COMP1204 DATA MANGEMENT

UNIX COURSEWORK

February 26, 2016

Vedant Chokshi
Student ID: 27748456
University of Southampton

Contents

1 SHELL SCRIPTS

1.1 countreviews.sh

```
#!/bin/bash

out=$(mktemp /tmp/countreviews.XXX)
for f in $1/*
do
    hotelName=$(basename "$f" .dat)
    numOfReviews=$(grep -c "<Author>" $f)
    echo -e $hotelName "\t" $numOfReviews >> $out
done
sort -k2nr $out
```

Firstly, a temporary output file is created in the */tmp* directory and stored in a variable, *out*. Using a temporary file meant that there was no need for the **rm** command to remove the file and hence allows the code to run faster. Secondly, there is a **for loop** which loops over each file in a specified folder given as an argument(referred to in bash through *$1*) from when the shell script is executed on command line. For each file:

1. The **basename** command is used to store the hotel name in a variable, *hotelName*. Using ".dat" as the suffix for **basename** removed the extension so only the *hotelName* is printed.

2. The option "-c" in **grep** scans through the all the lines in a given file, looking for the specified string and counts the number of occurrences of the specified string. Hence, the **grep -c** command is used to count the number of occurrences for "<Author>" in the file which is identical to the number of reviews. The number of reviews is stored in a variable, *numOfReviews*.

3. The *hotelName* and *numOfreviews* for the file being processed in the loop are **echo**ed to the temporary file created earlier. The option "-e" is used to allow tabbing with "\t" for a more appealing output stream. The data is appended in the *out* file with the use of ">>" so the *hotelName* and *numOfreviews* are added for each file without being overwritten.

Finally, the **sort -k2nr** command is used to sort the temporary file output the hotel with the most reviews to the hotel with the least.. The option "-k2n" sorts the second column and the option "-r" is used to reverse the order of the output.

1.2 averagereviews.sh

```
#!/bin/bash

out=$(mktemp /tmp/output.XXX)
for f in $1/*
do
   hotelName=$(basename "$f" .dat)
   overallAverage=$(awk '
   BEGIN {
   FS=">"
   } {
      if("<Overall"==$1) {
         sum+=$2;
         ++numOfReviews;
      }
   } END {
      printf "%.2f", sum / numOfReviews;
   }' $f)
   echo -e ${hotelName} "\t" ${overallAverage} >> $out
done
sort -k2nr $out
```

Like countreviews.sh, a temporary output file is created and stored in a variable, *out*. Then there is a **for loop** which loops over each file in a specified folder given as an argument(referred to in bash through *$1*) from when the shell script is executed on command line. For each file:

1. The **basename** command is used to store the hotel name in a variable, *hotelName* (*see countreviews.sh for more detail*)

2. Then the **awk** command is used to create a "Field Separator", *FS* which scans through all the lines in the file and splits the line at ">". If the first field(*$1*) of the split string equals "<Overall" then add second field(*$2*) to the variable *sum* and increment the variable *numOfReviews* by 1. Once all lines have been processed, the **printf** command is used to calculate the average overall by doing *sum* / *numOfReviews*.The result is rounded by using ".%2f" and stored inside the variable, *overallAverage*.

3. The *hotelName* and *numOfreviews* for the file being processed in the loop are **echo**ed to the temporary file created earlier. The option "-e" is used to allow tabbing with "\t" for a more appealing output stream. The data is appended in the *out* file with the use of "»" so the *hotelName* and *numOfreviews* are added for each file without being overwritten.

Finally, the **sort -k2nr** command is used to sort the temporary file output the hotel with the most reviews to the hotel with the least.. The option "-k2n" sorts the second column and the option "-r" is used to reverse the order of the output.

1.3 statistical_sig.sh

The statistical_sig.sh script is one script split into two parts

1.3.1 Part 1

```
#!/bin/bash

for f in reviews_folder/$1.dat reviews_folder/$2.dat
do
   nums+=$(awk '
   BEGIN {
   FS=">"
   } {
      if("<Overall"==$1) {
         n += 1;
         delta = $2 - mean;
         mean += delta / n;
         M2 += delta * (substr($0,10,1) - mean);
      }
   } END {
      print n;
      printf  mean " ";
      printf  M2 / (n-1) " ";
      printf sqrt(M2 / (n-1)) " ";
   }' $f)
done
```

The **for loop** loops through the 2 files in the *reviews_folder* directory. The 2 files are specified as the arguments(referred to in bash through *$1* and *$2*). For each file the **awk** command is used to create a "Field Separator", *FS* which scans through all the lines in the file and splits the line at ">". If the first field(*$1*) of the split string equals "<Overall" then add the second field(*$2*) to an algorithm which determines the number of reviews, mean, variance and standard deviation and stores them into *nums* as one string with each statistic separated by space.

1.3.2 Part 2

```
array=($nums)

#array[0]=N Hotel 1
#array[1]=Mean Hotel 1
#array[2]=Variance Hotel 1
#array[3]=Standard Deviation Hotel 1
#array[4]=N Hotel 2
#array[5]=Mean Hotel 2
#array[6]=Variance Hotel 2
#array[7]=Standard Deviation Hotel 2
tValue=$(echo "(${array[1]} - ${array[5]}) /sqrt((((${array[0]}-1)*${array[2]} + (${
    array[4]}-1)*${array[6]}) / (${array[0]} + ${array[4]} - 2)) * (1/${array[0]} + 1/
    ${array[4]}))" | bc -l)

printf "t: %.2f\n" $tValue
printf  "Mean $1: %.2f, SD: %.2f\n" ${array[1]} ${array[3]}
printf  "Mean $2: %.2f, SD: %.2f\n" ${array[5]} ${array[7]}
echo $tValue'>'1.960 | bc -l
```

Afterwards, an array called *array* is created. The calculated number of reviews, mean, variance and standard deviation for both reviews are stored into an array with each variable stored into separate index of *array*. Next, the t-statistic is calculated by simply substituting the corresponding statistic in the *array* to the t-statistic formula for 2 samples:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\left(\frac{(n_1-1)s_{X1}^2+(n_2-1)s_{X2}^2}{n_1+n_2-2}\right)\cdot\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}$$

The calculation is then **echo**ed and piped to **bc -l** which is the basic calculator and allows the floating point calculations to take place.
(*see Hypothesis Testing for more detail*)

Then, the t-statistic, mean and standard deviation are printed for each hotel with the use of the command **printf**. This command is more useful than echo in this case as it has the ability to work with floating points and hence round variables to 2 decimal places with the use of ".%2f". Finally, the t-statistic is compared to critical value(1.960 in this case) which outputs 0 if t-statistic is < critical value, otherwise outputs 1.

2 HYPOTHESIS TESTING

2.1 Problem

We are to test if the there is a significant statistical difference between the means of the overall for the top 2 hotels given by the output of *averagereviews.sh*.

2.2 Hypothesis Test

2.2.1 Hypotheses

H_0 : There is no significant difference between the means of the overall of the hotels

H_1 : There is a significant difference between the means of the overall of the hotels

2.2.2 Using

- *t = 0.03* (2 d.p)
- *5% significance*
- *2 tailed test*
- *Degrees of freedom = 449*
- *Critical value = 1.960*

2.3 Outcome

```
$ ./statistical_sig.sh hotel_188937 hotel_203921
t: 0.03
Mean hotel_188937: 4.78, SD: 0.63
Mean hotel_203921: 4.78, SD: 0.53
0
```

0.03 < 1.960 so there is significant evidence to reject H_1 and accept the null hypothesis, H_0. This shows that there is no significant difference between the means of the overall of hotel_188937 and hotel_203921.

3 DISCUSSION

This part discusses some of the improvements that TripAdvisor can make improve their way of collecting reviews.

- TripAdvisor allows any random user to write a review for any hotel. This may not be the best idea as hotels can pay people to give positive reviews for their hotel or negative reviews for rival hotels, helping increase their business.
  - A solution to this problem could be to only allow the users with TripAdvisor accounts to write reviews. The accounts could be tracked to see if there is large number of reviews being written in a short period of time to help detect fake reviews resulting in a more accurate, hence more reliable dataset.
  - Another solution could be to get the hotels to give a unique code for each booking which allows users to write a review for the particular hotel. This is very a accurate of collecting data as it would only enable people who have visited the hotel to write reviews, eliminating mostly all fake reviews.
  - Also, the accounts could be given a score which is generated by how many people have agreed with their reviews. Those accounts could be given a "verified" badge, making them trustworthy and so people are likely to trust them more.
- Another problem TripAdvisior face in collecting data this way is that the "overall" values do not correspond to the individual rating and so it would make more sense for the Overall value to be the averages of the individual ratings.
- One more problem with collecting data like this is that it can become outdated. The purpose of reviews is partly to also help the hotels improve their service. Therefore, if a hotel improves its service from bad to excellent, the bad reviews it has previously had will affect its "overall rating". This can be fixed by removing the oldest 2 weeks of data every 2 weeks of new data, leaving only the up-to-data of the hotels.

Implementing these solutions into the website and data collection could greatly improve TripAdvisor and help make TripAdvisor more reliable, allowing the company to provide better service for the users.