COMP1204: Unix Coursework

Huw Jones
27618153

February 24, 2016

1 Scripts

1.1 Count Reviews: Single File

```
grep -c "<Author>" $1
```

As every review has an author, it made sense to search for the “<Author>” string. Using the “-c” argument for grep returns a search count, rather than the string of occurrences. “$1” is used to get the first (real) argument when the script is executed on the command line.

1.2 Count Reviews: Directory

1.2.1 Explanation

```
function getReviewCount () {
  grep -c "<Author>" $1
}
```

This function returns the number of reviews in a hotel data file.

```
if [ -d $1 ]
then

...
fi
```

This checks whether the argument passed was a directory, and if so, execute the appropriate code.

```
for file in $1/*
do
  getReviewCount $file
done;
```

This section is executed if the argument passed as a directory. It iterates over the files in the directory, then calls the “getReviewCount” function for each file.

```
else
  getReviewCount $1
fi
```

This code is executed if the argument passed to the script is not a directory. It executes the “getReviewCount” as it should normally do for a file.

This script works for both 3.1.1: 1 and 2.

1.3 Count Reviews: Sorted

```
reviewCount=""
```

Initialises variable “reviewCount” to an empty string.

```
function getTrimmedHotelFile() {
  echo $1 | sed -e 's:^[/a-zA-Z_0-9\-]*\/::' -e 's:.dat::'
}
```

This functions uses sed substitution to remove all directory prefixes up to the last forward slash in the file path (/) from “$file”. It then uses another substitution to remove the .dat extension. Then, it assigns the result to “hotelName”.

The regex works as follows.

```
^
```

Matches from the start of the string

```
^[/a-zA-Z_0-9\-]*
```

Matches 0 or more all alphanumeric characters, dashes, forward slashes and underscores from the start of the string.

```
\/
```

Matches a forward slash. The backslash is used to escape it as forward slash is a special character.

```
^[/a-zA-Z_0-9\-]*\/
```

So, overall this regex matches all directories up to the last slash before the file name. This allows the script to remove the directory prefix. If sed supported “\s\S”, I would have used that instead of the alphanumeric mess that is currently used.

```
currentCount="$hotelName"$'\t'"$(getReviewCount $file)"
```

Sets “currentCount” to the hotel ID and the output of “getReviewCount”. The tab is used to make the output prettier than just using spaces.

```
reviewCount="$reviewCount"$'\n'"$currentCount"
```

Appends “currentCount” to the “reviewCount” using a newline so we can maintain 1 hotel per line.

```
echo -e "$reviewCount" | sort -k2nr
```

Echoes the contents of “reviewCount” to sdtOut whilst maintaining escape characters. The contents of “reviewCount” is then piped into the stdIn of sort, which reverse sorts the hotel list by number of reviews. Sort takes a “-k” argument that is used to specify the sort column. Here, “-k2nr” sorts the string by the second column (number of reviews)(“k2”), as a numeric type (“n”), reversely (“r”) (to sort by the greater number first).

1.4 Average Reviews

```
# Gets the average review of the hotel
function getAverageScore() {
  echo -e "$(getScores␣$1)" | awk '
  BEGIN {
    TotalScore=0;
    n=0;
  }

  {
    TotalScore += $0;
    n++;
  }

  END {
    printf ("%.2f\n", (TotalScore / n));
  }
  '
}
```

This function uses “awk” to calculate the mean. First, it gets a list of scores and pipes it to awk. Then, it instantiates “TotalScore” and “n” to 0. Next, it loops through the lines (of scores), adds the score to the total, then increments “n”. Finally, it uses “printf” to print the result of the $\frac{\sum x}{n}$ ($\bar{x}$) to 2 decimal places.

```
# Checks argument passed was a directory
if [ -d $1 ]
then
  # Loops through all files and prints HOTEL_ID AVERAGE_REVIEW
  for file in $1/*
  do
    currentHotel="$(getTrimmedHotelFile␣$file)"$'\t'"$(getAverageScore␣$file)"
    hotels="$hotels"$'\n'"$currentHotel"
  done

  # Sort hotels by second column (rating)
  echo -e "$hotels" | sort -k2nr
else
  echo "$(getTrimmedHotelFile␣$1)␣$(getAverageScore␣$1)"
fi
```

This part of the script checks a directory was passed as the first argument. Otherwise, it loops over every file in the data directory, then prints the hotel ID with the average review score, sorted by average review.

1.5 Statistical Significance

```
function getN() {
  echo -e "$(getScores␣$1)" | wc -l
}
```

Returns the number of reviews a hotel has.

```
function getSD() {
  file="$1"
  mean="$2"
  echo -e "$(getScores␣$1)" | awk '
    BEGIN {
      Total=0;
      n=0;
    }

    {
      Total += (($0 - '$mean')^2);
      n++;
    }

    END {
      var = Total/(n-1);
      printf("%.9f\n", sqrt(var));
    }
  '
}
```

This function calculates the standard deviation (for a sample). It uses the formula $\sigma = \frac{\sum(x_i-\bar{x})^2}{(n-1)}$. However, to maintain accuracy, it does not round the result to 2 decimal places, this is done later on when the result is printed to the terminal.

```
# Calculates the common standard deviation
function getSx1x2() {
  S2X1=$1
  nX1=$2
  S2X2=$3
  nX2=$4

  awk '
  BEGIN {
    numer=( ('$nX1' - 1) * ('$S2X1')^2) + ( ('$nX2' - 1) * ('$S2X2')^2);
    denom=( '$nX1' + '$nX2' - 2 );
    frac = numer / denom;
    printf ("%.9f\n", sqrt(frac) );
  }
  '
}
```

This function calulates sX_1X_2 and takes 4 arguments: S^2X_1, n_1, S^2X_2, and n_2. It uses awk to calulate $sX_1X_2 = \sqrt{\frac{(n_1-1)s^2_{X1}+(n_2-1)s^2_{X2}}{n_1+n_2-2}}$.

```
# Calculates the t-statistic
function getT_Statistic() {
  M1=$1
  n1=$2
  M2=$3
  n2=$4
  Sx1x2=$5
  awk '
  BEGIN {
    numer=( '$M1' - '$M2' );
    denom=( '$Sx1x2' * sqrt((1/'$n1') + (1/'$n2')) );
    printf ("%.9f\n", (numer/denom) );
  }
  '
}
```

Using the formula $t = \frac{\bar{X}_1-\bar{X}_2}{sX_1X_2\cdot\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}$, this function cal the t-statistic. It takes 4 arguments, "M1"/"M2" ($\bar{X}_1$, $\bar{X}_2$) and "n1"/"n2" (n_1, n_2).

```
function round(){
  awk 'BEGIN { printf ("%.'$2'f\n", '$1') }'
}
```

Rounds a float to a given number of decimal places. It takes 2 arguments, “$1”, the number to round, and “$2””, the number of places.

2 Hypothesis Testing

3 Discussion

3.1 Storage

TripAdvisor currently uses a tag based format to save their reviews. The format requires the use of one file per hotel.

3.1.1 Format

Comments are provided using double forward slash (C style).

```
// Hotel Identifier
<Overall Rating>n
<Avg. Price>$nnn
<URL>xxx
          // newline
<Author>
<Content>
<Date>MMM d, YYYY
<img>        //optional
<No. Reader>n
<No. Helpful>n
<Overall>
<Value>
<Rooms>
<Location>
<Cleanliness>
<Check in / front desk>
<Service>
<Business service>
```

This format is very open for interpretation. For example, there is no end deliminator for each field. Since there seems to be no concrete specification of how the data is to be interpreted, software could assume that the end of a line signifies the end of a field. However, there are fields that span multiple lines - namely the “Content” field.

Compared to the JSON or XML specifications, there is no logical structure to the data. Structuring your data properly reduces the time to process it, and makes it easier to comprehend; not only for humans, but for machines too.

3.1.2 Repetition

Secondly, storing the data in this format creates a lot of repetition. For each review, the tag names are specified. With hotels in the hundreds of thousands and each hotel having a multiple reviews, there is a lot of duplication. On a scale this large, that amount of data duplication should be avoided.

A Relational Database Management System (RDBMS) would be beneficial in storing this data. Depending on the schema selected, this would impose some limits (e.g.: maximum content length), however they should not affect the storage of reviews compared to the current system.

3.1.3 Data Relationships

An RDBMS would allow the individual hotel details to be migrated away from being stored with reviews. Not only would this allow for the schema to integrate with the rest of the site, it would allow the reviews to be separated.

A foreign key constraint (FKC) could be used to maintain a relationship with the review and the hotel. Additionally, the review could also be linked to a user table. Linking reviews to specific users could also introduce the possibility of a reputation-based system.

3.1.4 Reputation

A reputation based system could offer the ability for reviewers with more, higher quality reviews to affect the bias of a hotel’s ranking (similar to StackOverflow, but better). User reputation could also be used to filter trustworthy, or untrustworthy reviews. This means bots, or spamming users, would be able to get filtered out from the pool of trustworthy reviews. On the other hand, this would impeded new users’ ability to raise their reputation due to the competition with existing reviewers (a problem StackOverflow encounters).

The sytem should also take into account the number of reviews, average rating, and user reputation in order to produce a metric that can rank hotels accurately and as unbiased as possible (within reason). Take Amazon for example, it could recommend a product with 2, 5 star reviews. However, is a product with 2, 5 star reviews better than a product with 2,000 reviews and an average rating of 4.5? Somehow the system would need to balance this to allow well reviewed hotels to not take precedance over slightly less known, but equally as well reviewed hotels. Due to the complex nature of this system, caching hotel rankings and periodically updating them, would offer better performance then calculating a hotel's rank everytime it's page is requested.

3.1.5 Speed

RDBMS's can also be orders of magnitude faster than the existing file based system. This means querying or performing statistical analysis on the data (as completed in this report) would be able to be completed with a greater speed and ease. RDBMS provides Structured Query Language (SQL) as a means of querying the database. SQL also contains functions, and can be used to create functions that perform the statistical anaylsis.