Il progetto Gutenberg: calcolo delle frequenze delle parole in un corpus monolingue.

Irene Parlanti 587940 e Camilla Zucchi 490366

SPECIFICHE INIZIALI

Il progetto Gutenberg è composto di 7 GB di materiale, in particolare testo, immagini e file HTML. Il tutto è diviso in più cartelle e sottocartelle, contenenti infine dei file zip, di cui ci interessa estrarre i dati testuali per calcolare le frequenze dei termini allo scopo di elaborarne un grafico dove siano visibili le relazioni tra parole e occorrenze delle stesse.

Nelle due classi Analisi e Calcolo ci sono altrettanti main. In sequenza lanciamo:

•Analisi, che si occupa di unzippare i files grazie al richiamo della classe UnzipFile; in seguito, i files unzippati vengono messi nella cartella Unzipped;

•Calcolo, invece, che ripulisce i file.txt, elimina il superfluo: al suo interno il metodo selezionaTesti si occupa di cancellare i file readme.txt e quei file identificati dall'aggiunta del suffisso numerico –0 e –8. Il metodo successivo dal nome seleziona estrae i file di lingua inglese e li priva delle porzioni non utili all'analisi, cioè quelle precedenti *** START OF e seguenti *** END OF, e poi finiscono nella cartella Output.

INDIVIDUAZIONE DEI FILE

La consegna del progetto richiedeva in primis l'estrazione dei file con estensione .zip. Nella classe Analisi, abbiamo così predisposto un metodo ad hoc, listFiles.

Il metodo unzip successivamente richiamato si occupa di creare un oggetto della classe UnzipFile e richiamarne il metodo zipExtraction che, come da nome, completa l'estrazione, controllando che il file non sia una cartella e che abbia estensione .txt. Difatti, non è raro trovare file immagine, audio o video, che non ci interessano per l'analisi. I file unzippati vanno poi nella cartella unzipped.

ANALISI TESTUALE

Il passo successivo è il passaggio della cartella con i file unzippati ad una classe che ne inizi l'analisi e questo viene effettuato dal metodo caricaFile della classe Calcolo.

La classe Calcolo è strutturata come segue, ha:

- due ArrayList, la prima dal nome testo conterrà tutte le parole tokenizzate di un testo alla volta, e l'altra nomiTestiSelezionati conterrà tutti i nomi dei testi selezionati;
- due Hashmap, una a due dimensioni <String, Integer>, in cui verranno inserite tutte le parole e la loro frequenza, l'altra <Integer, Integer> in cui saranno registrate la frequenza e la cardinalità delle parole con quella frequenza, dati utili per l'elaborazione del grafico della legge di Zipf;
- infine, i quattro contatori presenti serviranno per l'output finale in console.

Il metodo selezionaTesti, che segue il caricaFile, a sua volta richiama il metodo leggiTesto e provvede alla tokenizzazione, richiamando la libreria OpenLP SimpleTokenizer. I controlli svolti dal metodo

guardano se la parola “readme” è nome del file e in tal caso si tratta di un file di istruzioni inutile; poi si verifica la presenza o meno di doppioni nell’ArrayList nomiTestiSelezionati, una volta eliminati i suffissi -0 e -8, inseriti per segnalare una copia dello stesso testo in una codifica diversa.

Il metodo leggiTesto legge il file di testo tramite le classi di Java FileChannel e ByteBuffer e lo riporta tutto a caratteri minuscoli.

Il metodo seleziona, poi, ha un doppio scopo: controllare che la lingua del testo sia l’inglese e prendere in considerazione solo la porzione di testo desiderata, andando ad eliminare sezioni testuali inserite dal progetto Gutenberg all’inizio e alla fine di ciascuna opera.

Il metodo analisiTesti va ad analizzare ciascun token e, se già presente nella prima hashmap, ne aumenta l’occorrenza, altrimenti crea una nuova key.

RISULTATI

Il penultimo metodo della classe Calcolo si chiama salvaRisultati, salva e stampa la prima hashmap in freq_parole.txt, mettendo a sinistra il token e a destra la sua frequenza.

Sempre lo stesso metodo crea un’altra hashmap <Integer, Integer>, di cui sopra, che viene stampata nel file freq_freq.txt e che riporta a sinistra la frequenza delle parole e destra il numero di parole che hanno quella frequenza.

NUMERO TOTALE DI FILE ANALIZZATI: 26093

NUMERO TOTALE DI FILE SELEZIONATI: 19392

NUMERO TOTALE DI PAROLE: 1237844079

NUMERO TOTALE DI PAROLE DISTINTE: 1525697

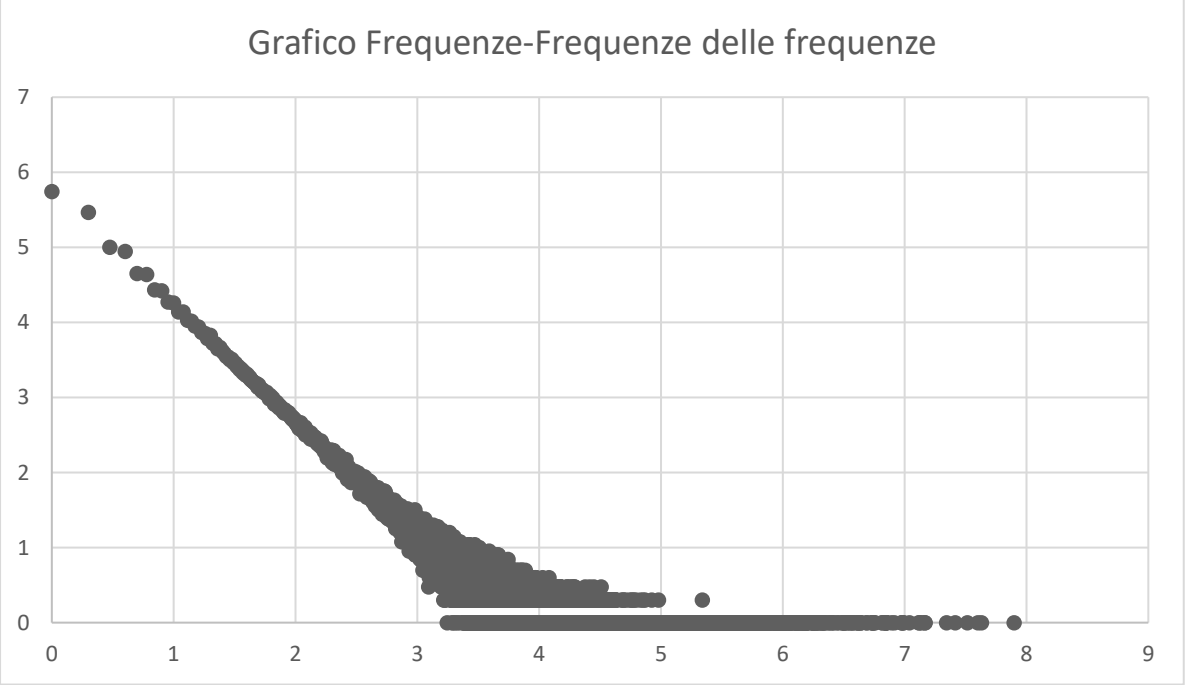
TESTA DEL FILE DELLE FREQUENZE:

Parola	Frequenza
the	79048047
of	42604676
and	40067292
to	32767924
a	25948222
in	22059631
that	14766327
he	13748768
was	13187523
it	13086494

TESTA E CODA DEL FILE DELLE FREQUENZE DELLE FREQUENZE:

Frequenza	Occorrenze
1	553052
2	291919
3	100591
4	88268
5	44993
6	43518
7	27164
8	26371
9	18788
10	18120
...	...
13187523	1
13748768	1
14673762	1
14766327	1
22059631	1
25948222	1
32767924	1
40067292	1
42604676	1
79048047	1

GRAFICO DI ZIPF:



APPENDICE: listato dei programmi usati

CLASSE Analisi

```
package progetto_romani;

import java.io.File;
import java.io.IOException;
import java.time.Duration;
import java.time.Instant;

public class Analisi {


	public static void main(String [] args)throws IOException{
		Instant start = Instant.now();
		System.out.println("Estrazione in corso");
		listFiles("C:\\Users\\irene\\eclipse-
workspace\\progetto_romani\\Gutenberg");
		System.out.println("Estrazione completata");
		Instant finish = Instant.now();
		long timeElapsed = Duration.between(start,
finish).toMinutes();
		System.out.println("Durata di esecuzione: " + timeElapsed
+ " minuti");
	}

	private static void listFiles(String path) throws IOException {
		File folder = new File(path);
		File [] files = folder.listFiles();
		for(File file : files) {
			if(file.isFile()) {
				try {
					if(file.getName().endsWith(".ZIP")) {
						unzip(file.getPath());
					}
				}catch(Exception e) {
					e.printStackTrace();
				}
			}
			else if(file.isDirectory()) {
				listFiles(file.getPath());
			}
		}
	}

	private static void unzip(String filePath) {
		String zipFilePath = filePath;
```

```
            String destDirectory = "C:\\Users\\irene\\eclipse-
workspace\\progetto_romani\\Unzipped";
            UnzipFile unzipper = new UnzipFile();
            try {
                  unzipper.zipExtraction(zipFilePath, destDirectory);
            }catch(Exception e) {
                  e.printStackTrace();
            }
      }

}
```

CLASSE UnzipFile

```
package progetto_romani;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

public class UnzipFile {

	public File newFile(File destinationDir, ZipEntry zipEntry)
throws IOException{
		File destFile = new
File(destinationDir,zipEntry.getName());
		String destDirPath = destinationDir.getPath();
		String destFilePath = destFile.getPath();
		if(!destFilePath.startsWith(destDirPath +
File.separator)) {
			throw new IOException("Entry is outside of the
target dir: " + zipEntry.getName());
		}
		return destFile;
	}

	public void zipExtraction(String filePath, String dest) throws
FileNotFoundException, IOException{
		try(ZipInputStream zis = new ZipInputStream(new
FileInputStream(filePath))){
			ZipEntry entry = zis.getNextEntry();
			while(entry != null) {
				File file = new File(dest,entry.getName());
				if(entry.getName().endsWith(".txt")) {
					if(entry.isDirectory()) {
						file.mkdirs();
					}else {
						File parent = file.getParentFile();
						if(parent.exists()) {
							parent.mkdirs();
						}
						try(BufferedOutputStream bos = new
BufferedOutputStream(new FileOutputStream(file))){
							byte[] buffer = new
byte[Math.toIntExact(entry.getSize())];
							int location;
```

```
      while((location=zis.read(buffer))!= -1) {
                                                bos.write(buffer, 0,
location);
                                            }
                                        }
                                    }
                                }
                                entry = zis.getNextEntry();
                            }
                        }

      }
}
```

CLASSE Calcolo

```
package progetto_romani;

import static java.util.stream.Collectors.toMap;
import java.io.BufferedWriter;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map;

import opennlp.tools.tokenize.SimpleTokenizer;
import opennlp.tools.tokenize.Tokenizer;

public class Calcolo {

     ArrayList<String> testo = new ArrayList<String>();
    ArrayList<String> nomiTesti = new ArrayList<String>();
    HashMap<String, Integer> h = new HashMap<>();
    HashMap <Integer, Integer> t = new HashMap<>();
    int count;

    public int contatoreTestiAnalizzati = 0;
    public int contatoreTestiSelezionati = 0;
    public int contatoreTypes = 0;
    public int contatoreTestiDoppi = 0;

    public static void main(String[] args)throws IOException {
           Instant start = Instant.now();
           System.out.println("Inizio selezione e analisi del
testi");
           selezionaEAnalizza("C:\\Users\\irene\\eclipse-
workspace\\progetto_romani\\Unzipped");
           Instant finish = Instant.now();
           long timeElapsed = Duration.between(start,
finish).toMinutes();
           System.out.println("Durata di esecuzione: " + timeElapsed
+ " minuti");
```

```
	}

	public void caricaFile(String path) throws IOException{
		File folder = new File(path);
		File[] files = folder.listFiles();

		for (File file : files) {
			if(file.isFile()){
				try {
					selezionaTesti(file.getPath(),
file.getName());
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
			else if (file.isDirectory()) {
				caricaFile(file.getPath());
			}
		}

	}

	private void selezionaTesti(String path, String nome) {

		String file = leggiTesto(path);
		testo = new ArrayList<String>();
		Tokenizer tokenizer = SimpleTokenizer.INSTANCE;
		testo = new
ArrayList<String>(Arrays.asList(tokenizer.tokenize(file)));
		contatoreTestiAnalizzati++;
		if(nome.contains("readme.txt") == false) {
			if(seleziona(testo, nome)== true) {
				nome = nome.replaceAll("(-8|-0)", "");
				if(nomiTesti.contains(nome) == false &&
nomiTesti.contains(nome+"-0") == false) {
					nomiTesti.add(nome);
					contatoreTestiSelezionati++;
				}else {
					contatoreTestiDoppi++;
				}
				analisiTesti();

			}
		}
	}

	private String leggiTesto(String s) {
		String lettura = "";
```

```
      try (RandomAccessFile reader = new RandomAccessFile(s, "r");
                  FileChannel channel = reader.getChannel();
                  ByteArrayOutputStream out = new
ByteArrayOutputStream()) {
                        int bufferSize = 1024;
                        if (bufferSize>channel.size()) {
                              bufferSize = (int)channel.size();
                        }
                        ByteBuffer buff =
ByteBuffer.allocate(bufferSize);
                        while(channel.read(buff) > 0) {
                              out.write(buff.array(), 0,
buff.position());
                              buff.clear();
                        }
                        lettura = new String (out.toByteArray(),
StandardCharsets.US_ASCII);
                        lettura = lettura.toLowerCase();
                  }catch(IOException e) {
                        e.printStackTrace();
                  }
                  return lettura;
    }

    private boolean seleziona (ArrayList<String> testo, String nome)
{
      boolean flag = false;
      String lang = "english";
      int langInd = testo.indexOf("language");

      if(testo.get(langInd+2).equalsIgnoreCase(lang) &&
testo.get(langInd+3).equals("character")) {
            flag = true;
            int pChiave = testo.indexOf("***");
            if (testo.get(langInd+7).equals("mp3")) flag = false;
            for(int i=pChiave; i<pChiave+5; i++) {
                  if(testo.get(i+1).matches("start") &&
testo.get(i+2).matches("of") && testo.get(i+4).matches("project")) {
                        testo.subList(0, 1+7).clear();
                  }
            }
            int pChiave2 = testo.indexOf("end");
            if(pChiave2 == -1) flag = false;
            else {
                  try {
                        for(int i = pChiave2; i<testo.size();i++) {
                              if(testo.get(i+1).matches("of") &&
testo.get(i+3).matches("project")) {
```

```
                                    testo.subList(i-4,
testo.size()).clear();
                              }
                        }
                  }catch(IndexOutOfBoundsException e) {
                        System.out.println("Errore");
                        System.out.println("La fine del testo " + nome
+ " non e' codificata secondo i canoni della libreria Gutenberg");
                  }
            }
      }
      return flag;
    }

    private void analisiTesti() {
      for(int i = 0; i < testo.size(); i++) {
            if(testo.get(i).matches("[a-zA-Z]+")) {
                  if(h.containsKey(testo.get(i))) {
                        int count = h.get(testo.get(i));
                        h.put(testo.get(i), count+1);
                  }
                  else {
                        h.put(testo.get(i), 1);
                  }
            }
      }


    }



    public void salvaRisultati() {
      String outputPath = "C:\\Users\\irene\\eclipse-
workspace\\progetto_romani\\freq_parole.txt";
      File file = new File(outputPath);
      BufferedWriter bf = null;
      contatoreTypes = h.size();
      Map<String, Integer> sorted = h
                  .entrySet()
                  .stream()

      .sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
                  .collect(

      toMap(Map.Entry::getKey,Map.Entry::getValue,(e1, e2)->e2,
                                          LinkedHashMap::new));
      try {
            bf = new BufferedWriter(new FileWriter(file));
```

```
            for(Map.Entry<String, Integer> entry:sorted.entrySet()) {
                  bf.write(entry.getKey()+"  "+ entry.getValue());
                  bf.newLine();
            }
            bf.flush();
            bf.close();
      } catch(Exception e) {
            e.printStackTrace();
      }
      try {

            FileWriter writer = new FileWriter("freq_freq.txt");
            BufferedWriter output =  new BufferedWriter(writer);

            for(Map.Entry <String, Integer> e : sorted.entrySet()) {
                  Integer value = t.get(e.getValue());
                  if(value == null)
                        t.put(e.getValue(), 1);
                  else
                        t.put(e.getValue(), value + 1);
            }
            LinkedHashMap<Integer, Integer> sortedMap = new
LinkedHashMap<>();

            t.entrySet()
                .stream()
                .sorted(Map.Entry.comparingByKey())
                .forEachOrdered(x -> sortedMap.put(x.getKey(),
x.getValue()));

            for(Map.Entry <Integer, Integer> e :
sortedMap.entrySet()) {
                  output.write(e.getKey()+"  " + e.getValue());
                  output.newLine();
            }

            output.flush();
            output.close();
      } catch(Exception e) {
            e.printStackTrace();
      }
    }

    private static void selezionaEAnalizza(String path) throws
IOException{
            Calcolo calcolo = new Calcolo();
            calcolo.caricaFile(path);
            calcolo.salvaRisultati();
```

```
            System.out.println("Testi analizzati : " +
calcolo.contatoreTestiAnalizzati);
            System.out.println("Testi selezionati : " +
calcolo.contatoreTestiSelezionati);
            System.out.println("Numero testi duplicati esclusi : " +
calcolo.contatoreTestiDoppi);
            System.out.println("Numero types : " +
calcolo.contatoreTypes);
      }



}
```