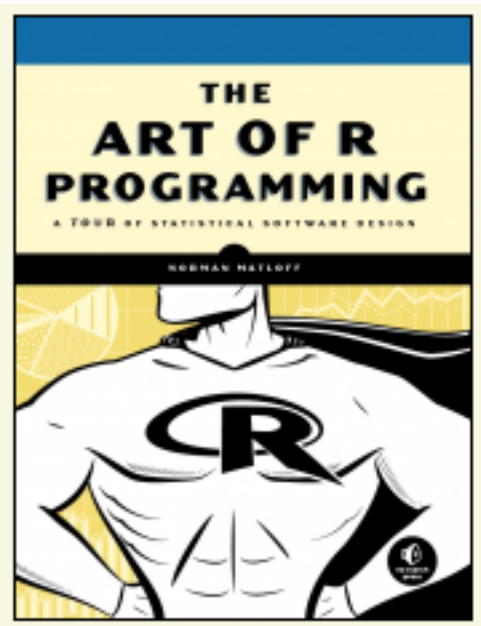
Introduction to R for Programmers

Class Material

http://polisci2.ucsd.edu/cfariss/Home/Christopher_J_Fariss.html



http://nostarch.com/artofr.htm

Getting Started

The working Directory

The workspace

Getting help

Declaring Objects in R

There are several alternatives

<-

->

=

the "_" underscore was the original declaration symbol

Vectors

Not to be confused with vectors in space.

Scalars are vectors too

Indexed beginning with 1.

There is no error if you use a 0 index however so be careful.

```
a <- c(1,2,3,4,5)
```

the c() function in R concatenates together elements of the same type (we'll put together elements not of the same type later).

Libraries/Packages

```
# install before loading
install.packages("PACKAGE_NAME")

# load packages:
library(PACKAGE_NAME)

# note that "#" is the comment character in R and that quotes are only necessary in 1 of 2 functions above (I have no idea why).

# there are some quirky features to R as the book describes
```

Lists

Lists are like vectors except that can combine elements of multiple types. You can even combine lists of lists of lists ... and so on.

Figuring out what a list contains however is a little bit tricky, at least compared to vectors, matrices and arrays.

```
# a simple list, with 3 elements:
# (1) a named character
# (2) an unamed scalar
# (3) a named vector
l1 <- list(name="list1", 3, values=c(3,8,9))
```

Vectors, Matrices and Arrays

You can declare vectors, matrices and arrays with the following functions:

vector(), matrix(), array()

each function takes different arguments. vector() is seldom used however because its easier to use c().

matrix() is used more often than array (at least in political science but probably not ecology) because the arguments are clearer.

a 2-dimensional array() is equivalent to an object declared as a matrix.

Loops!

Loops should be avoided at all costs in R but you'll use them often.

Seriously, though loops are relatively slow because of the way R processes the functions contained within the loop.

We'll see some timed demonstrations later on in the 2nd and 3rd R scripts.

As a rule of thumb, you should avoid loops if you are doing more than 10,000 combined iterations.

Short loops are fine. But loops don't scale well at all.

Reading data into R

R has many many ways to solve problems.

It also has lots of ways to read in and store data.

We will go over a few that I use a lot.

Specifically, csv files and sometimes R files.

You're choice should depend on the size and type of the data file.

For Stata users: Remember library(foreign)

Reading data into R

R has many many ways to solve problems.

It also has lots of ways to read in and store data.

We will go over a few that I use a lot.

Specifically, csv files and sometimes R files.

You're choice should depend on the size and type of the data file.

For Stata users: Remember library(foreign)

Data Frames

Think of a data frame in R as a hybrid object that looks like a matrix but can include multiple data types like a list.

Dataframes are useful objects when using statistical functions like linear regression using the lm() function.

Let's read in a csv file make sure its declared as a data frame.

Math!

As you might imagine, R can do it all.

We'll go over a few but we are just scratching the surface.

Problem 1

1a. use a for loop to go through the data frame and generate a new variable that is a function of at least 2 existing variables using one of the mathmatical functions above

1b.
If 1a seems to easy then tryout the subset command on the macro dataframe. See if you can make a random subset of half of the records from that file. Then, if there's time, run a linear model using the lm() function. Then use the predict function to see how well the model fits to the out of sample data (the data not randomly selected).

All of these functions are discussed in more detail here: http://polisci2.ucsd.edu/cfariss/code/BootCamp_p2.R

Vectorization

Now we want to learn how to use R in a way that minimizes are use of loops when possible. There are a lot of ways to do this of course.

The apply() functions are very useful for this purpose:

apply() # for matrices
lapply() # for lists
sapply() # for lists without all the list formatting
tapply() # for tables

Simulations and Distributions

R has all the bells and whistles here too. All of the basic discrete and continuous functions are available and its simple to combine these into all manner of mixed distributions for simulation purposes.

If you are interested I have some mixed distribution code I can share.

You should also check out Chapter 3 in this book:
"Ecological Models and Data in R"
http://www.math.mcmaster.ca/~bolker/emdbook/

Discrete Distributions

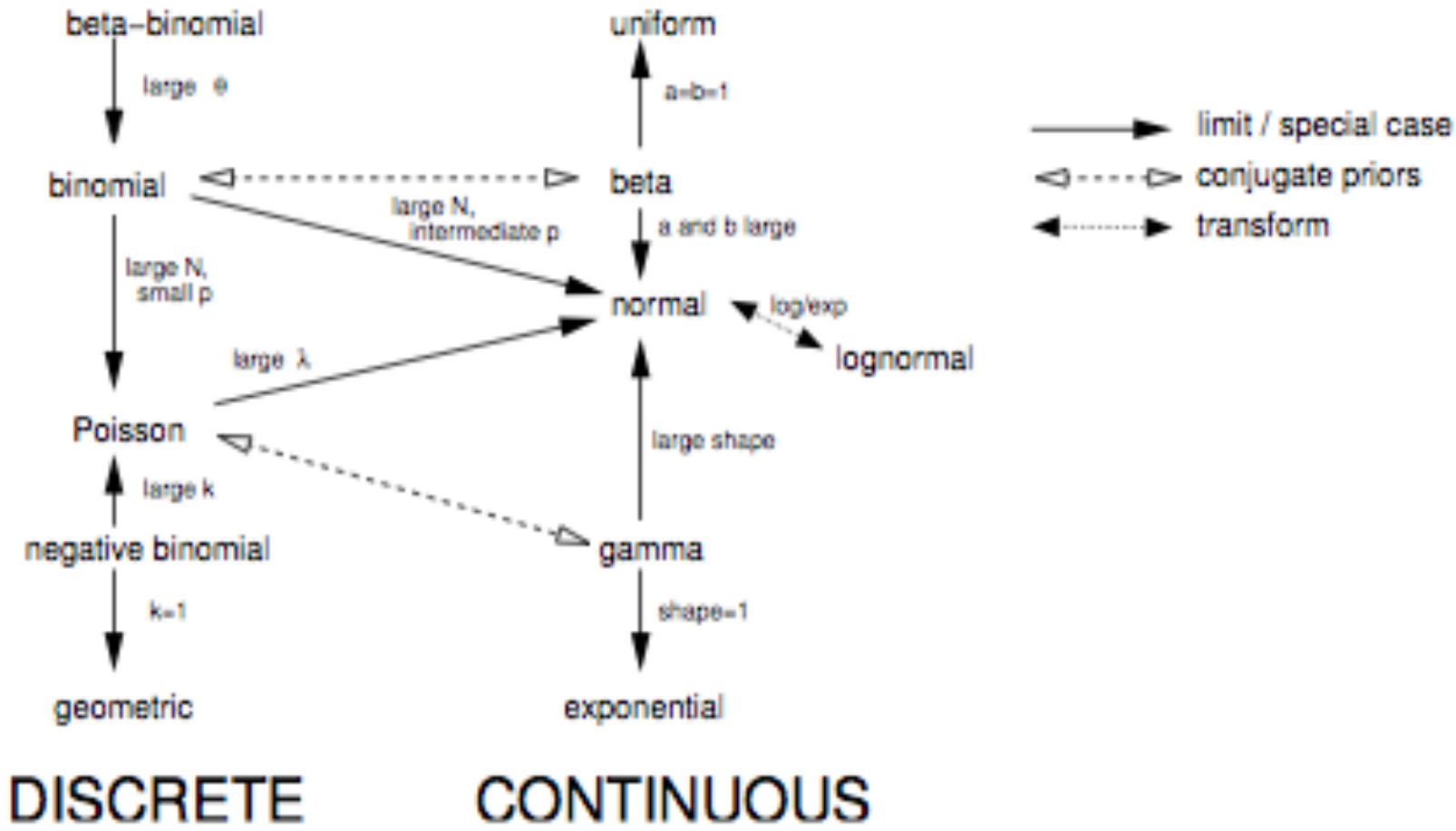
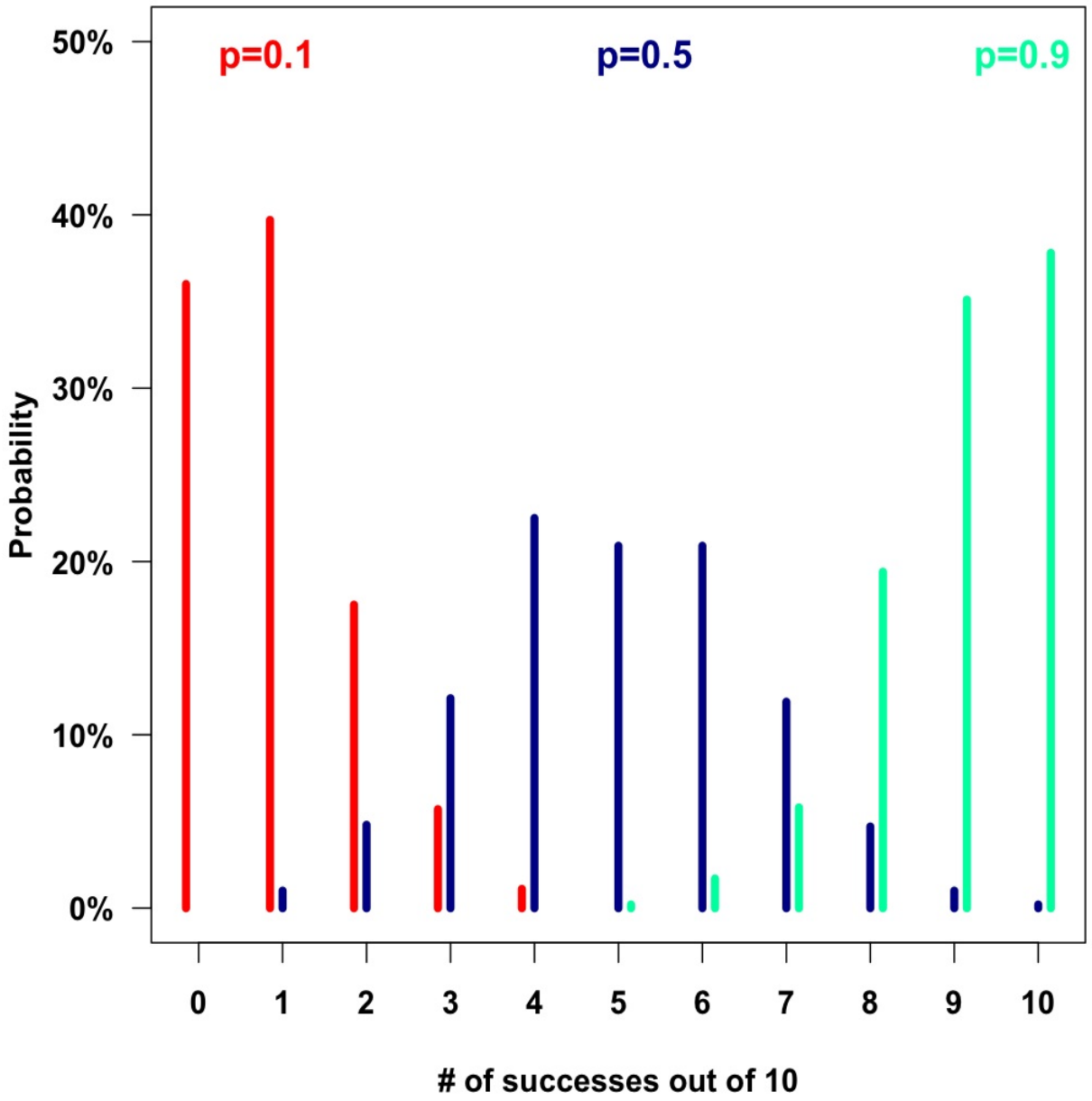


Figure 4.17 Relationships among probability distributions.

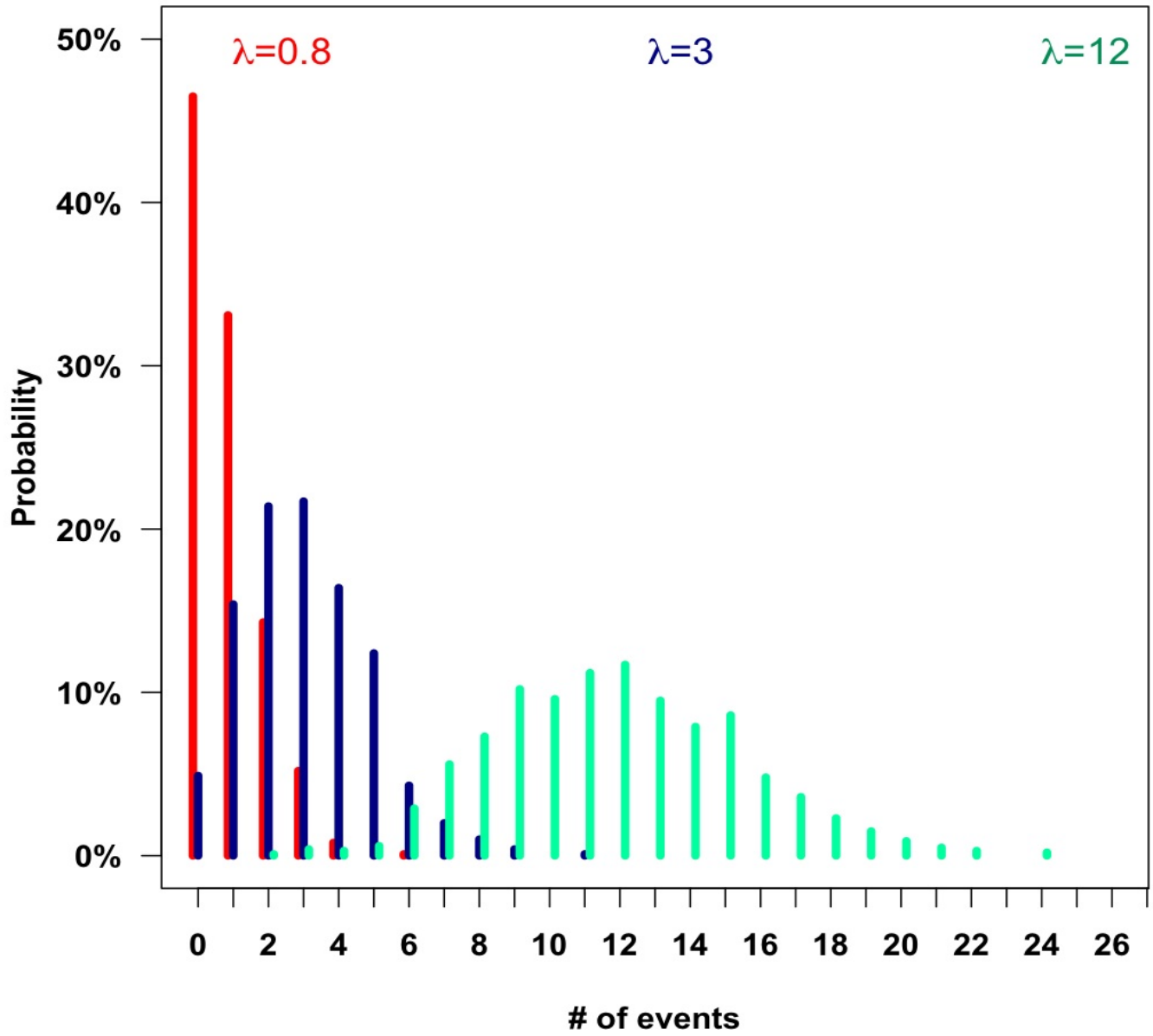
Binomial Distribution



Binomial Distribution

range	discrete, $0 \leq x \leq N$
distribution	$\binom{N}{x} p^x (1-p)^{N-x}$
R	`dbinom`, `pbinom`, `qbinom`, `rbinom`
parameters	p [real, 0–1], probability of success [`prob`]
	N [positive integer], number of trials [`size`]
mean	Np
variance	$Np(1-p)$
CV	$\sqrt{(1-p)/(Np)}$
Conjugate prior	Beta

Poisson Distribution

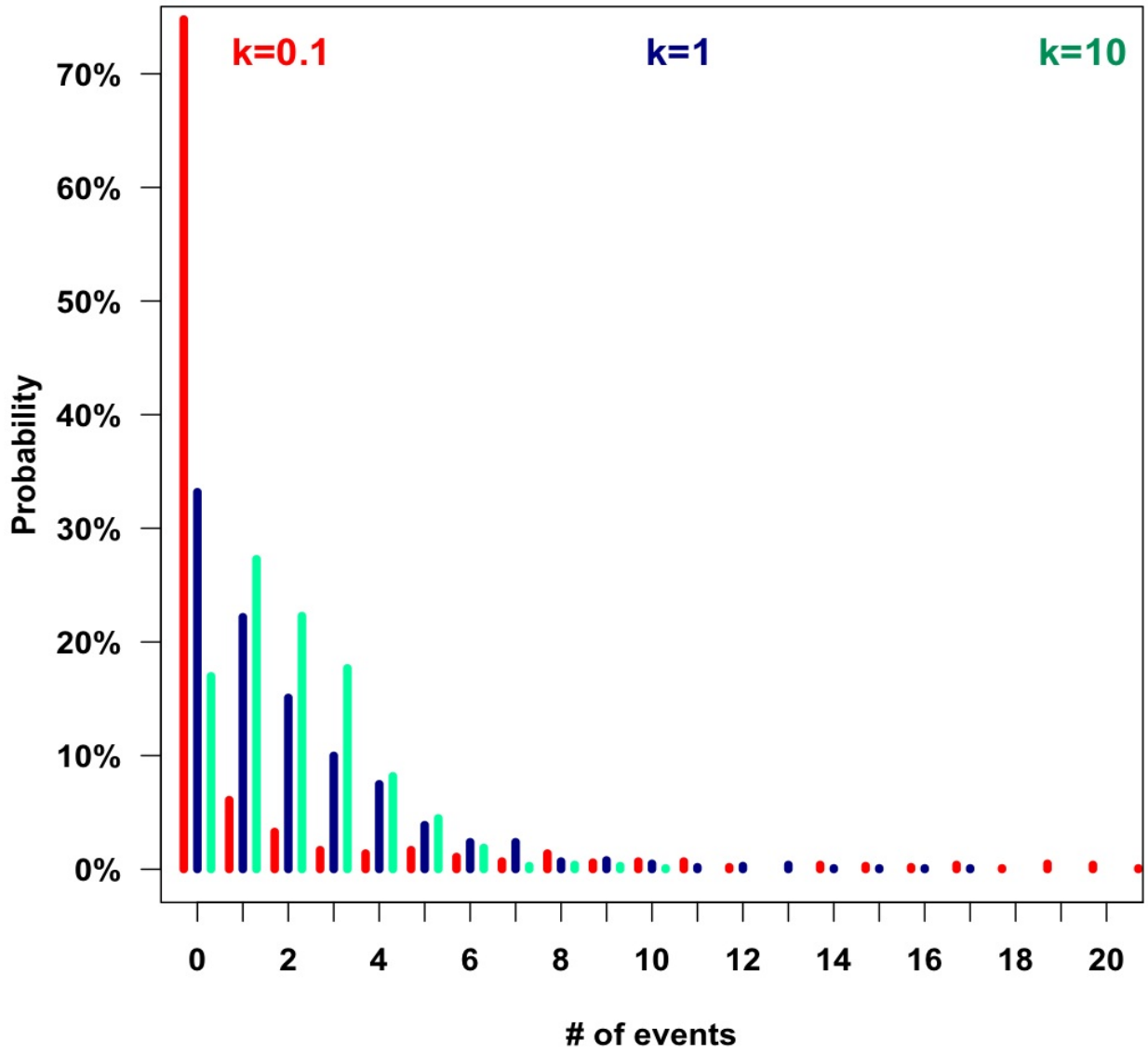


Poisson Distribution

Summary:

range	discrete $(0 \leq x)$
distribution	$\frac{e^{-\lambda}\lambda^n}{n!}$ **or** $\frac{e^{-rt}(rt)^n}{n!}$
R	`dpois`, `ppois`, `qpois`, `rpois`
parameters	λ (real, positive), expected number per sample [`lambda`] **or** r (real, positive), expected number per unit effort, area, time, etc.
mean	λ (**or** rt)
variance	λ (**or** rt)
CV	$1/\sqrt{\lambda}$ (**or** $1/\sqrt{rt}$)
Conjugate prior	Gamma

Negative Binomial Distribution



Negative Binomial Distribution

Summary:

range	discrete, $x \geq 0$
distribution	$\frac{(n+x-1)!}{(n-1!)x!}p^n(1-p)^x$
	or $\frac{\Gamma(k+x)}{\Gamma(k)x!}(k/(k+\mu))^k(\mu/(k+\mu))^x$
R	`dnbinom`, `pnbinom`, `qnbinom`, `rnbinom`
parameters	p $(0 < p < 1)$ probability per trial [`prob`]
	or μ (real, positive) expected number of counts [`mu`]
	n (positive integer) number of successes awaited [`size`]
	or k (real, positive), overdispersion parameter [`size`]
	(= shape parameter of underlying heterogeneity)
mean	$\mu = n(1-p)/p$
variance	$\mu + \mu^2/k = n(1-p)/p^2$
CV	$\sqrt{\frac{(1+\mu/k)}{\mu}} = 1/\sqrt{n(1-p)}$
Conjugate prior	No simple conjugate prior (Bradlow et al., 2002)

R's default coin-flipping (n =`size`, p =`prob`) parameterization. In order to use the "ecological" (μ =`mu`, k =`size`) parameterization, you *must* name the `mu` parameter explicitly (e.g. `dnbinom(5,size=0.6,mu=1)`).

Continuous Distributions

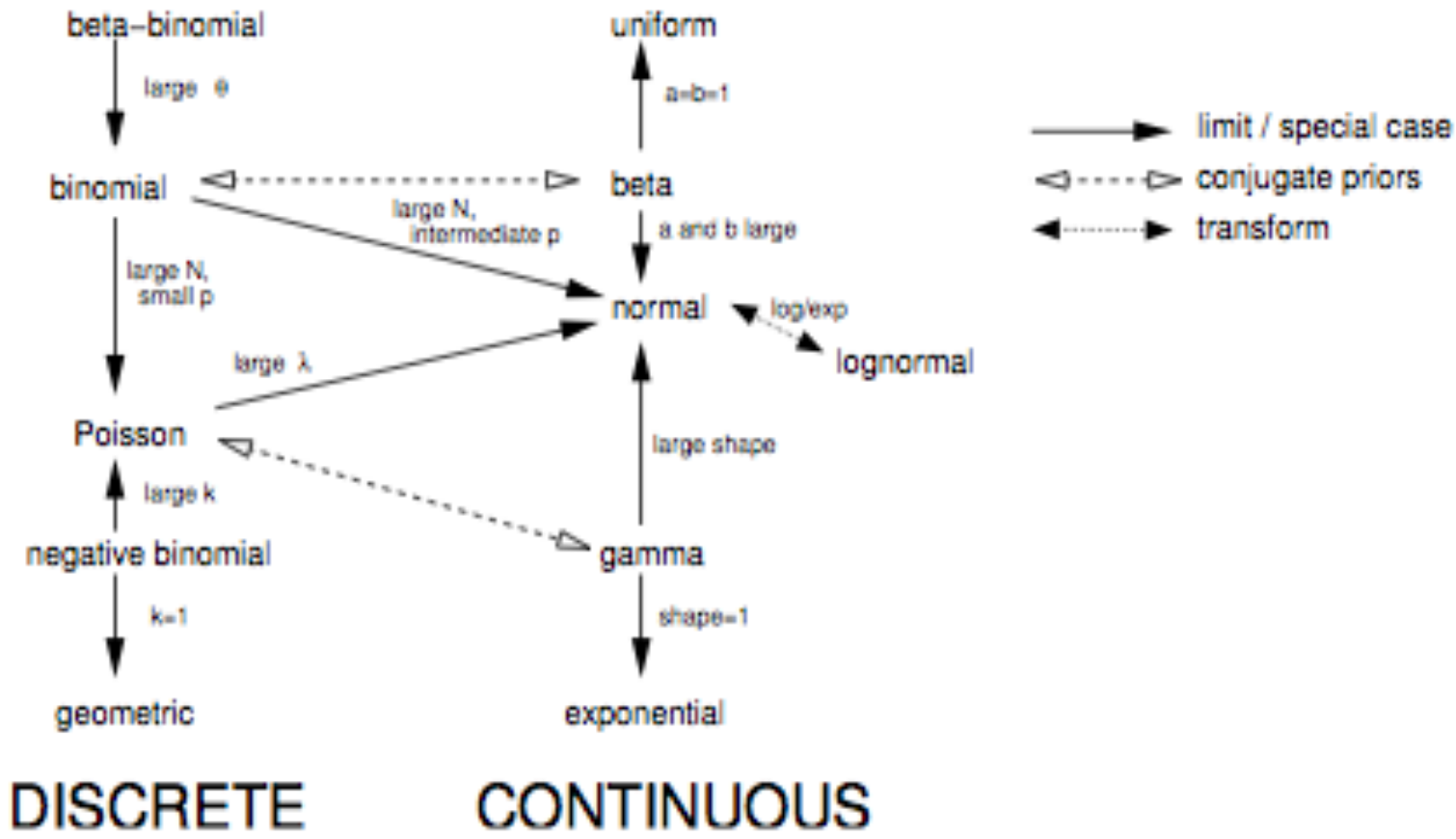
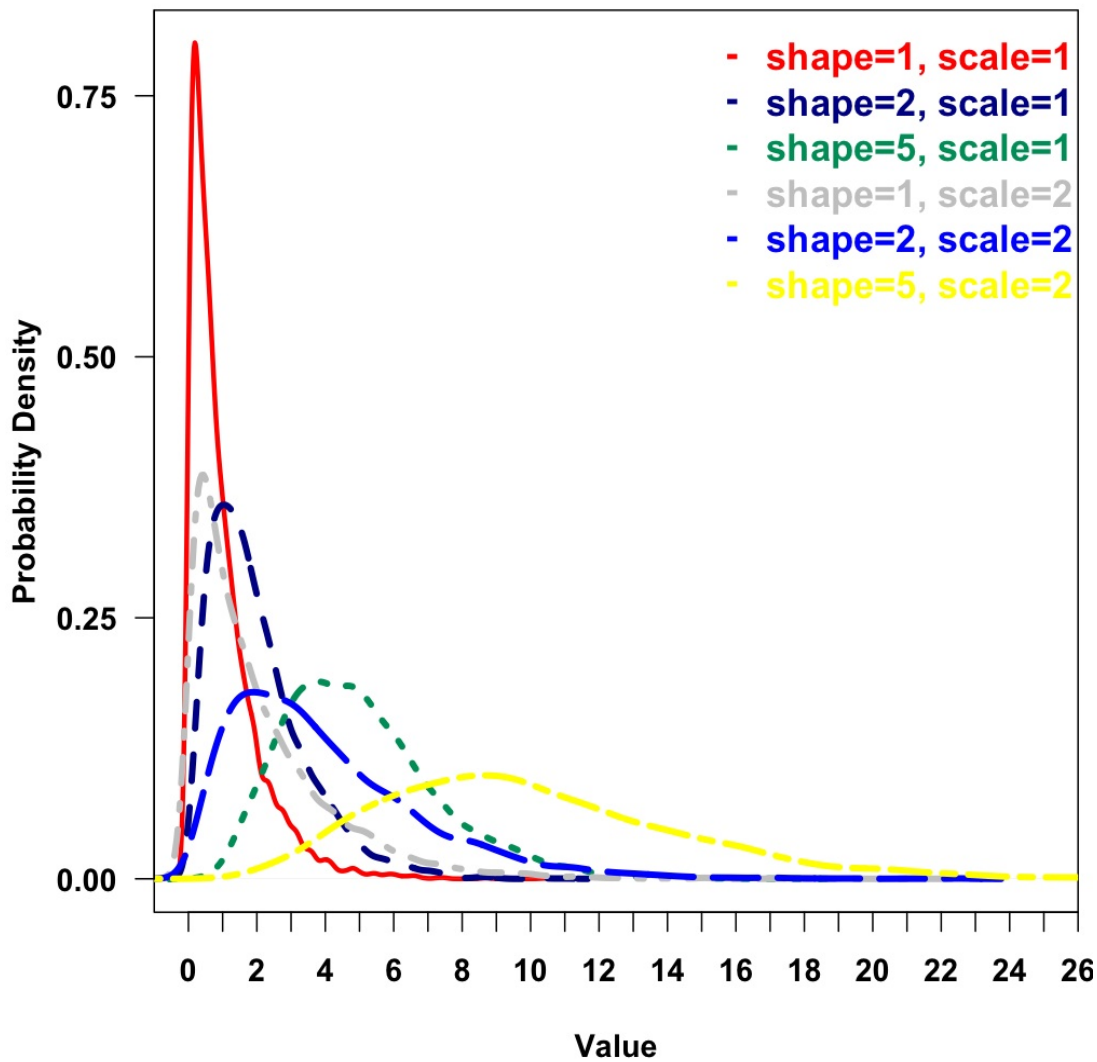


Figure 4.17 Relationships among probability distributions.

Gamma Distribution

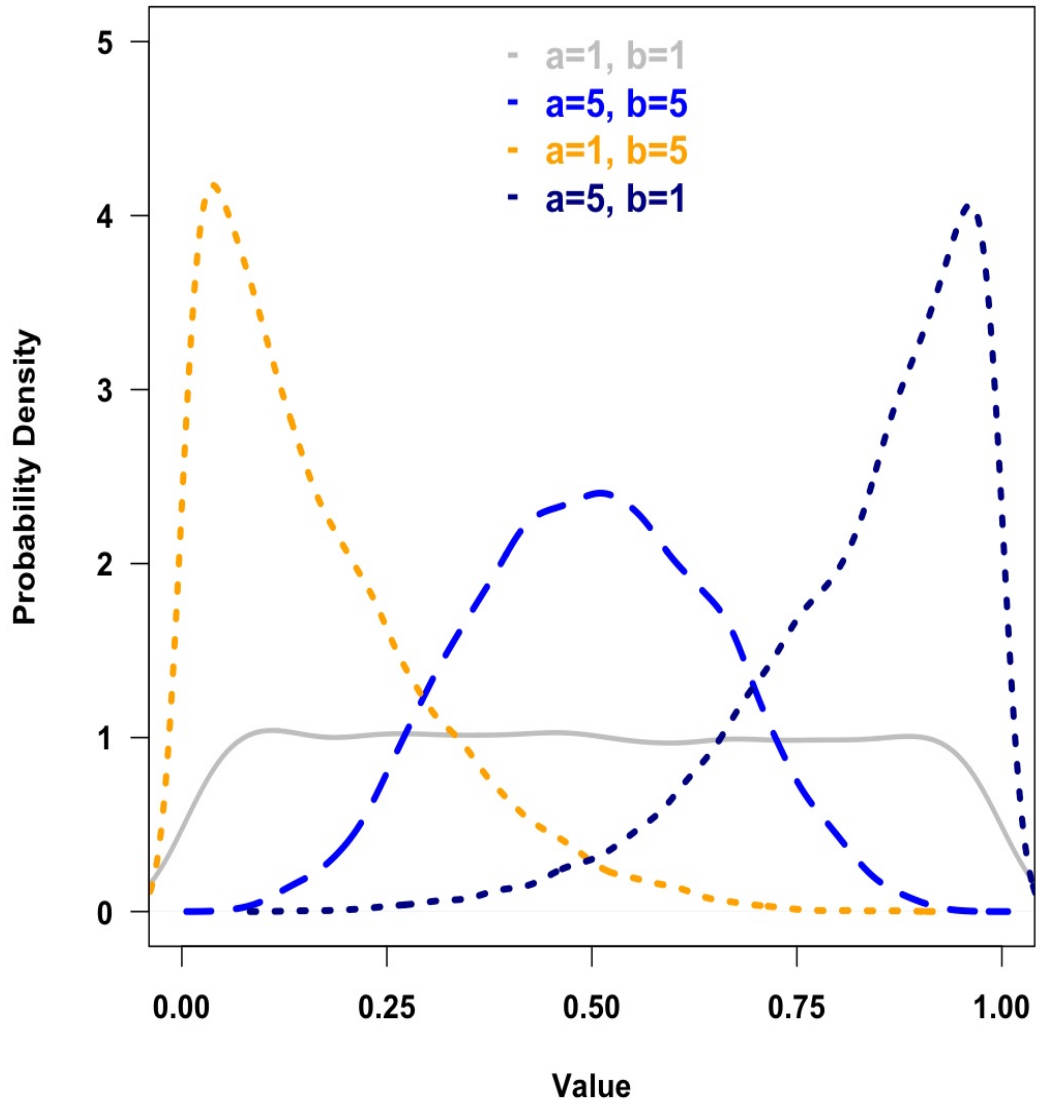


Gamma Distribution

range	positive real values
R	`dgamma`, `pgamma`, `qgamma`, `rgamma`
distribution	$\frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s}$
parameters	s (real, positive), scale: length per event [`scale`]
	or r (real, positive), rate $= 1/s$; rate at which events occur [`rate`]
	a (real, positive), shape: number of events [`shape`]
mean	as or a/r
variance	as^2 or a/r^2
CV	$1/\sqrt{a}$

Examples: almost any environmental variable with a large variance where negative values don't make sense: nitrogen concentrations, light intensity, etc..

Beta Distribution



Beta Distribution

Summary:

range	real, 0 to 1
R	`dbeta`, `pbeta`, `qbeta`, `rbeta`
density	$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$
parameters	a (real, positive), shape 1: number of successes $+1$ [`shape1`]
	b (real, positive), shape 2: number of failures $+1$ [`shape2`]
mean	$a/(a+b)$
mode	$(a-1)/(a+b-2)$
variance	$ab/((a+b)^2)(a+b+1)$
CV	$\sqrt{(b/a)/(a+b+1)}$

Strategies for compounding and generalizing distributions

In other words, what do you do when your data do not fit any of these simple distributions?

Strategy 1: Add covariates

Look for systematic differences within data that explain non-standard shape of distribution

Example: **bimodal** or **multimodal** distributions, which represent data that are a collection of objects from different populations with different means

Strategy 2: Make a mixture model

What if there aren't systematic differences?

Finite mixtures
- Assume that observations are drawn from discrete set of unobserved categories, each with own distribution
- All categories typically have same type of distribution, but with different means or variances
- These often fit multimodal data
- Use parameters of each component of the mixture + set of probabilities describing amount of each component

Zero-inflated models = common type
- Combine standard discrete probability distribution with some additional process that can lead to zero count

Finite mixture distribution

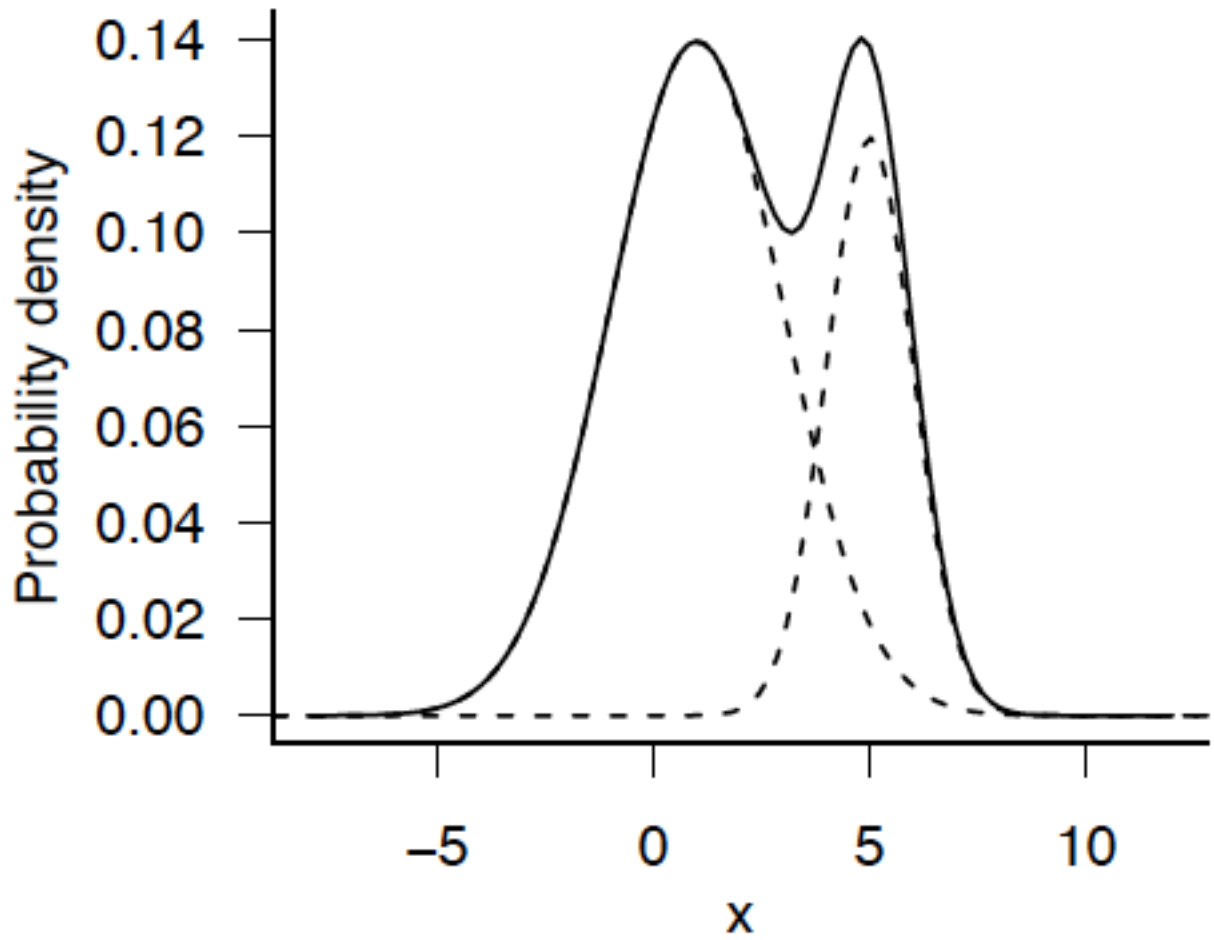
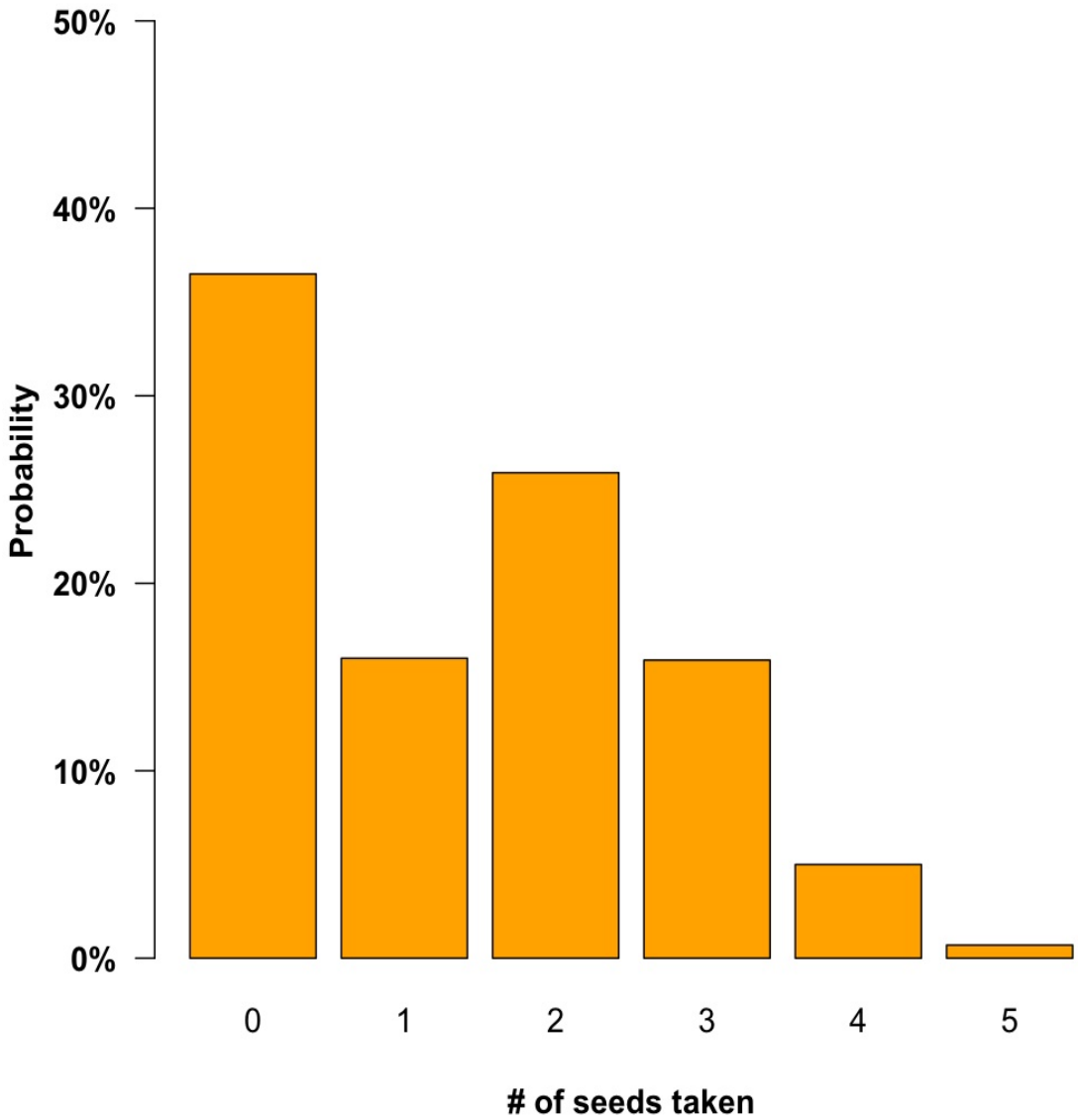


Figure 4.18 Finite mixture distribution: 70% Normal($\mu = 1, \sigma = 2$), 30% Normal($\mu = 5, \sigma = 1$).

Zero Inflated Binomial distribution



Problem 2

Simulate values from a normal distribution (or any one of the other distributions you prefer) as a function of the index values from the object you store the draws in.

For a matrix make the values in each cell a function of that cell's first and second dimension. This should be straightforward using 2 for loops.

Start with a vector and one for loop if you're still getting used to loops and the idea of nesting them.

Once your loops are working try to re-create the same matrix or vector without using loops or using one less loop if you are using a matrix.

If you are feeling really ambitious try this with a 3 dimensional array.

User Defined Functions

```
foo <- function(a, b)
{
    return (a+b)
}
```

If you are repeating a process over and over again it may be worthwhile to write your own function. They are also necessary for optimization problems using the optim() function (I have code for this too if you're interested).

bootstrapin.boogie()

This next function looks complicated but it combines functions that we've already seen earlier in the workshop.

And it has a cool name!

Fun with Regular Expressions

hopefully we haven't run out of time.

regular expression process and analyze text

dealing with text can be very computationally demanding which is why I am introducing these functions.

Problem 3

See if you can generate a binary document-by-term matrix using the corpus of 10 tweets readin above

hint: you will need to use a couple of functions that I have not covered, which are:

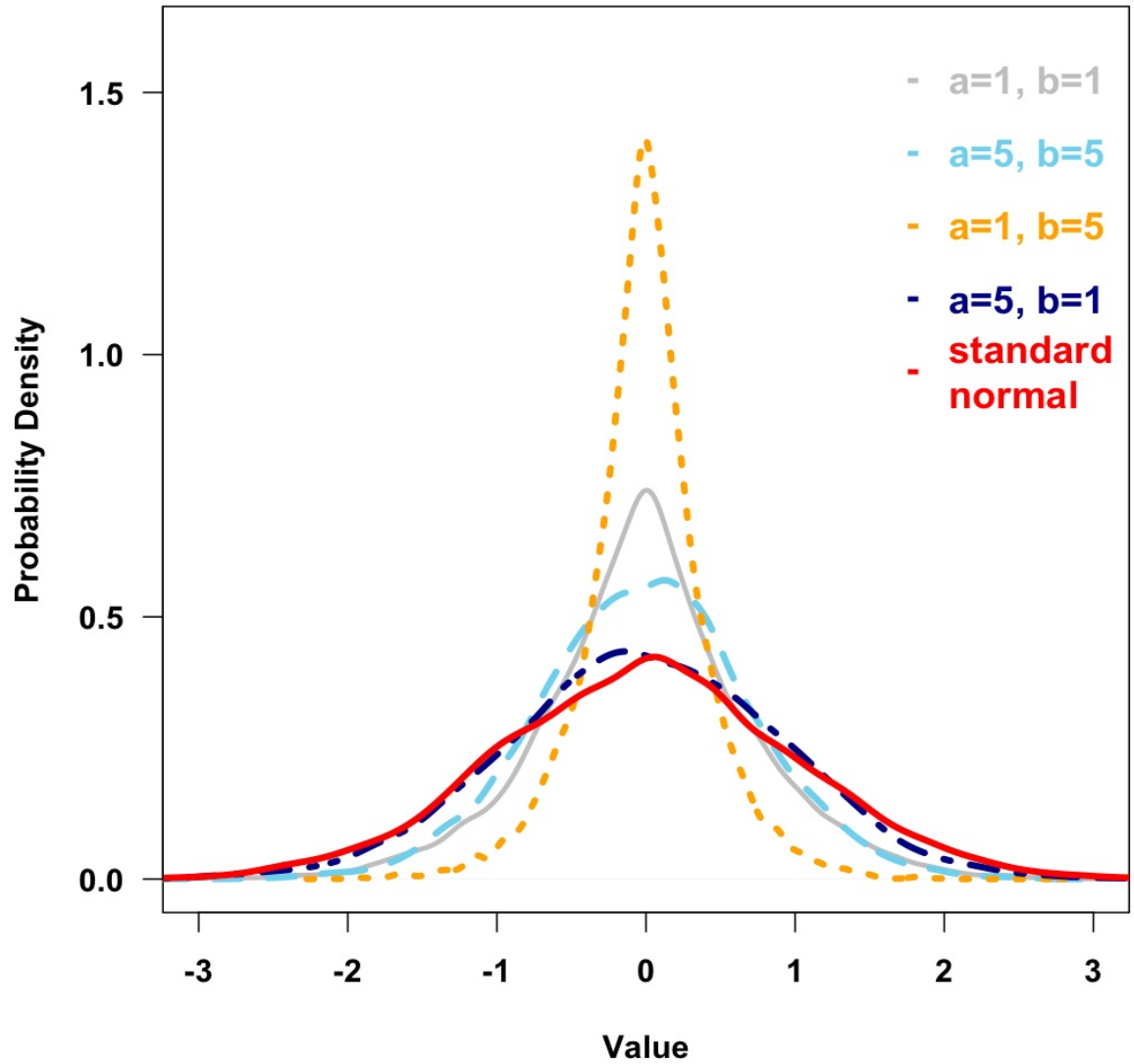
unqiue() and strsplit()

If there's still time ...

http://polisci2.ucsd.edu/cfariss/code/SIMols.R

http://polisci2.ucsd.edu/cfariss/code/SIMlogit.R

Normal distribution with beta distributed variance



Thanks!