Lecture 2: Introduction to Counting
Modeling Social Data, Spring 2017
Columbia University

January 27, 2017

Notes from bh2589

1 Why counting?

We can get some inspirations from the following example.

E.g. 2016 American Presidential Polls

Suppose we want to investigate the relationship between the election result and some related factors, such as: Age, Sex, Race, Party and etc. (i.e. P(SupportAge, Sex, Race, Party)), and we categorize Age into 100 groups(1-99, 100+), Sex into 2 groups(Male/Female) , Race into 5 groups(White/Black/Asian/Hispanic/Others), Party into 3 groups(Republican/ Democratic/Independent), so there should be 100*2*5*3 = 3000 groups in total theoretically. However, normally by doing surveys, we can only collect around 1000 responses of which the number is much less than the number of groups we want to investigate.

So here is the problem: It is difficult to obtain reliable estimates due to small sample sizes or sparsity.
By means of counting, we can solve this problem.

Potential solutions:

1. Sacrifice granularity for precision, by binning observations into larger, but fewer groups, so that the total number of groups can be significantly decreased.

   E.g. Bin Age from 100 groups(1-100 years old) in to fewer groups(like 5 groups: 18-29, 30-49, 50-64, 65+), then the total number of groups can be decreased by 20 times.

2. Develop more sophisticated methods that generalize well form small samples.

   E.g. Fit a model like: Support $= \beta_0 + \beta_1 Age + \beta_2 Age^2 + ...$

   'Rather than: Support$= \beta_0 + \beta_1 Age + ...$

3. Obtain larger samples through other means, so we can just count and divide to make estimates via relative frequencies.

   E.g. If we can collect around 1 million responses, we can have more than 100 responses for each group, so that we can make a good estimate of support within a few percentage points.

- **Advantages and Disadvantages**

1. The good:

   Shift away from sophisticated statistical methods on small samples to simpler methods on large samples.

2. The bad:

   Even simple methods are computationally challenging at large scales.

2 How to count? (Counting at small/medium scales on a single machine.)

- **Procedures**

1. Split: Arrange observations into groups of interest.
2. Apply: Compute distributions and statistics within each group.
3. Combine: Collect results across groups.

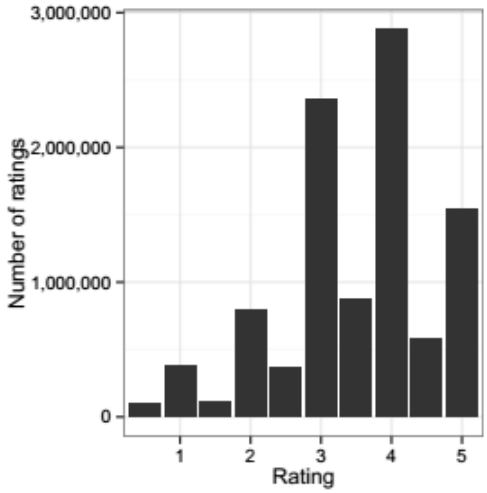
E.g. Calculating the means of the numbers in various groups (Assume there are N numbers in total, and they are from three different groups A, B, C)

Group	Number
A	1
B	2
C	3
A	4
B	5
C	6
A	7

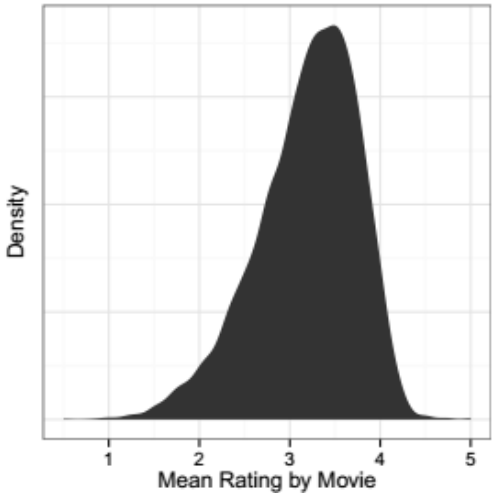
Dummy way	Firstly, search the numbers of Group A from the first element to the last one; For Group B and C, do the same thing, after splitting all of the numbers into 3 groups, we can calculate the means respectively.
	Time: NG ; Space: N
Better way (Computational cost is lower.)	Search the elements in the group one by one, and categorize each element into the corresponding group (A or B or C), then calculate the means respectively.
	Time: 2N ; Space: 2N

E.g. The generic group-by operation Split: Mark each observation as (group, value) and place value in bucket in corresponding group. Apply: Apply a function over values in bucket output group and result. Combine: Combine the final results.

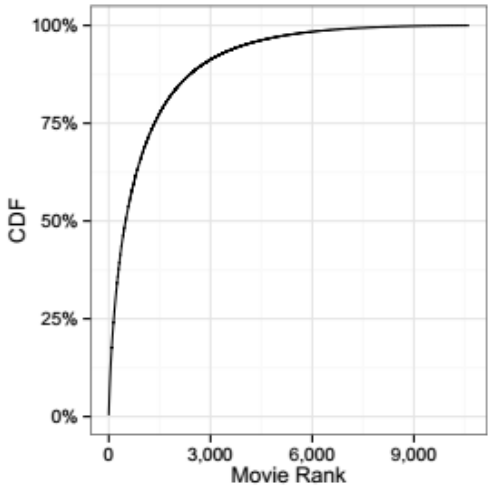
E.g. Movielens (What we only care about is how many ratings there are, not rating itself.)



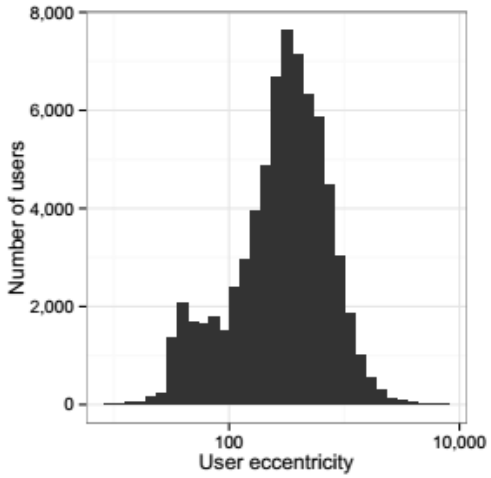
This picture shows how many ratings there are at each star level (5 star levels in total).
The data is grouped by rating value for each group (2 groups in total in this picture).



This picture shows the distribution of the density of movies with different mean ratings.
The data is grouped by movie id for each for each group and then compute the average ratings.



This picture shows the fraction of ratings given to the most popular movies (with the most ratings).
The data first is grouped by movie id and then count ratings and sort by the number of ratings, finally calculate the percentage of the group size of each movie and cumulate them gradually.



We can infer the median rank of each users rated movies from the distribution from this picture.

First, we need to join movie ranks to ratings and then we group date by user id for each group, finally we compute the median movie rank.

- **What do we do when the full dataset exceeds available memory?**

(Wrong) Sampling: Unreliable estimates for rare groups

(Wrong) Random access from disk: It is much slower though the storage are more.

(Right) **Streaming:** Read data one observation at a time, storing only needed state.

E.g. The combination group-by operation

Mark each observation as (group, value), if a new group appears, we initialize the result, or we update the result for corresponding group as function of existing result and current value. Finally, for each group, we output the result.

- **The group-by operation**

For arbitrary input data:

Memory	Scenario	Distributions	Statistics
N	Small dataset	Yes	General
V*G	Small distributions	Yes	General
G	Small # groups	No	Combinable
V	Small # outcomes	No	No
1	Large # both	No	No

N = total number of observations
G = number of distinct groups
V = largest number of distinct values within group

For pre-grouped input data:

Memory	Scenario	Distributions	Statistics
N	Small dataset	Yes	General
V*G	Small distributions	Yes	General
G	Small # groups	No	Combinable
V	Small # outcomes	Yes	General
1	Large # both	No	Combinable

N = total number of observations
G = number of distinct groups
V = largest number of distinct values within group

E.g.

Median rating by movie for Netflix: N - 100M ratings; G - 20K movies; V - 10 half-star values.

Since V*G = 200K, we can store per-group histograms for arbitrary statistics.

Median rating by video for YouTube: N - 10B ratings; G - 1B videos; V - 10 half-star values.

Under this circumstance, V*G = 10B, we cant store the data anymore because per-group histograms are too large to store in memory.

Mean rating by video for YouTube: N - 10B ratings; G - 1B videos; V - 10 half-star values.

In this situation, G - 1B, we can use streaming to compute combinable statistics.

Notes from hsa2136

3 Counting

Regular Statistics is basically just conditional probability: trying to find $P(y|x)$. If we have a small sample size, we're going to have large expected error.

The uncertainty we get is inversely related to the square root of the sample size:

$$\sqrt{\frac{p(1-p)}{N}}$$

The problem gets worse when we condition on more variables. For example, if we condition on multiple features such as $P(y|x_1, x_2, ...)$

Concrete example: 100 ages, 2 sexes, 5 races, 3 parties $\rightarrow$ 3000 groups. So, if we want to have a good enough N for each group, we need LOTS of samples.

One thing to do: bin features e.g. age between 18-24 is a group.

Another solution is to make a huge, non-representative study - e.g. poll all XBox users (mostly 9-17 year old guys). This causes a new problem: computation is very hard on such a large sample.

New Framework: split/apply/combine: split the data into groups, apply the computation (e.g. mean), combine the groups to get a sample-wide statistic.

Examples:

Bad Way:

1. Scan through X searching for a
2. Compile list of y-values
3. compute mean
4. repeat for b, c, ... etc.

Time: N*G (# of samples times # of groups)
Space: N

Better Way:

1. Scan through X
2. Compile list of y-values for each value of x
3. compute mean

Time: 2N
Space: 2N

Even Better:

1. Scan through X
2. Sum up total for each group and a counter
3. compute mean

Time: 2N
Space: 2G

Notes from sd2920

4 Introduction

- A few million, your computer wont notice anything
- A few billion, your laptop can handle it, but youll notice mistakes
- A few trillion, you need distributed computing (consider Hadoop)

4.1 Counting at scale: Estimating a Distribution

How to represent a distribution: p(y|x): y = expected result, x = variable for which the result is conditioned
Example from the slides: p(support | age, race, sex ...)

- Given election example: p(People who voted Hillary|Hispanics) = 50%
- We are trying to find a distribution that models how the Hispanic demographic voted. Let x = Hispanics, y = voted for Hillary
- We can try a binomial distribution:
  - If the true underlying value of p(y|x) is 50%, we can model sampling 500 people with: rbinom(500, 1, .5)
  - This is like flipping a coin 500 times with 1 meaning heads and 0 meaning tails
  - Running: *barplot(table(rbinom(50000, 100, .5)))* yields what looks just like a normal distribution! Hence we have just estimated a distribution given some conditioning knowledge
  - If we had a mixed binomial probability (by other conditional variables perhaps), then our resulting distribution would be a more complex mixture model.
- Calculating error?
  - $\sqrt{\frac{(p(p-1)}{N}}$
  - This formula roughly makes sense because as the number of things (N) gets larger, the error goes down.
  - Hey this looks like the central limit theorem: $\sqrt{\frac{\sigma}{N}}$

Counting shifts us away from sophisticated statistical methods on small samples and to simpler methods on larger samples.

4.2 Split Apply Combine Paradigm

This paradigm entails loading all your data into memory.

- Split: Arrange observations into groups of interest (individuals preferences/observations)
- Apply: Compute distributions/statistics
- Combine: Collect observations across groups
- Complexity: Time: 2N, N to split into groups, N to run distribution calculations, Space: 2N, copying grouped data to another place

Note: If all we want is a running mean, and we know that before we get our data, then we can just use the running mean algorithm on the stream of data as we see it. Split/Apply/Combine allows us to do more with the data

Things to look out for with Split/Apply/Combine:

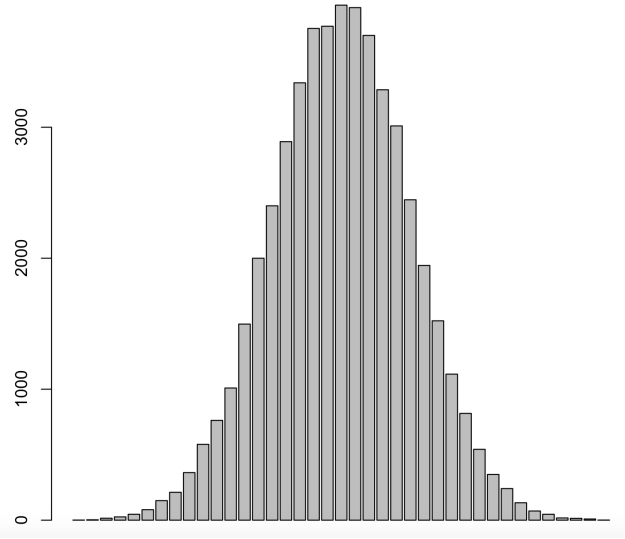


Figure 1: The result of *barplot(table(rbinom(50000, 100, .5)))*
It looks exactly like a normal distribution.

- If, for example we're calculating the variance: $\frac{1}{N}\Sigma(x_i - \bar{x})^2$
- Note that the formula can be simplified to $\frac{1}{N}\Sigma({x_i}^2 - 2\bar{x}x_i + \bar{x}^2)$
- But even with the simplified expression, we have a squared term which can lead to significant problems if the number is a high value float (with a mean of 10^6 the squared term becomes 10^12 which can lead to overflows
- Another note: The median is simply the middle number in the distribution, but it is computationally infeasible to calculate the median in streaming data without storing all values.

Algorithm Shell for Split/Apply/Combine:

for each observation as (group, value) **do**:
  place value in bucket for corresponding group
end for
for each group **do**:
  apply function over values in group
  output group and result
end for

This shell is useful when we are calculating within group statistics when we have required memory.

4.3 The Long Tail

The long tail is a statistical phenomenon where there is a small set of super popular things, followed by a very large set of moderately popular/unpopular things.
The long tail is often given as a reason why Netflix beat Blockbuster (even though in the early early days Netflix would have to mail you the movie when you would be able to just go drive to a Blockbuster).
Sampling on distributions with long tails is dangerous because its hard to model the tail drop off, so the distribution can easily be misrepresented.

Algorithm for dealing with long tail streaming data:

for each observation as (group, value) **do**:
  if new group: **then** Initialize result
  end if
  Update result for corresponding group as function of existing result and current value
end for
for each group **do**:
  apply function over values in group
  output group and result
end for

Though this algorithm seems really similar to what we've already seen, its really good for computing a subset of within group statistics like min, max, mean, variance with little memory usage.
Netflix Example: everyone likes the top 20 most popular movies, but we all also have weird esoteric tastes that are hard to service (at places like Blockbuster). For Blockbuster with physical locations it was easy to keep like 5000 most popular movies, but then our individual tastes get put aside for what's popular.

Perhaps we can use Split/Apply Combine here as well.

Pseudocode for Algorithm:

function GROUP BY ID(Dataset)
end function
for each Group in Dataset **do**:
  function SORT BY POPULARITY(Group)
  end function
end for
for each Group in Dataset **do**:
  function CUMULATIVE SUM(Group, Size of Data Set)
  end function
end for

Assume we have just grouped by ID

Movie ID	Number of users wanting to watch	Cumulative percentage breakdown
Jurassic Park	100	40.98 %
The Return of the King	201	82.37 %
Dances with Wolves	43	17.62 %

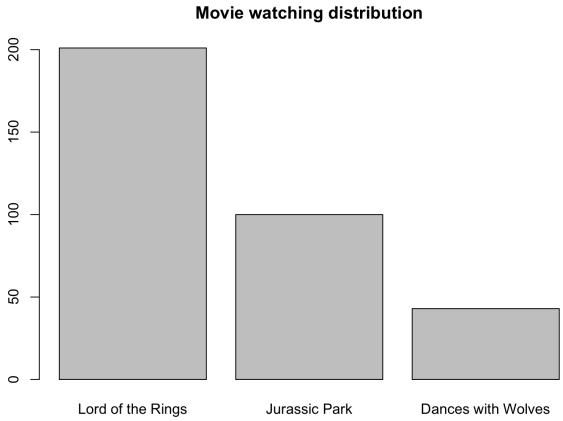


Figure 2: The result of graphing the cumulative movie preferences of all 344 users. Note how the distribution quickly tapers off at the end.

Handy chart for performing group-by functions on Pregrouped data:

Memory	Scenario	Distributions	Statistics
N	Small dataset	Yes	General
V*G	Small Distributions	Yes	General
G	Small # groups	No	Combinable
V	Small # outcomes	Yes	General
1	Large # both	No	Combinable

Legend:

- N = total number of observations
- G = number of distinct groups
- V = largest number of distinct vales within group

4.4 Practical Shell Stuff:

- **man**: gives users guide to the next function argument
- **ls**: lists directory contents, optional arguments include details of ownership/execute privileges
  - ls -a -l: these options allow you to see all files in a listed format
  - ls -alh: file sizes
  - ls -l *.sh: all .sh files
- **bash**: Bourne Again Shell, interactive programming environment where programs/executables can be run and directories can be navigated
- **more**: allows for quick examination of a file in the shell (so without using a text editor to walk through the file)
- **less**: Allows for quick exploration of a file like more but also allows one to go 'backwards' or to previous content in file
  - double quotes in the shell: allows for expansion of variables from the shell
    - all characters are treated the same except for $ and ' which are expanded in shell
  - single quotes: expression is strictly, literally, interpreted

Notes from sh3266

5 Mentions Beyond Lecture Slides

First, we discuss conditional probability with an example of a demographic survey shown in *slide 2*. Note any comments or disclaimers denoted with asterisk * explaining any estimates for the research. When the margin of estimate is high, the uncertainty is also high.

$$p(y|x)$$

$$uncertainty = \sqrt{p(1-p)/N} \approx \sqrt{\sigma/N}$$

We see that as N (sample size) increases, uncertainty decreases.

Unlike in the real world, computers can compute experiments many times. For example, through a programmed simulation of coin flipping, we can visualize the outcome with a histogram and determine an empirical probability. The coin example was simulated in class in R. $rbinom(n, 1, p)$ in R simulates a single flip with p probability n times. $set.seed(k)$ sets a seed for a deterministic result even when the simulation is repeated. The results would differ each time $rbinom$ is run (because the simulation is based on some random number generating algorithm) if a seed was not set.

slide 5 states a problem with categorical binning (polling problem). Some bins are more populated than others; significant distribution cannot be derived from bins with small samples. Later slides suggest potential solutions. One approach is to develop a sophisticated model to generalize multiple bins and group them in order to have fewer bins.

Split/Apply/Combine techniques in R are introduced, but we will cover them in more detail later (demonstrated in class during lecture 3). It is always better to use apply functions than for loops, especially for large data. Iterating through a massive matrix is extremely expensive. Instead, R will split the data set, apply a desired function, such as mean(), to subsets, then combine the results to return.

Different ways of computing the mean and the running mean of a data set were demonstrated in class. Given a table consisting of two columns (bin type column and value column), populating each group first, as shown below, before computing the mean is more efficient than naively iterating through the table to find values corresponding to each bin to calculate the mean.

$$a : 2, 5, 6, 10, 3; b : 1, 66, 33; c : 4, 20, 6...etc.$$

The time complexity for computing the mean of each bin using this "group by" method is 2N with 2N space and NlogN sorting time.

We compute the running mean by calculating the sum of each bin as we iterate through the list and dividing by the total occurrence thus far. In R, $filter()$ function calculates the moving average.

$$mean : \bar{x}, var : \frac{1}{N}\sum(x_i - \bar{x})^2$$

"The Anatomy of the Long Tail" is a reference to the end tails of a distribution curve. It describes the phenomenon of there being few popular things and many unpopular things. For example, there are very few movies that are highly popular, and many movies that are not.

Finally, we discuss some solutions to dealing with large data sets that exceed memory. Random sampling is deprecated because significant extreme points are likely to get filtered out. Loading data from the disk is too slow. One potential approach is to stream data in. Streaming is one model of counting that trades off flexibility

for scalability without sacrificing runtime. We compute the running mean and variance as the data streams in without storing the entire data set in memory.

6 Lecture Slide Content

Slide 5 The Problem

As we deal with big data throughout the course, counting is crucial not only to understand data but also to pre-process and analyze it. Types of data vary from political popularity polls to preferred games or TV shows. The responses collected can be binned into different categories such as age, sex, and race before computing statistical significance. However, there are often not enough responses to derive meaningful values for each bin. This phenomenon Known as the "curse of dimensionality" in dynamic optimization problems also occurs in combinatorics and sampling as seen in class. As the number of dimensions (categories) increases, the volume of the sample space also increases and the data becomes sparse. Slides 6-12 propose potential solutions and discuss their trade offs.

Slide 6-12 Potential Solution

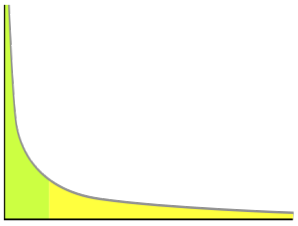
One approach is to develop an algorithm that generalizes multiple bins as one larger bin. For example, map multiple features x_1 and x_2 through some linear combination $B_0 + B_1\ x_1 + B_2\ x_2$. A possible algorithm could use Principal Component Analysis to reduce dimensionality. Additionally, collect more responses through various methods. But this is not always realistic. The benefit of dimensionality reduction is that a simpler method can apply to larger data bin. But as the data gets larger, computing under reasonable time becomes challenging.

Slide 15-17 Counting with R

R uses special techniques to compute counting problems using Split/Apply/Combine as described above to reduce runtime for general functions. $apply()$ function in R is generally more efficient than for loops. R supports a convenient library called $dplyr$ that has $filter(), select(), arrange(), groupby()$ functions that apply operations to subsets of data. $filter()$ allows you to select a subset of rows in a data frame. Unlike $filter()$, $select()$ chooses by columns instead of rows. $arrange()$ reorders rows. $group_by()$ uses the "split-apply-combine" concept to perform operations.

Slide 18-32 Anatomy of the Long Tail

The power law graph below showing popularity ranking describes the long tail:



The green region to the left represent the few highly rated products and the yellow region represents the unpopular majority. Examples of the long tail (Movielens and Netflix) were shown in class in lecture slides.

Slide 33-43 Group By and Memory Problem

$group_by()$ function groups a data frame by specified criteria of rows. Functions such as $filter()$ can be applied to grouped data. We discussed ways that different data can be grouped. There may be situations when grouped data is too larged to be stored in memory. Streaming algorithms described above can be a potential solution.

7 Command Line Demo

Finally, the professor demonstrated command line operations including ls with options, cat, cut, grep, awk, and generating histograms. Demo code can be found on GitHub.