Statistics Exercise 02

Enrico Cotti Cottini
Università di Padova
Matricola 2077993

April 2, 2024

1 Esercizio 15

In un ballottaggio tra due candidati, A e B, votano $N+M$ persone, con $N \in \mathbb{N}$, $M \in \{0, \dots, N\}$: N elettori sono completamente indecisi e votano a caso, senza preferenza tra A e B, mentre il gruppo di M persone sostiene il candidato A. Vogliamo trovare la probabilità che vinca A.

Possiamo descrivere il comportamento elettorale delle N persone indecise tramite una variabile aleatoria S_N con distribuzione binomiale di parametri N e $1/2$, definita su un opportuno spazio di probabilità $(\Omega, \mathcal{F}, P)$; S_N rappresenta il numero di voti che il candidato A riceve dal gruppo delle persone indecise. La probabilità che vinca A è allora data da

$$P\left(S_N + M > \frac{N+M}{2}\right) = P\left(S_N > \frac{N-M}{2}\right) = \sum_{k=\left\lfloor \frac{N-M}{2} \right\rfloor + 1}^{N} \mathrm{pBin}\left(N, \frac{1}{2}\right)(k).$$

Si scriva un programma che calcoli numericamente questa probabilità in funzione di N, M come sopra, in grado di trattare il caso $N + M = 10^6$, $M \leq 5000$.

Per la consegna servono:

- lo svolgimento dell'Esercizio 14 e una giustificazione matematica della procedura utilizzata per il calcolo della probabilità di vittoria elettorale richiesta;
- lo pseudo-codice del programma e il codice commentato in un linguaggio standard come C++ o Python (il codice anche in un file separato);
- un grafico in formato PDF che riporti la probabilità di vittoria elettorale in funzione di M quando $N + M = 10^6$ e M varia da 0 a 5000 (in passi da dieci).

2 Scelta e giustificazioni della procedura

In un ballottaggio tra due candidati, A e B, abbiamo $N + M$ persone votanti in totale, dove N elettori sono completamente indecisi, mentre il gruppo di M persone sostiene il candidato A.

il comportamento elettorale delle N persone indecise è descritto tramite una variabile aleatoria S_N con distribuzione binomiale di parametri N e $1/2$, questo vuol dire che le N persone indecise corrispondono a N tentativi ciascuno dei quali ha una possibilità di $1/2$ di avverarsi, quindi S_N equivale al numero di successi di un'estrazione di N tentativi consecutivi, di cui come gia detto, ognuno ha possibiliyà $1/2$ di avverarsi.

Dunque la probabilità $P(A)$ ovvero la probabilità che A vinca le elezioni equivale alla probabilita che $S_N + M$ (i votanti di A) sia Maggiore alla metà del numero totale dei votanti $\frac{N+M}{2}$, ossia:

$$P\left(S_N + M > \frac{N+M}{2}\right)$$

che equivale a

$$P\left(S_N > \frac{N-M}{2}\right)$$

Noi sappiamo che La funzione di massa di probabilità(funzione di massa perchè S_N è una variabile aleatoria discreta dato che N è un valore finito, Dimostrata alla fine del foglio della lezione del 25 marzo 2024) per S_N di paramtri $(N, 1/2)$ è data da:

$$P(S_N = k) = \binom{N}{k} \frac{1}{2}^k (1 - \frac{1}{2})^{N-k}$$

dove

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}$$

La probabilità $P(S_N = x_i)$ può essere interpretata come la probabilità che S_N assuma un valore specifico x_i tra i possibili valori $x_1, x_2, \ldots, x_n$.

Dire che $P\left(S_N > \frac{N-M}{2}\right)$ che equivale a sommare tutte le probabilità parziali $P(S_N = k_i)$ dove k deve prendere i valori $k_i \in \mathbb{N}$ maggiori di $\frac{N-M}{2}$ fino al massimo che puo assumere S_N ovvero N, quindi:

$$\frac{N-M}{2} < k_1 = \frac{N-M}{2} + 1 < ... < k_n = N$$

Quindi per ogni K_i la funzione di massa diventa:

$$\sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} \text{pBin}\left(N, \frac{1}{2}\right)(k) = \sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} P(S_N = k)$$

$$= \sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} \binom{N}{k} \frac{1}{2}^k (1 - \frac{1}{2})^{N-k} = \sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} \frac{N!}{k!(N-k)!} \frac{1}{2}^k (1 - \frac{1}{2})^{N-k}$$

Dunque in definitiva la probabilita che A vinca $P(A)$ equivale a:

$$P\left(S_N + M > \frac{N+M}{2}\right) = P\left(S_N > \frac{N-M}{2}\right) = \sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} \text{pBin}\left(N, \frac{1}{2}\right)(k).$$

$$= \sum_{k=\lfloor \frac{N-M}{2} \rfloor + 1}^{N} \frac{N!}{k!(N-k)!} \frac{1}{2}^{k} (1 - \frac{1}{2})^{N-k}$$

2.1 Considerazioni sulla complessita'

Il problema principale di questa formula e' il calcolo del coeficiente binomiale

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}$$

questa operazione per N e k grandi diventa infattibile data la complessita' fattoriale O(n!) di questo calcolo sia in termini di tempo che di costo, quindi e' inpensabile l'utilizzo di questa formulazione per $N + M$ nell'ordine di 10^6

2.2 Soluzione possibile

Cercando una soluzione a questo problema ho deciso di documentarmi sull' implementazione della funzione pmf (Probability Mass Function) e pdf(Probability density Function) della libreria Scipy di python, progetto open source distibuito con licenza BSD.

https://github.com/scipy/scipy

La funzione PMF e' definita in stats/discrete distns.py come segue:

Algorithm 1

```
1: function _PMF(x, n, p)
2:     return _boost._binom_pdf(x, n, p)
3: end function
```

Come possiamo vedere la libreria Boost viene utilizata ausiliariamente a Scipy e la vera implementazione di pdf(density) e' di Boost(libreria c++ opensource).

in Boost la funzione pmf e':

Algorithm 2

function PDF(dist, k)
  ...
  controlli
  ...
  using *boost::math::ibeta_ derivative*
  return ibeta_derivative($k+1, n-k+1$, dist.success_fraction())/$(n+1)$
end function

```
487
488         // Probability of getting exactly k successes
489         // if C(n, k) is the binomial coefficient then:
490         //
491         // f(k; n,p) = C(n, k) * p^k * (1-p)^(n-k)
492         //           = (n!/(k!(n-k)!)) * p^k * (1-p)^(n-k)
493         //           = (tgamma(n+1) / (tgamma(k+1)*tgamma(n-k+1))) * p^k * (1-p)^(n-k)
494         //           = p^k (1-p)^(n-k) / (beta(k+1, n-k+1) * (n+1))
495         //           = ibeta_derivative(k+1, n-k+1, p) / (n+1)
496         //
497         using boost::math::ibeta_derivative; // a, b, x
498         return ibeta_derivative(k+1, n-k+1, dist.success_fraction(), Policy()) / (n+1);
499
500       } // pdf
```

Figure 1: commento sul codice.

Nelle righe commentate e' presente la formulazione matematica per esprimere la funzione pmf senza il coeficente binomiale e con la derivata della funzione beta incompleta, sarebbe:

$$
\begin{aligned}
f(k;n,p) &= C(n,k)\cdot p^k\cdot(1-p)^{n-k} \\
&= \frac{n!}{k!(n-k)!}\cdot p^k\cdot(1-p)^{n-k} \\
&= \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}\cdot p^k\cdot(1-p)^{n-k} \\
&= p^k(1-p)^{n-k}/\left(\beta(k+1,n-k+1)\cdot(n+1)\right) \\
&= \text{ibeta\_derivative}(k+1,n-k+1,p)/(n+1)
\end{aligned}
$$

Dove sono presenti:

1. Il coefficiente binomiale $C(n,k)$.
2. Il fattoriale $n!$,
3. La funzione gamma $\Gamma(x)$, che generalizza il concetto di fattoriale ai numeri reali e complessi.

4. $\beta(x, y)$ rappresenta la funzione beta, che è la funzione integrale della distribuzione beta.

5. La funzione ibeta_derivative(a, b, x) rappresenta la derivata parziale della funzione beta incompleta.

Quindi questa formulazione utilizza la derivata parziale della funzione beta incompleta e la funzione beta incompleta.

2.3 Implementazione

L'implementazione delle funzioni beta incomplete nelle librerie Boost C++ si basa sull'Algoritmo 708, intitolato "Significant digit computation of the incomplete beta function ratios" di DiDonato e Morris. Queste funzioni calcolano la funzione beta incompleta.

spiegazione semplificata di questa implementazione:

1. **Implementazione Comune**: Tutte e quattro le funzioni condividono un'implementazione comune, che riceve sia x che y. A seconda della situazione, può essere restituito p o q, dove p e q sono correlati.

2. **Frazione Continua**: Sono utilizzate diverse rappresentazioni di frazioni continue in base ai valori di a e b:

   - Una frazione continua che si trova sui libri di testo è disponibile ma non utilizzata nell'implementazione a causa di problemi di velocità e accuratezza.
   - Quando sia a che b sono maggiori di 1, viene utilizzata una frazione continua di DiDonato e Morris.
   - Per valori piccoli di b e x, viene utilizzata una rappresentazione in serie.
   - Quando b è molto più piccolo di a, viene utilizzata una rappresentazione in serie con una funzione gamma incompleta.

L'implementazione seleziona il metodo appropriato in base ai valori di a, b e x.

Per documentazione piu approfondita :

1. Boost C++ Libraries:

   - Beta Derivative
   - Incomplete Beta Function

2. Wikipedia:

   - Beta Function
   - Incomplete Beta Function

3. Wolfram MathWorld:
    - Incomplete Beta Function
4. SciPy Documentation:
    - scipy.special.betainc
5. DLMF (Digital Library of Mathematical Functions):
    - Incomplete Beta Function

3 pseudo-codice

Nel file PMF.py ho implementato 2 funzioni pmf che associano a ciascun valore di una variabile aleatoria discreta la probabilità che la variabile aleatoria assuma quel valore. la prima pmf calcola la probabilita' utilizzando la formula sopracitata che utilizza la derivata della funzione beta incompleta betainc() la quale e' a sua volta implementata dalla libreria scipy. La seconda scipy_pmf utilizza direttamente binom.pmf la funzione pmf implementata dalla libreria scipy.
Ho fatto queste due distinzioni perche' la funzione pmf e' molto piu rapida della seconda probabilmente per qualche differenza di implementazione interna della libreria scipy.
L'esecuzione del programma con la prima funzione impiega circa 12 secondi sulla mia macchina mentre la seconda impiega circa 300 secondi.

Algorithm 3 Calcolo della funzione di massa di probabilità (PMF) tramite derivata della funzione beta incompleta

```
    function IBETA_DERIVATIVE(a, b, x, h = 1e − 5)
        fx ← betainc(a, b, x)
 3:     fx_plus_h ← betainc(a, b, x + h)
        derivative ← (fx_plus_h − fx) / h
        return derivative
 6: end function
    function PMF(n, k, p)
        ibeta_deriv ← ibeta_derivative(k + 1, n − k + 1, p)
 9:     pmf ← ibeta_deriv / (n+1)
        return pmf
    end function
12: function SCIPY_PMF(n, k, p)
        pmf_value ← binom.pmf(k, n, p)
        return pmf_value
15: end function
```

Nel file main.py e' presente il codice per la sommatoria delle probabilita' che compongono la probabilita della vittoria di A, in piu' questa probabilita' viene

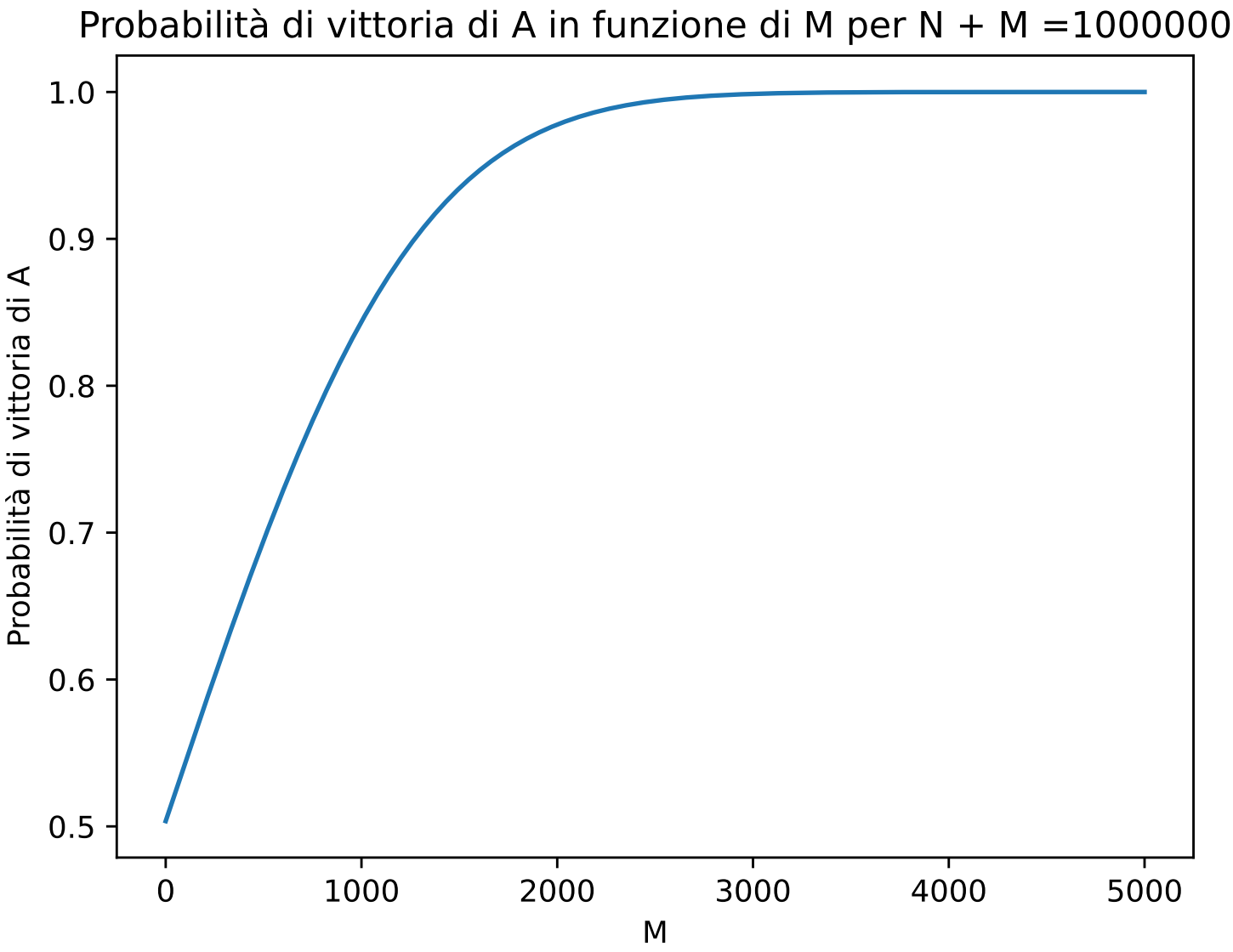
calcolata n volte per diversi valori di M (compresi tra 0 e 5000) dove n = step. E poi il codice per plottare il risultato.

Algorithm 4 Calcolo di A_win

function DADA(asdad)
    $N + M = 1000000$
    $M\_values = \text{np.linspace}(0, M_{\max}, \text{step}, \text{dtype} = \text{int})$
    $N\_values = N + M - M\_values$
    $p = 0.5$
    for (M, N) in zip$(M_values, N_values - M_values)$ **do**
        $k\_values = \text{np.arange}\left(\left\lfloor \frac{N-M}{2} \right\rfloor + 1, N\right)$
        $p\_A = \sum_{k=\left\lfloor \frac{N-M}{2} \right\rfloor + 1}^{N} pmf(N, k, p)$
    end for
end function

4 Grafico N+M in funzione di M crescente

grafico in formato PDF che riporti la probabilità di vittoria elettorale in funzione di M quando N + M = 1000000 e M varia da 0 a 5000 Possiamo osservare che con l'aumentare dei Votanti sicuri di M allora la probabilita' della vittoria di A aumenta fino a stabilizzarsi su un valore vicino al 100% pagina successiva


Probabilità di vittoria di A in funzione di M per N + M =1000000
Probabilità di vittoria di A
1.0
0.9
0.8
0.7
0.6
0.5
0
1000
2000
3000
4000
5000
M