Similitud entre documents

Programació Declarativa, Aplicacions
8 de novembre del 2024

Wilber Eduardo Bermeo Quito
Universitat de Girona

Jordi Badia Auladell
41591544T
jordibadiauladell@gmail.com

Aniol Juanola Vilalta
41559862N
u1978893@campus.udg.edu

Continguts

a) Funcions auxiliars

1. Filtració de paraules

```
private def filterWords(string: String): Array[String] = {
  string
    .toLowerCase
    .replaceAll("[-_'.,;:()!?¿¡\\[\\]{}1234567890$%&/\\\\@|#ºª+*^<>€=\\n\\t\\r]", " ")
    .split("\\W+")
    .filter(_.nonEmpty)
}
```

Descripció: preprocessa un text en forma d'`String` per retornar una llista normalitzada de les paraules que el conformen.

Paràmetres:

- `string`: `String` amb el text a preprocessar.

Retorna: una llista d'`String` amb les paraules del text entrat en a minúscula i suprimint tots aquells caràcters que no siguin lletres per espais.

Exemple d'execució:

Input:

```
filterWords("hola! ¿Que tal? Quin; exemple!mes;xulo.11 anys@")
```

Output:

```
Array(hola, que, tal, quin, exemple, mes, xulo, anys)
```

2. Mostra per consola

```
private def printFirstNFreq(list: List[(String, Int)]): Unit = {
  val totalWordCount = list.map(_._2).sum
  printf("Num de Paraules:\t%d\t\t\t\tDiferents:\t%d\n", totalWordCount, list.length)
  printf("Paraules\t\t\tOcurrencies\t\t\tFrequencia\n")
  println("------------------------------------------------------------")
  for (i <- list.take(10)) {
   printf("%-" + 20 + "s%-" + 20 + "d%.2f\n", i._1, i._2, (i._2.toDouble / totalWordCount)
* 100)
  }
}
```

Descripció: mostra per consola algunes estadístiques de la llista entrada. Mostra el total de paraules i el total de paraules diferents. Mostra també les 10 primeres paraules amb les seves respectives ocurrències i freqüències.

Paràmetres:

- `list`: `List[(String, Int)]` on l'enter és el nombre d'ocurrències de la cadena de caràcters.

b) Freqüències de paraules

```
def freq(string: String, print: Boolean = true): List[(String, Int)] = {
  val aux = filterWords(string)
    .groupBy(identity)
    .view.mapValues(_.length)
    .toList
    .sortBy(-_._2) //ordena descendentment pel segon element de la tupla

  if (print)
    printFirstNFreq(aux)

  aux
}
```

Descripció: preprocessa i separa les paraules d'un text, mapejant-les al nombre d'ocurrències de cada una.

Paràmetres:

- `string`: `String` amb el text a comptabilitzar-ne les paraules.
- `print`: `Boolean` opcional per imprimir o no el resultat de la funció.

Retorna: una llista de tuples `(String, Int)` amb cada paraula i les seves ocurrències, respectivament.

Exemple d'execució:

Input:

```
freq("hola hola adeu adeu adeu vinga vinga jaja jeje jiji")
```

Output:

```
Num de Paraules:  10        Diferents:  6
Paraules      Ocurrencies     Frequencia
---------------------------------------------
adeu                3                   30,00
hola                2                   20,00
vinga               2                   20,00
jaja                1                   10,00
jiji                1                   10,00
jeje                1                   10,00
```

c) Sense *stop-words*

```
def nonstopfreq(string: String, stopWords: List[String], print: Boolean = true):
List[(String, Int)] = {
  val aux = freq(string, print = false)
    .filter { case (w, _) => !stopWords.contains(w) }

  if (print)
    printFirstNFreq(aux)

  aux
}
```

Descripció: preprocessa i separa les paraules d'un text, mapejant-les al nombre d'ocurrències de cada un, i treient-ne una sèrie de paraules que es consideren irrellevants.

Paràmetres:

- `string`: `String` amb el text a comptabilitzar-ne les paraules.
- `stopWords`: `List[String]` és una llista de paraules considerades irrellevants.
- `print`: `Boolean` opcional per imprimir o no el resultat de la funció.

Retorna: una llista de tuples (`String`, `Int`) amb cada paraula i les seves ocurrències, respectivament, excloent aquelles claus que figuren a la llista `stopWords`.

Exemple d'execució:

Input:

```
nonstopfreq("hola hola adeu adeu adeu vinga vinga jaja jeje jiji", List("hola",
"jaja")
```

Output:

```
                  Num de Paraules:  7       Diferents:  4
                  Paraules      Ocurrencies     Frequencia
                  ---------------------------------------------
                  adeu                3                   42,86
                  vinga               2                   28,57
                  jiji                1                   14,29
                  jeje                1                   14,29
```

d) Distribució de paraules

```
def paraulesfreqfreq(string: String, print: Boolean = true): Map[Int, Int] = {
  val aux = freq(string, false)

  val freqCounts = aux
    .groupBy(_._2)
    .view.mapValues(_.length).toMap
    .toSeq
    .sortBy(_._2)

  if (print) {
    println("\nLes 10 frequencies mes frequents:")
    freqCounts.takeRight(10).reverse.foreach {
      case (freq, count) =>
        printf("%d paraules apareixen %d vegades\n", count, freq)
    }
    println()
    println("Les 5 frequencies menys frequents:")
    freqCounts.take(5).foreach {
      case (freq, count) =>
        printf("%d paraules apareixen %d vegades\n", count, freq)
    }
  }

  freqCounts.toMap
}
```

Descripció: preprocessa un text i compta les ocurrències de les diferents freqüències que constitueixen les paraules que el formen.

Paràmetres:

- `string`: `String` amb el text a comptabilitzar-ne les freqüències de les paraules.
- `print`: `Boolean` opcional per imprimir o no el resultat de la funció.

Retorna: un mapa de clau `Int` i valor `Int` amb cada freqüència i les seves ocurrències, respectivament..

Exemple d'execució:

Input:

```
paraulesfreqfreq("hola hola adeu adeu adeu vinga vinga jaja jeje jiji")
```

Output:

```
                        Les 10 frequencies mes frequents:
                        3 paraules apareixen 1 vegades
                        2 paraules apareixen 2 vegades
                        1 paraules apareixen 3 vegades

                        Les 5 frequencies menys frequents:
                        1 paraules apareixen 3 vegades
                        2 paraules apareixen 2 vegades
                        3 paraules apareixen 1 vegades
```

e) N-grames

```
def ngrames(n: Int, string: String, print: Boolean = true): Seq[(String, Int)] = {
  val aux = filterWords(string)
    .sliding(n)
    .map(_.mkString(" "))
    .toSeq
    .groupBy(identity)
    .view.mapValues(_.length)
    .toSeq
    .sortBy(-_._2)

  if (print) {
    aux.take(10).foreach { case (string, count) =>
      printf("%-" + 40 + "s%d\n", string, count)
    }
  }

  aux.toList
}
```

Descripció: preprocessa i separa les paraules d'un text en ngrames de mida `n`, mapejant-les al nombre d'ocurrències de cada un.

Paràmetres:

- `n`: `Int` amb el nombre de paraules de cada ngrama.
- `string`: `String` amb el text a comptabilitzar-ne les freqüències de cada ngrama.
- `print`: `Boolean` opcional per imprimir o no el resultat de la funció.

Retorna: una seqüència de tuples (`String`, `Int`) amb cada ngrama i les seves ocurrències, respectivament.

Exemple d'execució:

Input:

```
ngrames(3, "hola hola adeu hola hola adeu vinga jaja jeje jiji")
```

Output:

```
                    hola hola adeu                          2
                    jaja jeje jiji                          1
                    vinga jaja jeje                         1
                    hola adeu hola                          1
                    adeu vinga jaja                         1
                    hola adeu vinga                         1
                    adeu hola hola                          1
```

f) Vector *space model*

```
def cosineSim(text1: String, text2: String, n: Int, stopWords: List[String] = List()):
Double = {
  // Freq
  val freq1 = if (n == 1) nonstopfreq(text1, stopWords, print = false) else ngrames(n,
text1, print = false)
  val freq2 = if (n == 1) nonstopfreq(text2, stopWords, print = false) else ngrames(n,
text2, print = false)

  // Frequency normalization
  val normalizedFreq1 = normalizedFrequencies(freq1)
  val normalizedFreq2 = normalizedFrequencies(freq2)

  // Unified set
  val allWords = (normalizedFreq1.keySet ++ normalizedFreq2.keySet).toList

  // Aligned vectors generation
  val vector1 = allWords.map(word => normalizedFreq1.getOrElse(word, 0.0))
  val vector2 = allWords.map(word => normalizedFreq2.getOrElse(word, 0.0))

  // Scalar product and vector magnitudes
  val dotProduct = vector1.zip(vector2).map { case (a, b) => a * b }.sum
  val magnitude1 = math.sqrt(vector1.map(a => a * a).sum)
  val magnitude2 = math.sqrt(vector2.map(b => b * b).sum)

  if (magnitude1 == 0 || magnitude2 == 0) 0.0
  else dotProduct / (magnitude1 * magnitude2)
}
```

Descripció: donats dos `String`, els preprocessa ignorant-ne les *stopwords* i calcula el coeficient de similitud de cosinus.

Paràmetres:

- `text1`: `String` amb el primer text a comparar.
- `text2`: `String` amb el segon text a comparar.
- `n` : `Int` nombre n de valors a utilitzar per la similitud de cosinus. Si $n = 1$, es tenen en compte les *stopwords* i es filtren. Si $n > 1$, s'utilitza la funció `ngrama` per a generar la freqüència.
  - **Precondició:** $n \geq 1$.
- `stopWords`: `List[String]` amb les paraules a ignorar.

Retorna: Coeficient de similitud de cosinus (entre 0 i 1).

Exemple d'execució:

Input:

```
println(cosineSim("hola", "hola", 1))
println(cosineSim("hola", "adeu", 1))
println(cosineSim("hola adeu", "adeu hola", 1))
println(cosineSim("hola adeu", "adeu hola", 2))
println(cosineSim("hola hola hola adeu", "hola hola adeu", 1))
println(cosineSim("hola hola hola adeu", "hola hola adeu", 2))
println(cosineSim("holap", "holak", 1))
println(cosineSim("hola a b c d e", "hola f g h i j k l m n o p q", 1))
println(cosineSim("hola a b c d e", "hola f g h i j k l m n o p q", 3))
```

Output:

```
1.0
0.0
0.9999999999999998
0.0
0.9899494936611666
0.9486832980505138
0.0
0.11322770341445959
0.0
```

1. Normalize frequency

```
private def normalizedFrequencies(words: Seq[(String, Int)]): Map[String, Double] = {
  val wordCounts = words.groupBy(_._1).view.mapValues(_.map(_._2).sum).toMap
  val maxFreq = wordCounts.values.max.toDouble

  wordCounts.map { case (word, freq) => (word, freq / maxFreq) }
}
```

Descripció: normalitza les freqüències d'un conjunt de paraules.

Paràmetres:

- `words`: `Seq[(String,Int)]` amb les paraules i les corresponents freqüències.

Retorna: un mapa de clau `String` i valor `Double` amb cada paraula i la seva freqüència normalitzada.

g) Taula de comparació entre els diferents documents

Nota: en els digrames i trigrames, a part de per ser coherents amb l'enunciat, s'ha decidit deixar-hi les *stopwords* ja que son importants a l'hora d'avaluar frases (o sintaxis), i no només paraules (o lèxic).

1. cosineSim (treient *stopwords*)

	pg2500.txt	pg74-net.txt	pg2500-net.txt	pg11.txt	pg74.txt	pg11-net.txt	pg12.txt	pg12-net.txt
pg2500.txt	1,000	0,265	0,971	0,277	0,299	0,209	0,261	0,198
pg74-net.txt	0,265	1,000	0,269	0,215	0,989	0,219	0,209	0,210
pg2500-net.txt	0,971	0,269	1,000	0,208	0,268	0,213	0,201	0,202
pg11.txt	0,277	0,215	0,208	1,000	0,259	0,952	0,876	0,824
pg74.txt	0,299	0,989	0,268	0,259	1,000	0,218	0,247	0,208
pg11-net.txt	0,209	0,219	0,213	0,952	0,218	1,000	0,832	0,864
pg12.txt	0,261	0,209	0,201	0,876	0,247	0,832	1,000	0,963
pg12-net.txt	0,198	0,210	0,202	0,824	0,208	0,864	0,963	1,000

2. Digrama (sense treure *stopwords*)

	pg2500.txt	pg74-net.txt	pg2500-net.txt	pg11.txt	pg74.txt	pg11-net.txt	pg12.txt	pg12-net.txt
pg2500.txt	1,000	0,646	0,979	0,499	0,675	0,433	0,515	0,448
pg74-net.txt	0,646	1,000	0,649	0,590	0,991	0,585	0,636	0,635
pg2500-net.txt	0,979	0,649	1,000	0,449	0,652	0,433	0,465	0,449
pg11.txt	0,499	0,590	0,449	1,000	0,620	0,969	0,799	0,754
pg74.txt	0,675	0,991	0,652	0,620	1,000	0,583	0,665	0,631
pg11-net.txt	0,433	0,585	0,433	0,969	0,583	1,000	0,754	0,772
pg12.txt	0,515	0,636	0,465	0,799	0,665	0,754	1,000	0,967
pg12-net.txt	0,448	0,635	0,449	0,754	0,631	0,772	0,967	1,000

3. Trigrama (sense treure *stopwords*)

	pg2500.txt	pg74-net.txt	pg2500-net.txt	pg11.txt	pg74.txt	pg11-net.txt	pg12.txt	pg12-net.txt
pg2500.txt	1,000	0,110	0,931	0,199	0,204	0,058	0,203	0,065
pg74-net.txt	0,110	1,000	0,116	0,153	0,962	0,165	0,176	0,189
pg2500-net.txt	0,931	0,116	1,000	0,057	0,112	0,061	0,065	0,069
pg11.txt	0,199	0,153	0,057	1,000	0,255	0,915	0,381	0,244
pg74.txt	0,204	0,962	0,112	0,255	1,000	0,159	0,274	0,183
pg11-net.txt	0,058	0,165	0,061	0,915	0,159	1,000	0,245	0,266
pg12.txt	0,203	0,176	0,065	0,381	0,274	0,245	1,000	0,920
pg12-net.txt	0,065	0,189	0,069	0,244	0,183	0,266	0,920	1,000

4. Conclusió

Com a norma general, el digrama tendeix a trobar mes similituds màximes que el consinesim, i a la vegada, aquest en troba més que el trigrama. Això es deu a que en comparar el lèxic (cosinesim) s'obtenen resultats en base a la freqüència de paraules útils (sense *stopwords*), mentre que el digrama i el trigrama comparen sintaxis.

Si bé és cert que digrama troba més similituds (ja que el *dataset* pot tenir molts parells iguals d'*stopwords*, com "don t"), el trigrama és més exigent en la seva classificació: en trobar dos textos similars, els considera molt similars (> 0.9) i quan no els troba similars, els considera realment diferents (< 0.3), potser també per les seqüències d'*stopwords*.