CS 470

Data Mining

Homework 5

Jason Ji

Collaborations: Didn't collaborate with any other students.
Translated the code of minHash from python to Java, original code is linked here:
https://github.com/chrisjmccormick/MinHash

Test Cases Description

The data sets used in this assignment are articles_100/1000/2500/10000.train, which represent a certain number of articles. Specifically, each line of the data represents an article with the article id followed by its content. Among these articles, some are plagiarisms of each other (where they have been just slightly modified). These near-duplicates are specified in articles_100/1000/2500/10000.truth. An example of an article is shown below:

t980 A man was shot dead and fifteen others injured when Zambian policemen clashed with citizens rioting in protest against alleged ritual murders by a local businessman, police said Monday. U.S. President George W. Bush expressed confidence on Monday about passing an immigration bill and said a Senate vote of no-confidence in Alberto Gonzales would have no bearing on his service as attorney general. French President Nicolas Sarkozy announced Tuesday in Washington that he would visit China later this month, joined by his wife, Carla Bruni-Sarkozy. These columns for release Tuesday, April 2, 2002 are moving today to clients of the New York Times News Service. Media and entertainment giant Viacom Inc. said Wednesday it may split into two divisions with one focussing on "growth" and the other, more traditional arm, aiming for "value". I don't know if "Harry Potter and the Order of the Phoenix" is a good movie – I haven't seen it. But I'm pretty certain that it shouldn't be judged as a movie at all. It is a visual representation of a book in which millions of people are Two car bombs blew up Monday in the working class town of Yehud, just hours after three Palestinian militants were killed by missiles fired by an Israeli helicopter in a pinpointed attack. Australia's farmers should remember rising fuel prices were hitting farmers worldwide and not just them, Deputy Prime Minister and National Party Leader John Anderson said Tuesday.

Implementation

I chose to implement option 1 of this assignment, which is to convert the original code written in Python into Java. Essentially, the MinHash algorithm detects the similarity between articles and identifies articles that are near duplicates of each other. The MinHash algorithm works as follows:

First, it converts every article into shingles that consist of 3 consecutive words. Then it generates random hash functions to mimic the random permutation of the shingles. Using these random functions, it then computes the signature matrix for each article by finding the minimum hashed value. Finally, it compares the MinHash signatures by counting the number of components in which the signatures are equal. It divides the number of matching components by the signature length to get a similarity value. Finally, near duplicates are detected by displaying pairs of documents/signatures with a similarity greater than a threshold (0.5).

Experiment and Results

I applied my minHash Java algorithm to all input datasets and compare the detected near duplicate articles with the truth files. It turned out that my Java implementation is fairly accurate, with perfect true positive and 0 false positive rates. Below is the result of my code with different inputs.

```
Converting documents to shingles...
Shingling 100 docs took 0.175554333 sec.
Average shingles per doc: 248.070000
Calculating Jaccard Similarities...
  (0 / 100)
Calculating all Jaccard Similarities took 0.245787958 sec.
Generating random hash functions...
Generating MinHash signatures for all documents...
Generating MinHash signatures took 0.052215375 sec.
Comparing all signatures...
Comparing MinHash signatures took 0.002321709 sec.
List of Document Pairs with J(d1,d2) more than 0.5
Values shown are the estimated Jaccard similarity and the actual Jaccard similarity.
                Est.Jaccard Similarity    Actual Jaccard Similarity
 t980 --> t2023   1.00                     0.98

t1088 --> t5015   1.00                     0.98

t1297 --> t4638   1.00                     0.98

t1768 --> t5248   1.00                     0.98

t1952 --> t3495   1.00                     0.98

True positives: 5 / 5
False positives: 0
```

Input = article_100.train

```
Converting documents to shingles...
Shingling 1000 docs took 0.457264125 sec.
Average shingles per doc: 252.239000
Calculating Jaccard Similarities...
  (0 / 1000)
  (100 / 1000)
  (200 / 1000)
  (300 / 1000)
  (400 / 1000)
  (500 / 1000)
  (600 / 1000)
  (700 / 1000)
  (800 / 1000)
  (900 / 1000)
Calculating all Jaccard Similarities took 15.916533 sec.
Generating random hash functions...
Generating MinHash signatures for all documents...
Generating MinHash signatures took 0.148397083 sec.
Comparing all signatures...
Comparing MinHash signatures took 0.079837791 sec.
List of Document Pairs with J(d1,d2) more than 0.5
Values shown are the estimated Jaccard similarity and the actual Jaccard similarity.
                  Est.Jaccard Similarity    Actual Jaccard Similarity
 t980 --> t2023   1.00                      0.98

t1088 --> t5015   1.00                      0.98

t1297 --> t4638   1.00                      0.98

t1768 --> t5248   1.00                      0.98

t1952 --> t3495   1.00                      0.98

t2535 --> t8642   1.00                      0.98

t2839 --> t9303   1.00                      0.98

t2957 --> t7111   1.00                      0.98

t3268 --> t7998   0.90                      0.98

t3466 --> t7563   1.00                      0.98

True positives: 10 / 10
False positives: 0
```

Input = article_1000.train

```
Converting documents to shingles...
Shingling 2500 docs took 0.722826417 sec.
Average shingles per doc: 252.386800
Generating random hash functions...
Generating MinHash signatures for all documents...
Generating MinHash signatures took 0.386723666 sec.
Comparing all signatures...
Comparing MinHash signatures took 0.341094833 sec.
List of Document Pairs with J(d1,d2) more than 0.5
Values shown are the estimated Jaccard similarity and the actual Jaccard similarity.
                  Est.Jaccard Similarity    Actual Jaccard Similarity
 t787 --> t9596   0.90                      0.98

 t906 --> t5442   1.00                      0.98

 t969 --> t6244   1.00                      0.98

 t980 --> t2023   1.00                      0.98

t1088 --> t5015   0.90                      0.98

t1297 --> t4638   1.00                      0.98

t1768 --> t5248   0.90                      0.98

t1952 --> t3495   1.00                      0.98

t2535 --> t8642   1.00                      0.98

t2839 --> t9303   1.00                      0.98

t2957 --> t7111   1.00                      0.98

t3268 --> t7998   1.00                      0.98

t3466 --> t7563   1.00                      0.98

t3575 --> t8979   1.00                      0.98

t3725 --> t4099   1.00                      0.98

t4467 --> t6205   0.90                      0.98

t4530 --> t7907   0.90                      0.98

t5551 --> t7693   1.00                      0.98

t7270 --> t8387   0.90                      0.98

t7527 --> t8101   1.00                      0.98

True positives: 20 / 20
False positives: 0
```

Input = article_2500.train

```
Converting documents to shingles...
Shingling 10000 docs took 1.713414916 sec.
Average shingles per doc: 251.989300
Generating random hash functions...
Generating MinHash signatures for all documents...
Generating MinHash signatures took 0.701445083 sec.
Comparing all signatures...
Comparing MinHash signatures took 2.775206291 sec.
List of Document Pairs with J(d1,d2) more than 0.5
Values shown are the estimated Jaccard similarity and the actual Jaccard similarity.
                Est.Jaccard Similarity    Actual Jaccard Similarity
t6520 --> t6906   1.00                    1.00

 t980 --> t2023   0.90                    0.98

t9248 --> t4211   1.00                    1.00

t4969 --> t2390   1.00                    1.00

t8090 --> t1898   1.00                    1.00

 t379 --> t3446   1.00                    1.00

t8805 --> t8306   0.90                    0.98

t6092 --> t3783   1.00                    1.00

t1513 -->  t764   1.00                    1.00

t5442 -->  t906   0.80                    0.98

t4530 --> t7907   1.00                    0.98

t8826 --> t9285   1.00                    0.98

t3136 --> t8469   1.00                    1.00

t1726 --> t9170   1.00                    1.00

t5239 --> t2001   1.00                    1.00

t7958 --> t1621   1.00                    1.00

 t104 --> t4172   1.00                    1.00

t2475 --> t1142   1.00                    0.98

t3072 --> t7923   1.00                    1.00

t7563 --> t3466   1.00                    0.98
```

...Rows omitted...

```
True positives: 80 / 80
False positives: 0
```

Input = article_10000.train

Insights and Lessons Learned

From the experiment results shown above, we can see that my Java minHash algorithm has high accuracy in duplicate detection. For all input files, it produces a perfect true positive rate with no false positive. It also runs very fast, with the longest taking around 3 seconds to compute the Jaccard similarities (estimated).

When looking at the estimated Jaccard Similarity and the actual Jaccard similarity, we can see that minHash is a good estimation. Most of the estimated score is very close to the actual similarity score, and it helps us accurately classify the duplicates. In addition, if we look at the result of processing article_1000.train, the computation of actual Jaccard Similarity takes about 15 seconds, while the computation of estimated Jaccard Similarity through minHash only takes about 0.15 seconds. This is a huge improvement in efficiency while maintaining accuracy.

I also learned a few lessons when converting from Python to Java. For example, some of the packages used by the original Python code are not available in Java. Java doesn't have the binascii package, so I had to use the StandardCharsets package to hash shingles to 32-byte integers. I chose this package because it is a standardized way of converting a String to an array of bytes. Intersection and union are also implemented differently in Java compared to Python. It involves creating two additional variables to store the intersection and union of two sets. It seems that Python is more powerful and convenient with some operations, and offers more sophisticated libraries that are easier to use than Java. In addition, I initially used ArrayList of Double objects to store the data, which resulted in running out of Java heap memory. I then changed to arrays with doubles and fixed the issue. We need to consider how the data is stored when translating code that deals with large data.