Assignment-2-Template

September 18, 2020

[1]:
```
import sys
import re
import numpy as np

from numpy import dot
from numpy.linalg import norm
```

[2]:
```
# Set the file paths on your local machine
# Change this line later on your python script when you want to run this on the␣
↪CLOUD (GC or AWS)

wikiPagesFile="/home/kia/wikiextractor/WikiPagesOutput/
↪WikipediaPages_oneDocPerLine_1000Lines_small.txt"
wikiCategoryFile="/home/kia/wikiextractor/wiki-categorylinks.csv.bz2"
```

[3]:
```
# Read two files into RDDs

wikiCategoryLinks=sc.textFile(wikiCategoryFile)

wikiCats=wikiCategoryLinks.map(lambda x: x.split(",")).map(lambda x: (x[0].
↪replace('"', ''), x[1].replace('"', '') ))

# Now the wikipages
wikiPages = sc.textFile(wikiPagesFile)

wikiCategoryLinks.take(2)
```

[3]:
```
['"10","Redirects_from_moves"', '"10","Redirects_with_old_history"']
```

[4]:
```
wikiCats.take(1)
```

[4]:
```
[('10', 'Redirects_from_moves')]
```

[5]:
```
df = spark.read.csv(wikiPagesFile)

# Uncomment this line if you want to take look inside the file.
# df.take(1)
```

[6]:
```python
# wikiPages.take(1)
```

[7]:
```python
# Assumption: Each document is stored in one line of the text file
# We need this count later ...
numberOfDocs = wikiPages.count()

print(numberOfDocs)
# Each entry in validLines will be a line from the text file
validLines = wikiPages.filter(lambda x : 'id' in x and 'url=' in x)

# Now, we transform it into a set of (docID, text) pairs
keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('"␣
↪url=')], x[x.index('">') + 2:][:-6]))

# keyAndText.take(1)
```

```
1000
```

[8]:
```python
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


def build_zero_one_array (listOfIndices):

    returnVal = np.zeros (20000)

    for index in listOfIndices:
        if returnVal[index] == 0: returnVal[index] = 1

    return returnVal


def stringVector(x):
    returnVal = str(x[0])
    for j in x[1]:
        returnVal += ',' + str(j)
    return returnVal
```

```python
def cousinSim (x,y):
        normA = np.linalg.norm(x)
        normB = np.linalg.norm(y)
        return np.dot(x,y)/(normA*normB)
```

[9]:
```python
# Now, we transform it into a set of (docID, text) pairs
keyAndText = validLines.map(lambda x : (x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">') + 2:][:-6]))

# Now, we split the text in each (docID, text) pair into a list of words
# After this step, we have a data set with
# (docID, ["word1", "word2", "word3", ...])
# We use a regular expression here to make
# sure that the program does not break down on some of the documents

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

# remove all non letter characters
keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))
# better solution here is to use NLTK tokenizer

# Now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])
# to ("word1", 1) ("word2", 1)...
allWords = keyAndListOfWords.???

# Now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.
allCounts = allWords.???

# Get the top 20,000 words in a local array in a sorted format based on frequency
# If you want to run it on your laptio, it may a longer time for top 20k words.
topWords = allCounts.???

#
print("Top Words in Corpus:", allCounts.top(10, key=lambda x: x[1]))

# We'll create a RDD that has a set of (word, dictNum) pairs
# start by creating an RDD that has the number 0 through 20000
# 20000 is the number of words that will be in our dictionary
topWordsK = 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 = topWordsK.map (lambda x : (topWords[x][0], x))


print("Word Postions in our Feature Matrix. Last 20 words in 20k positions: ",␣
↪dictionary.top(20, lambda x : x[1]))
```

```
Top Words in Corpus: [('the', 74530), ('of', 34512), ('and', 28479), ('in',
27758), ('to', 22583), ('a', 21212), ('was', 12160), ('as', 8811), ('for',
8773), ('on', 8435)]
Word Postions in our Feature Matrix. Last 20 words in 20k positions:
[('quebecor', 19999), ('poten', 19998), ('kasada', 19997), ('yadnya', 19996),
('drift', 19995), ('iata', 19994), ('satire', 19993), ('expreso', 19992),
('olimpico', 19991), ('auxiliaries', 19990), ('tenses', 19989), ('petherick',
19988), ('stowe', 19987), ('infimum', 19986), ('parramatta', 19985), ('rimpac',
19984), ('hyderabad', 19983), ('cubes', 19982), ('meats', 19981), ('chaat',
19980)]
```

[10]:

```
################## TASK 2  ##################

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

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


# Now join and link them, to get a set of ("word1", (dictionaryPos, docID))␣
↪pairs
allDictionaryWords = ???.join(???)

# Now, we drop the actual word itself to get a set of (docID, dictionaryPos)␣
↪pairs
justDocAndPos = allDictionaryWords.???


# Now get a set of (docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...])␣
↪pairs
allDictionaryWordsInEachDoc = justDocAndPos.groupByKey()
```

```
# The following line this gets us a set of
# (docID,  [dictionaryPos1, dictionaryPos2, dictionaryPos3...]) pairs
# and converts the dictionary positions to a bag-of-words numpy array...
allDocsAsNumpyArrays = allDictionaryWordsInEachDoc.map(lambda x: (x[0],␣
↪buildArray(x[1])))

print(allDocsAsNumpyArrays.take(3))
```

```
[('431962', array([0.08145766, 0.02357985, 0.03429796, …, 0.        , 0.
,
       0.        ])), ('432044', array([0.04477612, 0.0261194 , 0.02985075, …,
0.        , 0.        ,
       0.        ])), ('432055', array([0.06666667, 0.02666667, 0.01333333, …,
0.        , 0.        ,
       0.        ]))]
```

[11]:
```
# Now, create 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.???

# 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 numberOfDocs␣
↪(number of docs)
multiplier = np.full(20000, numberOfDocs)

# 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(np.full(20000, numberOfDocs), ???))

# Finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf * idf␣
↪vectors
allDocsAsNumpyArraysTFidf = allDocsAsNumpyArrays.map(lambda x: (x[0], np.
↪multiply(x[1], idfArray)))

allDocsAsNumpyArraysTFidf.take(2)

# use the buildArray function to build the feature array
```

```
# allDocsAsNumpyArrays = allDictionaryWordsInEachDoc.map(lambda x: (x[0],␣
↪buildArray(x[1])))


# print(allDocsAsNumpyArraysTFidf.take(2))
```

[11]:
```
[('431962',
  array([0.00688066, 0.00196613, 0.00481585, ..., 0.        , 0.        ,
         0.        ])),
 ('432044',
  array([0.0037822 , 0.00217788, 0.00419141, ..., 0.        , 0.        ,
         0.        ]))]
```

[12]:
```
wikiCats.take(1)
```

[12]:
```
[('10', 'Redirects_from_moves')]
```

[19]:
```
# Now, we join it with categories, and map it after join so that we have only␣
↪the wikipageID
# This joun can take time on your laptop.
# You can do the join once and generate a new wikiCats data and store it. Our␣
↪WikiCategories includes all categories
# of wikipedia.

featuresRDD = wikiCats.join(???).map(lambda x: (x[1][0], ???))

# Cache this important data because we need to run kNN on this data set.
featuresRDD.cache()
featuresRDD.take(10)
```

[19]:
```
[('1721_births',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
         0.        ])),
 ('1793_deaths',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
         0.        ])),
 ('18th-century_British_scientists',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
         0.        ])),
 ('18th-century_English_medical_doctors',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
         0.        ])),
 ('Alumni_of_the_University_of_Aberdeen',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
         0.        ])),
 ('Articles_incorporating_Cite_DNB_template',
  array([0.00521662, 0.00277279, 0.00633689, ..., 0.        , 0.        ,
```

```
        0.        ])),
 ('Articles_incorporating_DNB_text_with_Wikisource_reference',
  array([0.00521662, 0.00277279, 0.00633689, …, 0.        , 0.        ,
         0.        ])),
 ('Articles_with_hCards',
  array([0.00521662, 0.00277279, 0.00633689, …, 0.        , 0.        ,
         0.        ])),
 ('Commons_category_link_is_on_Wikidata',
  array([0.00521662, 0.00277279, 0.00633689, …, 0.        , 0.        ,
         0.        ])),
 ('English_scientists',
  array([0.00521662, 0.00277279, 0.00633689, …, 0.        , 0.        ,
         0.        ]))]
```

[14]:
```
# Let us count and see how large is this data set.
wikiAndCatsJoind.count()
```

[14]:
```
13780
```

[20]:
```
# Finally, we have a function that returns the prediction for the label of a␣
↪string, using a kNN algorithm
def getPrediction (textInput, k):
    # Create an RDD out of the textIput
    myDoc = sc.parallelize (('', textInput))

    # Flat map the text to (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 will give us a set of (word, (dictionaryPos, 1)) pairs
    allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda␣
↪x: (x[1][1], x[1][0])).groupByKey ()

    # Get tf array for the input string
    myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])

    # Get the tf * idf array for the input string
    myArray = np.multiply (???, ???)

    # Get the distance from the input text string to all database documents,␣
↪using cosine similarity (np.dot() )
    distances = featuresRDD.map (lambda x : (x[0], np.dot (x[1], myArray)))
    # distances = allDocsAsNumpyArraysTFidf.map (lambda x : (x[0], cousinSim␣
↪(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 (docID, 1) pairs
    docIDRepresented = sc.parallelize(topK).map (lambda x : (x[0], 1))

    # now, for each docID, get the count of the number of times this document␣
↪ID appeared in the top k
    numTimes = docIDRepresented.???

    # Return the top 1 of them.
    # Ask yourself: Why we are using twice top() operation here?
    return numTimes.top(k, lambda x: x[1])
```

[21]:
```
print(getPrediction('Sport Basketball Volleyball Soccer', 10))
```

```
[('1931_births', 1), ('All_disambiguation_pages', 1), ('2015_deaths', 1),
('Disambiguation_pages_with_short_description', 1), ('Bullfighters', 1),
("Air_Force_Falcons_men's_basketball_coaches", 1),
('Human_name_disambiguation_pages', 1), ('All_article_disambiguation_pages', 1),
('Lists_of_sportspeople_by_sport', 1), ('All_articles_with_dead_external_links',
1)]
```

[22]:
```
print(getPrediction('What is the capital city of Australia?', 10))
```

```
[('All_set_index_articles', 2), ('Articles_with_short_description', 2),
('All_Wikipedia_articles_written_in_Australian_English', 2),
('Royal_Australian_Navy_ship_names', 1), ('Set_indices_on_ships', 1),
('Use_dmy_dates_from_April_2018', 1), ('Use_Australian_English_from_April_2018',
1)]
```

[23]:
```
print(getPrediction('How many goals Vancouver score last year?', 10))
```

```
[('All_stub_articles', 2), ('1979_births', 1),
('1991_Canadian_television_series_debuts', 1), ('CBC_Television_shows', 1),
('1990s_Canadian_teen_drama_television_series', 1),
('1994_Canadian_television_series_endings', 1),
('Canadian_television_program_stubs', 1), ('Television_shows_set_in_Vancouver',
1), ('Ak_Bars_Kazan_players', 1)]
```

[ ]:
```
# Congradulations, you have implemented a prediction system based on Wikipedia␣
↪data.
# You can use this system to generate automated Tags or Categories for any kind␣
↪of text
# that you put in your query.
# This data model can predict categories for any input text.
```