5243 Introduction to Data Mining

Assignment 4

Johns Gresham

Introduction

To find the similarity between two documents we can directly compare all of the features of the document. However, if we are look to find a similar document given a document, taking time to compare the document with all the others is time consuming. Instead we can use min hashing to speed up document comparisons. For this assignment the true similarity between two documents is the actual jaccard similarity. We will compare this to the estimated jaccard similarity calculated from minhashing.

External Sources

- Ekzhu
  - *MinHash* algorithm and estimated jaccard similarity measurement. I also used their algorithm to compute the actual jaccard similarity
  - *https://github.com/ekzhu/datasketch*
- NLTK 3.0.5
  - *Nltk.tokenize* and *nltk.corpus.stopwords* helped in preprocessing the data.
- NumPy 1.9.2
  - Required *numpy.array* format for NLTK
- BeautifulSoup4.4.0
  - Used when parsing the article information from the html files.

Data Formatting

Instead of the feature vectors being an array of values, such as a 1 or 0 if a word exists, the feature vectors were instead represented sparsely and in strings. For example, a document containing these words could be represented like so.

```
data1 = ['minhash', 'is', 'a', 'probabilistic', 'data', 'structure', 'for',
         'estimating', 'the', 'similarity', 'between', 'datasets']
```

It wasn't difficult to format the data like this considering we had this information in a similar for before converting it to numbers (0, 1, or maybe tf-idf value) in assignments 2 and 3.

No n-shingles/n-grams above n=1 were used in this assignment. For more accurate similarity measures that account for some semantic context would use n-grams.

Efficiency and Efficacy

The time for the minhashing and estimating jaccard similarity was tracked for a different number of permutations. A simple mean square error between the true and estimated jaccard similarities for every comparison between two documents was used.

Num of permutations (k)	Time	MSE
16	85.58s	0.00265
32	86.09s	0.00123
64	87.36s	0.000679
128	98.6s	0.000610
256	93.29s	0.000254

Time to calculate actual jaccard similarity was 43.22s.

Discussion

The efficiency of minhashing might not be shown in this assignment because the actual jaccard similarity took a shorted time to calculate then with minhashing. Perhaps there aren't enough documents to notice the reduced algorithmic big-O complexity of minhashing. With lots of more documents, however, one would expect minhashing should take a much shorter time. Perhaps there wasn't the right amount of memory on OSU CSE machines for minhashing to take advantage of, even though it should use less memory than true jaccard calculations. I would've expected a linear growth in running time for the number of permutations, however it isn't obvious from the slight variations in running time ~2% across trials varying the number of permutations by over 100.

The efficacy of minhashing was measure by MSE, which was extremely low for all trials of k. The low error indicates minhashing can accurately estimate the jaccard similarity. Also, as the number of permutations increased, the MSE decreased linearly expect from the step from 64 to 128 iterations.

File Locations

directory : /home/7/gresham/5243/lab4/

makefile

readme: contains run instructions

report.docx

minhashing.py (from Ekzhu source)

lab4.py