IS622 Week 5 HW

Brian Chu | Sept 27, 2015

Exercise 4.2.1 : 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., "CS101") and likewise, studentID's 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.

Key attribute = university

(b) Estimate the fraction of students who have a GPA of 3.5 or more.

Key attribute = studentID grouped by university (since studentID is not unique)

(c) Estimate the fraction of courses where at least half the students got "A."

Key attribute = courseID grouped by university (since courseID is not unique)

Exercise 4.3.3: As a function of n, the number of bits and m the number of members in the set S, what number of hash functions minimizes the false-positive rate?

Let r = ratio of m members to n bits = $\frac{m}{n}$
Let y = false-positive rate
Let k = number of hash functions

Therefore, we want to minimize $F(y) = (1 - e^{-rk})^k$ as a function of r and k

The derivative is $F'(y) = (1 - e^{-rk})^k \cdot (ln(1 - e^{-rk}) + \frac{rke^{-rk}}{1-e^{-rk}})$

We set F'(y) equal to 0 and solve for k to find the number of hash functions that minimizes the false-positive rate. This is rather computationally difficult so I demonstrate by subbing in values of k between 0 and 10 into the original function until a minimum is reached. I will use the values $m = 1$, $n = 8$, and $r = \frac{m}{n} = \frac{1}{8}$. I am also assuming no decimal number of hash functions is possible.

```
m <- 1
n <- 8
r <- m/n

for (k in 0:10) {
  y <- (1 - exp(-k*r))^k
  print(paste("k=",k, "y=", round(y,4)))
}
```

```
## [1] "k= 0 y= 1"
## [1] "k= 1 y= 0.1175"
## [1] "k= 2 y= 0.0489"
```

```
## [1] "k= 3 y= 0.0306"
## [1] "k= 4 y= 0.024"
## [1] "k= 5 y= 0.0217"
## [1] "k= 6 y= 0.0216"
## [1] "k= 7 y= 0.0229"
## [1] "k= 8 y= 0.0255"
## [1] "k= 9 y= 0.0292"
## [1] "k= 10 y= 0.0342"
```

Therefore, about 6 hash functions minimizes the error rate when r = $\frac{1}{8}$.

Let's substitute values back into our derivative function and see if the slope indeed reaches 0 around the range 5-7.

```
kk <- seq(5,7,.1)
for (k in kk) {
  y <- ((1 - exp(-r*k))^k) * ((log(1 - exp(-r*k))) + (r*k*exp(-r*k) / (1-exp(-r*k))))
  print(paste("k=",k, "y=", round(y,4)))
}
```

```
## [1] "k= 5 y= -0.001"
## [1] "k= 5.1 y= -8e-04"
## [1] "k= 5.2 y= -6e-04"
## [1] "k= 5.3 y= -4e-04"
## [1] "k= 5.4 y= -2e-04"
## [1] "k= 5.5 y= -1e-04"
## [1] "k= 5.6 y= 1e-04"
## [1] "k= 5.7 y= 2e-04"
## [1] "k= 5.8 y= 4e-04"
## [1] "k= 5.9 y= 5e-04"
## [1] "k= 6 y= 7e-04"
## [1] "k= 6.1 y= 8e-04"
## [1] "k= 6.2 y= 0.001"
## [1] "k= 6.3 y= 0.0011"
## [1] "k= 6.4 y= 0.0012"
## [1] "k= 6.5 y= 0.0014"
## [1] "k= 6.6 y= 0.0015"
## [1] "k= 6.7 y= 0.0016"
## [1] "k= 6.8 y= 0.0017"
## [1] "k= 6.9 y= 0.0019"
## [1] "k= 7 y= 0.002"
```

We do in fact find that the slope reaches 0 in this range, between 5.5 and 5.6 to be exact.

Also, because y gets smaller as r gets larger, we can deduce the number of k hash functions needed gets smaller as r gets larger too.

Exercise 4.5.3: Suppose we are given the stream of Exercise 4.5.1, to which we apply the Alon-Matias-Szegedy Algorithm to estimate the surprise number. For each possible value of i, if Xi is a variable starting position i, what is the value of Xi.value?

```
stream <- c(3, 1, 4, 1, 3, 4, 2, 1, 2)
n <- length(stream)

element <- integer(length(stream))
value <- integer(length(stream))

for (i in 1:n) {
  element[i] <- stream[i]
  value[i] <- sum(stream[i:n] == element[i])
}

df <- data.frame(element, value)
df
```

```
##   element value
## 1       3     2
## 2       1     3
## 3       4     2
## 4       1     2
## 5       3     1
## 6       4     1
## 7       2     2
## 8       1     1
## 9       2     1
```

The estimate of the surprise (second) moment is given by the formula $\sum_{i=1}^{n} \frac{n(2 \cdot X.value - 1)}{n}$

```
e <- integer(n)

for (i in 1:nrow(df)) {
  e[i] <- n*(2 * value[i] - 1)
}

second_order_moment <- sum(e)/n
print(second_order_moment)
```

```
## [1] 21
```

The estimate is 21.