CW1

Luke Marden

November 2019

1 Introduction

1. calculateFeatureVector - This works out the amount of each word in a dictionary is used in a passage of text
2. documentSimularityDistance - This uses the above algorithm and works out the similarity of two pieces of texts based on the closeness of their feature vectors
3. calculateClosestMatch - This calculates the closest document to each other.

2 calculateFeatureVector

Algorithm 1 calculateFeatureVector(**A**,*lengthA*,**Q**,*lengthQ*) **return F**

Require: An array **A** of length *lengthA* and **Q** of length *lengthQ*
Ensure: An integer array **F**, is equal to how common a word occurs from **Q** in **A**

```
1: for i ← 0 to lengthQ do
2:     F_i ← 0                          ▷ initialise every value in the array to zero
3: for i ← 0 to lengthA do
4:     for j ← 0 to lengthQ do
5:         if A_i = Q_j then
6:             F_j ← = F_j + 1
   return F
```

Algorithm Analysis

1. Determining the fundamental operation:
   $A_i = Q_j$
2. Case explanation:
   Assumed worst case

3. Run time complexity function:

$$f(n) = \sum_{i=0}^{lengthQ} \sum_{j=0}^{lengthA} 1 \tag{1}$$

4. Order of the algorithm:

3 calcaluteDocumentSimularityDistance

Algorithm 2 calcaluteDocumentSimularityDistance(**F1**,n,**F2**,n) **return** DSD

Require: Two integer arrays **F1** and **F2**, both of length n
Ensure: An integer DSD, is equal to how similar **F1** and **F2** are.

```
1: DSD ← 0                          ▷ initialise every value in the array to zero
2: for i ← 0 to n do
3:     if F1_i >= F2_i then
4:         DSD ← DSD + (F1_i - F2_i)
5:     else
6:         DSD ← DSD + (F2_i - F1_i)
   return DSD
```

4 calculateClosestMatch

Algorithm Analysis

1. Determining the fundamental operation:

2. Case explanation:
   Assumed worst case

3. Run time complexity function:

4. Order of the algorithm:

Algorithm 3 calculateClosestMatch(**Q**,x,**S**,y,z) **return closestMatches**

Require: Two string arrays, one linear, **Q** of length x and the other two dimensional, **S** of length y by z

Ensure: An integer DSD, is equal to how similar **F1** and **F2** are.

```
1: for i ← 0 to y do
2:     ClosestMatches_i ← 0          ▷ initialise every value in the array to zero
3: for i ← 0 to y do
4:     for j ← 0 to y do
5:         DSD_j ← 0                 ▷ initialise every value in the array to zero
6:     for j ← 0 to y do
7:         xFVector_j ← 0            ▷ initialise every value in the array to zero
8:     position ← 0
9:     min ← 1000
10:    for j ← 0 to y do
11:        for k ← 0 to y do
12:            SFVector_k ← calculateFeatureVector( S_k, Q)
13:        DSD_j ← calculateDocumentSimularityDistance( XFVector, SFVector)
14:        if DSD_j = 0 then
15:            continue
16:        else
17:            if min > DSD_j then
18:                min ← DSD_j
19:                position ← j
    ClosestMatches_i ← position
    return ClosestMatches
```