Universitat de Girona

Similitud entre documents - Programació declarativa. Aplicacions

Curs 2022/23

GEINF

Similitud entre documents

Gabriel Lopez 41511387K – u1968394@campus.udg.edu

Angel Molero 41512839R – u1967593@campus.udg.edu

Professor: Mateu Villaret Auselle

Índex

A Freqüència de paraules **2**

B Sense stop-words **3**

C Distribució de paraules **4**

D Ngrames **5**

E Vector space model **6**

A Freqüència de paraules

- Mètode **freq**:

```scala
/**
 *  Calcula la cantidad que aparecen las diferentes palabras del texto sin contar con las stop-words.
 *
 * @param texto : texto a analizar.
 * @return List[(String, Int)] : lista de palabras y la cantidad que aparece.
 */
def freq(texto: String): List[(String, Int)] = {
  val modtexto = texto.toLowerCase().replaceAll( regex = "[^a-z]",  replacement = " ").split( regex = " ").filter(_.nonEmpty).toList
  val palNoRep = modtexto.distinct
  val freq = palNoRep.map(p => (p, modtexto.count(x=>x==p))).sortWith(_._2 > _._2)
  freq
}
```

- Mètode **printFreq**:

```scala
/**
 * La función printFreq muestra la frecuencia de las palabras leidas.
 * @param frequency una lista de tuplas de tamaño 2 con el elemento de la
 *                  tupla siendo una cadena de caracteres y el segundo un número entero.
 * @return void.
 */
def printFreq(frequency: List[(String, Int)]): Unit = {
  var numParaules = 0
  frequency.foreach(x => numParaules = numParaules + x._2)
  println("Num de Paraules: " + numParaules + " Diferents: " + frequency.length)
  println("Paraules ocurrencies frequencia")
  println("----------------------------------------")
  for ((str, i) <- frequency.take(10)) {
    val n = calculFreq(numParaules, i)
    println(str + "\t" + i + "\t" + redondear(n, ndigits = 2))
  }
}
```

- Codi **main**:

```scala
def main(args: Array[String]): Unit = {
  val texto = scala.io.Source.fromFile("src/main/scala/pg11.txt").mkString

  printFreq(freq(texto))
```

- Resultat de l'execució:

```
Num de Paraules: 30419 Diferents: 3007
Paraules ocurrencies frequencia
--------------------------------------
the 1818    5.98
and 940 3.09
to  809 2.66
a   690 2.27
of  631 2.07
it  610 2.01
she 553 1.82
i   545 1.79
you 481 1.58
said    462 1.52
```

B Sense stop-words

- Mètode **nonstopfreq**:

```scala
/**
 *  Calcula la cantidad que aparecen las diferentes palabras del texto sin contar con las stop-words.
 *
 * @param texto : texto a analizar.
 * @return List[(String, Int)] : lista de palabras y la cantidad que aparece.
 */
def nonstopfreq(texto: String): List[(String, Int)] = {
  val stopwords = scala.io.Source.fromFile("src/main/scala/english-stop.txt").mkString
  val stop = stopwords.split( regex = "\n").toList
  val modtexto = texto.toLowerCase().replaceAll( regex = "[^a-z]",  replacement = " ").split( regex = " ").filter(_.nonEmpty)
  val f = modtexto.filter(!stop.contains(_))
  f.distinct.map(p => (p, f.count(x => x == p))).sortWith(_._2 > _._2).toList
}
```

- Mètode **printFreq**:

```scala
/**
 * La función printFreq muestra la frecuencia de las palabras leidas.
 * @param frequency una lista de tuplas de tamaño 2 con el elemento de la
 *                  tupla siendo una cadena de caracteres y el segundo un número entero.
 * @return void.
 */
def printFreq(frequency: List[(String, Int)]): Unit = {
  var numParaules = 0
  frequency.foreach(x => numParaules = numParaules + x._2)
  println("Num de Paraules: " + numParaules + " Diferents: " + frequency.length)
  println("Paraules ocurrencies frequencia")
  println("--------------------------------------")
  for ((str, i) <- frequency.take(10)) {
    val n = calculFreq(numParaules, i)
    println(str + "\t" + i + "\t" + redondear(n, ndigits = 2))
  }
}
```

- Codi **main**

```scala
def main(args: Array[String]): Unit = {
  val texto = scala.io.Source.fromFile("src/main/scala/pg11.txt").mkString

  printFreq(nonstopfreq(texto))
  println()
```

- Resultat Execució:

```
Num de Paraules: 10038 Diferents: 2623
Paraules ocurrencies frequencia
---------------------------------------
alice   403 4.01
gutenberg   93  0.93
project 87  0.87
queen   75  0.75
thought 74  0.74
time    71  0.71
king    63  0.63
turtle  59  0.59
began   58  0.58
tm  57  0.57
```

C Distribució de paraules

- Mètode **paraulafreqfreq**:

```scala
/**
 * Calcula cuantas palabras aparecen el mismo número de veces.
 *
 * @param texto : texto1
 * @return List[(Int, Int)] : la frecuencia de aparición de los diferentes palabras.
 */
def paraulafreqfreq(texto: String): List[(Int, Int)] = {
  val f = freq(texto)
  f.groupBy(_._2).map(x => (x._1, x._2.size)).toList.sortWith(_._2 > _._2)
}
```

- Codi **main**

```scala
def main(args: Array[String]): Unit = {
  val texto = scala.io.Source.fromFile("src/main/scala/pg11.txt").mkString
  val pff = paraulafreqfreq(texto)

  println("Les 10 frequencies mes frequents:")
  pff.take(10).foreach(x => println( x._2 + " paraules apareixen " + x._1 + " vegades"))

  println("\nLes 5 frequencies menys frequents:")
  pff.takeRight(5).foreach(x => println( x._2 + " paraules apareixen " + x._1 + " vegades"))
```

- Resultat Execució:

```
Les 10 frequencies mes frequents:
1330 paraules apareixen 1 vegades
468 paraules apareixen 2 vegades
264 paraules apareixen 3 vegades
176 paraules apareixen 4 vegades
101 paraules apareixen 5 vegades
74 paraules apareixen 8 vegades
72 paraules apareixen 6 vegades
66 paraules apareixen 7 vegades
39 paraules apareixen 9 vegades
35 paraules apareixen 10 vegades

Les 5 frequencies menys frequents:
1 paraules apareixen 68 vegades
1 paraules apareixen 178 vegades
1 paraules apareixen 227 vegades
1 paraules apareixen 940 vegades
1 paraules apareixen 100 vegades
```

D Ngrames

- Mètode **ngrames**:

```scala
/**
 *  Calcula los n grames de un texto.
 *
 * @param n : tamaño del grama.
 * @param texto : texto del que queremos calcular los n grames.
 * @return List[(String, Int)] : lista de n grames y la cantidad que aparece.
 */
def ngrames(n: Int, texto: String): List[(String, Int)] = {
  val modtexto = texto.toLowerCase().replaceAll( regex = "[^a-z]",  replacement = " ").split( regex = " ").filter(_.nonEmpty)
  val grames = modtexto.sliding(n).toList
  val nrep = grames.map(x => (x.mkString(" "), 0))
  nrep.groupBy(_._1).map(x => (x._1, x._2.size)).toList.sortWith(_._2 > _._2)
}
```

- Codi **main**:

```
def main(args: Array[String]): Unit = {
  val texto = scala.io.Source.fromFile("src/main/scala/pg11.txt").mkString

  val ng = ngrames(3, texto)
  println()
  ng.take(10).foreach(x => println(x))
```

- Resultat Execució:

```
(project gutenberg tm,57)
(the mock turtle,53)
(i don t,31)
(the march hare,30)
(said the king,29)
(the project gutenberg,29)
(said the hatter,21)
(the white rabbit,21)
(said the mock,19)
(said to herself,19)
```

E Vector space model

- Mètode **cosinesim**: Per realitzar la comprovació mitjançant digrames i trigrames, s'ha introduit un paràmetre per indicar si volem que s'analitzi paraula per paraula o per ngrames.

```
/**
 *  Calcula lo similitud del cosinus entre dos textos.
 * @param t1 : texto1
 * @param t2 : texto2
 * @return BigDecimal : resultado final de la similitud del cosinus.
 */
def cosinesim(t1: String, t2: String, n: Int = 0): BigDecimal = {
  var l1 = List[(String, Int)]()
  var l2 = List[(String, Int)]()
  if(n == 0){
    l1 = nonstopfreq(t1)
    l2 = nonstopfreq(t2)
  } else {
    l1 = ngrames(n, t1)
    l2 = ngrames(n, t2)
  }
  val totalPal1 = sumatotal(l1)
  val totalPal2 = sumatotal(l2)
  // Calculamos la frecuencia maxima para calcular la frecuencia normalizada de los otros elementos
  val freqNormMax1 = redondear(calculFreq(totalPal1, l1.head._2), ndigits = 2)
  val freqNormMax2 = redondear(calculFreq(totalPal2, l2.head._2), ndigits = 2)
  val freq1 = frecuenciaNormal(l1, freqNormMax1, totalPal1)
  val freq2 = frecuenciaNormal(l2, freqNormMax2, totalPal2)

  // Calculamos el producto escalar
  val numerador = vectorAlignment(freq1.sortBy(_._1), freq2.sortBy(_._1), result = 0.0)//productoEscalar(group)
  // Calculamos el denominador
  val ai = raizSumatorio(freq1)
  val bi = raizSumatorio(freq2)
  redondear(numerador/(ai*bi), ndigits = 3)
}
```

- Mètodes Auxiliars:
  - Mètode **nonstopfreq**: el codi s'ha indicat anteriorment.
  - Mètode **ngrames**: el codi s'ha indicat anteriorment.
  - Mètode **sumatotal**:

```scala
/**
 * Calcula la suma de todos los elementos de una lista.
 *
 * @param l1 : lista de palabras y la cantidad que aparece.
 * @return Int : suma de todos los elementos.
 */
private def sumatotal(l1: List[(String, Int)]): Int = {
  l1.foldLeft(0)((x, y) => (x + y._2))
}
```

  - Mètode **redondear**:

```scala
/**
 * Redondea la frecuencia de las palabras.
 *
 * @param num numero decimal que redondearemos.
 * @param ndigits cantidad de decimales que queremos.
 * @return BigDecimal resultado de redondear
 */
def redondear(num: BigDecimal, ndigits: Int): BigDecimal = {
  num.setScale(ndigits, BigDecimal.RoundingMode.HALF_UP)
}
```

  - Mètode **calculFreq**:

```scala
/**
 * Calcula la frecuencia de las palabras.
 *
 * @param numParaules : número total de palabras.
 * @param i : número de veces que aparece una palabra.
 * @return BigDecimal : frecuencia de la palabra.
 */
private def calculFreq(numParaules: Int, i: Int): BigDecimal = {
  BigDecimal((i.toFloat / numParaules) * 100)
}
```

– Mètode **frecuenciaNormal**:

```scala
/**
 * Calcula la frecuencia normal.
 *
 * @param l1 : lista de palabras y la cantidad que aparece.
 * @param freqNormMax1 : frecuaencia normal máxima.
 * @param sumaTotal : suma total de apariciones de todas las palabras.
 * @return List[(String, BigDecimal)] : lista de palabras y su frecuencia normal.
 */
private def frecuenciaNormal(l1: List[(String, Int)], freqNormMax1: BigDecimal, sumaTotal: Double): List[(String, BigDecimal)] = {
  l1.map(x => (x._1, redondear(BigDecimal((x._2.toFloat / sumaTotal) * 100), ndigits = 2) / freqNormMax1))
}
```

– Mètode **VectorAlignment**:

```scala
/**
 *  Calculamos el producto escalar
 *
 * @param v1 : lista de palabras y su frecuencia.
 * @param v2 : lista de palabras y su frecuencia.
 * @return Double : producto escalar final
 */
def vectorAlignment(v1: List[(String, Double)], v2: List[(String, Double)], result: Double): Double = {
  if (v1.isEmpty || v2.isEmpty) result
  else {
    val comp = v1.head._1.compareTo(v2.head._1)
    if (comp == 0) vectorAlignment(v1.tail, v2.tail, result + v1.head._2 * v2.head._2)
    else if (comp < 0) vectorAlignment(v1.tail, v2, result)
    else vectorAlignment(v1, v2.tail, result)
  }
}
```

– Mètode **raizSumatorio**:

```scala
/**
 * Calcula la raíz cuadrada de la suma de todos los elementos.
 *
 * @param freq1 : lista con todas las palabras y su frequencia.
 * @return Double : resultado de la raiz cuadrada.
 */
private def raizSumatorio(freq1: List[(String, BigDecimal)]): Double = {
  Math.sqrt(freq1.foldLeft(BigDecimal(0))((x, y) => x + y._2 * y._2).doubleValue)
}
```

- Codi **main**: A l'inici del codi hi ha una constant anomenada **n**, amb la qual, podem indicar si volem que cosinesim analitzi paraula per paraula o per ngrames.

```scala
def main(args: Array[String]): Unit = {
  val n = 0;
  val files = getListOfFiles( dir = "src/main/scala/inputFiles/")

  var resultats: Map[(String, String), BigDecimal] = Map()

  for(i <- files.indices){
    val text1 = scala.io.Source.fromFile(files(i)).mkString

    resultats += ((files(i).getName, files(i).getName) -> 1.000)

    for(j <- i + 1 ≤ until < files.length){
      val text2 = scala.io.Source.fromFile(files(j)).mkString
      val aux = cosinesim(text1, text2, n)
      resultats += ((files(i).getName, files(j).getName) -> aux)
      resultats += ((files(j).getName, files(i).getName) -> aux)
    }

  }

  print("\t\t\t\t")
  for(i <- 0 ≤ until < files.length){
    var line = files(i).getName
    while(line.length < 14){
      line += " "
    }
    print(line+"\t")
  }
  println()
```

```
for(i <- files.indices){

  var line = files(i).getName
  while(line.length < 18){
    line += " "
  }
  for(j <- files.indices) {
    line += "%1.3f".format(resultats.find(x => x._1._1 == files(i).getName && x._1._2 == files(j).getName).get._2) + "\t\t\t"
  }
  println(line)
}
```

- Resultat execució ($\mathbf{n = 0}$):

	pg11-net.txt	pg11.txt	pg12-net.txt	pg12.txt	pg2500-net.txt	pg2500.txt	pg74-net.txt	pg74.txt
pg11-net.txt	1,000	0,951	0,864	0,831	0,213	0,210	0,219	0,217
pg11.txt	0,951	1,000	0,823	0,876	0,209	0,278	0,214	0,258
pg12-net.txt	0,864	0,823	1,000	0,962	0,202	0,198	0,208	0,207
pg12.txt	0,831	0,876	0,962	1,000	0,202	0,262	0,208	0,247
pg2500-net.txt	0,213	0,209	0,202	0,202	1,000	0,971	0,269	0,268
pg2500.txt	0,210	0,278	0,198	0,262	0,971	1,000	0,265	0,299
pg74-net.txt	0,219	0,214	0,208	0,208	0,269	0,265	1,000	0,988
pg74.txt	0,217	0,258	0,207	0,247	0,268	0,299	0,988	1,000

- Resultat execució ($\mathbf{n = 2}$):

	pg11-net.txt	pg11.txt	pg12-net.txt	pg12.txt	pg2500-net.txt	pg2500.txt	pg74-net.txt	pg74.txt
pg11-net.txt	1,000	0,965	0,771	0,748	0,425	0,432	0,582	0,581
pg11.txt	0,965	1,000	0,752	0,792	0,439	0,493	0,585	0,613
pg12-net.txt	0,771	0,752	1,000	0,962	0,443	0,446	0,629	0,626
pg12.txt	0,748	0,792	0,962	1,000	0,453	0,504	0,626	0,651
pg2500-net.txt	0,425	0,439	0,443	0,453	1,000	0,933	0,633	0,636
pg2500.txt	0,432	0,493	0,446	0,504	0,933	1,000	0,644	0,671
pg74-net.txt	0,582	0,585	0,629	0,626	0,633	0,644	1,000	0,990
pg74.txt	0,581	0,613	0,626	0,651	0,636	0,671	0,990	1,000

- Resultat execució ($\mathbf{n = 3}$):

	pg11-net.txt	pg11.txt	pg12-net.txt	pg12.txt	pg2500-net.txt	pg2500.txt	pg74-net.txt	pg74.txt
pg11-net.txt	1,000	0,894	0,320	0,284	0,061	0,057	0,209	0,198
pg11.txt	0,894	1,000	0,284	0,434	0,055	0,225	0,185	0,293
pg12-net.txt	0,320	0,284	1,000	0,893	0,071	0,068	0,243	0,230
pg12.txt	0,284	0,434	0,893	1,000	0,065	0,227	0,214	0,313
pg2500-net.txt	0,061	0,055	0,071	0,065	1,000	0,620	0,142	0,134
pg2500.txt	0,057	0,225	0,068	0,227	0,620	1,000	0,139	0,265
pg74-net.txt	0,209	0,185	0,243	0,214	0,142	0,139	1,000	0,943
pg74.txt	0,198	0,293	0,230	0,313	0,134	0,265	0,943	1,000