Homework 1 - Data Mining - Sapienza

Ivan Fardin 1747864

November 1st, 2020

Contents

1 Problem 1

1.1

sample space $\Omega = \{$

2 of Hearts, 3 of Hearts, 4 of Hearts, 5 of Hearts, 6 of Hearts, 7 of Hearts, 8 of Hearts, 9 of Hearts, 10 of Hearts, J of Hearts, Q of Hearts, K of Hearts, A of Hearts,
2 of Diamonds, ..., A of Diamonds,
2 of Spades, ..., A of Spades,
2 of Clubs, ..., A of Clubs $\}$

$|\Omega| = 13 \cdot 4 = 52$ (13 cards for each suit)

$$\Pr(\text{2 of Hearts}) = ... = \Pr(\text{A of Clubs}) = \frac{1}{|\Omega|} = \frac{1}{52}$$

$$\Pr(\text{"Pick a 2 in the deck"}) = ... = \Pr(\text{"Pick an ace in the deck"}) = \frac{4}{52} = \frac{1}{13}$$

$$\Pr(\text{"Pick a Hearts in the deck"}) = ... = \Pr(\text{"Pick a Clubs in the deck"}) = \frac{13}{52} = \frac{1}{4}$$

1.2

1.2.1

$$\Pr(\text{"Pick at least an ace in the first three cards"}) = 1 - \frac{\binom{4}{0}\binom{48}{3}}{\binom{52}{3}} = 1 - \frac{48!\ 49!}{45!\ 52!} \approx 1 - 0.78 = 0.22$$

We have $\binom{48}{3}$ combinations to pick 3 cards all of different rank from the ace out of $\binom{52}{3}$ possible hands. Since we are looking for at least an ace, we can compute the complement of this probability.

1.2.2

$$\Pr(\text{"Pick exactly an ace in the first five cards"}) = \frac{\binom{4}{1}\binom{48}{4}}{\binom{52}{5}} = \frac{48!\ 5!\ 47!}{3!\ 44!\ 52!} \approx 0.3$$

We have $\binom{4}{1}$ combinations to pick an ace and $\binom{48}{4}$ to pick four other cards of a different rank out of $\binom{52}{5}$ possible hands.

1.2.3

$$\Pr(\text{"Pick the first three cards of the same rank"}) = \frac{13\binom{4}{3}}{\binom{52}{3}} = \frac{13 \cdot 4!\ 49!}{52!} \approx 0.0024$$

We have 13 ways to make three of a kind ($\binom{4}{3}$ combinations) out of $\binom{52}{3}$ possible hands.

1.2.4

$$\Pr(\text{"Pick all Diamonds in the first five cards"}) = \frac{\binom{13}{5}}{\binom{52}{5}} = \frac{13!\ 47!}{52!\ 8!} \approx 0.0005$$

We have $\binom{13}{5}$ combinations to pick 5 out of 13 Diamonds out of $\binom{52}{5}$ possible hands.

1.2.5

$$\Pr(\text{"Pick a full house in the first five cards"}) = \frac{13\ \binom{4}{3}\ 12\ \binom{4}{2}}{\binom{52}{5}} = \frac{13 \cdot 12 \cdot 4!\ 5!\ 47!}{52!} \approx 0.0014$$

We have 13 ways to make three of a kind ($\binom{4}{3}$ combinations) and 12 ways to constitute a pair among the remaining ranks ($\binom{4}{2}$ combinations) out of $\binom{52}{5}$ possible hands.

1.3

I developed a simple Python program called *cards.py* to perform simulations of picking the first five cards of a deck in order to check my answers.

It makes use of two arrays: one for the card numbers and the other for the four suits.
So, it simply picks one card at a time from the deck by randomly choosing two items from the two arrays and if the card is not already drawn prints it, otherwise picks a new one until a non-drawn card is printed.

2 Problem 2

sample space Ω of the sum of $n = 3$ regular dice = { n, ..., 6n } = { 3, ..., 18 }

sample space Ω' of throwing $n = 3$ regular dice = { $(a, b, c) : 1 \leq a, b, c \leq 6$ } = { (1,1,1), ..., (6,6,6) }

$|\Omega| = 6n - n = 5n = 15$

$|\Omega'| = 6^n = 6^3$

To calculate the probability of seeing a sum of 11 or a sum of 16, we need to partition the two numbers into the sum of three integers in the range from 1 to 6 (faces of a die).

$11 = 6 + 4 + 1 = 6 + 3 + 2 = 5 + 5 + 1 = 5 + 4 + 2 = 3 + 5 + 3 = 3 + 4 + 4$
$16 = 6 + 6 + 4 = 6 + 5 + 5$

The number of permutations for each partition is $\dfrac{n!}{n_1!\ n_2!\ n_3!}$ where:

- n is the number of dice,
- n_1 is the number of repetition of the same value in the sum different from n_2 and n_3,
- n_2 the number of repetition of the same value in the sum different from n_1 and n_3,
- n_3 is the number of repetition of the third value in the sum different from n_1 and n_2

The number of permutations of the partitions of 11 is $3! + 3! + \dfrac{3!}{2!} + 3! + \dfrac{3!}{2!} + \dfrac{3!}{2!} = 3\ (3! + 3) = 27$

The number of permutations of the partitions of 16 is $\dfrac{3!}{2!} + \dfrac{3!}{2!} = 2 \times 3 = 6$

$\Pr(\text{"Sum of 11"}) = \dfrac{27}{6^3} = 0.125$

$\Pr(\text{"Sum of 16"}) = \dfrac{6}{6^3} = \dfrac{1}{6^2} = 0.02\overline{7}$

$\Pr(\text{"Sum of 11"} \cup \text{"Sum of 16"}) = \Pr(\text{"Sum of 11"}) + \Pr(\text{"Sum of 16"}) = 0.125 + 0.02\overline{7} = 0.152\overline{7}$

3 Problem 3

sample space Ω = { TP, FP, TN, FN } = { $(P_t, P), (P_t, N), (N_t, N), (N_t, P)$:
each pair represents (result outcome, person infected) }

$|\Omega| = 4$

$$\text{Sensitivity of the rapid test} = \frac{TP}{TP + FN} = 84.7\%$$

$$\text{Specificity of the rapid test} = \frac{TN}{TN + FP} = 85.7\%$$

Let's assume 1% of the population currently has COVID-19 $\Rightarrow$ Pr("person is infected") = Pr(P) = 1%

Result of the rapid test = positive $\Rightarrow$ probability that this person is infected with COVID-19?

The result can be either a TP or an FP $\Rightarrow$ Pr(P_t) = Pr(TP) + Pr(FP)

Pr("positive test given person is infected") = Pr($P_t|P$) = Sensitivity
Pr("negative test given person is not infected") = Pr($N_t|N$) = Specificity

$\Rightarrow$

$$\text{Pr(TP)} = \text{Pr}((P_t, P)) = \text{Pr}(P_t|P)\ \text{Pr(P)} \iff \text{Pr}(P_t|P) = \frac{Pr(P_t \cap P)}{Pr(P)} = \frac{Pr(P|P_t)Pr(P_t)}{Pr(P)} = 84.7\%$$

$$\text{Pr(TN)} = \text{Pr}((N_t, N)) = \text{Pr}(N_t|N)\ \text{Pr(N)} \iff \text{Pr}(N_t|N) = 85.7\%$$

$$\text{Pr(FP)} = \text{Pr}((P_t, N)) = \text{Pr}(P_t|N)\ \text{Pr(N)} \iff \text{Pr}(P_t|N) = 1 \text{ - Specificity} = 14.3\%$$
$$\text{Pr(FN)} = \text{Pr}((N_t, P)) = \text{Pr}(N_t|P)\ \text{Pr(P)} \iff \text{Pr}(N_t|P) = 1 \text{ - Sensitivity} = 15.3\%$$

$$\Rightarrow \text{Pr}(P|P_t) = \frac{Pr(P_t|P)Pr(P)}{Pr(P_t)} = \frac{Pr(P_t|P)Pr(P)}{Pr(P_t|P)Pr(P) + Pr(P_t|N)Pr(N)} = \frac{0.847 \cdot 0.01}{0.847 \cdot 0.01 + 0.143 \cdot 0.99} =$$
$$= 0.0565 = 5.65\%$$

The probability that a person is infected with COVID-19 given test is positive can be calculated using Bayes theorem.
The fact that the probability of positive tests is the sum between the probability of being either a TP or an FP is an application of the Law of Total Probability.

The probability that a person is infected with COVID-19 given test is positive is strongly affected by the assumption that only 1% of the population currently has COVID-19. If we assume that not 1% but 10% of the population currently has COVID-19, the probability rises to $\approx$ 40%, and if we assume 15% it becomes 51%.
So, I think that such a test can be useful along with other more accurate tests.

4 Problem 4

4.1

sample space Ω of the $G_{n,p}$ model $= \{\ (v_i, v_j) : 1 \le i < j \le n$ and $(v_i, v_j) \in M$ with probability $p\ \}$ where n is the number of nodes, M is the set of edges and $(v_i, v_j) = (v_j, v_i)$ so I count it once.

The maximum number of edges is equal to the number of edges in a complete graph $\binom{n}{2} = \dfrac{n\ (n-1)}{2}$

$$|\Omega| = m = \binom{\frac{n\ (n-1)}{2}}{m}\ p^m\ (1-p)^{\frac{n\ (n-1)}{2} - m} \qquad \text{(number of edges)}$$

4.2

$\Pr((v_i, v_j) \in M) = p$

4.3

Suppose n even, find the probability that the graph contains exactly two cycles of size $\frac{n}{2}$ and no other edges. The number of edges must be n since each node must be connected to two: the previous and next.

$$\Pr(\text{"graph has exactly } n \text{ edges"}) = \binom{\frac{n\ (n-1)}{2}}{n}\ p^n\ (1-p)^{\frac{n\ (n-1)}{2}\ -n} = \frac{1}{n\ (n-3)!}\ p^n\ (1-p)^{\frac{n\ (n-3)}{2}}$$

Furthermore, there are no nodes or edges in common between the two cycles otherwise the two cycles would not be the same size or a third cycle would exist. A cycle is a sequence of vertices in which each node appears once
$v_1, ..., v_{l_1}, v_1$ where $l_1 = \frac{n}{2}$, with $v_i \ne v_j$ for $i, j \in \{1, ..., l_1\} = M_1$
$v'_1, ..., v'_{l_1}, v'_1$ where $l_1 = \frac{n}{2}$, with $v'_i \ne v'_j$ for $i, j \in \{1, ..., l_2\} = M_2$
$l_1 + l_1 = n$ and $(v_i, v_{i+1}) \ne (v'_i, v'_{i+1})$ for all edges $\in M_1, M_2$

There exist $\binom{n}{\frac{n}{2}}$ combinations of $\frac{n}{2}$ items out of n. Each combination denotes a partition of the graph into two sets, we need exactly n edges to connect the n nodes and create two components

$$\Pr(\text{"graph contains exactly two cycles of size } \tfrac{n}{2} \text{ and no more edges"}) = \binom{n}{\frac{n}{2}} \binom{\frac{n\ (n-1)}{2}}{n}\ p^n\ (1-p)^{\frac{n\ (n-1)}{2} - n}$$

4.4

Unlike **4.3** where we only take partitions of size $\frac{n}{2}$, here we need to take also partitions that make the sum equal to n with a minimum number of items of a partition equal to three (a cycle has at least 3 nodes).

$\Pr(\text{"graph contains exactly two cycles of any size with no common nodes and no more edges"}) =$

$$= \sum_{i=3}^{\frac{n}{2}} \binom{n}{i} \binom{n}{n-i} \binom{\frac{n\ (n-1)}{2}}{n}\ p^n\ (1-p)^{\frac{n\ (n-1)}{2} - n}$$

4.5

The probability a vertex has degree k is a Bernoulli Random Variable

$$P(X = deg(v) = k) = \binom{n-1}{k}\ p^k\ (1-p)^{n-1-k}$$

We have $\binom{n-1}{k}$ ways of choosing k edges out of the $n-1$ possible edges, p^k is the probability that the k edges are present and p^{n-k-1} is the probability that the remaining $n-k-1$ edges are not.

So, the expected degree of the vertex i is the expected value of a Bernoulli Random Variable

$$X_{ij} = \begin{cases} 1 & if\ the\ edge\ (v_i, v_j) \in M \\ 0 & otherwise \end{cases} \tag{1}$$

$$E[X_{ij}] = 0 \cdot Pr(X_{ij} = 0) + 1 \cdot Pr(X_{ij} = 1) = p$$

$$E[X] = \sum_{k=0}^{n-1} k \cdot Pr(X = k) = E[\sum_{j=1}^{n-1} X_{ij}] = \sum_{j=1}^{n-1} E[X_{ij}] = p\ (n-1)$$

4.6

From **4.1**, **4.2** and **4.5** the expected number of edges is the expected value of Bernoulli Random Variable

$$E[Y] = \sum_{m=0}^{\frac{n\ (n-1)}{2}} m \cdot Pr(Y = m) = E[\sum_{m=0}^{\frac{n\ (n-1)}{2}} X_{ij}] = \sum_{m=0}^{\frac{n\ (n-1)}{2}} E[X_{ij}] = p\ \frac{n\ (n-1)}{2}$$

4.7

5 Problem 5

To find the 10 beers with the highest number of reviews from the *beers.txt* file, I wrote a simple bash script called *beers.sh* where I chained the following Unix commands through pipes, so that the output text of each process (stdout) is passed directly as input (stdin) to the next one.

```
cut -f 1 beers.txt | sort | uniq -c | sort -k 1 -t$'\t' -nr | head
```

I assumed that each line in the *beers.txt* file contains a review consisting of the beer name and the associated score.

From the *man* command:

- `cut -f 1 beers.txt`

  Selects only the first field (i.e. beer name, fields separated by tab) of the *beers.txt* file

- `sort`

  Sorts the beer names alphabetically

- `uniq -c`

  Filters adjacent matching lines from standard input, writing to standard output (sort the input first) for each beer name a prefix denoting the number of occurrences in the input

- `sort -k 1 -t$'\t' -nr b.txt`

  Sorts the standard input in reverse order according to string numerical value of the first column (separated by the tab character from the second one)

- `head`

  Prints the first 10 lines of the standard input to standard output

6 Problem 6

To find the top-10 beers with the highest average overall score among the beers that have had at least 100 reviews, I wrote a simple Python program called *beers.py*.

Since some beers appear in multiple lines of the *beers.txt* file, preprocessing is needed in order to sum the score of reviews referring to the same beer. So, I thought about using a hash table (in Python, the dictionary is implemented as a hash table) where each key is represented by a beer name and the value by a pair consisting of the overall score and the number of reviews associated with the beer.

After that, as in **Problem 5**, I read each line of the *beers.txt* file, split it according to the tab character then I added the score of the review to the overall score so far and I updated the number of reviews.

In the end, I sorted the dictionary by the average overall score of each entry within the dictionary from largest to smallest via the sorted function and printed the top 10 beers that have had at least 100 reviews.

7 Problem 7

7.1

As suggested, I used *Twython* to get the stream of the tweets in Rome as they are generated.

First, I wrote the *twitter-config.json* file where I stored sensitive information.
Then, in *Auth.py* I wrote a class to retrieve information from the above file.

After this initial step, in *RomeStreamTweets.py* I start two threads: a listener and a writer and wait for *Ctrl+C* command to properly close the program.

The listener code is implemented in *TwitterListener.py* where a Streamer object is created and a filtered query using a bounding box for Rome location is performed.

The streamer (implemented in *Streamer.py*) is a *TwythonStreamer* that receives tweets from the *Twitter Streaming API* and extract from them information of interest for the application like the username, the text and the exact geographic location (if any otherwise the place). When all the information is present, a minimalist version of the tweet is added to a list.

The writer code is implemented in *TwitterWriter.py* where the same list is continuously written into a JSON file called *tweets.json.*

7.2

To plot the tweet locations on Google Maps, I wrote a simple HTML page *RomeStreamTweetsMap.html* where based on the points saved in the *tweets.json* file, it puts markers on the map of Rome where the tweets are located.

To avoid explicitly inserting the Google Maps API key into the code via the following script

*<script src="https://maps.googleapis.com/maps/api/js?key=API_KEY&
callback=initMap&libraries=&v=weekly" defer></script>*

I wrote a *gmaps-config.json* file where I stored the key and read it from there by dynamically loading the map.

As workaround for the cross-origin policy issue, I used Python to start a web server on the local machine via *python3 -m http.server* for loading the json file containing the tweets and then I entered the following URL in a browser `http://localhost:8000/RomeStreamTweetsMap.html`.

So, to start capturing tweets and plotting them on the map, you need to launch the Python program via *python3 -m RomeStreamTweets.py* and access the above URL.