HW2 - October 31, 2016

Diane Tam, Ben Cheung

1) Let n$\hat{p}$ be the actual number of head. And let np be the expected number of heads.
We model our equation as $P(\|\hat{p} - p| \leq \delta) < \gamma$.

$$= P(|n\hat{p} - np| \geq n\delta) \geq 1 - \gamma \Rightarrow P(|n\hat{p} - np| \geq n(\frac{1}{2} - \frac{1}{3} = \frac{1}{6})$$

$$\leq 2e^{\frac{-n(\frac{1}{6})^2}{3p}} \geq .999$$

Let p = 1, we are placing an upper bound on this p

$$\frac{.999}{2} \leq e^{\frac{-n}{108}}$$

$$ln(\frac{.999}{2}) \leq \frac{-n}{108}$$

$$-108(ln(\frac{.999}{2})) \geq n$$

$$n = 74.97$$

2 a) We can start by finding that the i^{th} bit in the first array could remain a zero after one hashing with probability $(1 - \frac{1}{k/n})$, because for each hash array, there is k/n spaces. So for a random hash function, there is equal probability for each of these spaces to be hit. After hashing m elements to this array, the probability is raised to the m^{th} power: $(1 - \frac{1}{k/n})^m$. Each hash function array is also the same probability for the i^{th} element to be zero after m elements are hashed. we thus multiply all of these together and get $(1 - \frac{1}{k/n})^{km}$, since there are k hash functions.

$$(1 - \frac{1}{k/n})^{km} = (1 - \frac{1}{k/n})^{km * \frac{n}{k} * \frac{k}{n}} \approx e^{-mk * \frac{k}{n}} = e^{\frac{-mk^2}{n}}$$

This is the probability that the i^{th} bit in all the hash arrays all remain zero after m elements are hashed in each function. Therefore the probability that the i^{th} bit is 1 is $1 - e^{\frac{-mk^2}{n}}$. Therefore, probability of false positive is $(1 - e^{\frac{-mk^2}{n}})^k$

This differs from the regular approach by the exponent in e (regular false positive: $(1 - e^{-mk/n})^k$). For the modified approach e is raised to an extra power of k. We can see that The false positive rate rises with the modified algorithm.

2 b) The false positive rate increases as the number of hash functions go up. This is because the more hash functions we have, the more opportunity we have for collisions as more bits get set to 1 in the pre-processing stage. If there are too few hash functions, the false positive rate also is higher because the hash function must fit all of our pre-processing words into a fixed size array. Since we have way more words than spaces in the array, we are bound to have collision.

Please see code comments for implementation explanations. run python bloom.py

3 a) In class we defined the majority item if it takes up more than half the stream: $\frac{m}{2} + 1$, which is $\frac{m}{k+1}$ since k =1 in this case. To generalize, if we are looking for the k most frequent items in a stream, then each of the k most frequent items must appear at least $\frac{m}{K+1}$ times.

Similar to the proof of contradiction from class, we can generalize the proof for the k most frequent items. Assume that the algorithm is incorrect, such that there is an item X in the stream that has frequency greater than $\frac{m}{k+1}$ but was not in storage at the end of the algorithm. During the algorithm, When X appears either it's stored or not. If it is already stored, then it must have gotten decremented by another item since it was not remaining in the end (let's associate that item with X) or incremented as more X appears in the stream. And if X is not stored, it decrements all values that are stored. Every appearance of X must be associated with a unique item.

3 b) In terms of the hashtags returned as the ones with the greatest frequency, the algorithm a was able to return some of the actually top frequency hastags such as teamfollowback or teamiphone or team. But the downside was that the results of algorithm a (which is called heavy hitter in our files) had a bunch of values left in the storage that were not frequent numbers, there simply weren't 500 hashtags that were truly frequent (over frequency m/k 5600).For space requirements, CMS general algorithm uses 6 arrays for the 6 hash functions. Each array is of length e/epsilon which we set for 2720. For the algorithm a, only used two parallel lists of size k, the number of buckets (which was 500). One list was for the counter, the other for the hashtag name. So the algorithm a used much much less space, but was not as accurate as the CMS implementation.