Heidelberg University
Winter semester 2024/25

Group Artificial Intelligence for Programming (AIP)
Prof. Dr. Artur Andrzejak, Kim Tuyen Le

Problem Set 4 for lecture Mining Massive Datasets

Due November 18, 2024, 23:59 CET

Exercise 1 (2 points)

Consider a cluster of n machines. Each has a probability p of failing in a given period of time T.

a) What is the probability of at least one machine failure during this period of time?

Hint: Consider the complementary event that no machine fails during that time.

b) For $0 \leq k \leq n$, what is the probability p_k (in terms of k, n, and p), of exactly k machines failing during T? Give an explanation.

c) Show that the probabilities from Part b) satisfy: $p_1 + p_2 + \cdots + p_n = 1 - (1-p)^n$.

Exercise 2 (1 point)

Give two examples for hash functions as alternatives to the arithmetic remainder function `mod`. See this video[1] which was created in the weeks after Adobe's password breach in October 2013 and explain the connection of hash functions to password security. What is a rainbow table? What is salt in this context?

Exercise 3 (2 points)

Explain the details of the data structure `HashMap` from Java standard library by answering the following questions (to this aim, analyze the source code and/or documentation of the implementation).

- Describe briefly the internal data structure used by `HashMap` for storing of map entries.
- How is a new <key, value> pair added to a map? Consider the possibility that a collision of hash values may occur.
- How is an entry retrieved given a search key? Consider the possibility that a bucket can have more than one entry.

Exercise 4 (1 point)

Suppose you have to design/implement hash functions for a Bloom filter algorithm. Propose 3 hash functions which are as independent from each other as possible and briefly justify your choice.

[1] Computerphile: How NOT to store passwords! https://www.youtube.com/watch?v=8ZtInClXe1Q

Exercise 5 **(2 points)**

Consider a Bloom filter with a bit array of $n = 5$ bits.

a) Compute the probability that a random element gets hashed by a hash function to a given bit in the bit array by using the probability formula from Lecture 5 (slide *Analysis: Throwing Darts (2)*). Explain your answer.

b) Now we want to use the following hash functions ($k = 2$):
$h_1(x) = x \mod 5$,
$h_2(x) = 2 \cdot x + 3 \mod 5$.
Is every bit equally likely to be hit by one of the two hash functions? Suppose the numbers 4 and 1 are in the set that we want to find with the filter. Show the state of the bit array after you apply the hash functions to these numbers.

c) Consider again the hash functions and the bit array state as in Part b). How likely is it for a number in the stream to be a false positive? Use the false positive probability formula from Lecture 5 (slide *Bloom Filter -- Analysis*). Explain your answer.

Exercise 6 **(2 points)**

Consider the Bloom filtering technique and the set of keys S with m members. Assume that you can control the size n of the bit array (bitset) used in the filter ($n \geq m$) in steps $m/8$.

a) Plot the optimal values of the number of hash functions k (as integers) for $n = m, 2m, \ldots, 20m$. Describe your observations, if any. (*0.5 points*)

b) Assume that you want to minimize the size n (which drives up your memory cost) for a given upper bound b on the expected false positive probability. Determine the optimal values for n (and so k) under this objective for $b_1 = 0.01$, $b_2 = 0.005$, and $b_3 = 0.001$. Also state the (expected) false positive probability for each of these parameter combinations. Explain your approach and explain why you do not need to know the actual value of m.

Hint: you might need to write a short program to perform the concrete calculation. (*1.5 points*)

Exercise 7 **(2 points)**

Consider again the Bloom filtering technique. Prove that the optimal number k_{opt} of hash functions for given size n of the bitset and a given number m of keys is $k_{opt} = n/m \ln(2)$ (as claimed in the Lecture 5, slide *Bloom Filter – Analysis (2)*).

Exercise 8 **(2 points)**

Suppose we have a stream of tuples with the schema `Grades(university, courseID, studentID, grade)`. Assume universities are unique, but a courseID is unique only within a university (i.e., different universities may have different courses with the same ID, e.g., “IPI”) and likewise, studentIDs are unique only within a university (different universities may assign the same ID to different students). Suppose we want to answer certain queries approximately from a 1/20th sample of the data. For each of the queries below, indicate how you would construct the sample. That is, tell what the key attributes should be.

a) For each university, estimate the average number of students in a course. (*1 point*)

b) Estimate the fraction of students who have an average grade of 2.0 or better. (*0.5 points*)

c) Estimate the fraction of courses where at least half of the students got the grade 1.7 or better. (*0.5 points*)

Exercise 9 (2 points)

Analyze the WordCount example for Spark Structured Streaming from the lecture, also described in the Structured Streaming Programming Guide as "Quick Example"[2]. Run (under Linux) the sample code as described in the guide:

```
spark-submit structured_network_wordcount.py localhost 9999
```

Then, in another terminal window, run the following commands:

```
mkfifo fifo
cat > fifo &
nc -lk 9999 < fifo &
```

Now the *netcat* process is running in the background, listening on the named pipe `fifo`. All that is sent through this named pipe will be forwarded by *netcat* to port 9999, where in turn the Spark Structured Streaming program is listening. When you forward the standard output of any process with the help of the `>` operator to `fifo`, this output will be streamed to the `structured_network_wordcount.py` program.

a) Run the command

```
echo "A small step for a man, a giant leap for mankind." \
        > fifo
```

and describe what happens. Does the output of the streaming program behave as expected? (*0.25 points*)

b) Run the command

```
yes "k thx bai bai" > fifo
```

What do you notice? How large is roughly the throughput when you assume one byte per character? (*0.25 points*)

c) Modify the sample code to compute: (i) the word lengths and (ii) the average length of words received from the network in real-time. Display both the count of each word length and the average word length in the console, updated continuously as new data arrives. Name your code as `structured_network_wordlength.py`, test and submit it with your solution. (*1.25 points*)

Hint: You might use `length()` and `agg()` functions.

d) Run your `structured_network_wordlength.py` as described above (i.e., using `fifo`) on "*The Complete Works of William Shakespeare*"[3]. Report your results. (*0.25 points*)

[2] https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#quick-example

[3] Available here https://www.gutenberg.org/files/100/100-0.txt and in heiBOX under *MMD-Exercises\data* (consider only the content between "*** START OF THE PROJECT ... ***" and "*** END OF THE PROJECT ... ***")