FIT5197 2018 S1 Assignment 1

Yang (Kelvin) Li (questions, some answers) and Wray Buntine (some answers)

12 March 2018

Question 1: calculate conditional probability of an event

Tossing a fair die 7 times. Consider the following events, A = `each value appears at least once and $B$ =`the outcome is alternate in numbers (i.e., no two values are adajacent). What is $p(A|B)$? Solve this analytically (a formula) and experimentally by simulation, in each case printing the value found.

```
# analytical solution,

# p(B) - anything for first toss, next 6 can be any 5 values (out of 6)
# they do not have to give this explicitly, it can be hidden in the formula

(5/6)^6
```

```
## [1] 0.334898
```

```
# full space of possibilities

(6^7)
```

```
## [1] 279936
```

```
# p(A, B) - several ways of doing this, at least,

(6*factorial(5)*(choose(7,2) - 6) / (6^7))
```

```
## [1] 0.03858025
```

```
(6*factorial(5)*(6*5/2) / (6^7))
```

```
## [1] 0.03858025
```

```
# p(A|B) - ratio

(6*factorial(5)*(choose(7,2) - 6) / (6^7)) / (5/6)^6
```

```
## [1] 0.1152
```

```
# experimental solution

#  one way (of many) to do a "no alternate" check
check_alt = function(x) {
  alt = TRUE
  for (i in 1:(length(x) - 1)) {
    if (x[i] == x[i + 1]) {
      alt = FALSE
      break
    }
  }
  return(alt)
}
```

```
cntAlt = cntAppearOnce = 0
n = 100000
for (i in 1:n) {

  samples = sample(6, 7, replace = TRUE)
  if (check_alt(samples)) {
    cntAlt = cntAlt + 1
    if (length(unique(samples)) == 6) {
      cntAppearOnce = cntAppearOnce + 1
    }
  }

}
cntAppearOnce / cntAlt
```

```
## [1] 0.1146958
```

Question 2: entropy

Given the dataset which contains 2 discrete random variables X and Y, and 100 samples with missing values (NA). Do the following:

- handle NAs by mode imputation, and plot variables in a histogram with proper axis labels and title.
- calculate p(X), p(Y), p(X, Y), p(X|Y), p(Y|X).
- calculate H(X), H(Y), H(X|Y) and H(Y|X)

```
df <- read.csv(file="FIT5197_2018_S1_Assignment1_Q2_data.csv",header=TRUE, sep=",")
#  do this to get parameters for the imputation (mode, etc)
summary(df)
```

```
##        X                Y
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :1.0000
##  Mean   :0.4556   Mean   :0.5402
##  3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
##  NA's   :10       NA's   :13
```

```
# imputation
#  this example is "univariate sample", could do "mode"
#  which means assign 0 to X and 1 to Y

for (i in 1:nrow(df)){
     if ( is.na(df$X[i]) ) df$X[i] = rbinom(1,1,0.4556)
}
for (i in 1:nrow(df)){
     if ( is.na(df$Y[i]) ) df$Y[i] = rbinom(1,1,0.5402)
}

# check done OK
summary(df)
```

```
##        X              Y
##  Min.   :0.00   Min.   :0.00
##  1st Qu.:0.00   1st Qu.:0.00  
##  Median :0.00   Median :1.00  
##  Mean   :0.43   Mean   :0.52  
##  3rd Qu.:1.00   3rd Qu.:1.00  
##  Max.   :1.00   Max.   :1.00  
```

```
# get marginal totals
t <- table(df)
t
```

```
##    Y
## X    0  1
##   0 25 32
##   1 23 20
```

```
#  many ways to compute these, as long as the computation is in R and correct
#  could be dumb enumeration of values with for loops!
#
# WARNING:   answer will vary depending on imputation method ...

# p(X)
px=margin.table(t,1)/margin.table(t)
px
```

```
## X
##    0    1 
## 0.57 0.43
```

```
# p(Y)
py=margin.table(t,2)/margin.table(t)
py
```

```
## Y
##    0    1 
## 0.48 0.52
```

```
# p(X,Y)
prop.table(t)
```

```
##    Y
## X      0    1
##   0 0.25 0.32
##   1 0.23 0.20
```

```
# p(X|Y)
prop.table(t,2)
```

```
##    Y
## X           0         1
##   0 0.5208333 0.6153846
##   1 0.4791667 0.3846154
```

```
# p(Y|X)
prop.table(t,1)
```

```
##    Y
## X           0         1
##   0 0.4385965 0.5614035
##   1 0.5348837 0.4651163
```

```r
#  am too lazy to write my own, and don't care how they do it
#install.packages("entropy")
library("entropy")
# gives base e by default

# H(X)
entropy(px,unit="log2")
```

```
## [1] 0.985815
```

```r
# H(Y)
entropy(py,unit="log2")
```

```
## [1] 0.9988455
```

```r
# H(X|Y)
py[1]*entropy(prop.table(t,2)[1:2,1],unit="log2") + py[2]*entropy(prop.table(t,2)[1:2,2],unit="log2")
```

```
##         0
## 0.9792417
```

```r
# H(Y|X)
px[1]*entropy(prop.table(t,1)[1,1:2],unit="log2") + px[2]*entropy(prop.table(t,1)[2,1:2],unit="log2")
```

```
##         0
## 0.9922722
```

Question 3: correlations and covariance

Assuming X and Y be two independent standard Gaussian random variables, $U = X - Y$ and $V = 2X + 3Y$. What is the correlation and covariance between X and Y? Solve the questions analytically, then confirm your analytical results using $1,000,000$ simulation.

```r
#  analytically, cov = -1

#  analytically, corr = -1/sqrt(26) = -0.196116

# generate data
n = 1000000
x = rnorm(n)
y = rnorm(n)
u = x - y
v = 2 * x + 3 * y

# cov function

mean(u * v)
```

```
## [1] -1.000698
```

```r
# corr function

mean(u * v) / (sd(u) * sd(v))
```

```
## [1] -0.1964444
```

Question 4: maximum likelihood estimation of parameters

Solve MLE of Poisson/Bernoulli/Binomial parameter analytically and numerically (without using existing mle functions in R) Hint: you should first define a log likelihood function. Then use the optimize() function in R to find the optimal parameters for the given dataset.

```r
data <- c(4, 3, 2, 4, 6, 3, 4, 0, 5, 6, 4, 4, 4, 5, 3, 3, 4, 5, 4, 5)

# analytical soln
mean(data)
```

```
## [1] 3.9
```

```r
# computational function
fn <- function(lambda) {
   val <- 0
   for (i in 1:20) {
      # OK for skipping factorial(data[i])
      val <- val + data[i]*log(lambda) - lambda
   }
   return(val)
}

# test out
fn(3.9)
```

```
## [1] 28.15617
```

```r
fn(5)
```

```
## [1] 25.53616
```

```r
fn(3)
```

```
## [1] 25.69176
```

```r
optimize(fn, c(0,10), maximum = TRUE, tol = .Machine$double.eps^0.25)
```

```
## $maximum
## [1] 3.899995
##
## $objective
## [1] 28.15617
```

Question 5: central limit theorem

Sample a sequence of 10 i.i.d. random variables from a poisson distribution with $\lambda = 10$. Experimentally justify the central limit theorem using simulation with sample size 100, 1000, 10000 and 100000. You should compute both theoretical and sample mean and sd. In addition, plot each result in a histogram with the theoretical Gaussian curve.

```r
lambda = 10
nRandVar = 10
f = function(x, mean = cltMeanTheo, sd=cltSdTheo) dnorm(x, mean = mean, sd = sd)

n = 100
sampleMeans = rep(0, n)
for (i in 1:n) {
  sampleMeans[i] = mean(rpois(nRandVar, lambda))
}
(cltMeanTheo = lambda)
```

```
## [1] 10
```

```r
(cltSdTheo = sqrt(lambda/nRandVar))
```

```
## [1] 1
```

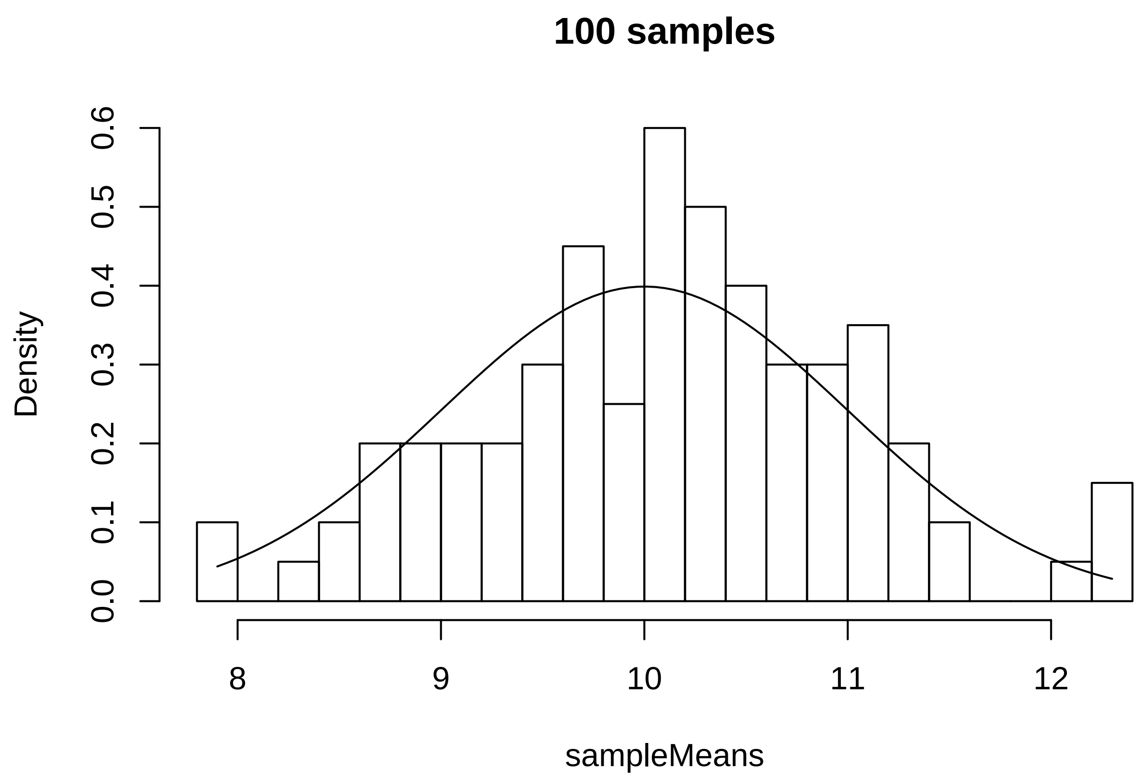
```r
(cltMeanEmp = mean(sampleMeans))
```

```
## [1] 10.162
```

```r
(cltSdEmp = sd(sampleMeans))
```

```
## [1] 0.9160499
```

```r
h = hist(sampleMeans, main = "100 samples", breaks = 30, freq = F)
curve(f, from = min(sampleMeans), to = max(sampleMeans), add = T)
```



```r
n = 1000
sampleMeans = rep(0, n)
for (i in 1:n) {
  sampleMeans[i] = mean(rpois(nRandVar, lambda))
}
(cltMeanTheo = lambda)
```

```
## [1] 10
```

```
(cltSdTheo = sqrt(lambda/nRandVar))
```

```
## [1] 1
```

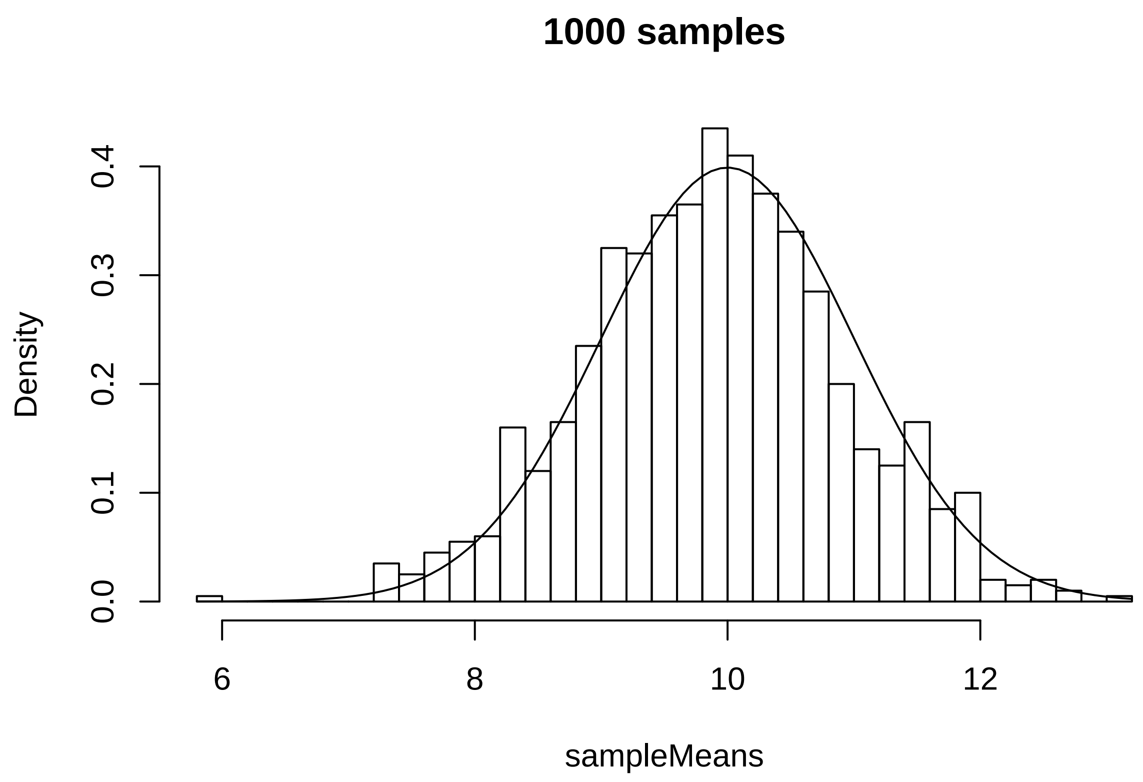
```
(cltMeanEmp = mean(sampleMeans))
```

```
## [1] 9.9445
```

```
(cltSdEmp = sd(sampleMeans))
```

```
## [1] 1.015619
```

```
h = hist(sampleMeans, main = "1000 samples", breaks = 30, freq = F)
curve(f, from = min(sampleMeans), to = max(sampleMeans), add = T)
```



```
n = 10000
sampleMeans = rep(0, n)
for (i in 1:n) {
  sampleMeans[i] = mean(rpois(nRandVar, lambda))
}
(cltMeanTheo = lambda)
```

```
## [1] 10
```

```
(cltSdTheo = sqrt(lambda/nRandVar))
```

```
## [1] 1
```

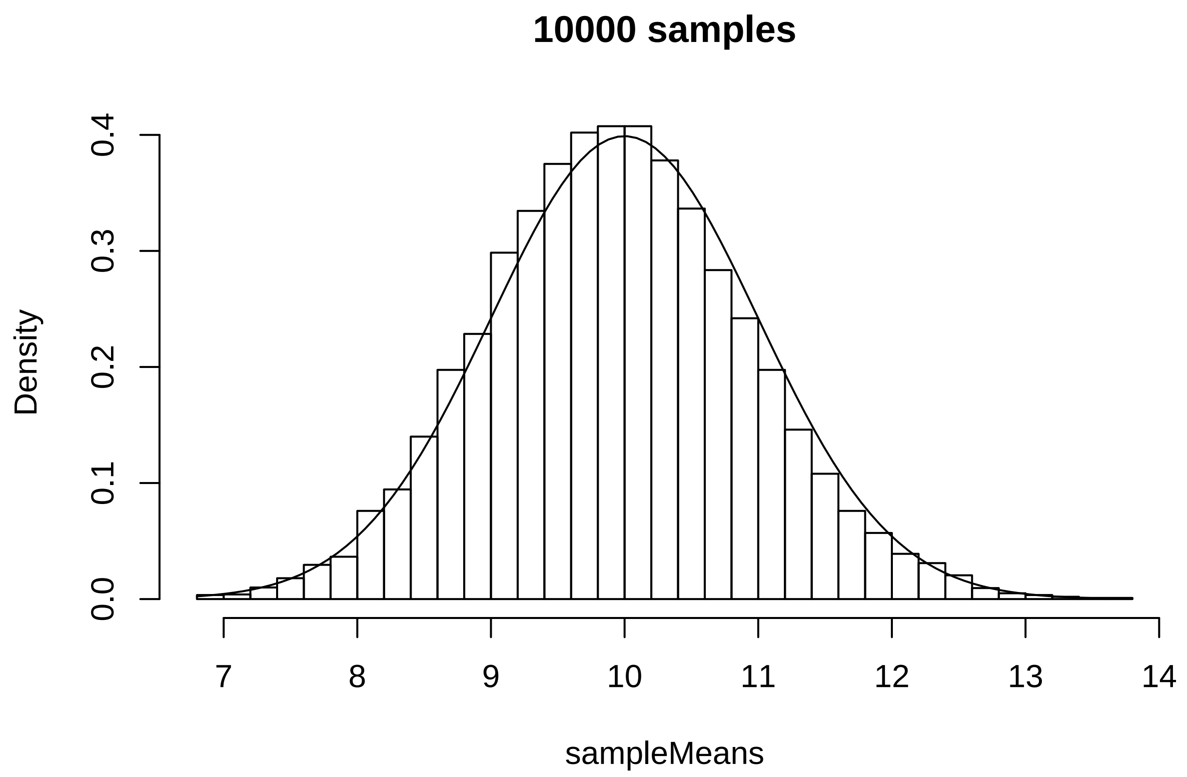
```
(cltMeanEmp = mean(sampleMeans))
```

```
## [1] 9.99106
```

```
(cltSdEmp = sd(sampleMeans))
```

```
## [1] 0.9812565
```

```
h = hist(sampleMeans, main = "10000 samples", breaks = 30, freq = F)
curve(f, from = min(sampleMeans), to = max(sampleMeans), add = T)
```



```
n = 100000
sampleMeans = rep(0, n)
for (i in 1:n) {
  sampleMeans[i] = mean(rpois(nRandVar, lambda))
}
(cltMeanTheo = lambda)
```

```
## [1] 10
```

```
(cltSdTheo = sqrt(lambda/nRandVar))
```

```
## [1] 1
```

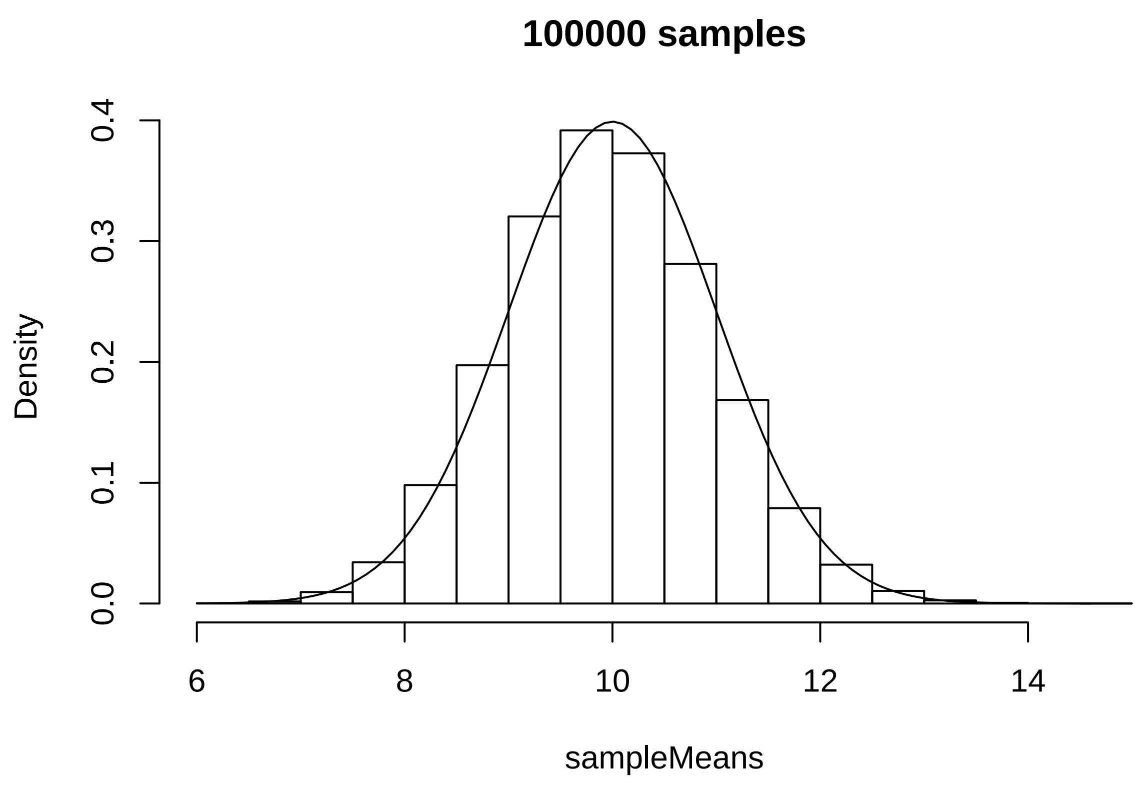
```
(cltMeanEmp = mean(sampleMeans))
```

```
## [1] 9.999079
```

```
(cltSdEmp = sd(sampleMeans))
```

```
## [1] 0.9995452
```

```r
hist(sampleMeans, main = "100000 samples", breaks = 30, freq = F)
curve(f, from = min(sampleMeans), to = max(sampleMeans), add = T)
```


100000 samples
Density
sampleMeans