Data Mining Assignment 2

Author: Lukas Gust

Due: Jan. 30th

1 Creating k-grams

Description:

You will construct several types of k-grams for all documents. All documents only have at most 27 characters: all lower case letters and space.

[G1] Construct 2-grams based on characters, for all documents.

[G2] Construct 3-grams based on characters, for all documents.

[G3] Construct 2-grams based on words, for all documents.

Remember, that you should only store each k-gram once, duplicates are ignored.

A: How many distinct k-grams are there for each document with each type of k-gram?

B: Compute the Jaccard similarity between all pairs of documents for each type of k-gram.

Solution:

A: Using a simple python script and built in set data structure we get the following number of k-grams for each document for each k-gram.

```
G1
D1.txt: # of grams:266
D2.txt: # of grams:265
D3.txt: # of grams:258
D4.txt: # of grams:259

G2
D1.txt: # of grams:815
D2.txt: # of grams:804
D3.txt: # of grams:757
D4.txt: # of grams:771

G3
D1.txt: # of grams:308
D2.txt: # of grams:307
D3.txt: # of grams:294
D4.txt: # of grams:260
```

B: Using simple python built in set methods we can compute the union and intersection of these sets to compute the Jaccard similarity for every combination of documents for each type of k-gram.

```
Jaccard similarity for G1
Similarity (D1.txt, D2.txt): 0.9962406015037594
Similarity (D1.txt, D3.txt): 0.9124087591240876
Similarity (D1.txt, D4.txt): 0.7326732673267327
Similarity (D2.txt, D3.txt): 0.9157509157509157
Similarity (D2.txt, D4.txt): 0.7293729372937293
Similarity (D3.txt, D4.txt): 0.6950819672131148

Jaccard similarity for G2
Similarity (D1.txt, D2.txt): 0.9648058252427184
Similarity (D1.txt, D3.txt): 0.7312775330396476
Similarity (D1.txt, D4.txt): 0.35555555555555557
Similarity (D2.txt, D3.txt): 0.7402452619843924
Similarity (D2.txt, D4.txt): 0.3496143958868895
Similarity (D3.txt, D4.txt): 0.3510167992926614

Jaccard similarity for G3
Similarity (D1.txt, D2.txt): 0.7672413793103449
Similarity (D1.txt, D3.txt): 0.2754237288135593
Similarity (D1.txt, D4.txt): 0.012477718360071301
Similarity (D2.txt, D3.txt): 0.3008658008658009
Similarity (D2.txt, D4.txt): 0.016129032258064516
Similarity (D3.txt, D4.txt): 0.014652014652014652
```

As we can see, D1 and D2 are very similar. And the word gram (G3) doesn't do as well to capture the similarity.

2 Min Hashing

Description:

We will consider a hash family $\mathcal{H}$ so that any hash function $h \in \mathcal{H}$ maps from $h : \{k\text{-grams}\} \to [m]$ for m large enough.

A: Using grams G2, build a min-hash signature for document D1 and D2 using $t = \{20, 60, 150, 300, 600\}$ hash functions. For each value of t report the approximate Jaccard similarity between the pair of documents D1 and D2, estimating the Jaccard similarity:

$$\hat{\mathrm{JS}}_t(a, b) = \frac{1}{t} \sum_{i=1}^{t} \begin{cases} 1 & \text{if } a_i = b_i \\ 0 & \text{if } a_i \neq b_i . \end{cases}$$

B: What seems to be a good value for t? You may run more experiments. Justify you answer in terms of both accuracy and time.

Solution:

A: Using python's built-in hash function with a random salt a, we have $h_a(x) = (\mathrm{hash}(x) + a)\ \&\ m$, where m is very large. We run the experiment 100 times to compute an average Jaccard similarity.

```
t=20:  0.9770000000000008
t=60:  0.9640000000000011
t=150: 0.9625999999999999
t=300: 0.9650333333333329
t=600: 0.9648666666666668
```

B: As we can see, and from running this multiple times, $t = 60$. This is because it produces a small error for the amount of time it takes to compute the min-hash for $t = 60$ compared to $t = 600$. $t = 150$ isn't too bad either, maybe somewhere in between the two would be optimal. Note: The m used in the experiment is very large due to the hash function that we used. If it is much smaller then the Jaccard similarity does not approximate well. e.g. if $m = 10000$ the approximate Jaccard similarity is 1.0 for all t from above.

End