In [1]:
```
from pyspark.sql import functions as F
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
5	application_1635792162170_0006	pyspark	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

In [2]:
```
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])
```

In [3]:
```
countWords("s3://chrisjermainebucket/text/Holmes.txt")
```

```
[('the', 5404), ('', 3145), ('and', 2798), ('of', 2720), ('to', 2700),
('a', 2575), ('I', 2533), ('in', 1702), ('that', 1559), ('was', 1360),
('his', 1096), ('is', 1076), ('you', 1029), ('he', 1014), ('it', 976),
('my', 901), ('have', 893), ('with', 843), ('had', 806), ('as', 776), ('
which', 753), ('at', 739), ('for', 697), ('be', 612), ('not', 598), ('fr
om', 485), ('upon', 460), ('said', 448), ('but', 441), ('me', 414), ('we
', 413), ('this', 407), ('been', 385), ('very', 371), ('her', 367), ('yo
ur', 359), ('"I', 349), ('were', 336), ('by', 334), ('on', 334), ('an',
329), ('all', 321), ('so', 317), ('are', 316), ('would', 313), ('she', 3
05), ('It', 290), ('no', 286), ('one', 283), ('could', 280), ('has', 27
7), ('there', 275), ('The', 273), ('into', 272), ('out', 272), ('He', 26
4), ('what', 264), ('or', 260), ('Mr.', 259), ('when', 257), ('little',
257), ('him', 253), ('who', 253), ('up', 250), ('will', 250), ('some', 2
27), ('do', 217), ('should', 207), ('down', 204), ('may', 201), ('Holmes
', 197), ('our', 195), ('man', 193), ('if', 189), ('see', 184), ('am', 1
81), ('shall', 170), ('must', 168), ('can', 165), ('about', 163), ('over
', 161), ('than', 159), ('any', 157), ('only', 155), ('more', 151), ('ca
me', 142), ('other', 140), ('they', 140), ('before', 138), ('know', 13
7), ('You', 136), ('think', 132), ('two', 128), ('Holmes,', 127), ('us',
126), ('did', 126), ('"It', 124), ('There', 121), ('might', 118), ('come
', 117), ('"You', 112), ('it.', 110), ('just', 110), ('such', 110), ('mu
ch', 107), ('back', 106), ('heard', 104), ('time', 102), ('made', 102),
('But', 100), ('where', 100), ('found', 100), ('"And', 99), ('Sherlock',
96), ('how', 96), ('now', 95), ('their', 95), ('it,', 94), ('own', 94),
('never', 92), ('then', 92), ('like', 90), ('after', 90), ('however,', 8
9), ('quite', 89), ('We', 89), ('most', 86), ('good', 85), ('through', 8
5), ('took', 84), ('tell', 84), ('them', 84), ('away', 84), ('She', 84),
('saw', 84), ('its', 84), ('And', 83), ('me,', 83), ('him.', 82), ('S
t.', 80), ('go', 80), ('Project', 80), ('way', 79), ('without', 79), ('f
ace', 79), ('Holmes.', 78), ('nothing', 78), ('Miss', 77), ('few', 77),
('make', 76), ('left', 76), ('matter', 75), ('every', 75), ('small', 7
5), ('door', 75), ('take', 74), ('last', 74), ('me.', 74), ('you,', 74),
('find', 74), ('until', 73), ('long', 73), ('young', 73), ('A', 73), ('"
The', 73), ('say', 72), ('case', 72), ('As', 72), ('"But', 72), ('he,',
69), ('these', 68), ('Then', 68), ('put', 67), ('first', 67), ('"Well,',
67), ('then,', 66), ('once', 65), ('seemed', 65), ('round', 65), ('thoug
ht', 64), ('right', 64), ('even', 64), ('him,', 64), ('while', 64), ('If
', 63), ('went', 63), ('seen', 62), ('old', 62), ('ever', 61), ('three',
61), ('himself', 61), ('he.', 61), ('hand', 61), ('still', 61), ('those
', 60), ('rather', 59), ('though', 59), ('something', 59), ('"Oh,', 58),
```

Task2

In [4]:

```
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_

# each entry in validLines will be a line from the text file (that has a
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(

# 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
# die on some of the documents
regex = re.compile('[^a-zA-Z]')
keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ',

# now get the top 20,000 words... first change (docID, ["word1", "word2"
# 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", 3423
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])
```

In [5]:

```
# 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) ("nextmo
# the number will be the spot in the dictionary used to tell us where th
# HINT: make use of topWords in the lambda that you supply
dictionary = twentyK.map(lambda x: (topWords[x][0], x))

# finally, print out some of the dictionary, just for debugging
dictionary.top(10)
# print(dictionary)
```

```
[('zz', 6504), ('zyxel', 13837), ('zyeh', 18665), ('zy', 8957), ('zx', 4
107), ('zw', 9699), ('zvm', 18871), ('zv', 3578), ('zurich', 15571), ('z
uma', 3634)]
```

Assignment 4

Task 1:

- First, get an RDD encoding your dictionary, where the RDD has a bunch of (word, posInDictionary) pairs.
- Next, create a second RDD that effectively has a bunch of (word, docID) pairs, where the word occurs in the given document (you get this just like the code from lab, where you flatMap the document corpus)

In [6]:

```
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_

# each entry in validLines will be a line from the text file (that has a
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(

# 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
# die on some of the documents
regex = re.compile('[^a-zA-Z]')
keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ',

# now get the top 20,000 words... first change (docID, ["word1", "word2"
# to ("word1", 1) ("word2", 1)...
word_docID_unmapped = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for
word_docID_unmapped_local = word_docID_unmapped.collect()
```

- Now, a lot of those words in the documents won't actually appear in the dictionary. But if you join the two RDDs, you'll have a bunch of (word, (docID, posInDictionary)) pairs, where the given document has the given word at the given position in the dictionary.

In [7]:

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

```
[('zz', 6504), ('zyxel', 13837), ('zyeh', 18665), ('zy', 8957), ('zx', 4
107), ('zw', 9699), ('zvm', 18871), ('zv', 3578), ('zurich', 15571), ('z
uma', 3634)]
```

In [8]:

```
word_docID_unmapped.top(10)
```

```
[('zzzzzzt', '20_newsgroups/rec.sport.baseball/104569'), ('zzzzzz', '20_
newsgroups/rec.sport.hockey/53841'), ('zzzzzz', '20_newsgroups/rec.spor
t.baseball/105004'), ('zzzzzz', '20_newsgroups/rec.sport.baseball/105002
'), ('zzzzzz', '20_newsgroups/rec.sport.baseball/104795'), ('zzzzzz', '2
0_newsgroups/rec.sport.baseball/104540'), ('zzzzzz', '20_newsgroups/rec.
motorcycles/105113'), ('zzzzzz', '20_newsgroups/rec.motorcycles/104730
'), ('zzzz', '20_newsgroups/comp.sys.ibm.pc.hardware/60262'), ('zzzz', '
20_newsgroups/comp.sys.ibm.pc.hardware/60262')]
```

In [9]:

```
word_docID_unmapped = word_docID_unmapped.join(dictionary)
```

In [10...

```
word_docID_unmapped.top(10)
```

```
[('zz', ('20_newsgroups/talk.politics.guns/54380', 6504)), ('zz', ('20_n
ewsgroups/talk.politics.guns/54380', 6504)), ('zz', ('20_newsgroups/tal
k.politics.guns/54380', 6504)), ('zz', ('20_newsgroups/sci.med/59185', 6
504)), ('zz', ('20_newsgroups/sci.crypt/15545', 6504)), ('zz', ('20_news
groups/sci.crypt/15545', 6504)), ('zz', ('20_newsgroups/rec.sport.baseba
ll/105004', 6504)), ('zz', ('20_newsgroups/rec.sport.baseball/105002', 6
504)), ('zz', ('20_newsgroups/rec.sport.baseball/104795', 6504)), ('zz',
('20_newsgroups/rec.sport.baseball/104540', 6504))]
```

- Next, process this RDD (using an appropriate Spark operation) so that you get a bunch of (docid, (listOfAllDictonaryPos)) pairs. Not surprisingly, listOfAllDictonaryPos lists all of the posInDictionary values found for that document.

In [11...

```
word_docID_mapped = word_docID_unmapped.map(lambda x: (x[1][0], x[1][1])
word_docID_mapped.top(1)
```

```
[('20_newsgroups/talk.religion.misc/84570', 19589)]
```

In [12...

```
word_docID_mapped = word_docID_mapped.groupByKey()
word_docID_mapped.top(1)
```

```
[('20_newsgroups/talk.religion.misc/84570', <pyspark.resultiterable.Resu
ltIterable object at 0x7f69f90c9c50>)]
```

In [13...

```
word_docID_mapped = word_docID_mapped.map(lambda x : (x[0], list(x[1])))
```

- Then finally, you will write a map () that will take that RDD and convert into the listOfAllDictonaryPos values to a NumPy array.

In [14...

```
import numpy as np

def list_to_np(ls):
    arr = np.zeros(20000)
    for i in ls:
        arr[i] += 1
    return arr

res = word_docID_mapped.map(lambda x: ((x[0]), list_to_np(x[1])))
```

In [15…

```
# [tup[1] for tup in res if tupp[0] == "20_newsgroups/comp.graphics/3726
import numpy as np

result1 = np.array(res.lookup("20_newsgroups/comp.graphics/37261"))
result1[result1.nonzero()]
```

```
array([ 8.,  2.,  6.,  3., 12.,  4.,  3.,  6.,  2.,  1.,  1.,  5.,  2.,
        2.,  2.,  3.,  1.,  1.,  1.,  1.,  3.,  1.,  1.,  2.,  3.,  4.,
        1.,  1.,  1.,  1.,  1.,  3.,  1.,  1.,  1.,  2.,  1.,  1.,  1.,
        2.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  2.,
        1.,  1.,  2.,  2.,  1.,  2.,  1.,  1.,  1.,  3.,  4.,  1.,  1.,
        1.,  1.,  2.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  2.,  1.,
        1.,  1.,  1.,  5.,  2.,  2.,  1.,  1.,  5.,  1.,  4.,  1.,  1.,
        1.,  2.,  1.,  2.,  1., 11.,  1.,  1.,  1.,  1.,  2.,  2.,  2.,
        5.,  1.,  2.,  1.,  1.,  1.,  1.,  2.,  1.,  2.,  2.,  4.,  1.,
        1.,  1.,  5.,  1.,  1.,  1.,  1.,  2.,  4.,  1.,  1.,  1.,  3.,
        1.,  1.,  1.,  1.,  3.,  2.,  2.,  1.,  1.,  6.,  1.,  6.,  1.,
        1.,  3.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  7.,  2.,  1.,  1.,
        1.,  1.,  1.])
```

In [16…

```
import numpy as np
result2 = np.array(res.lookup("20_newsgroups/talk.politics.mideast/75944
result2[result2.nonzero()]
```

```
array([135.,  37.,  71.,  28.,  49.,  19.,  46.,  16.,  13.,  22.,   9.,
        22.,  11.,   7.,   7.,   6.,   4.,   6.,  12.,  11.,  10.,   3.,
        10.,   4.,   2.,  21.,   5.,   4.,   2.,   2.,   1.,   1.,   1.,
         5.,   1.,  23.,   5.,   2.,   1.,   6.,   8.,   4.,   7.,   3.,
         3.,   2.,   1.,   1.,   6.,   4.,   4.,   7.,   1.,   8.,   7.,
        13.,   4.,   4.,  10.,   3.,   3.,   2.,   2.,   3.,   7.,   4.,
         1.,   2.,   4.,   8.,   4.,   7.,   2.,   1.,   1.,   2.,   1.,
         2.,   2.,   1.,   5.,   3.,   3.,   3.,   1.,   1.,   1.,   2.,
         1.,   4.,   3.,   1.,   3.,   3.,   4.,   7.,   1.,   2.,   1.,
         3.,   2.,   1.,   4.,   6.,   3.,  11.,   1.,   6.,   3.,   1.,
         3.,   1.,   2.,   1.,   1.,   1.,   3.,   3.,   2.,   5.,   2.,
         2.,   2.,   2.,   1.,   1.,   1.,   1.,   1.,   3.,   1.,   1.,
         1.,   1.,   3.,   3.,   4.,   1.,   1.,   5.,   1.,   1.,   2.,
         6.,   2.,   2.,   1.,   1.,   1.,   1.,   1.,   1.,   3.,   5.,
         1.,   1.,   1.,   1.,   2.,   1.,   1.,   1.,   6.,   1.,   1.,
         2.,   1.,   3.,   2.,   1.,   1.,   3.,   2.,   2.,   3.,   8.,
         1.,   1.,   1.,   1.,   2.,   3.,   1.,   2.,   6.,   1.,   1.,
         2.,   1.,  13.,   4.,   1.,   1.,   1.,   1.,   1.,   3.,   1.,
         1.,   2.,   2.,   1.,   1.,   1.,   2.,   1.,   1.,   1.,   1.,
         3.,   1.,   2.,   4.,  26.,   1.,   1.,   3.,   2.,   2.,   1.,
         3.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   2.,   6.,
         1.,   1.,   1.,   6.,   1.,   1.,   1.,   4.,   2.,   1.,   1.,
         1.,   1.,   4.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,
         3.,   4.,   4.,   4.,   3.,   1.,   3.,   1.,   1.,   1.,   1.,
         1.,   1.,   4.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,
         1.,   1.,   1.,   1.,   2.,   1.,   1.,   2.,   1.,   9.,   1.,
         1.,   2.,   1.,   1.,   1.,   5.,   1.,   1.,   2.,   1.,   1.,
         2.,   1.,   1.,   1.,   1.,   6.,   1.,   1.,   1.,   1.,   1.,
         3.,   1.,   2.,   1.,   1.,   1.,   2.,   1.,   2.,   9.,   1.,
         1.,   1.,   2.,   1.,   1.,   2.,   2.,   2.,   1.,   1.,   2.,
         4.,    1.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,   11.,    2.,
         1.,    1.,    1.,    1.,    1.,    1.,    1.,    2.,    1.,    1.,    1.,
         1.,    1.,    1.,    2.,    9.,    1.,    2.,    7.,    3.,    3.,    2.,
         1.,    1.,    2.,    1.,    1.,    1.,    2.,    2.,    1.,    1.,    1.,
         3.,    8.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    8.,    1.,
         2.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,
         1.,   10.,    1.,    1.,    1.,    2.,    1.,    2.,    3.,    4.,    1.,
         1.,    1.,    1.,    1.,    1.,    1.,    3.,    1.,    1.,    1.,    1.,
         1.,    1.,    8.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    6.,
         1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    2.,    1.,
         1.,    1.,    1.,    1.,   11.,    2.,    1.,    1.,    1.,    1.,    1.,
         2.,    1.,    1.,    3.,    3.,    1.,    1.,    1.,    1.,    1.,    1.,
         1.,    1.,    1.,    1.,    1.,    2.,    1.,    3.,    1.,    1.,    1.,
         1.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,    2.,    1.,    4.,
         2.,    3.,    1.,    1.,    1.,    4.,    1.,    1.,    4.,    1.,    1.,
         1.,    2.,    4.,    1.,    1.,    1.,    1.,    2.,    3.,    1.,    1.,
         1.,    2.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,    1.,    1.,
         1.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,    1.,    2.,    1.,
         2.,    1.,    2.,    2.,    1.,    1.,    1.,    1.,    1.,    4.,    1.,
         1.,    1.,    8.,    1.,    1.,    1.,    2.,    1.,    1.,    2.,    3.,
         1.,    1.,    1.,    1.,    1.,    3.,    1.,    1.,    1.,    1.,    1.,
         4.,    1.,    1.,    1.,    3.,    2.,    1.,    1.,    1.,    1.,    1.,
         2.,    1.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,    1.,    1.,
         1.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,    1.,    1.,    1.,
         1.,    1.,    1.,    1.,    1.,    2.,    1.,    1.,    1.,    1.,    1.,
         4.,    3.,    2.,    1.,    2.,    1.,    2.,    1.]])
```

In [17…

```
import numpy as np
result3 = np.array(res.lookup("20_newsgroups/sci.med/58763"))
result3[result3.nonzero()]
```

```
array([4., 4., 3., 2., 1., 1., 4., 3., 1., 2., 1., 5., 1., 2., 1., 1.,
1.,
       2., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 1., 1., 1., 1., 1., 1., 2., 2., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 1., 1., 1., 2., 1., 1., 1., 1., 2., 1., 1., 1., 2., 1., 5.,
1.,
       1., 1., 1., 1., 1., 1., 2., 1., 2., 1., 1., 1., 1., 1., 1., 1.,
1.,
       1., 3., 1., 1.])
```

Task 2

In [18…

```
res.top(5)
```

```
[('20_newsgroups/talk.religion.misc/84570', array([2., 1., 1., ..., 0.,
0., 0.])), ('20_newsgroups/talk.religion.misc/84569', array([20.,  1., 1
7., ...,  0.,  0.,  0.])), ('20_newsgroups/talk.religion.misc/84568', ar
ray([12.,  3.,  5., ...,  0.,  0.,  0.])), ('20_newsgroups/talk.religio
n.misc/84567', array([6., 6., 4., ..., 0., 0., 0.])), ('20_newsgroups/ta
lk.religion.misc/84566', array([2., 3., 3., ..., 0., 0., 0.]))]
```

- First, use clip (https://numpy.org/doc/stable/reference/generated/numpy.clip.html) in a map to make it so that your count arrays have only 0s or 1s. Then simply sum up all of the arrays using Spark. This will give you a vector that has, for each word, the number of documents that have that word. You can use this to then define a map that uses this vector to convert each documents TF vector (or count vector) into a TF-IDF vector.

In [19…

```
# TF(i, d) = (Number of occurences of word i in d) / (Total number of wo
# IDF(i) = log((Size of corpus (number of docs)) / (Number of documents
import numpy as np

corpus_size = res.count()

def calc_IDF(whole_arr):
    temp_arr = whole_arr.map(lambda x: np.clip(x[1], 0, 1))
    temp_arr = temp_arr.sum()
    temp_arr = corpus_size * np.reciprocal(temp_arr)
    temp_arr = np.log(temp_arr)
    return temp_arr

IDF = calc_IDF(res)

def calc_TFIDF(arr):
    t = np.sum(arr)
    TF = arr / t
    TFIDF = np.multiply(TF, IDF)
    return TFIDF

res_TFIDF = res.map(lambda x: (x[0], calc_TFIDF(x[1])))
```

In [20…

```
res_TFIDF.top(5)
```

```
[('20_newsgroups/talk.religion.misc/84570', array([0.00112697, 0.0009955
2, 0.00142786, ..., 0.        , 0.        ,
       0.        ])), ('20_newsgroups/talk.religion.misc/84569', array
([0.00455734, 0.00040258, 0.00981596, ..., 0.        , 0.        ,
       0.        ])), ('20_newsgroups/talk.religion.misc/84568', array
([0.00498443, 0.00220152, 0.00526267, ..., 0.        , 0.        ,
       0.        ])), ('20_newsgroups/talk.religion.misc/84567', array
([0.002067  , 0.00365182, 0.00349182, ..., 0.        , 0.        ,
       0.        ])), ('20_newsgroups/talk.religion.misc/84566', array
([0.00093192, 0.00246966, 0.00354218, ..., 0.        , 0.        ,
       0.        ]))]
```

In [21…

```
result4 = np.array(res_TFIDF.lookup("20_newsgroups/comp.graphics/37261")
result4[result4.nonzero()]
```

```
array([1.92555059e-03, 8.50478606e-04, 3.65947527e-03, 1.27515309e-03,
       7.26881462e-03, 2.19975741e-03, 2.66216132e-03, 6.35871363e-03,
       3.43014584e-03, 1.65693809e-03, 9.44999233e-03, 3.95217425e-03,
       4.19787791e-03, 5.18781198e-03, 6.54906258e-03, 9.61804463e-06,
       9.42766276e-03, 2.87951994e-03, 5.38035007e-04, 6.14710796e-03,
       9.49332595e-03, 1.50258572e-02, 3.37418661e-03, 4.94317579e-03,
       5.98488082e-03, 4.54842076e-03, 4.99141867e-03, 1.65903978e-02,
       4.91237317e-03, 7.28885166e-03, 6.36745621e-03, 1.33917572e-02,
       6.28356577e-03, 7.13964429e-03, 7.13821425e-03, 1.59751652e-02,
       6.41882998e-03, 7.64721754e-03, 1.62776158e-02, 8.57654687e-03,
       7.64721754e-03, 8.98232287e-03, 8.13687431e-03, 7.89645803e-03,
       8.90098531e-03, 8.51964347e-03, 8.08508990e-03, 1.72643992e-02,
       8.86948389e-03, 9.22023607e-03, 2.01086375e-02, 2.13909906e-02,
       1.05521284e-02, 1.99249503e-02, 1.03349188e-02, 9.82126974e-03,
       1.04846708e-02, 3.22766724e-02, 4.18291549e-02, 1.12750179e-02,
       1.09072460e-02, 1.15831427e-02, 1.14176787e-02, 2.29934203e-02,
       1.12501771e-02, 1.20844660e-02, 1.10766217e-02, 1.17285929e-02,
       1.13455801e-02, 1.16161455e-02, 1.18747446e-02, 1.17746633e-02,
       2.43627756e-02, 1.30455569e-02, 1.22276075e-02, 1.23083739e-02,
       1.35702072e-02, 6.68355875e-02, 2.64048707e-02, 2.64227453e-02,
       1.24333419e-02, 1.27369322e-02, 6.59676107e-02, 1.24404248e-02,
       5.50191332e-02, 1.27837904e-02, 1.36103441e-02, 1.34238833e-02,
       2.67718630e-02, 1.36407694e-02, 3.04637923e-02, 1.38510787e-02,
       1.67550858e-01, 1.46863490e-02, 1.49479000e-02, 1.46724068e-02,
       1.51344992e-02, 3.01422477e-02, 2.99871466e-02, 2.96874341e-02,
       8.16808649e-02, 1.54008745e-02, 3.19745942e-02, 1.59666661e-02,
       1.61138574e-02, 1.61138574e-02, 1.61354292e-02, 3.41042871e-02,
       1.63132576e-02, 3.39910789e-02, 3.56393971e-02, 7.18675593e-02,
       1.78924764e-02, 1.74457237e-02, 1.71389064e-02, 1.04179133e-01,
       1.97732182e-02, 1.82007228e-02, 2.02620792e-02, 1.83663202e-02,
       4.20396654e-02, 7.50891528e-02, 1.88696852e-02, 1.89194795e-02,
       1.91265005e-02, 5.63168646e-02, 1.92350761e-02, 1.88206286e-02,
       1.93473329e-02, 1.92350761e-02, 5.97170587e-02, 3.92916929e-02,
       3.99479240e-02, 2.01148879e-02, 2.02620792e-02, 1.32024656e-01,
       2.13165617e-02, 1.40466044e-01, 2.11158179e-02, 2.34110073e-02,
       6.81339249e-02, 2.11158179e-02, 2.17587186e-02, 4.35174373e-02,
       2.21339792e-02, 2.27113083e-02, 2.27113083e-02, 2.28728651e-02,
       2.04083036e-01, 4.64434820e-02, 2.30427105e-02, 2.42992986e-02,
       2.40539080e-02, 2.48524578e-02, 2.42992986e-02])
```

In [22…

```
result5 = np.array(res_TFIDF.lookup("20_newsgroups/talk.politics.mideast
result5[result5.nonzero()]
```

```
array([5.54725111e-03, 2.68605086e-03, 7.39273306e-03, 2.03178717e-03,
       5.06707738e-03, 1.94545485e-03, 4.31868710e-03, 2.42388609e-03,
       2.46472840e-03, 4.14154280e-03, 1.62831955e-03, 6.62550483e-03,
       3.22073049e-03, 2.43657927e-03, 1.69721414e-03, 1.15698386e-03,
       1.93593964e-03, 4.04824180e-03, 3.94159141e-03, 2.94504525e-03,
       9.88048494e-04, 4.42826235e-03, 1.49072312e-03, 5.28678488e-04,
       1.04214371e-02, 1.86094552e-03, 1.49772974e-03, 1.04087154e-03,
       1.64197257e-06, 2.22622265e-03, 4.73573904e-05, 1.23392775e-02,
       1.00104738e-03, 9.83171309e-04, 9.65207547e-04, 3.03599781e-03,
       3.98260227e-03, 2.13198368e-03, 3.57653570e-03, 1.57413222e-03,
       2.17955958e-03, 1.07444211e-03, 5.40226952e-04, 5.59517954e-04,
       3.84777461e-03, 1.44634481e-03, 2.21205679e-03, 6.24049705e-03,
       5.63104209e-04, 4.86079553e-03, 4.90455048e-03, 7.84729739e-03,
       1.77626038e-03, 2.30413648e-03, 7.10498595e-03, 1.96850758e-03,
       2.16703483e-03, 1.31035475e-03, 1.36934217e-03, 1.94164808e-03,
       5.81927334e-03, 2.86998439e-03, 1.36267792e-03, 1.45303972e-03,
       3.47796655e-03, 6.75110950e-03, 2.96447004e-03, 5.13676332e-03,
       1.55299386e-03, 1.39374925e-03, 7.44483602e-04, 1.48960041e-03,
       8.19877288e-04, 1.73294509e-03, 1.59879298e-03, 8.04183656e-04,
       5.12938309e-03, 2.50805724e-03, 2.55637383e-03, 2.43592804e-03,
       8.68439688e-04, 8.90132820e-04, 1.51500388e-03, 1.87415561e-03,
       1.47175815e-03, 3.73352652e-03, 2.91159940e-03, 8.95481969e-04,
       2.74919301e-03, 2.73594560e-03, 3.94921536e-03, 6.43284322e-03,
       9.41246420e-04, 2.15148451e-03, 9.90881645e-04, 2.99283121e-03,
       2.01554418e-03, 9.63339843e-04, 4.03647370e-03, 6.18664562e-03,
       3.09437419e-03, 1.14412419e-02, 1.04727765e-03, 6.46812994e-03,
       2.94738368e-03, 8.38630128e-04, 3.02887238e-03, 1.24561624e-03,
       2.07560755e-03, 1.04475964e-03, 1.06858419e-03, 9.54857618e-04,
       3.66614576e-03, 3.47542870e-03, 2.19044071e-03, 5.46533058e-03,
       2.17407776e-03, 2.28621293e-03, 2.17989452e-03, 2.14543455e-03,
       1.21594255e-03, 1.35797367e-03, 1.14847082e-03, 1.20463876e-03,
       1.23571864e-03, 3.65586326e-03, 1.48294476e-03, 1.19122644e-03,
       1.72118288e-03, 1.19472526e-03, 4.10036054e-03, 3.68466389e-03,
       5.05531136e-03, 1.23345898e-03, 1.23120831e-03, 6.42692941e-03,
       1.26171657e-03, 1.22971283e-03, 2.77888071e-03, 8.50936495e-03,
       2.67244625e-03, 2.57242105e-03, 1.30665677e-03, 1.32609646e-03,
       1.36646971e-03, 1.33113712e-03, 1.39175645e-03, 1.34196217e-03,
       4.06737502e-03, 6.66015665e-03, 1.33203133e-03, 1.35516979e-03,
       1.47751669e-03, 1.40044312e-03, 2.74441105e-03, 1.77340069e-03,
       1.40789911e-03, 1.49754646e-03, 9.17519717e-03, 1.79193649e-03,
       1.55162439e-03, 2.80764004e-03, 1.41476369e-03, 4.55867309e-03,
       2.84412471e-03, 1.46947107e-03, 1.52289276e-03, 4.36336743e-03,
       2.94734231e-03, 3.08925841e-03, 4.56617143e-03, 1.48000188e-02,
       1.48294476e-03, 1.51417984e-03, 1.65512923e-03, 1.48607009e-03,
       3.03248439e-03, 4.51193830e-03, 1.53771952e-03, 3.30814720e-03,
       9.82160631e-03, 1.65301994e-03, 1.64310687e-03, 3.16100856e-03,
       1.45076350e-03, 2.28045094e-02, 6.28527927e-03, 1.58375442e-03,
       1.55915286e-03, 1.65883955e-03, 1.99742033e-03, 1.66418261e-03,
       5.26825408e-03, 1.60462324e-03, 1.63591216e-03, 3.35883510e-03,
       3.65182541e-03, 1.64725962e-03, 1.80143740e-03, 1.66471970e-03,
       3.48471461e-03, 1.94125903e-03, 1.70249386e-03, 1.62229003e-03,
       1.67886632e-03, 5.49065092e-03, 1.62981732e-03, 3.51850320e-03,
       7.18938406e-03, 6.82825870e-02, 1.83673546e-03, 1.84112325e-03,
       7.45846129e-03, 3.46394698e-03, 3.56254061e-03, 1.92484759e-03,
       5.31435087e-03, 1.76371529e-03, 1.79734601e-03, 1.80212219e-03,
       1.75988711e-03, 1.85524697e-03, 1.71351579e-03, 1.81110074e-03,
       1.85902314e-03, 3.59062274e-03, 1.59576275e-02, 1.81810567e-03,
       1.76886018e-03, 1.74112573e-03, 1.21087323e-02, 1.99838912e-03,
       1.81529327e-03, 1.91555955e-03, 7.20301378e-03, 3.92903592e-03,
       2.03030115e-03, 2.05654965e-03, 2.34281276e-03, 1.89338843e-03,
       7.71648164e-03, 1.91976248e-03, 2.03441986e-03, 1.84185787e-03,
       1.93256207e-03, 1.94477563e-03, 1.94213613e-03, 1.82949731e-03,
       1.91305272e-03, 5.48418222e-03, 8.20480615e-03, 8.00912394e-03,
       8.13354947e-03, 5.95776745e-03, 2.10946125e-03, 6.04535353e-03,
       1.90559822e-03, 1.92145244e-03, 2.04170062e-03, 2.01312387e-03,
       1.92229932e-03, 2.20154005e-03, 8.24779536e-03, 2.01611747e-03,
       1.98687176e-03, 2.14854661e-03, 2.10009803e-03, 2.06630563e-03,
       2.00619983e-03, 2.01611747e-03, 2.01113690e-03, 2.01411998e-03,
       2.01014603e-03, 1.95906495e-03, 2.11658686e-03, 3.97945835e-03,
       1.99548771e-03, 2.15746909e-03, 5.10374880e-03, 2.13727797e-03,
       2.08962793e-02, 2.24034273e-03, 2.11064263e-03, 5.25250669e-03,
       2.36648380e-03, 2.05120154e-03, 2.06521328e-03, 1.10922888e-02,
       2.07623295e-03, 2.01212951e-03, 4.13261127e-03, 2.02825299e-03,
       2.45923005e-03, 4.92725845e-03, 2.30488266e-03, 2.10125999e-03,
       2.10476025e-03, 2.13234090e-03, 1.39000653e-02, 2.03961082e-03,
       2.20154005e-03, 2.10593184e-03, 2.20993562e-03, 2.50247258e-03,
       6.33903847e-03, 2.38155912e-03, 4.51083611e-03, 2.33221324e-03,
       2.31159243e-03, 2.26466008e-03, 4.34884514e-03, 2.13851901e-03,
       4.69998231e-03, 2.40214731e-02, 2.26466008e-03, 2.59476063e-03,
       2.36093261e-03, 4.90972969e-03, 2.35543541e-03, 2.07958118e-03,
       4.43691552e-03, 4.36484421e-03, 4.39477714e-03, 2.34459883e-03,
       2.26933837e-03, 4.41987124e-03, 9.23963003e-03, 2.19190026e-03,
       2.32180881e-03, 2.34999115e-03, 4.53243042e-03, 2.21702845e-03,
       2.41702455e-03, 2.23002422e-03, 2.25085309e-03, 2.55588260e-02,
       4.77078324e-03, 2.31667754e-03, 2.37397121e-03, 2.23295320e-03,
       2.22132726e-03, 2.40296685e-03, 2.41095697e-03, 2.29332887e-03,
       4.66442648e-03, 2.29990209e-03, 2.34999115e-03, 2.24482326e-03,
       2.77338907e-03, 2.37209004e-03, 2.29496502e-03, 5.41022468e-03,
       2.24798561e-02, 2.40894882e-03, 4.71452376e-03, 1.66575680e-02,
       7.05540026e-03, 7.23891680e-03, 5.09859807e-03, 2.33396608e-03,
       2.62039535e-03, 4.72925468e-03, 2.30488266e-03, 2.46806290e-03,
       2.55706059e-03, 5.02402716e-03, 5.00968555e-03, 2.47253163e-03,
       2.37965257e-03, 2.46362922e-03, 7.49328537e-03, 2.17505363e-02,
       2.57559639e-03, 2.37021511e-03, 2.52910792e-03, 2.53159272e-03,
       2.45923005e-03, 2.37209004e-03, 2.68843588e-03, 2.17228767e-02,
       2.64116954e-03, 4.99552358e-03, 2.49308994e-03, 2.39704752e-03,
       2.42728420e-03, 2.46362922e-03, 2.35361483e-03, 2.53659533e-03,
       2.62039535e-03, 2.42110609e-03, 2.74001794e-03, 2.61172064e-03,
       2.69838260e-03, 2.63815465e-02, 2.43353057e-03, 2.61172064e-03,
       2.53159272e-03, 5.12458961e-03, 2.51684516e-03, 4.94506326e-03,
       8.18795431e-03, 1.16250166e-02, 2.55967164e-03, 2.52910792e-03,
       2.62920526e-03, 2.65339253e-03, 2.59755220e-03, 2.51684516e-03,
       2.54929904e-03, 9.54212225e-03, 2.55187440e-03, 2.66905257e-03,
       2.54418324e-03, 2.68515882e-03, 2.55967164e-03, 2.87798371e-03,
       2.18914302e-02, 2.71535959e-03, 2.76204506e-03, 2.70173746e-03,
       2.92093787e-03, 2.60035762e-03, 2.58100591e-03, 2.72579603e-03,
       1.72955902e-02, 2.81293604e-03, 2.68515882e-03, 2.67223731e-03,
       2.84236304e-03, 2.78496540e-03, 2.77338907e-03, 2.65031192e-03,
       2.93601319e-03, 2.74001794e-03, 6.08428374e-03, 2.83379786e-03,
       2.75831375e-03, 2.74726536e-03, 2.75092422e-03, 2.86436158e-03,
       3.33298306e-02, 5.72872316e-03, 2.75831375e-03, 2.69838260e-03,
       2.89667272e-03, 2.70511234e-03, 2.74363004e-03, 5.54677814e-03,
       2.77722157e-03, 2.80885450e-03, 8.82338696e-03, 8.29740347e-03,
       2.96206809e-03, 3.04831997e-03, 2.79281680e-03, 3.12114805e-03,
       2.99500698e-03, 2.79678377e-03, 2.77722157e-03, 2.83379786e-03,
       2.82956345e-03, 2.83379786e-03, 2.89194223e-03, 6.16053165e-03,
       2.81293604e-03, 8.60659775e-03, 2.76204506e-03, 2.81293604e-03,
       2.82956345e-03, 2.91110647e-03, 2.76958238e-03, 2.81293604e-03,
       2.81293604e-03, 3.17291018e-03, 2.82956345e-03, 2.97829766e-03,
       3.05456635e-03, 6.33043804e-03, 2.96206809e-03, 1.21932799e-02,
       5.96762547e-03, 9.26063248e-03, 3.12825873e-03, 3.08687749e-03,
       2.93601319e-03, 1.22182654e-02, 2.94112899e-03, 2.95150159e-03,
       1.30977132e-02, 2.92591844e-03, 3.17291018e-03, 3.37564268e-03,
       6.54885662e-03, 1.35930294e-02, 3.16521902e-03, 3.00642740e-03,
       3.02400263e-03, 3.14275417e-03, 6.10913270e-03, 9.18264755e-03,
       3.08026583e-03, 3.06088252e-03, 3.08026583e-03, 6.27091997e-03,
       3.02998460e-03, 3.03603055e-03, 6.10913270e-03, 3.22139340e-03,
       3.15014370e-03, 3.10033744e-03, 3.21301856e-03, 3.20476599e-03,
       3.10718954e-03, 3.20476599e-03, 3.18070742e-03, 3.15014370e-03,
       6.31526221e-03, 3.16521902e-03, 3.22139340e-03, 3.12114805e-03,
       3.34333154e-03, 3.17291018e-03, 6.42603712e-03, 3.15014370e-03,
       6.36141484e-03, 3.17291018e-03, 6.39326433e-03, 6.47704949e-03,
       3.32277323e-03, 3.17291018e-03, 3.18070742e-03, 3.24728912e-03,
       3.21301856e-03, 1.38883831e-02, 3.22989418e-03, 3.33295894e-03,
       3.45910000e-03, 3.10178184e-02, 3.43397182e-03, 3.25619153e-03,
       3.30293642e-03, 7.42920637e-03, 3.49901949e-03, 3.37564268e-03,
       6.68666308e-03, 1.16316819e-02, 3.31276782e-03, 3.38683693e-03,
       3.36466581e-03, 3.42181455e-03, 3.32277323e-03, 1.03773000e-02,
       3.32277323e-03, 3.34333154e-03, 3.40991324e-03, 3.36466581e-03,
       3.71460318e-03, 1.38883831e-02, 3.47209577e-03, 3.54197365e-03,
       3.42181455e-03, 1.03773000e-02, 7.20969702e-03, 3.51297800e-03,
       3.40991324e-03, 3.45910000e-03, 3.52728994e-03, 3.44639630e-03,
       7.05457987e-03, 3.55704897e-03, 3.54197365e-03, 3.47209577e-03,
       7.11409794e-03, 3.52728994e-03, 3.51297800e-03, 3.75649576e-03,
       3.57253737e-03, 3.60484851e-03, 3.71460318e-03, 4.35619727e-03,
       3.82580177e-03, 3.71460318e-03, 7.51299152e-03, 3.67560213e-03,
       3.73516149e-03, 3.77866688e-03, 3.73516149e-03, 3.77866688e-03,
       3.71460318e-03, 3.77866688e-03, 3.87722731e-03, 3.90480795e-03,
       3.82580177e-03, 3.82580177e-03, 3.90480795e-03, 7.99335692e-03,
       3.93380360e-03, 4.14832571e-03, 4.10643313e-03, 4.10643313e-03,
       4.03094902e-03, 1.67742926e-02, 1.32683369e-02, 8.71239453e-03,
       4.29663790e-03, 8.38714629e-03, 4.10643313e-03, 8.59327580e-03,
       4.14832571e-03])
```

In [23…

```
result6 = np.array(res_TFIDF.lookup("20_newsgroups/sci.med/58763"))
result6[result6.nonzero()]
```

```
array([2.34482371e-03, 4.14265386e-03, 4.42577019e-03, 2.92148526e-03,
       1.33936842e-03, 2.16121698e-03, 1.08191180e-02, 8.05687963e-03,
       2.58109075e-03, 8.59275516e-03, 4.17703243e-03, 4.12642026e-03,
       9.62545664e-03, 4.20143956e-03, 6.31741620e-03, 5.31671210e-03,
       1.48492077e-02, 2.34245926e-05, 6.75606643e-04, 2.85621422e-03,
       1.39301109e-02, 7.98215533e-03, 5.15843543e-03, 7.88937191e-03,
       8.61158132e-03, 6.33508993e-03, 9.41006605e-03, 1.04688183e-02,
       1.10776054e-02, 1.06254158e-02, 1.15837546e-02, 1.47099141e-02,
       1.39793137e-02, 1.47149138e-02, 1.53791960e-02, 1.36221220e-02,
       1.58114516e-02, 3.46935867e-02, 3.42222836e-02, 1.70775536e-02,
       1.82594869e-02, 1.83374808e-02, 2.12790916e-02, 1.99693017e-02,
       2.16014850e-02, 2.17616535e-02, 2.77570110e-02, 2.30871061e-02,
       2.31437988e-02, 2.52717384e-02, 2.50165686e-02, 5.76677542e-02,
       2.58076567e-02, 2.72681028e-02, 3.02984539e-02, 2.82909350e-02,
       6.64934520e-02, 3.03504273e-02, 3.12699320e-02, 3.30500207e-02,
       6.64437477e-02, 4.11898951e-02, 2.22132596e-01, 3.43093815e-02,
       3.48071684e-02, 3.87869947e-02, 3.79869411e-02, 4.06113152e-02,
       3.78096918e-02, 3.85914817e-02, 8.45144911e-02, 4.43275669e-02,
       8.94617535e-02, 4.18132151e-02, 4.35768377e-02, 4.37580704e-02,
       4.41332318e-02, 4.24887787e-02, 4.76963992e-02, 4.50471728e-02,
       5.07453196e-02, 4.99174635e-02, 1.62708477e-01, 5.61201497e-02,
       5.70171306e-02])
```

Task 3

- Next, your task is to build a kNN classifier, embodied by the Python function predictLabel. This function will take as input a text string and a number k, and then output the name of one of the 20 newsgroups.
- This algorithm first **converts the input string into a TF-IDF vector** (using the dictionary and count information computed over the original corpus). **It then finds the k documents in the corpus that are “closest” to the query vector (where distance**

is computed using the L_2 norm), and returns the newsgroup label that is most

In [24...

```
wordDict = dictionary.collect()
```

In [25...

```
wordDict = dict(wordDict)
```

In [26...

```
from collections import Counter
def predictLabel(k, inputStr):
    initArr = np.zeros(20000)
    inputWords = regex.sub(' ', inputStr).lower().split()
    for word in inputWords:
        if word in wordDict.keys():
            initArr[wordDict[word]] += 1
    # Convert initArr to TFIDF
    input_TFIDF = calc_TFIDF(initArr)
    # Calculate 2 Norm with other vectors
    twoNorm = res_TFIDF.map(lambda x: (x[0], np.linalg.norm(x[1] - input
    a = twoNorm.collect()
    a.sort(key=lambda x: x[1])

    resultCategories = []
    for i in range(k):
        resultCategories.append(a[i][0])

    resultLabels = [i.split('/')[1] for i in resultCategories]
    resultLabelsCounter = Counter(resultLabels)
    return resultLabelsCounter.most_common(1)[0][0]

topK = predictLabel(10, 'Graphics are pictures and movies created using
```

In [27...

```
predictLabel (10, 'Graphics are pictures and movies created using comput
```

```
'comp.graphics'
```

In [28...

```
predictLabel (10, 'A deity is a concept conceived in diverse ways in var
```

```
'talk.religion.misc'
```

In [29...

```
predictLabel (10, 'Egypt, officially the Arab Republic of Egypt, is a tr
```

```
'alt.atheism'
```

In [30…

```
predictLabel (10, 'The term atheism originated from the Greek atheos, me
```

```
'alt.atheism'
```

In [31…

```
predictLabel (10, 'President Dwight D. Eisenhower established NASA in 19
```

```
'sci.space'
```

In [32…

```
predictLabel (10, 'The transistor is the fundamental building block of m
```

```
'talk.politics.misc'
```

In [33…

```
predictLabel (10, 'The Colt Single Action Army which is also known as th
```

```
'talk.politics.guns'
```

In [34…

```
predictLabel (10, 'Howe was recruited by the Red Wings and made his NHL
```

```
'talk.politics.mideast'
```

In [ ]: