Pràctica de programció funcional + orientada a objectes

Similitud entre documents

Ismael El Habri, Lluís Trilla

16 d'octubre de 2018

Índex

Capítol 1

Codi de la pràctica

Hem dividit el nostre codi en dos fitxers, SimilitudEntreDocuments.scala i MapReduceFramework.scala.

1.1 Fitxer SimilitudEntreDocuments.scala

Aquest fitxer inclou les següents funcions:

1.1.1 Funcions de freqüència

Funció Freqüència

```
//Rebent una String i un enter n dóna com a resultat una llista amb tuples (n-grames,
    freqüència)
def freq(text:String, n:Int):List[(String, Int)] =
  normalitza(text).split(" +").sliding(n).toList
    .map(_.mkString(" ")).groupBy(identity).mapValues(_.length).toList
```

Aquesta funció rep una String amb el contingut d'un fitxer i un enter, i torna una Llista amb tuples n-grames (on n és l'enter donat), Nombre. El nombre és el nombre de vagades que apareix en el fitxer la paraula amb la que va agrupat. En la funció usem la funció que explicarem a continuació, `normalitza`. A part, usem les següents funcions de Scala:

- `split`: separa una string en una llista de strings usant un delimitador donat
- `sliding`: per fer grups de n-elements amb els elements de la llista de Strings.
- `map`: L'usem per convertir cada grup resultant de la funció anterior en Strings.
- `mkString`: Per crear Strings a partir de la llista.
- `groupBy`: S'usa per agrupar els elements d'una llista donada una certa relació, en aquest cas volem agrupar els ideals, això dóna un resultat del tipus `Map[String,List[String]]`.

- `mapValues`: L'usem per reduir els resultats dels valors anteriors en un element de Diccionari, en aquest cas: String -¿ Int.

Funció de normalització

```
//Rep una String i la normalitza (canvia tot el que no és lletra per espais i passa la
    string a minúscules)
def normalitza(text:String):String =
  text.map(c=> if(c.isLetter) c else ' ').toLowerCase().trim
```

Funció que per cada element de una String fa un map per canviar tots els elements que no són lletres per espais, passa el resultat del map a minúscules.

De les funcions de Scala no n'usem cap de nova, apart de `trim` que ens treu els espais generats al principi i al final de la String normalitzada.

Funció de Freqüències sense *StopWords*

```
//Rep una String i una llista de Strings amb stop words, i fa el vector de frequencies
    filtran les stop words.
def nonStopFreq(text:String, stop:List[String], n:Int):List[(String, Int)] =
  normalitza(text).split(" +").filterNot{a =>
      stop.contains(a)}.sliding(n).toList.map(_.mkString("
      ")).groupBy(identity).mapValues(_.length).toList
```

Funció molt semblant a la de freqüències normals, però en aquest cas, abans del sliding, filtrem les *stopwords* donades.

Funció de distribució de paraules

```
//Obtenim les 10 frequències més frequents, i les 5 menys frequents
def paraulafreqfreq(llistaFrequencies:List[(String,Int)]): Unit = {
  val stringFrequencies:String = llistaFrequencies.map(_._2.toString.concat(" ")).mkString
  val freqfreqList = stringFrequencies
    .split(" +").groupBy(identity).mapValues(_.length).toList.sortBy(-_._2)
  println("Les 10 frequencies mes frequents:")
  for(frequencia <- freqfreqList.slice(0,10))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 +" vegades")
  println("Les 5 frequencies menys frequents:")
  for(frequencia <-
      freqfreqList.slice(freqfreqList.length-5,freqfreqList.length).sortBy(-_._2))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 +" vegades")
}
```

Funció on busquem les 10 freqüències més freqüents, i les 5 que ho son menys. Esta implementada aprofitant la mateixa funció freq original. Primer de tot transformem l'estructura de dades que conté les freqüències en una string amb les freqüències separades per espais, i després nomes cal usar la funció freq per tal d'obtenir les freqüències de freqüències

1.1.2 Funcions de Comparació

La única funció per comparar és la funció `cosinesim`, però abans, introduirem les funcions auxiliars utilitzades per fer-lo.

Funcions auxiliars

```
//Rep dos vectors de frequencies absolutes i les converteix a tf.
def freqAtf(llistaFreq:List[(String, Int)]):List[(String, Double)] = {
  val mesfrequent = llistaFreq.maxBy(_._2)._2
  llistaFreq.map{a=> (a._1, a._2.toDouble/mesfrequent)}
}

//Retorna la paraula no-stop més frequent del text introduït, junt amb la seva freqüència
def mesFrequent(text:String, stop:List[String], n:Int) = nonStopFreq(text, stop,
    n).maxBy(_._2)

//Busquem les paraules q no tenim a txt1 de txt2 i les afegim amb frequencia = 0 a txt1
    Al final ordenem alfabeticament, per tenir el mateix ordre en els dos vectors!
def alinearVector(aAlinear:List[(String, Double)], suport:List[(String,
    Double)]):List[(String, Double)] ={
  val aAlinearMap = aAlinear.toMap
  (aAlinear ::: (for (b<-suport if !aAlinearMap.contains(b._1)) yield (b _1, 0.0)))
      sortBy(_._1)
}
```

1. `freqAtf`: passa les freqüències absolutes a freqüència *tf*.
2. *mesFrequent*: ens dóna l'element més freqüent de la llista de freqüències.
3. *alinearVector*: Donat un vector a alinear, i un vector amb el qual s'ha de alinear, alinea el vector a alinear.

Capítol 2

Joc de proves

Capítol 3

Resultats