COMP1204 Unix Coursework Report

Davide Gamba
dg3g18@soton.ac.uk
ID: 30426243

March 14, 2019

1 Introduction

As a data scientist for TripAdvisor my job was to help them make sense of what hotels are performing well. I have been tasked with analysing all the files containing reviews for each hotel. The files are in .dat format and each hotel file contains a various number of reviews and each reviews contains many fields such as "Author", "Overall" etc...

2 Scripts

In order to perform the operations requested by the coursework specifications, two scripts were created, namely countreviews.sh and averagereviews.sh. The code of both scripts has been included below, along with an explanation of how it functions.

2.1 Countreviews.sh

In order to count the number of reviews for each hotel and to sort the list from the hotel with the highest number of reviews to the one with the lowest. The code for this script is listed below:

```
1  #!/bin/bash
2  cd $1
3  for file in *.dat
4  do
5      fileName=${file%.*}
6      fileName=${fileName##*/}
7  echo $fileName $(grep -c "<Author>" $file)
8  done | sort -k 2 -nr
9
```

Line 1 is used to determine which interpreter needs to be used (in this case bash).
Line 2 is used to move to the folder specified in the parameter. This works with

both absolute and relative paths.
The script then utilizes a for loop to iterate through the reviews folder. During the execution of every loop, the script takes the filename of each hotel and removes the path and the .dat extension from the name of the hotel (lines 5-6).
Line 7 is used to print the name of the hotel, along with its number of reviews, which is generated thanks to the command "grep -c" which counts every occurrence of the field "<Author>" for each file in the folder. Since every review only has one "<Author>" field, by counting the number of these fields in a hotel file, we get the total number of reviews for each hotel.
The last line is used to end the loop and to sort the list in decreasing order.

2.2 AverageReviews.sh

This script returns each hotel name next to the average review score for that hotel, in a list that is sorted from the highest ranking hotel to the lowest.

```
1   #!/bin/bash
2   cd $1
3   for file in *.dat
4   do
5     hotelName=${file%.*}
6     hotelName=${hotelName##*/}
7     average=$(awk -F '>' '/<Overall>/ {count+= $2; total += 1}
    END{print count/total}' $file)
8       echo -n $hotelName
9       printf " %0.2f\n" $average
10  done|sort -k2 -nr
11
```

Lines 1-6 are the same as for countreviews.sh.
Line 8 uses the awk command to retrieve the score after the <Overall>tag, and stores it in the variable "count", while the variable "total" keeps the count of the total number of reviews for the hotel. At the end, the awk command prints the average of the reviews. The result of this operation is stored in the variable "average".
Lines 8 and 9 are used to print the hotel name next to the average of its reviews (rounded to the second decimal digit).
The last line, as for countreviews.sh, is used to end the loop and to sort the list in decreasing order.

3 Discussion

3.1 Unstructured Markup vs Structured Data

There are mainly two ways to store data, according to the use that will be made of the data, it is possible to store it in a structured way (in a database) or in an unstructured way (tags and markups).
While unstructured data might be easier to read for humans, it becomes very

inefficient when that data has to be manipulated or read by a machine. Structured data, on the other hand, might appear confusing to humans, but it makes sense from a machine-language standpoint and can be processed by apposite programs way faster and more efficiently then unstructured data.
The format in which TripAdvisor stores the reviews for its hotels is unstructured as it utilizes tags to divide the data into different fields or sections. Again, this makes it very easy for a human to read and understand the data, but it requires a lot more work to actually process that data with scripts or external programs. This is reflected in the scripts above, where since the data was not in a database, everything needed to be treated as a string.
If the data was structured in a database, operations like the one performed by countreviews.sh and averagereviews.sh would have been much quicker, with less code required.

3.2 Ideas on how to improve the user-verification and ranking system

It is known that TripAdvisor has faced some real challenges, in particular regarding fake or untrustworthy reviews the aim of which was to artificially lower or heighten the score of certain businesses.
If every review has the same weight, then a person writing a fake review has the same power to hurt or benefit a business as everybody else. For this reason many websites like Steam or Amazon, give a different weight to "verified" reviews and to "unverified" ones. In practice, once the websites has verified that, for example, you actually bought an particular item from Amazon, and that you left a review for that item, then your review becomes verified and other users who might still be undecided on whether or not to buy that particular item, can see that your review is trustworthy, at least to a great extent.
TripAdvisor, in my opinion, could implement a similar system, in which people who have actually spent at least a night at a hotel could upload their receipt, which can then be verified by the system. If the verification is successful, then the review can have more weight on the overall score of the hotel.
Another problem could be with bots operating from the same computer. TripAdvisor should also implement a system to check whether an improbable number of reviews for the same hotel come from the same IP address.
Another Idea would be to give less weight to single reviews that are way off the average of the hotel, both in positive and in negative. Of course this solution would need to be paired with an algorithm able to take a definite time span (for example of one month) and calculate an average over the last month on which to base the judgement on whether or not a review could be considered untrustworthy. In this way, hotels that have actually improved their services over the previous month would not be affected negatively by the algorithm.
Lastly, and probably the most complicated of the ideas proposed, would be to create an algorithm that determines the trustworthiness of the reviews based on human-detectable factors like improbable bad grammar or reviews that look very similar to one another with the same score. Of course, these factors are

hard to determine, especially with an algorithm. The program will probably have to use machine learning to create a pattern of a trustworthy review and then compare it to the reviews that don't match the pattern and are therefore flagged as untrustworthy.
This last solution would be the hardest to implement but could potentially be the most efficient one.