Université Grenoble Alpes
Master MIASHS – DCISS - 2021
Documents autorisés
2 heures
barème indicatif

Examen de programmation2
Cooccurrences de mots dans texte

Le but de ce devoir est de développer un programme qui compte les cooccurrences de mots dans un texte et qui permet ensuite de calculer une mesure d'association entre mots du texte.

Nous appelons une cooccurrence, deux mots qui apparaissent successivement dans le texte. Par exemple, dans la phrase « *Le chien mange* », <le, chien> et <chien, mange> sont des cooccurrences.

A partir de l'analyse des cooccurrences et des occurrences de mots dans un texte, on peut calculer une mesure d'association entre mots. Par exemple, dans un discours d'un candidat à l'élection présidentielle française de 2012, la cooccurrence <moi,président> a été répétée 15 fois (sur les 481 mots que compte l'extrait). Les mots « moi » et « président » sont également répétés indépendamment 15 fois dans tout le discours. Les mots « moi » et « président » sont donc en association parfaites, c'est-à-dire qu'ils apparaissent toujours ensemble.

Pour calculer la mesure d'association, on utilise l'information mutuelle normalisée (mesure déjà implémentée dans le code fourni).

Les différentes tâches à réaliser sont les suivantes :

1- Compléter la classe `WordReader` (4pts)

Cette classe permet de lire un flux mot par mot. Un mot est suite contiguë de lettres ou de chiffres. Tous les autres caractères sont considérés comme des séparateurs de mots. Plusieurs caractères peuvent séparer deux mots. Pour tester si un caractère est une lettre ou un chiffre, on utilisera la fonction `Character.isLetterOrDigit(char c)` qui retourne `true` si le caractère `c` est effectivement une lettre ou un chiffre.

2- Compléter la classe générique `Pair` (4pts)

Cette classe permet de représenter une paire non ordonnée d'objets du même type. Non ordonnée signifie que la paire <x1,x2> est égale à la paire <x2,x1>. On utilisera cette classe pour représenter les cooccurrences dans la suite du travail.

3- Compléter la classe `CooccurrenceSim` (8pts)

Cette classe permet de compter les cooccurrences et occurrences, de calculer la mesure d'association entre deux mots et construire un dictionnaire ordonné qui associe à chaque valeur de la mesure d'association, l'ensemble des cooccurrences (Pair) qui ont cette valeur.

4- Compléter la classe principale `AnalyseTexte` (4pts)

Elle ouvre et lit fichier texte donné en paramètre et affiche ensuite le dictionnaire des valeurs d'association et les cooccurrences associées dans l'ordre décroissant des valeurs.

```
import java.io.*;
/**
 * Classe filtre permettant de lire un flux de caractères
 * mot par mot. Un mot est une suite contiguë de lettres ou de chiffres.
 */
public class WordReader extends _____ {
	/**
	 * Représente le mot en cours de construction.
	 * un caractère est ajouté en utilisant la méthode append(char c)
	 * de StringBuilder.
	 * Attention, append(int i) existe aussi.
	 */
	private StringBuilder sb;

	protected WordReader(Reader in) {

		_____

	}
	/**
	 * Lit le prochain mot du flux
	 * @return le dernier mot lu dans le flux ou
	 *          null si on est arrivé à la fin
	 * @throws IOException
	 */
	public String readWord() throws IOException {

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

	}
}
```

```java
/**
 * Classe représentant une paire d'objets du type X
 * @param <X> le type des données de la paire
 */
public class Pair<X> {

	private final X e1;
	private final X e2;

	public Pair(X e1, X e2) {
		this.e1=e1;
		this.e2=e2;
	}

	public X getE1() { return e1; }
	public X getE2() { return e2; }

	/**
	 * Deux paires sont égales si leurs deux éléments sont égaux
	 * quelque soit l'ordre (on suppose e1 et e2 non null)
	 */
	@Override
	public boolean equals(Object o) {

		_____

		_____

		_____

		_____

		_____

		_____

		_____
	}

	/**
	 * Le hashcode d'une paire est la somme des hashcodes de ses deux éléments
	 */
	@Override
	public int hashCode() {

		_____
	}

	/**
	 * Une paire est représentée de la manière suivante : <e1,e2>
	 */
	@Override
	public String toString() {

		_____
	}
}
```

```java
import java.util.*;
/**
 * Permet de compter les cooccurrences et occurrences
 * et de calculer l'association entre mots */
public class CooccurrenceSim {
	/* Permet d'associer à chaque paire de mots, son nombre de cooccurrences*/

	private Map<Pair<String>, Integer> cooccurrences = ____________________ ;

	/* Permet d'associer à chaque mot, son nombre d'occurrences */

	private Map<String, Integer> occurrences = ____________________ ;

	/* Nombre total d'occurrences de mots dans le texte */
	private int nbOccurrences = 0;

	/* Mot précédement ajouté via la méthode pushWord */
	private String previous = null;

	/**
	 * Permet d'ajouter un mot. Les dictionnaires occurrences
	 * et cooccurrences sont mis à jour ainsi que nbOccurrences et previous
	 */
	public void pushWord(String word) {

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

		_____

	}
```

```
	/* calcule l'association entre deux mots */
	public double getSim(String w1, String w2) {
		return getSim(new Pair<>(w1, w2));
	}

	/* calcule l'association entre deux mots */
	private double getSim(Pair<String> p) {
		if (!cooccurrences.containsKey(p)) {
			return -1;
		}
		int cooc = cooccurrences.get(p);
		int occw1 = occurrences.get(p.getE1());
		int occw2 = occurrences.get(p.getE2());

		double freq = (double) cooc / nbOccurrences;
		double pmi =  Math.log(((double) (cooc * nbOccurrences))
				/ (occw1 * occw2));
		return pmi/-Math.log(freq);

	}

	/* Retourne un dictionnaire qui indexe les cooccurrences en fonction
	 * de leur valeur d'association (i.e. la valeur retournée par getSim */
	public TreeMap<Double,Set<Pair<String>>> getSims() {
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

```
	}
}
```

```java
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.*;

public class AnalyseTexte {

	public static void main(String[] args) {
		Path p = Paths.get(args[0]);

		if (!Files.exists(p)) {
			System.err.println("Le fichier "+p+" n'existe pas");
			System.exit(0);
		}

		try (WordReader wr = ____________________________________________________){

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		____________________________________________________________________________

		} catch (IOException e) {
			System.err.println("Une erreurs s'est produite"+
					" lors de la lecture du fichier");

			e.printStackTrace();
		}
	}
}
```