Assignment 12, Introduction to Statistics

Oleg Sivokon

<2015-03-27 Fri>

Contents

1 Problems

1.1 Problem 1

Given the lottery ticket can have six numbers chosen from 1 through 6. Each play selects a six digits number and the players are awarded according to the number of digits they guessed.

1. What is the chance of guessing all numbers?
2. What is the chance of guessing exactly three of all numbers?
3. What is the chance of the winning number to be a palindrome?

1.1.1 Answer 1

The chance of guessing all numbers can be calculated as a product of probabilities of guessing each number independently. Probability of guessing one number is one in six, thus the total probability of guessing the number is $\frac{1}{6^6} = 2.143\,347 \times 10^{-5}$.

1.1.2 Answer 2

The probability of guessing exactly three numbers is the probability of guessing three numbers times the probability of guessing other three not winning numbers, as many times as we can choose combinations of three out of six, i.e.: $\binom{6}{3} \times \frac{1}{6^3} \times \frac{5^3}{6^3} = 5.358\,367\,8 \times 10^{-2}$.

1.1.3 Answer 3

The probability of a six-digit number being a palindrome is the product of first and last numbers being the same, second and fifth being the same and third and fourth being the same. Observe now that the condition of being the same is equivalent to requiring that one of the numbers of the pair be exactly of the six possible results, hence the probability of two given numbers matching is exactly $\frac{1}{6}$, thus total probability is $\frac{1}{6^3} = 4.629\,629\,7 \times 10^{-3}$.

Here's the calculation that verifies the results:

```
(defun generate-ticket ()
  (loop :repeat 6 :collect (random 6)))

(defun exactly-3-match (a b)
  (= 3 (loop :for i :in a :for j :in b
          :when (= i j) :count 1)))

(defun palindromep (tested) (equal tested (reverse tested)))

(defun num->ticket (n)
  (nreverse
   (loop :repeat 6 :collect (mod n 6) :do (setf n (floor n 6)))))

(defun ticket->num (ticket)
  (reduce (lambda (a b) (+ (* 6 a) b)) ticket :initial-value 0))

(defun next-ticket (previous)
  (num->ticket (1+ (ticket->num previous))))

(defparameter *all-tickets* (expt 6 6))

(defun chance-of-winning ()
  (/ (loop :with ticket := (generate-ticket)
        :repeat *all-tickets*
        :for attempt := '(0 0 0 0 0 0) :then (next-ticket attempt)
        :when (equal attempt ticket) :count 1)
     *all-tickets*))

(defun chance-of-three-matching ()
  (/ (loop :with ticket := (generate-ticket)
        :repeat *all-tickets*
        :for attempt := '(0 0 0 0 0 0) :then (next-ticket attempt)
        :when (exactly-3-match ticket attempt) :count 1)
     *all-tickets*))

(defun chance-of-palindrome ()
  (/ (loop :repeat *all-tickets*
        :for attempt := '(0 0 0 0 0 0) :then (next-ticket attempt)
        :when (palindromep attempt) :count 1)
     *all-tickets*))

(format t "~&Chance of winning the lotery:     ~f~%~
             Chance of guessing exactly three: ~f~%~
             Chance of palindrome ticket:      ~f"
        (chance-of-winning)
        (chance-of-three-matching)
        (chance-of-palindrome))
```

```
Chance of winning the lotery:     0.00002143347
Chance of guessing exactly three: 0.053583678
Chance of palindrome ticket:      0.0046296297
```

1.2 Problem 2

Given five country flags, four town flags and two army flags all hung together on a thread.

1. What is the chance that three first flags are the country flags?
2. What is the chance that all the flags of the same kind hung together?
3. What is the chance that between two army flags, there will be only the country flags?
4. If three flags selected at random, what is the chance that at least two of them are of the same kind?

1.2.1 Answser 4

The chance of the first three flags being the country flags is the chance of the first being the country flag, times second, times third. The chance of the first is 5 in 11, the chance of second is 4 in 10 and the chance of third is 3 in 9. Thus the total chance is $\frac{5*4*3}{11*10*9} = 6.060\,606\,060\,58 \times 10^{-2}$.

1.2.2 Answer 5

Observe, first, that there are only 3! possibilities for such arrangments, i.e.

1. **country**, **city**, **army**.
2. **country**, **army**, **city**.

3. ...
4. **army**, **city**, **country**.

Given the total number of ways the flags can be hung:

$$\frac{11!}{5!4!2!} = 6930,$$

it gives that there is only $\frac{3!}{6930} = 8.658\,008\,658\,01 \times 10^{-4}$ chance the flags will hang in the specified order.

1.2.3 Answer 6

The number of ways country flags can be hung between the army flags are either one, or two, or three, or four, or five, for each case there is a number of ways the group of flags can be positioned on the rope.

1. There are $11 - 3 + 1 = 9$ ways to position the flags on the rope.
2. There are $11 - 4 + 1 = 8$ ways to position the flags on the rope.
3. There are $11 - 5 + 1 = 7$ ways to position the flags on the rope.
4. There are $11 - 6 + 1 = 6$ ways to position the flags on the rope.
5. There are $11 - 7 + 1 = 5$ ways to position the flags on the rope.

Also observe that for each case, the remaining combinations of flags are defined by the number of flags we can permute times the number of ways we can position the army flags.

1. $\frac{8!}{4!4!} = 70$.
2. $\frac{7!}{3!4!} = 35$.
3. $\frac{6!}{2!4!} = 15$.

4. $\frac{5!}{1!4!} = 5$.
5. $\frac{4!}{4!} = 1$.

Summing this all up gives: $\frac{70*9+35*8+15*7+5*6+1*5}{6930} = 1.515\,151\,515\,15 \times 10^{-1}$.

1.2.4 Answer 7

We can divide this problem into two sub-problems:

1. In how many ways can we slect three first flags s.t. the first ant the second or first and the last will match. We will have thee disjoint probabilities for each kind of flag weighted by their relative probablity:

$$\frac{5}{11} \times \left(\frac{4}{10} + \frac{4}{9}\right) \times 4 + 210 = 3.030\,303\,030\,3 \times 10^{-1}$$
$$\frac{4}{11} \times \left(\frac{3}{10} + \frac{3}{9}\right) \times 5 + 210 = 1.939\,393\,939\,39 \times 10^{-1}$$
$$\frac{2}{11} \times \left(\frac{1}{10} + \frac{1}{9}\right) \times 5 + 410 = 3.636\,363\,636\,36 \times 10^{-2}.$$

2. And the probability that the last two flags are the same, this probability is again weighted by the first flag selected and summed for two remaining kinds of flags:

$$\frac{5}{11} \times \left(\frac{4}{10} \times \frac{3}{9} + \frac{2}{10} \times \frac{1}{9}\right) = 7.070\,707\,070\,67 \times 10^{-2}$$
$$\frac{4}{11} \times \left(\frac{5}{10} \times \frac{4}{9} + \frac{2}{10} \times \frac{1}{9}\right) = 8.888\,888\,888\,86 \times 10^{-2}$$
$$\frac{2}{11} \times \left(\frac{5}{10} \times \frac{4}{9} + \frac{4}{10} \times \frac{3}{9}\right) = 6.464\,646\,464\,63 \times 10^{-2}.$$

Summing it up gives

$$\begin{aligned}
&7.070\,707\,070\,67 \times 10^{-2}+\\
&8.888\,888\,888\,86 \times 10^{-2}+\\
&6.464\,646\,464\,63 \times 10^{-2}+\\
&\ \ 3.030\,303\,030\,3 \times 10^{-1}+\\
&1.939\,393\,939\,39 \times 10^{-1}+\\
&3.636\,363\,636\,36 \times 10^{-2} = 7.575\,757\,575\,75 \times 10^{-1}.
\end{aligned}$$

The code to verify the answers:

```lisp
(defun shift-elements (vec low high)
  (prog1 vec
    (loop :for i :from high :downto low :do
       (setf (aref vec (1+ i)) (aref vec i)))))

(defun initialize-perms (vec element &optional (low 0))
  (prog1 vec
    (loop :with j := low
       :for i :from low :below (length vec)
       :if (eql (aref vec i) element) :do
       (shift-elements vec j (1- i))
       (setf (aref vec j) element j (1+ j)))))

(defun can-move-index (vec element)
  (loop :for i :from (1- (length vec)) :downto 0
     :for current := (aref vec i)
     :with prev := nil
     :when (and prev
                (not (eql prev element))
                (eql current element))
     :do (return i)
     :end :do (setf prev current)))

(defun move-index (vec index)
  (prog1 vec
    (psetf (aref vec (1+ index)) (aref vec index)
           (aref vec index) (aref vec (1+ index)))))

(defun permute-group (vec element &optional (low 0))
  (cons
   (copy-seq (initialize-perms vec element low))
   (loop :with init := (initialize-perms vec element low)
      :with last := low
      :for moving := (can-move-index init element)
      :while moving
      :do (move-index init moving)
      :when (< moving last) :do
      (initialize-perms init element (1+ moving))
      :end
```

```
       :collect (copy-seq init)
       :do (setf last moving))))

(defun canonical (element repeat &optional (previous #()))
  (loop :with result := (make-array (+ repeat (length previous)))
     :for i :below repeat :do
     (setf (aref result i) element)
     :finally
     (return
       (prog1 result
         (loop :for j :from i :below (length result) :do
            (setf (aref result j) (aref previous (- j i))))))))

(defun permutations-with-repetition (groups)
  (loop :with first := (car groups)
     :with perms := (list (canonical (car first) (cdr first)))
     :for (key . value) :in (cdr groups)
     :do (setf perms
               (loop :for perm :in perms
                  :nconc (permute-group
                          (canonical key value perm) key)))
     :finally (return perms)))

(defparameter *all-flags*
  (permutations-with-repetition '((a . 5) (b . 4) (c . 2))))

(defun first-three-a ()
  (/ (loop :for flags :in *all-flags*
        :when (equal (coerce (subseq flags 0 3) 'list) '(a a a))
        :count 1)
     (length *all-flags*)))

(defun togetherp (flags)
  (= 2 (loop :with previous := nil
          :for elt :across flags
          :when (and previous (not (eql elt previous)))
          :count 1 :end
          :do (setf previous elt))))

(defun flags-hang-together ()
  (/ (loop :for flags :in *all-flags*
        :when (togetherp flags)
        :count 1)
     (length *all-flags*)))

(defun between-army-p (flags)
  (not
   (loop :with flags-seen := 0
      :with previous := nil
      :for elt :across flags :do
      (case elt
        (b (when (= flags-seen 1) (return t)))
        (c (when (eql previous 'c) (return t))
           (incf flags-seen)))
      (setf previous elt))))
```

```
(defun between-army ()
  (/ (loop :for flags :in *all-flags*
        :when (between-army-p flags)
        :count 1)
     (length *all-flags*)))

(defun first-three-duplicate ()
  (/ (loop :for flags :in *all-flags*
        :when (or (eql (aref flags 0) (aref flags 1))
                  (eql (aref flags 1) (aref flags 2))
                  (eql (aref flags 0) (aref flags 2)))
        :count 1)
     (length *all-flags*)))

(format t "~&Chance three first flags are country flags: ~f~%~
           Chance all flags hang together:              ~f~%~
           Chance only city flags between army flags:   ~f~%~
           Chance of duplicate in first three flags:    ~f~%"
        (first-three-a)
        (flags-hang-together)
        (between-army)
        (first-three-duplicate))
```

```
Chance three first flags are country flags: 0.060606062
Chance all flags hang together:             0.0008658009
Chance only city flags between army flags:  0.15151516
Chance of duplicate in first three flags:   0.75757575
```