getPrediction ("Best French restaurant")
'about religion'
>>>
```