Scientific Programming

October 26, 2018

1 Words

1. There are **97723** unique words.
2. There are **25751** words with apostrophes.
3. There are **159** words with non-ASCII characters.
4. There are **10** pairs of -OG / -OGUE words in the database. These are given in table 1.

-OG	-OGUE
ANALOG	ANALOGUE
CATALOG	CATALOGUE
DEMAGOG	DEMAGOGUE
DIALOG	DIALOGUE
EPILOG	EPILOGUE
MONOLOG	MONOLOGUE
PEDAGOG	PEDAGOGUE
PROLOG	PROLOGUE
SYNAGOG	SYNAGOGUE
TRAVELOG	TRAVELOGUE

Table 1: -OG/-OGUE words

5. This has been done (see code).
6. A histogram of scrabble scores is given below. The highest scoring word is **PIZZAZZ** for 45 points.

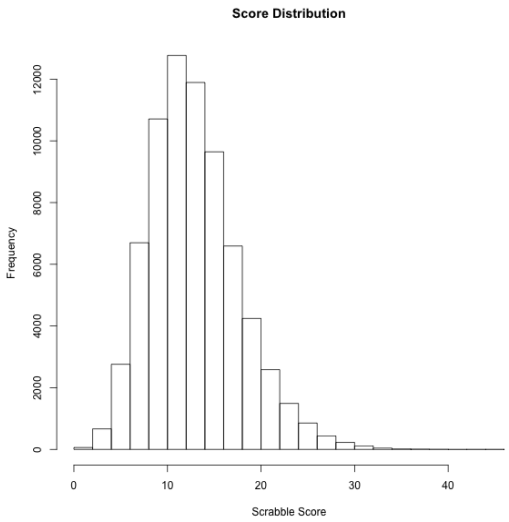

Figure 1: Histogram of scrabble scores

7. There are **120** words with reverse complements in the database. These are:
   A AL ARIZ AS B BIRD C CI CL D E F FLO FM G GILL H HE HF HG HO HORN HZ I IN IO IR J K L LN LOU LR LT LYRA M MILO MN MO MR N ND NOV O OK OORT ORLY OZ P PL POLK Q R RH RN RX S SI T U UR V W WM X Y YB Z ZIBO ZN ZOLA ASP BEVY BID BIG BOIL BOORISH BOORS BY ELM GIRT GO GRID GRIT HI HILLY HOOFS HOVELS HULLS IF KHZ LO LORN LS MA MI MILD MILK MILS NU OX PORN RS SH SHRILLY TRIG TRY TS US VELD VOLE VS WILD WIRY WIZARD WORD WORN WOVE WRIT WRY

8. From the list of [F A L U Y P L N I], we can construct **39** words at least 4 letters long and containing the letter 'A' that are in the database. These are:
   ALLY AINU FALL FAUN FAIL FAIN FLAY FLAP FLAN FLAIL FULANI FINAL FINALLY LAIN LULA LUNA LILA LINA ULNA YALU YUAN PALL PAUL PAULI PAIL PAILFUL PAIN PAINFUL PAINFULLY PLAY PLAYFUL PLAN PLAIN PLAINLY PIAF PILAF PILAU NAIL INLAY

2 Examination marking

The filled-out table of student results is given in table 2.

student	score	grade	rank
1	19	B	6
2	24	A	2
3	14	D	10
4	17	C	8
5	20	B	5
6	23	A	3
7	29	A	1
8	7	F	12
9	22	A	4
10	17	C	8
11	9	F	11
12	18	C	7

Table 2: Student examination results

To test for students cheating, we define a pairwise penalty function between two students which quantifies how similar their answers are. If we denote the average proportion of correctly answered questions p, the probability of two students both answering a question correctly given both students attempting the question and no cheating occurring is

$$P(c, c|ans, no\_cheat) = p^2$$

On the other hand, the probability that they both give the same incorrect answer is

$$P(i, i|ans, no\_cheat) = (\frac{1-p}{4})^2$$

Since probabilities are multiplicative, their logs are additive and we can therefore define a probabilistic similarity score based on log likelihoods which we denote S

$$S(a, b) = -N(c, c) * log_{10}(p^2) - N(i, i) * log_{10}((\frac{1-p}{4})^2)$$

Where $N(c, c)$ and $N(i, i)$ are the number of identical correct and incorrect answers respectively between students a and b. In the present case, $p = 0.608$ giving $-log_{10}(p^2) = 0.43$ and $-log_{10}((\frac{1-p}{4})^2) = 2.02$. A funtion has been written that will automatically check if two students have similar results based on this similarity measure (see code).

It is hard to predict what the similarity distribution would be in the absence of cheating, as it will be biased towards more similarity if some questions are easier than others, leading to a higher proportion of students picking similar

questions than expected by random chance, or if some wrong options are more likely to be picked than others.

However, we can visualize the distribution of all 66 pairwise similarity scores (figure 2). If we assume that the majority of students do not cheat, we would expect pairs of cheating students to give rise to outliers in the distribution. We can also visualize this in two dimensions by plotting a heatmap of similarity scores.

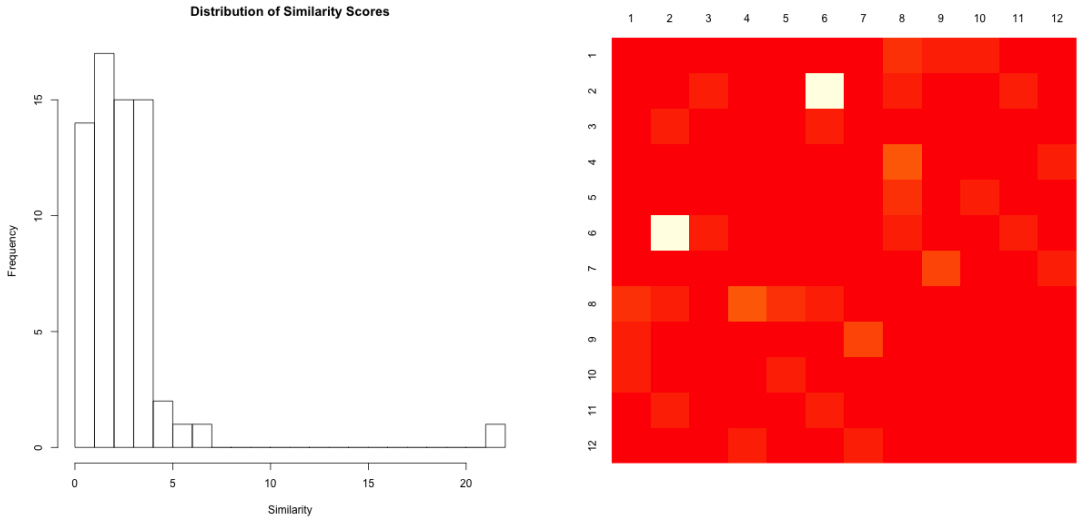

Figure 2: Histogram (left) and heatmap (right) of similarity scores

We clearly see that one pair of students is a significant outlier, and that this pair consists of students 2 and 6. To make this slightly more quantitative, we can rank the similarity scores with the top 5 pairs given in table 3. The number of standard deviations each pair is from the mean of all *other* similarity scores is also given, since we may assume that cheating scores will deviate from the non-cheating distribution.

Pair	Similarity	Standard Deviations
(2,6)	21.2	14.2
(4,8)	6.5	1.5
(7,9)	5.5	1.1
(1,8)	4.9	0.9
(5,8)	4.5	0.8

Table 3: Table of similarity scores

Again we see the pair (2, 6) as a significant outlier, whereas the other high-scoring pairs are not far enough from the mean to justify accusations of cheating. It is worth noting that this analysis is somewhat confounded by the (2,6) pair being a far outlier and thus bringing up the standard deviation of the set of similarity scores where it is not excluded, leading to relatively smaller deviations for the remaining pairs.

After concluding that the responses of students 2 and 6 are suspiciously similar, we can check the actual data and see that they answered 27 of the same questions and gave the same answer to every question they both answered, even though 6 of these were incorrect. We thus conclude that students 2 and 6 cheated, although it is not clear if there was a directionality to the cheating.

We can now remove the S(2,6) from our list of scores and rerun the analysis to see if any other pairs are suspiciously far off the distribution mean (table 4).

These scores do not seem particularly suspicious, although if we assume scores to be normally distributed, there is only a 0.02% chance of a score being 3.6 standard deviations from the mean which might be considered significant even after correcting for 65 comparisons.

This has been automatized further in the function catch_cheat where, given a Bonferroni-corrected cutoff of e.g. 0.01, pairs of potentially cheating students are iteratively flagged and removed until there are none exceeding this threshold (with a threshold of 0.01, only the pair (2,6) are flagged as cheating).

Pair	Similarity	Standard Deviations
(4,8)	6.5	3.6
(7,9)	5.5	2.6
(1,8)	4.9	2.1
(5,8)	4.5	1.8

Table 4: Table of similarity scores after removing S(2,6)

3 Appendix

```
##code for doing practical 1


####start by defining functions we will need

get_grade = function(x){
  #Given a fraction of correctly answered questions, returns a letter grade
  x = floor(100*x)
  if (x >= 70){return('A')}
  else if ( x >= 60 ){return('B')}
  else if ( x >= 50 ){return('C')}
  else if ( x >= 40 ){return('D')}
  else {return('F')}
}

cheat_calc = function( a1, a2, c = crib, pc = -log10(pcorr^2), pf = -log10( ((1-pcorr)/4)^2 ))
  #given penalty scores for identical correct (pc) and incorrect (pf) answers,
  #calculates a similarity score between two students with answer vectors a1 and a2
  corrs = (a1 == a2 & a1 == c)
  falses = (a1 == a2 & a1 != c & a1 != 'NA')
  score = pc * sum(corrs) + pf * sum(falses)
return(score)
}


getASCII = function(x){
  #returns TRUE if word x contains only ASCII characters, FALSE otherwise
  x = utf8ToInt(x)
  return(all(x < 128)) #ASCII characters are the first 127 utf8 characters
}

getScore = function(x, scores = sorted){
  #calculates the scrabble score of word x given an alphabetical list of character scores (sco
  score = 0
  for (i in 1:nchar(x)){ score = score + scores[substr(x, i, i)] }
  return(score)
}

revcomp = function(x, revcomplist = comp){
  #Given a word x, generates the reverse complement of x. revcomplist is a list of reverse con
  rc = ''
  for (i in nchar(x):1){rc = paste0(rc, revcomplist[substr(x, i, i)])}
  return(rc)
}

can_make = function(x, ls = c('A', 'F', 'L', 'U', 'Y', 'P', 'L', 'N', 'I') ){
  #Given a word x, returns TRUE if it can be made using characters in ls, FALSE otherwise.
```

```
  for (i in 1:nchar(x)){
    N = substr(x, i, i)
    if (N %in% ls){ ls = ls[-match(N, ls)]} #see if character in list of allowed characters
    else {return(FALSE)} #return false if no match
  }
  return(TRUE)
}

catch_cheat = function(scores, pairs, cutoff = 0.01){
  #given a list of scores and pairs, reports cheaters with a probability threshold of cutoff
  #Use: catch_cheat(sim_scores, pairs) after running code_sp.R
  p = 0
  while( p < cutoff ){
    top = scores[ order(scores, decreasing = 1)[1] ]
    ind =
      pair = pairs[ scores == top ]
    pairs = pairs[ scores != top ]; scores = scores[ scores != top ]
    m = mean(scores)
    sd = sd(scores)
    p = pnorm( (m-top)/sd )*(length(scores)+1)
    if (p < cutoff){
      cat('\nstudents', pair, 'appear to be cheating')
      cat('\nsimilarity is', top, 'corresponding to p=', p, '\n')
    }
    else{cat('\nno more students appear to be cheating, most suspicious is', pair, top, p)}
  }
}

###now do the actual exercises

#load file, convert to uppercase and take only unique elements
f = unique(toupper(as.vector(unlist(read.delim('files/usr-share-dict-words', header=FALSE)))))

cat('there are', length(f), 'unique words\n')

aps = grep("'", f) #find words with apostrophes
cat('there are', length(aps), 'words with apostrophes\n')

f = f[-aps] #remove words with apostrophes

ascii = sapply(f, getASCII) #get vector where each element is 1 if all ASCII, 0 otherwise
cat('there are', length(f[!ascii]), 'words with non-ASCIIs')

f = f[ascii] #remove non-ASCII words

og = f[grep('*OG$', f)]
ue = f[grep('*OGUE$', f)]

ogues = vector()
for (word in ue){
  #for each XOGUE word, seach our list of YOG words for X=Y
  cat('\n', word, substr(word, 1, nchar(word)-2), '\n')
  w = og[ og == substr(word, 1, nchar(word)-2) ]
  cat(w, '\n')
  ogues = c(ogues, w)
}
```

```
cat('\nThere are', length(ogues), 'ogue words. These are\n', ogues, '\n')

scores = read.table('files/scrabble.txt', header=FALSE)
alph = unlist(scores[,1]) #extract list of letters in table
sorted = numeric(25); sorted[as.integer(alph)] = unlist(scores[,4]) #factors are alphabetical
names(sorted) = sort(alph) #now alphabetical list of letter (name) and score (value)

scrabbles = sapply(f, getScore) #compute vector of scrabble scores

png(file='scrabble_hist.png', w=600, h=600)
hist(scrabbles, xlab = 'Scrabble Score', main = 'Score Distribution') #plot hist
dev.off()

m = max(scrabbles) #max score
mword = names(scrabbles[which(scrabbles == m)]) #highest scoring word
cat('\nmax word is', mword, 'with', m, 'points')

comp = rev(as.vector(sort(alph))); names(comp) = rev(comp) #vector of reverse complement lette
revcomps = vector()
#find all words with a reverse complement in the database
for (w in f){ if (revcomp(w) %in% f){revcomps = c(revcomps, w)}}

temp = revcomps
revcomps = sapply(temp, revcomp)
#generate list where names are words, values are reverse complements of those words
names(revcomps) = temp
cat('\n\n', revcomps, '\n')


####Now find the possible word combinations.

#First construct new database of allowed words, then check if they can be made from our charac

witha = f[ grep('A', f) ]; witha = witha[ nchar(witha) > 3 ]
ls = c('F', 'A', 'L', 'U', 'Y', 'P', 'L', 'N', 'I') #allowed characters
#test = c('APLN', 'ABBA', 'AFLUY') #test case

realwords = witha[ as.vector(unlist(sapply(witha, can_make))) ] #get possible words
cat('\nwe can construct', length(realwords), 'unique words from our letters. These are:\n')
print(realwords)


####Now do the student marking and cheating task

#read in data
crib = as.vector(unlist(read.delim('files/grading/crib.dat', header=FALSE)))
grade_scheme = read.table('files/grading/grade.txt', header=TRUE)

#initialize dataframe of results
res = data.frame(student = 1:12, score = numeric(12), grade = character(12), rank = numeric(12
                 stringsAsFactors = FALSE)
answers = vector('list', 12) #list of vectors of answers
scores = numeric(12) #vector of student scores
for (i in 1:12){ #treat each student individually
  ans = read.table(paste0('files/grading/student',i,'.dat'), header=TRUE, stringsAsFactors = F
  #make a list of 100 'NA', change the answered questions to the relevant answer
  answers[[i]] = rep('NA', 100); answers[[i]][ans[,'qn']]=ans[,'response']
```

```
  score = 0
  for (j in 1:dim(ans)[1]){ if (ans[j,2] == crib[ans[j,1]]){score = score+1} } #count scores
  cat('\n', i, 'score', score)
  res[i, 'score' ] = score; scores[i] = score
  cat('\n', 'grade', get_grade(score/30), '\n')
  res[i, 'grade' ] = get_grade(score/30)
}

scores = sort(scores, decreasing = TRUE) #sort by score to find ranks
for (i in 1:12){res[i, 'rank'] = which(scores == res[i, 'score'])[1]} #set ranks

###Test for cheating

pcorr = mean(scores)/30 #total probability of a question being answered correctly
pc = -log10(pcorr^2) #probabiliy that two students both provide a correct answer
pf = -log10( ((1-pcorr)/4)^2 ) #probability that two students both provide the same incorrect

cheats =  matrix(NA, 12, 12)  #initialize matrix of similarity scores
plotCheat = matrix(0, 12, 12) #need a slightly different data structure for plotting using 'im
sim_scores = numeric(0); pairs = numeric(0) #construct these for automatic cheat detection

for (i in 1:11){
  for (j in (i+1):12){
    cheat = cheat_calc( answers[[i]], answers[[j]]) #calculate pairwise similarity scores
    cheats[i,j] = cheats[j,i] = cheat
    plotCheat[13-i, j] = plotCheat [13-j, i] = cheat
    sim_scores = c(sim_scores, cheat); pairs = c(pairs, paste0('(',i,',',j,')') )
  }
}


png(file='cheat_heat.png', w=600, h=600)
image(t(plotCheat), col=heat.colors(20), axes = 0) #plot heatmap
axis(3, at = seq(0, 1, length = 12), labels=1:12,srt=45,tick=FALSE)
axis(2, at = seq(0, 1, length = 12), labels=12:1,srt=45,tick=FALSE)
dev.off()

u = sim_scores[ order(sim_scores, decreasing = 1)[1:5] ] #find the top similarity scores
cat( u )

inds = vector('list', length(u))
for (i in 1:length(u) ){ inds[[i]] = which( cheats == u[i], arr.ind = 1 )[1,] }
print(inds) #find the most similar pairs

png(file='cheat_hist.png', w=600, h=600) #plot histogram.
hist(sim_scores, 30, main = 'Distribution of Similarity Scores',
     xlab = 'Similarity', ylab = 'Frequency')
dev.off()

sds = numeric(5)
for (i in 1:5){
  #get the number of standard deviations of the top similarity scores from the mean of the sim
  #distribution excluding that particular score
  m = mean(sim_scores[sim_scores != u[i]], na.rm = 1)
  sd = sd(sim_scores[sim_scores != u[i]], na.rm = 1)
  sds[i] = (u[i]-m)/sd
  cat('\n', m, sd, sds[i], '\n')
```

```
}

new_scores = sim_scores[sim_scores != u[[1]]] #remove highest scoring score
for (i in 2:5){
  #get the number of standard deviations of the top similarity scores from the mean of the sim
  #distribution excluding that particular score after removing 2,6
  m = mean(new_scores[new_scores != u[i]], na.rm = 1)
  sd = sd(new_scores[new_scores != u[i]], na.rm = 1)
  sds[i] = (u[i]-m)/sd
  cat('\n', m, sd, sds[i], '\n')
}


#####Or we can do the above 'automatically', listing only the cheating results

catch_cheat(sim_scores, pairs)
```