

MCIS 6263

Assignment 1

Objectives:

- Refresh and sharpen your coding skills
- Become familiar with semi-structured data processing and data transformation

Description:

In this assignment, you are given a dataset of approximately 20,000 news documents collected from a set of newsgroups (mailing lists). The set of documents (email messages) is partitioned almost evenly across 20 different topics such as sport, electronics, politics, etc. The documents of each newsgroup are stored in one directory. Each news document is stored in a text file in a semi-structured format.

Here is a sample document:

From: dpc47852@uxa.cso.uiuc.edu (Daniel Paul Checkman)

Subject: Re: Is MSG sensitivity superstition?

Article-I.D.: news.C5wl4F.Dt

Organization: University of Illinois at Urbana

Lines: 22

bruce@Data-IO.COM (Bruce Reynolds) writes:

>Anecdotal evidence is worthless. Even doctors who have been using a drug
>or treatment for years, and who swear it is effective, are often suprised
>at the results of clinical trials. Whether or not MSG causes describable,
>reportable, documentable symptoms should be pretty simple to discover.

I tend to disagree- I think anecdotal evidence, provided there is a lot of it, and it is fairly consistent, will be very important. First, it points to the necessity of doing a study, and second, it at least says that the effects are all psychological (or possibly allergy in this case). As I've pointed out before, psychological effects are no less real than other effects. One person's "make-believe" can easily be another person's reality. Using psychadelic drugs in a bizarre and twisted example, the hallucinations one person experiences on an acid trip cannot be guaranteed to another person on an acid trip- there is no clinical evidence that those effects are always going to happen. Anyhow, that was a pretty lame example, but hopefully I made my point- it's all a matter of perception, and as long as someone ingesting MSG perceives it as causing bad effects, then s/he can definitely experience those affects. On the other hand, it could just be an allergy to the food it's in, or something. Still, anecdotal evidence is not worthless- it's the stuff that leads to the study being done.

-Dan

Each document starts with the email header which contains a number of attributes such as “From:”, “Subject:”, “Organization:”, etc. followed by the message body.

In this assignment, you are asked to do the following tasks:

1. Download the dataset and decompress the files
2. Parse the documents and extract key information
3. Compute some stats

The details follow:

1. Download the Dataset and Decompress the Files

Download the 20-newsgroups.zip file from the assignment page on blackboard. Decompress the .zip file anywhere you want. That should give you a folder named “20-newsgroups” which contains 20 subfolders, one subfolder for each category.

2. Task 1: Parse the Documents, Clean them from Noise, & Extract Key Information

Your first task is to write a **Java** program to process the documents in the dataset and parse the semi-structured content of each document to extract the following information:

- Category: Document category can be extracted from the name of the folder in which the document is stored (e.g. rec.sport.hockey, rec.autos, comp.sys.mac.hardware, etc.)
- Sender (From): This can be extracted from the “From:” field of message header (highlighted in yellow in the example above).
- Sender Affiliation (Organization): This can be extracted from the “Organization:” field of the message header. Some documents will have no Organization field. For those ones you can set the Organization value to “N/A”.
- Subject: This can be extracted from the “Subject:” field of the message header.
- Document Body: This is the message body (highlighted in green in the example above.) It’s the text that comes after all the header fields in the document. None of the header fields (such as “Lines:”, “Keywords”, “Article-ID”, etc.) should be included in the body.

Data Pre-processing/Cleaning:

As we mentioned in the class, Big Data can be noisy and, therefore, some pre-processing might be needed to clean the data and improve its quality. Your code should clean the data as follows:

- All the fields that are not mentioned above (and highlighted in yellow in the example) should be discarded. For example, the fields “Article-I.D.”, “Lines”, etc. from the example above should be discarded.
- All quoted text should be removed from the message body. In the example above, all the text highlighted in red should be discarded. Quoted text usually starts with a line that says something like: “XYX@Org.org (X Y Z) writes:” followed by one or more lines that start with ‘>’. All such lines should be discarded.

Hint: For simplicity, assume that any line that contains “:” should be discarded except the lines that correspond to the required header fields (e.g. From, Organization, etc.). This should help take care of the two previous preprocessing requirements.

- In the message body (highlighted in green in the example above) there might be some tab ‘\t’ characters or repeated spaces (which uses extra space for no useful reason). Make sure to clean the message body by replacing tab characters and repeated spaces with a single space character. You can use Regex to achieve this. Something like:

```
body = body.replaceAll("\\s+", " ");
```
- Since the body of most documents contains multiple lines, we can’t store the body content as it’s in the tsv file; otherwise, it will break the file format. That’s because according to our file format a new line means a new document. Therefore, you should replace each new line character in the “body” with some other indicator

Input/Output:

Your program should take the absolute path of the “20-newsgroups” folder from Task 1 and traverse all the subfolders and process the files in each folder. Your code should parse each file to extract the information mentioned above.

The output of this task should be one huge file that contains all the data in Tab Separated Values (.tsv) format. This format is commonly used to store Big Data. Your output file should include a line for each document in the set. Each line should include the aforementioned information in the same order mentioned above (i.e. Category, From, Organization, Subject, Body). The values in each line should be separated by tabs (the ‘\t’ character).

In the assignment documents, you’ll find a sample input/output for you to check and test your code.

3. Task 2: Compute stats

Your code in this task should take the “.tsv” file generated in Task 1 as input and compute the following stats:

- The total number of documents (which should be same as the number of lines in the file)
- The average word count of the document “body” in the data set. To get this, you need to compute the number of words in the “body” column for each document (line), then compute the average. For simplicity, assume that words are separated by single spaces.
- Compute the average number of documents per category. To compute this you need to compute the number of documents that belong to each category and then compute the average value.
- Find the category with the maximum average “body” word count.
- Find the category with the minimum average “body” word count.

The output of this task should be another Tab Separated Values (.tsv) file. Your file will contain one line only with tab separated values for the requested stats in the same order that they appear above.

What to submit?

1. Put all Task 1 source code in a folder named “task1-src”. Compress the folder and submit a file named “task1-src.zip”
2. Compile and export your Task 1 code to a single executable JAR file. Submit a file named task1.jar. It should be possible to execute your jar file using something like the following command

```
java -jar task1.jar DataSetFolderPath Output.tsv
```

3. Put all Task 2 source code in a folder named “task2-src”. Compress the folder and submit a file named “task2-src.zip”
4. Compile and export your Task 1 code to a single executable JAR file. Submit a file named task1.jar. It should be possible to execute your jar file using something like the following command:

```
java -jar task2.jar input.tsv Output.tsv
```

Important:

Make sure to submit your work exactly as instructed including the formats and filenames. Otherwise, your assignment will not be graded.