COMPSCI 514: Problem Set 1

Released: 9/13.

Due: 9/26 by 10:00am (by class time) in Gradescope.

Instructions:

- Each homework group should work together to produce a single solution set. One member should submit a solution pdf to Gradescope, marking the other members as part of their group.
- You may talk to members of other groups at a high level about the problems but not work through the solutions in detail together.
- Before we can grade the problem set in Gradescope, all members of your group must answer the Gradescope consent poll in Piazza. Or if you don't consent, contact me to make alternative grading arrangements.
- You must show your work/derive any answers as part of the solutions to receive full credit.

1. Messing Around with Markov's Inequality (10 points)

1. (2 points) Recall that Markov's inequality only applies to *non-negative* random variables. Consider a random variable $\mathbf{X}$, that always takes values greater than some floor a, which may be negative or positive. Give as tight an upper bound as you can on $\Pr(\mathbf{X} \geq t)$ (in terms of $\mathbb{E}[\mathbf{X}]$, a, and t).
2. (2 points) The maximum score on a midterm exam is 100% and the class average is 75%. What's the maximum fraction of students who could have scored below 25%.
3. (6 points) Let $\mathbf{X}$ be distributed uniformly on $[0, 1]$.
   (a) What is $\Pr(\mathbf{X} \geq 3/4)$?
   (b) Give an upper bound on $\Pr(\mathbf{X} \geq 3/4)$ using Markov's inequality.
   (c) Apply Markov's inequality to $\mathbf{X}^2$ to give a tighter upper bound.
   (d) What happens when you try to give a bound using higher moments? I.e., apply Markov's to $\mathbf{X}^r$ for a few values of $r > 2$ and describe what you observe.
   (e) **Bonus:** (3 points) Exhibit a monotonic function f so that applying Markov's to $f(\mathbf{X})$ gives as tight a bound on $\Pr(\mathbf{X} \geq 3/4)$ as you can.

2. Bloom Filters Meet Machine Learning (10 points)

Consider applying Bloom filters to database optimization: you want to store a large table of $(user, movie)$ pairs, with information about when the user watched the given movie, how they rated it, if they finished watching, etc. Most pairs in this table are empty since most users have not watched most movies. So, whenever a user watches a movie, you insert information on that pair into the database and also add the pair to a bloom filter. Before you make a possibly expensive query to the the table, you check the bloom filter and only query the table if it returns a hit.

1. (3 points) You have $n = 1$ billion $(user, movie)$ pairs with information stored in your database. You would like to store these pairs in a bloom filter with false positive rate of 5%. How large must your filter be to achieve this rate? I.e., how large must you set the bit array size m and how much storage in kBs, MBs, or GBs does it take? You may use the false positive rate calculation given in class of $\left(1 - e^{\frac{-kn}{m}}\right)^k$ – even though its classic derivation is incorrect, it is a good approximation. Throughout you may assume that you use the optimal number of hash functions $k = \ln 2 \cdot \frac{m}{n}$ (assume this quantity is an integer).

2. (4 points) The machine learning team at your company has developed a classifier that is able to predict if a user has watched a move with pretty good probability, without having to look at the user-movie database. The classifier is a function $y : (user, movie) \to \{0, 1\}$ which returns 1 on 90% of $(user, movie)$ pairs where the user has actually watched a movie and returns 0 on 97% of pairs where the user hasn't watched the movie.

   You decide to use this classifier to reduce the load on your bloom filter. You only add a $(user, movie)$ pair (u, m) to the filter if the user has watched the movie *and* $y(u, m) = 0$. When any pair (u, m) is queried you first check if $y(u, m) = 1$ – if so you query the pair in the your database. If not, you check your bloom filter, and if it returns a positive you then query the database.

   What is the overall false positive rate on a random pair (u, m) where u has not watched m? Give an expression in terms of the number of pairs in the database n, the filter size m, and the number of hash functions k.

3. (3 points) How large must you set m to still have a false positive rate of 5%. How much storage have you saved using this ML approach compared to the standard approach in (1).

3. Network Size Estimation via Colliding Crawls (10 points)

You want to estimate the number of nodes n in a large network (the Facebook social network, the web, etc.), by randomly crawling the network. Assume that after enough steps of randomly hopping from node to node in the network, the final node you land at is sampled uniformly at random from all nodes in the network.

1. (3 points) Consider sending out t independent random crawls that end at t random nodes. Let $\mathbf{C}$ denote the number of pairwise collisions between these nodes. Prove that $\mathbb{E}[\mathbf{C}] = \frac{\binom{t}{2}}{n}$. Show that $\mathrm{Var}[\mathbf{C}] \leq \frac{2t^2}{n}$.

2. (3 points) Consider estimating the network size as $\tilde{n} = \frac{\binom{t}{2}}{\mathbf{C}}$. How large must you set t so that $\Pr(|n - \tilde{n}| \geq \epsilon n) \leq \delta$ for $\delta > 0$? (Give an expression in terms of n, ϵ, and δ.) **Hint:** Start by showing how to pick t so that $\mathbf{C}$ is close to its expectation with good probability.

3. (1 point) Are you able to apply an exponential tail bound (Chernoff bound or Bernstein's inequality) to bound the accuracy the above estimation procedure?

4. (3 points) Describe and analyze a method that given failure probability $\delta > 0$, outputs $\tilde{n}$ with $\Pr(|n - \tilde{n}| \geq \epsilon n) \leq \delta$ and only has a $O(\log(1/\delta)$ dependence in its sample complexity.

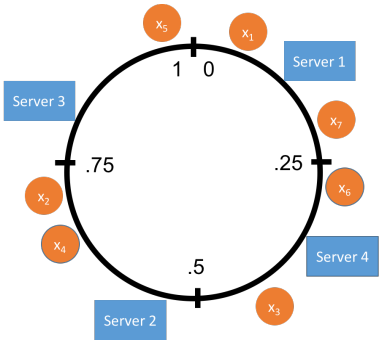
5. **Bonus:** (2 points) Why is the assumption that the random crawls land on uniformly random nodes unrealistic? How might this issue affect your estimator and how might you combat it?

You may assume that the number of crawls t is less than the network size n in all bounds. You do not need to work out precise constants in any bounds (i.e., may use Big-O notation.)

4. Randomized Load Balancing Meets Scalability (10 points)

Consider a large scale distributed database, storing items coming from some universe U. A random hash function $\mathbf{h} : U \to [k]$ is used to assign each item x to one of k servers. When that item is queried in the future, the hash function is used to identify which server it is stored on. In many applications, the number of servers scales dynamically, depending on the storage load, availability, etc. If a new server is added to the current set of k, since $\mathbf{h}$ maps only to $[k]$, we will have to pick a new hash function, rehash and move all the stored items.

Consider the following solution: pick a random hash function $\mathbf{h}$ which maps both items and servers to values chosen independently and uniformly at random in the range $[0, 1]$. Each item is assigned to the first server to its right, with wrap-around. I.e., item x is assigned to the server with the smallest hash value larger than $\mathbf{h}(x)$. If there are no servers with a larger hash value, the item is assigned to the server with the smallest hash value. When a new server is added to the system it is hashed to $[0, 1]$ and any items that should be assigned to it are moved.



1. (3 points) If we have n items, originally stored on k servers, and we add a new server using this method, what is the expected number of items that will be moved to that new server. **Hint:** First show that the expected size of the hash range assigned to any server when there are k servers is $\frac{1}{k}$. Then remember that the $(k+1)^{st}$ server is hashed in exactly the same way as the previous servers.

2. (4 points) Show that with probability $\geq 9/10$, with k servers in total no server is assigned a hash range of width greater than $\Omega\left(\frac{\log k}{k}\right)$. **Hint:** Fix a single server and then think about how the remaining $k - 1$ servers must be hashed so that its hash range has size $\Omega\left(\frac{\log k}{k}\right)$.

3. (3 points) Show that with probability $\geq 9/10$ the maximum load on a server is $O\left(\frac{n \log k}{k}\right)$.

You may assume that k and $\frac{n}{k}$ are both larger than 2.