Data Science

kNN Assignment

Chris Jermaine, Risa Myers, & Marmar Orooji

Rice University

Objectives

- Review the kNN assignment for Spark

Preprocessing the document text

- Read in the file
- Keep only lines that contain `id=`

```
validLines = corpus.filter(lambda x : 'id' in x)
validLines.take(3)
```

```
['<doc id="20_newsgroups/comp.graphics/37926" url="" title="20_newsgroups/comp.graphics/37926"> From: cst@garfield.catt
 '<doc id="20_newsgroups/comp.graphics/37944" url="" title="20_newsgroups/comp.graphics/37944"> From: egerter@gaul.csd.
 '<doc id="20_newsgroups/comp.graphics/38274" url="" title="20_newsgroups/comp.graphics/38274"> From: geigel@seas.gwu.e
```

Preprocessing the document text

```
# 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('"> ') + 3:x.index(' </doc>')]))
keyAndText.take(3)

[('20_newsgroups/comp.graphics/37926',
  ' From: cst@garfield.catt.ncsu.edu (Caroline Tsang) Subject: Graphics Library Package Date: 6 Apr 93 05:12:01 GMT Lir
 ('20_newsgroups/comp.graphics/37944',
  " From: egerter@gaul.csd.uwo.ca (Barry Egerter) Subject: Re: Graphics Library Package Date: Tue, 6 Apr 1993 15:50:32
 ('20_newsgroups/comp.graphics/38274',
  ' From: geigel@seas.gwu.edu (Joseph Geigel) Subject: Looking for AUTOCAD .DXF file parser Date: Wed, 14 Apr 1993 21:3
```

- What's going on here?
  - We're extracting the id and separating it from the body of the text
  - Result is a key-value pair

Preprocessing the document text

```
regex = re.compile('[^a-zA-Z]')
keyAndText.map(lambda x : (str(x[0]), regex.sub(' ', x[1]).lower().split()))

[('20_newsgroups/comp.graphics/37926',
  ['from',
   'cst',
   'garfield',
      ...
```

- What's going on here?
  - We're keeping the doc id with the text (`str(x[0])`)
  - We're replacing every non alphabetic character in the value text with a space (`regex.sub`)
  - We're converting the text to lower case (using string function `lower`)
  - We're splitting the text at spaces (`split`)

Why does this matter?

- We want "dog" and "dog." and "Dog!" to all become "dog"
- This helps TF-IDF work better

Why else does this matter?

- Apples to apples
- When we go to compare a new document to the training set, we have to apply the EXACT same transformations
  1. Replace every non alphabetic character in the text with a space (`regex.sub`)
  2. Convert the text to lower case (using string function `lower`)
  3. Split the text at spaces (`split`)

- RDD of top N words in the corpus
- Keep it as an RDD!
- Spark knows how to use it
- We can use it in RDD transformations (e.g. join)

Bag of Words

- Want (id1, [17, 0, 4, 3, 0, 0, 0, 2, 0, 5, ...]) for each document
- Where 17 is the number of times Word0 appears in the document identified by id1
- 0 is the number of times Word1 appears in the document identified by id1 ...
- Don't lose the docId!
- Think about RDD transformations
  - May operate on the key
  - You decide how to structure the elements in your RDD
- Think about how you can use `refDict` to keep only the those words
- As soon as possible, convert words to indexes
- Want (docId, [numPy array of word counts by word index])
- Recall the Python exercise!

- Sanity check: What should they length of your numPy array be?

TF-IDF

- Compute this for EVERY document at the same time
- No looping over documents - use RDDs!
- No looping over words - use numPy vectorized functions
- Compute each piece
- TF
  - # of occurrences of each top word in each document (from the bag of words)
  - Total # of top words in each document (Note: Not # of unique words)
    - The “total number of words” in “Today is a great day today” is six.

TF-IDF

- IDF
  - Total # of documents
  - # of documents having each top word

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & \dots \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} & \dots \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} & \dots \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} & \dots \\ w_{4,0} & w_{4,1} & w_{4,2} & w_{4,3} & \dots \end{bmatrix}$$

    We want to know how many $w_{x,0}$ entries are non-zero, how many $w_{x,1}$ entries are non-zero, etc.
  - Don't forget to take the log (`np.log`) in the IDF calculation
- Combine pieces (how? what do the pieces have in common?)
- Use numPy array functions!

kNN classifier prep the test doc

- You are given k and a text string
- You are classifying one text string at a time
- Apply the same prep / transformations
  1. Replace every non alphabetic character in the text with a space (`regex.sub`)
  2. Convert the text to lower case (using string function `lower`)
  3. Split the text at spaces (`split`)
  4. Construct a Bag of Words numPy array representation using the words in the document that are in `refDict`
- Compute TF-IDF
  1. Compute the number of words in the test document
  2. Remember to use the IDF value calculated from the training data
- Now, we have our features and we can compare our test doc to the training data
- Do it all at once! Use an RDD
- Do NOT create a queue (like outlier detection)

kNN classifier calculate distances and choose label

- Find the closest k documents
  - Is there an RDD transformation or action that can do this for you?
  - Which order should you sort in?
- Figure out what categories they cover
- Is there a clear winner?
- If not choose the category for the closest document (that is in the tie)

Best practices

- Use RDDs as much as possible
- Do NOT `collect` until you absolutely have to
- Use RDD transformations as much as possible
  - This means you often need (key, value) pairs
  - Your "value" can be as complex as you need!
- Use numPy functions as much as possible
- You can divide a vector of numbers by a single number
- Don’t lose your docId
- If you start writing a loop, think about whether or not you can use an RDD to do what you are trying to do

Is my code right?

- You can change the test data!
- Put the exact text you are trying to match in the training set
- Look for the 1 nearest neighbor

- Make a very small training set with a small set of words and / or labels

Closing thoughts

- Start with your comments
- Write out the transformations of your data visually, before starting to code (e.g. COLD to WARM)
- Expect them to change
- Start early!