Package animation: Animated Statistics Using R

Yihui Xie[1]

December 11, 2007

[1]School of Statistics, Renmin University of China, Beijing, 100872, China; Tel: 86-10-82509086; Fax: 86-10-82509086; Email: `paste("xieyihui", "@", "gmail.com", sep = "")`; Homepage: `http://www.yihui.name`; You may visit my project "Animated Statistics Using R" at `http://R.yihui.name` for a variety of animations in statistics in web pages.

Abstract

Animated pictures are undeniably both interesting and intuitional. This vignette mainly gives a brief overview to a large variety of animations in statstics, which could probably aid in teaching statistics, data analysis, and the presentation of statistical reports. The methods of making animations are also introduced. It is hoped that the traditional "static" statistics can be altered to some degree by such a kind of "animated" approach.

Contents

List of Figures

Chapter 1

Introduction

The concept of statistics, viewed from an analytical way, can be defined as "the study of algorithms for data analysis" ([1]). Nowadays statistical methods and models are increasing at an exploding speed, leading to more and more difficulties for people to understand those abstract mathematical and statistical algorithms. Basically this happens because sometimes it is really hard to imagine what on earth has happened behind a statistical algorithm, or how it works in processing data. While on the other hand, the size and complexity of data are also increasing, which results in the other problem for knowledge discovery. In the mean time, traditional *static* statistical reports (printed on paper in presses) for these complex data can be rather unsatisfactory for explainations of statistical results, and we need a more *active* way to present the fruits of our analyses.

> We're drowning in information and starving for knowledge.
>
> – Rutherford D. Rogers

To solve these problems, I adopted the approach of animation at last, because the human visual cortex is arguably the most powerful computing system we have access to, and visualization (especially animation) allows us to put information into a form which allows us to use the power of this computing system. Thus by virtue of our visual system we may be able to quickly understand a somewhat complicated method or result (usually in a simplified case).

However, there is currently very few work in such a literature for animations in statistics[1], therefore this vignette provides an integrated discussion on the animating of statistical models and data in the environment of R language ([5]).

[1]Most of work has been contributed to the computer science, media and entertainment industries.

Chapter 2

Tools for Animation

Perhaps most people would think of GIF images as the first choice for making animations, because it is well known that GIF is one of the only few image formats that has the ability to create animations[1], nevertheless ultimately I didn't adopt this format for several reasons[2].

Actually it is not so convenient to make animated image files in R, whereas we still have other choices, among which I list two main tools I have employed as follows in section 2.1 and 2.2, and other possible means are mentioned in section 2.3.

2.1 R graphical devices

The R package grDevices has offered a variety of graphics devices, and it's really a great help when we need to produce single image files – there are several choices such as PNG, JPEG, BMP, PDF, PS, TEX/LATEX and WMF, etc. All of them work very well when producing images files one by one, but the essential problem is that none of them is able to make animation files *directly*. At most what we can do is to produce a sequence of images.

Nevertheless, we may as well just use the Windows graphics devices (under Windows) or X Window System graphics devices (under Linux) or MacOS X Quartz devices (under MacOS X) inside R to make animations, i.e. draw graphs one after another in these devices. Again, there is an obvious drawback: it's inconvenient for users who don't have R installed in their computers to watch the animations, as the pictures are displayed *inside* R and we need an independent platform to show our animations.

For users who have installed R, animations can be made with the function *Sys.sleep()* in a loop. Obviously this function is intentionally used to slow down the loop so that we can see the whole process clearly. For example, we can show

[1]There are other formats such as APNG (Animated Portable Network Graphics), MNG (Multiple-image Network Graphics) and SVG (Scalable Vector Graphics), etc, but less popular (SVG might be promising in animations).

[2]You may read this page: `http://r.yihui.name/misc/gif_pdf_grDev.htm`

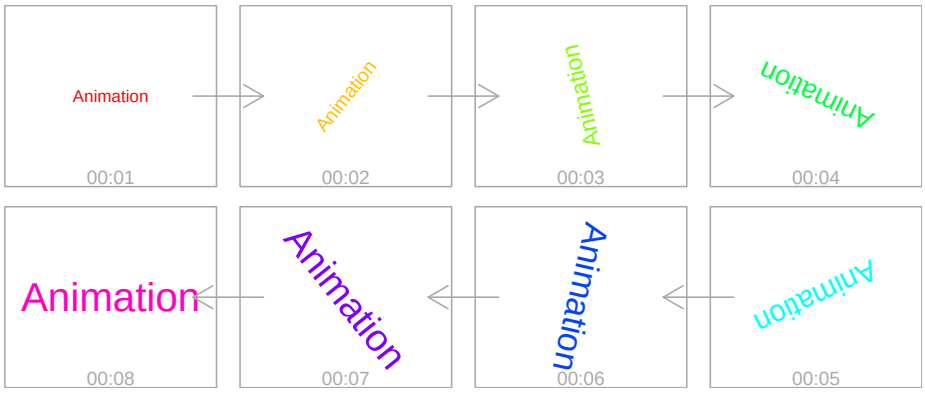

Figure 2.1: An illustration of the process of animations.

the process of rotating the word "Animation" in the loop below[3] (Figure 2.1 shows some sample frames of this animation):

```
> for (i in 1:360) {
+     plot(1, ann = F, type = "n", axes = F)
+     text(1, 1, "Animation", srt = i, col = rainbow(360)[i],
+         cex = 7 * i/360)
+     Sys.sleep(0.01)
+ }
```

For detailed instructions and explainations in R graphics system, you may refer to the book "R graphics" ([4]) by Paul Murrell, or read R-help on graphics functions (e.g. packages graphics, grDevices, etc) carefully.

2.2 HTML & JavaScript

Why use HTML & JavaScript? I believe there are at least three reasons:

- R already has built-in functions for reading and writing text files, so we can create HTML files easily, e.g. use *cat()* with arguments `file` and `append`;
- Although R has no devices for image formats such as GIF, there are still many other "static" image formats which can be well shown in *web pages*, e.g. JPEG, PNG, ...;
- Generally speaking, no additional programs are needed in order to display the animations, as long as the web browser supports JavaScript – surely most browsers can meet such a simple requirement.

[3] This code is used in the header of `http://R.yihui.name`. Argument `srt` controls the rotating degree, `col` for colors, and `cex` for magnification.

The work to do is just to create single image *frames* and try to show them in an HTML page. And how can we fulfill this? The answer is through JavaScript[4]. This idea has already been implemented in this package animation: all animation functions has a special argument `saveANI`, which determines whether to generate animation files or just to show animations inside R.

2.3 Other tools

Currently there are still at least other two choices for animations: the first one is to use the package rgl which takes advantage of the OpenGL system to make 3D visulizations. The user may conveniently interact with 3D elements in the plot (drag and rotate, etc). And the second one is Scalable Vector Graphics (SVG), which is a language for describing two-dimensional graphics and graphical applications in XML. SVG files are compact and provide high-quality graphics on the Web, in print, and on resource-limited handeld devices. In addition, SVG supports scripting and animation, so is ideal for interactive, data-driven, personalized graphics. Besides, SVG is a royalty-free vendor-neutral open standard developed under the W3C (World Wide Web Consortium) Process. Currently there are a few packages supporting the creation of SVG files, e.g. Cairo, cairoDevice, and RSvgDevice, etc[5].

[4]Refer to this page again for details: `http://r.yihui.name/misc/gif_pdf_grDev.htm`

[5]As far as I know, there is one other package gridSVG by Paul Murrell, but it is still highly experimental.

Chapter 3

Statistics and Animations

So what is the connection between statistics and animations? Generally there are three areas in which animations can be used, namely:

- algorithms involving iterations, e.g. K-Means cluster algorithm
- methods relevant to random numbers, e.g. simple random sampling
- statistics with dynamic trends, e.g. time series

Below I'll explain with examples how they work in animations respectively.

3.1 Iterative Algorithms

There are a large number of algorithms in statistics which involve iterations to optimize certain functions. For example, in the K-Means cluster analysis, the basic steps are described in Figure 3.1. What animation can do is to show the output of *each* iteration.

Here I use a cluster problem containing two numerical variables as a simplified case; actually the main reason is for the convenience of making scatterplots on the 2D plane: the x-axis and y-axis denote the two variabels respectively. During the process of iterations, we may present these two elements at each iteration:

- Location of each center: just average x-y locations within each cluster.
- Temporary cluster results: annotate each cluster by different point symbols.

As the iteration goes on, both the centers and cluster membership will change – this is just the source of animation. To sum up, the animation steps may be: show centers, compute distances and cluster, show cluster membership, re-compute centers and move, re-compute distances and change cluster membership, and so one and so forth. Figure 3.2 gives two iterations of locating centers and computing distances.

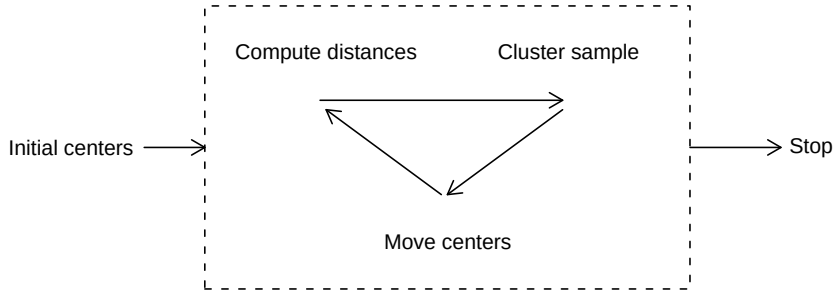

Figure 3.1: Basic steps of K-Means cluster algorithm: the iteration in the middle box will go on and on until maximum number of steps is achieved or clusters are converged.

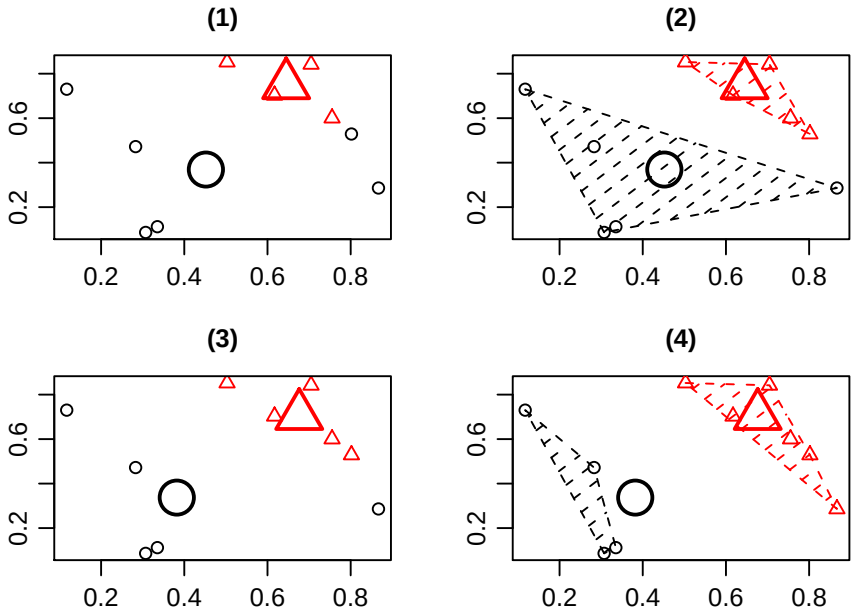

Figure 3.2: Sample iterations of K-Means cluster algorithm: (1) locate cluster centers based on the average of the last step; (2) compute distances and determine cluster membership again (centers are not moved!); (3) locate cluster centers again based on the result of (2); (4) compute distances and determine cluster membership again (centers not moved). Check carefully for the changes especially from (2) to (3) and from (3) to (4).

The animation function in the package animation is *kmeans.ani()*; see the help files for detailed usage.

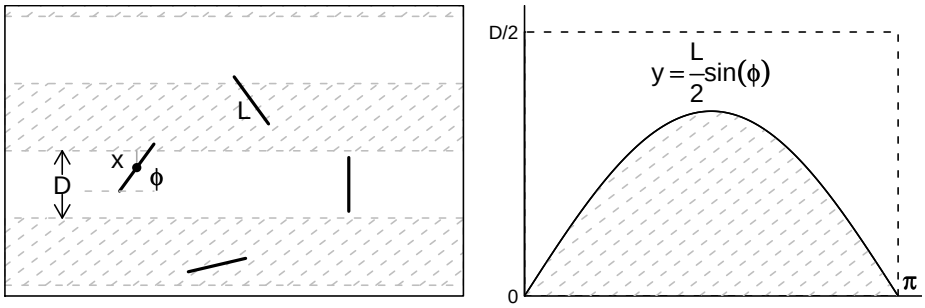

Figure 3.3: The problem of Buffon's Needle. (1) The left plot: bold segments stand for "needles"; D is the distance between lines; L is the length of needle; x is the distance from the middle of the needle to the nearest line; ϕ is the angle at which the needle falls; (2) The right plot: the needle will cross the lines if and only if $x \leq (L/2)\sin(\phi)$, i.e. the point (ϕ, x) should fall into the shadow area.

3.2 Random Numbers

Surely statistics cannot survive without randomness. We can see random numbers in subjects such as probability theory, survey sampling, and numerical simulation/optimization, etc. In areas which involve with random numbers, we may generate random numbers again and again to do simulations and get corresponding results – this is just where animations can play an important role.

Let's take the Buffon's Needle for example. This is one of the oldest problems in the field of geometrical probability and it's familiar to people who have basic knowledge of probability theory, so I wouldn't repeat the background here.

The critical parts for the simulation of this problem are:

- Randomly generate a location where the needle falls (only the middle point of the needle is needed).
- Randomly generate an angle ϕ at which the needle falls (a number in $[0, \pi]$).

After these two elements have been decided, we'll immediately know whether the needle will cross the lines or not. The problem and solution are explained in Figure 3.3, while the simulation is illustrated in Figure 3.4.

The animation function in the package animation is *buffon.needle()*; see the help files for detailed usage.

3.3 Dynamic Trends

We are always exploring relationships among our variables, as descriptions for single variables are far from enough in statistics; therefore the method of *conditioning* is fairly important. For instance, we may want to examine how a

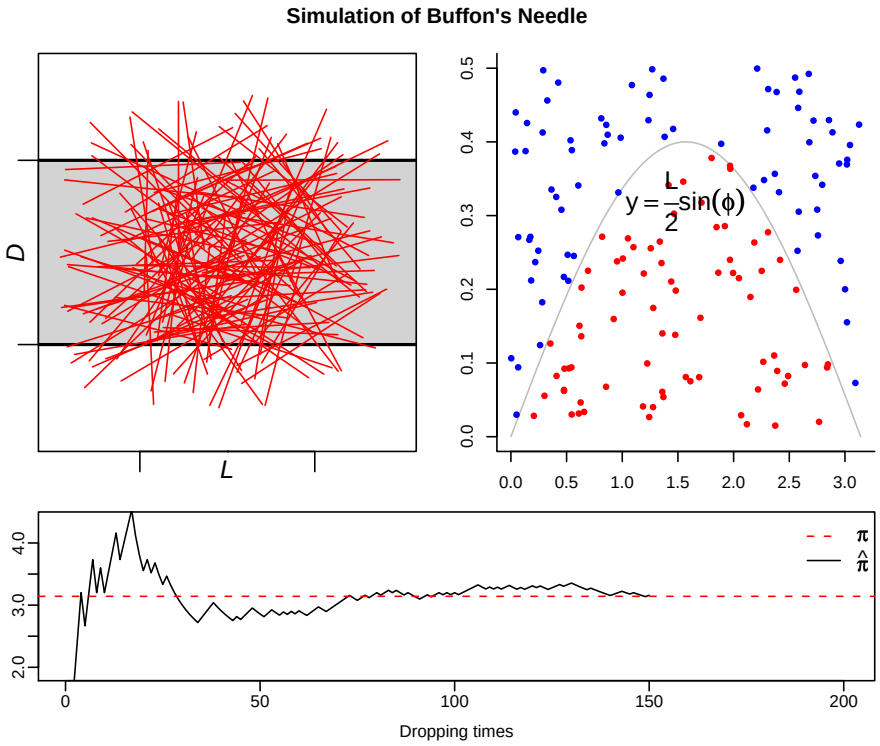

Figure 3.4: Simulation of Buffon's Needle. (1) Top left: simulation of dropping needles; (2) Top right: corresponding point pairs (ϕ, x); (3) Bottom: values of π calculated from the above simulations; actually this is the 150th animation frame taken from the whole process of 200 needle falls.

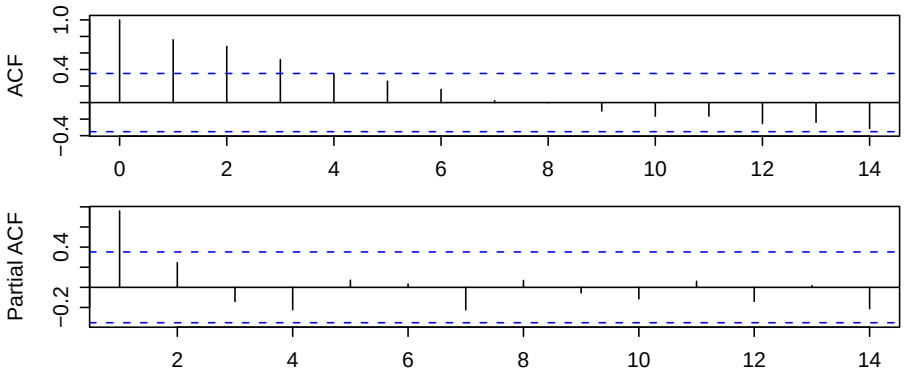

Figure 3.5: ACF and PACF for the number of visits to Yihui's website.

certain statistic A varies conditioned on a variable B. This "variation" (or simply "change") just builds the connection between statistics and the application of animations.

The most common case is the subject of time series[1], in which the conditioning variable is usually time.

For example, in time series analysis we often use the whole data set to fit an ARIMA model to examine the relationship between X_t and corresponding lagged terms such as $X_{t-1}, X_{t-2}, \ldots$, however, if we want to know how such a relationship varies over time, this single model surely cannot help. Here I simply employ an intuitional method called "*Moving Window Regression*" (MWR) to fulfill this idea. MWR is able to show the changes of coefficients over time, and what I do next is rather naive (just for demonstration). Further topics can be found in [6], etc.

The time series data is from the dataset `pageview` in animation, and we may have a look at the ACF and PACF plots in Figure 3.5 just for data between Oct 1 to 31, 2007.

```
> library(animation)
> data(pageview)
> x = pageview$visits[11:41]
> par(mfrow = c(2, 1))
> acf(x)
> pacf(x)
```

Not going further in the traditional ARIMA anlysis, I just use an AR(1) model for computation. Suppose there are n observations $\{x_1, x_2, \ldots, x_n\}$, and the MWR method is just to split the data into $n - k + 1$ subsets depending on the window width k: $\{x_1, \ldots, x_k\}$, $\{x_2, \ldots, x_{k+1}\}$, ..., $\{x_{n-k+1}, \ldots, x_n\}$, and at last compute AR(1) models on each subset. In the code below, I computed the coefficient ϕ for $n - k + 1$ AR(1) models $x_t = \phi x_{t-1} + \epsilon_t$ using *arima()* in package stats, and during the moving (in a loop), I marked out observations used in MVR by rectangles with different colors, and plotted the corresponding coefficients as well as "$\phi \pm 2$*s.e." in the lower part of the graph. The last line `Sys.sleep(1)` is to slow down the process of moving windows so that we can clearly see the "real moving"[2].

```
> library(animation)
> data(pageview)
> x = pageview$visits[11:41]
> k = 15
> base = 0:(k - 1)
> sx = 2.5 * (x - min(x))/(max(x) - min(x)) + 1.6
> plot(sx, ylim = c(-0.3, 4.2), cex = 1.5, yaxt = "n")
> axis(2, c(0, 0.6, 1.2), col.axis = "red")
> axis(2, seq(1.6, 4.1, length = 4), seq(min(x), max(x),
```

[1] But the applications are absolutely not limited to time series!

[2] Just for demonstration; in practical applications there's no need to slow down the computation.

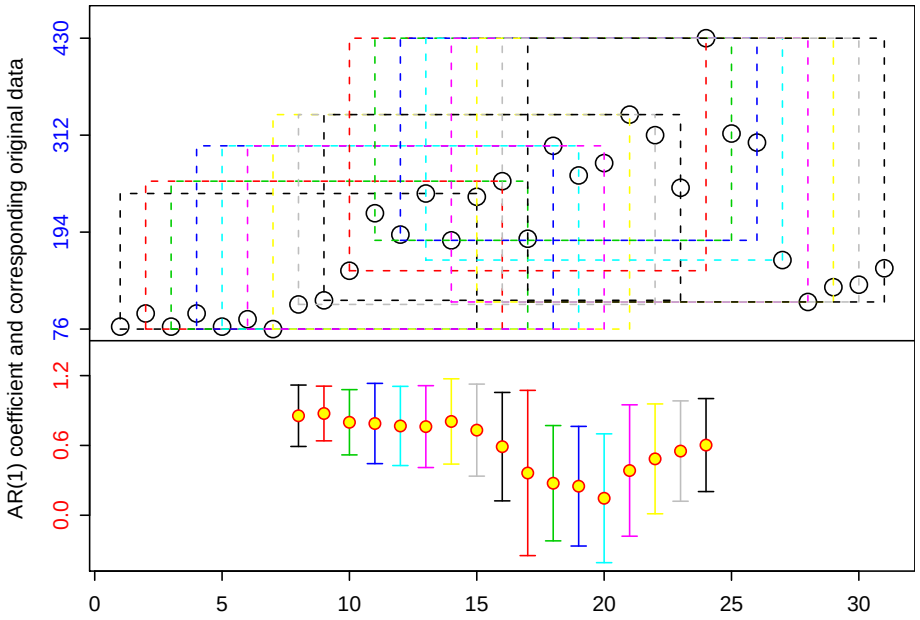

Figure 3.6: Illustration for Moving Window Auto-Regression.

```
+     length = 4), col.axis = "blue")
> abline(h = 1.5)
> for (i in 1:(length(x) - k + 1)) {
+     idx = base + i
+     m = arima(x[idx], order = c(1, 0, 0))
+     phi = coef(m)["ar1"]
+     se = sqrt(vcov(m)[1, 1])
+     rect(i, min(sx[idx]), i + k - 1, max(sx[idx]), lty = 2,
+         border = i)
+     arrows(i + k/2 - 0.5, phi - 2 * se, i + k/2 - 0.5,
+         phi + 2 * se, angle = 90, code = 3, length = 0.05,
+         col = i)
+     points(i + k/2 - 0.5, phi, pch = 21, col = "red",
+         bg = "yellow")
+     Sys.sleep(1)
+ }
```

Figure 3.6 shows the eventual result; we can roughly observe that the AR(1) coefficient ϕ is stable first, and begins to decrease in the 15 days centered at Oct 15, then increases from about Oct 13 (those 15 days are centered at Oct 20).

A more general animation function for MVR is still in my TODO list.

Chapter 4

Package **animation** Overview

The package animation is based on the most primitive idea of animation: make picture *frames* one after another with a certain duration (time interval between frames) specified. And this dull method has also been implemented in an HTML animation page using JavaScript to animate the image frames.

Currently there are two ways for animation: one is just to show animations in a graphical device (Windows, X Window, etc), and the other is to make animations in an HTML page so that people without R installed are also able to view the animations. There is a common argument `saveANI` in each animation function controlling the way to make animations:

`saveANI = TRUE` Convert the animation frames into PNG files, which will be used in the HTML animation page.

`saveANI = FALSE` Don't generate animation files: just show animation inside R.

To make an HTML animation page, you have to start a page first by *ani.start()*, then use any animation functions to generate PNG files in the `images` directory relative the HTML page, and at last use *ani.stop()* to complete writing the page. By default, *ani.stop()* will automatically open a web browser to view the HTML animation page[1].

Here is a sample session:

```
> ani.start()
> brownian.motion(saveANI = TRUE)
> ani.stop()
```

There a plenty of examples in the help pages of each animation functions. Just try them if you like.

Having provided a mechanism for generating animations, next I shall go into the huge project of statistical animations in the many branches.

[1] Use the function *browseURL()* in utils

Chapter 5

Statistical Animations Gallery

In section 3 I have explained some basic connections between animation and the discipline of statistics. In this section I just give a summary of the animation functions in the package animation. This gallery will be supplemented day by day.

5.1 Probability Theory

Probabbility theory is a subject relevant to randomness. As mentioned in section 3.2, animation can be closely related to this subject.

5.1.1 Probability in Flipping Coins

In the first class of learning probability we usually begin with the probability in flipping coins or tossing dice, and the function *flip.coin()* gives a simple simulation. Here the concept of a "coin" is acutally abstract: it can be anyting, and you just have to specify the true probabilities (or take the default `NULL`) for each "face".

Usage

```
flip.coin(saveANI = FALSE, faces = 2, prob = NULL, interval = 0.2,
    nmax = 100, border = "white", cl = 1:2, ...)
```

For example, we toss the coin for 50 times, and considering that sometimes the result of flipping a coin is neither head or tail (the coin just stands on the table!), we specify there are three possible results `Head, Tail, Stand` with probabilities `0.45, 0.1, 0.45` respectively, as the result `Stand` is not likely to happen. Figure 5.1 shows the result of flipping this coin.

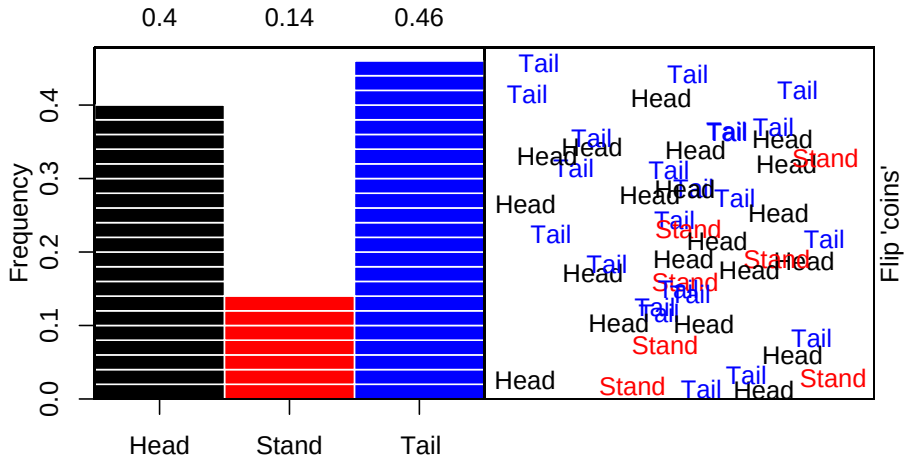

Figure 5.1: Probability of flipping a coin: Head? Tail? Or just stand on the table? Run in R to watch the animation.

```
> flip.coin(faces = c("Head", "Stand", "Tail"), interval = 0.2,
+     nmax = 50, type = "n", prob = c(0.45, 0.1, 0.45),
+     cl = c(1, 2, 4))
```

You may set larger times of flipping `nmax` to check whether the frequencies will approximate to the true probabilities.

5.1.2 Buffon's Needle

This problem has been mentioned in section 3.2, so I will not repeat again here.

Usage

```
buffon.needle(saveANI = FALSE, l = 0.8, d = 1, interval = 0.05,
    nmax = 100, redraw = TRUE, ...)
```

5.1.3 Brownian Motion

Brownian Motion, a.k.a "random walk", characterizes the trace of a point moving in a line or a plane (or in higher dimensions). Suppose the current location of the point is x_t, then the next location will be $x_{t+1} = x_t + \epsilon_{t+1}$ with i.i.d $\epsilon_t \sim N(\mu, \sigma^2)$.

It is very easy to simulate this process in R. If the initial location is 0, the next k locations can be computed simply by `cumsum(rnorm(k))`. The function *rnorm()* generates k i.i.d random numbers following Normal distribution, and *cumsum()* computes cumulative sums for these numbers, which is essentially the moving process of Brownian Motion.

The function *brownian.motion()* in animation has provided a simulation for Brownian Motion with animations.

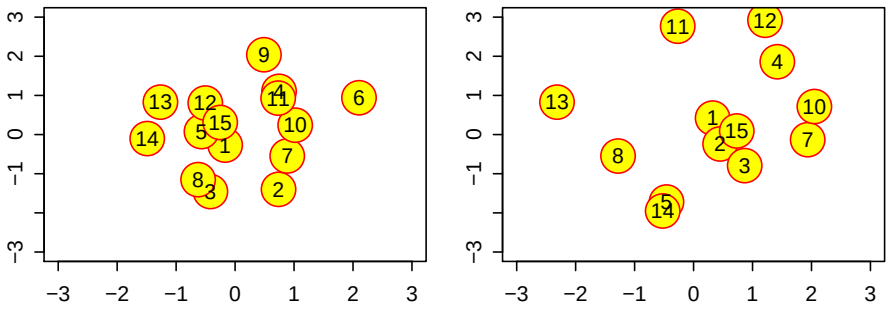

Figure 5.2: Two sample frames of Brownian Motion.

Usage

```
brownian.motion(saveANI = FALSE, n = 10, interval = 0.05,
    nmax = 100, main = "Demonstration of Brownian Motion",
    xlim = c(-20, 20), ylim = c(-20, 20), pch = 21, cex = 5,
    col = "red", bg = "yellow", ...)
```

For example, the code below shows the traces of 15 points moving in the 2D plane for 100 steps. Figure 5.2 shows two sample frames of the animation.

```
> brownian.motion(n = 15, nmax = 100)
```

5.1.4 Law of Large Numbers

TODO...

From frequency to probability.

5.1.5 Monte-Carlo Simulation for Computing Areas

TODO...

Monte-Carlo integration.

5.1.6 Central Limit Theorem

TODO...

Limit distribution of the sample mean $\bar{x}$.

5.2 Sampling Survey

Sampling survey is also a subject based on random numbers: the process of sampling is essentially generating random numbers for *indexing* the sampling frame. Therefore the problem behind sampling is just the manner to generate random numbers.

5.2.1 Simple Random Sampling

Simple Random Sampling is the purest form of probability sampling. Each member of the population has an equal and known chance of being selected. When there are very large populations, it is often difficult or impossible to identify every member of the population, so the pool of available subjects becomes biased.

In most cases, we conduct the sampling in a "*without-replacement*" manner, i.e. we don't put back the sample points once we pick them out. Correspondingly there is another way "sampling *with* replacement": every time before we do the sampling, we put all the individuals back again; although this is rare in practical sampling work, it's extremely important and closely related to the idea of bootstrapping (see section 5.9.1).

Here we only discuss the case of "Simple Random Sampling Without Replacement" (SRSWOR). The function *sample()* is convenient for us to conduct the sampling.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

To randomly sample 10 individuals from a population of 100 elements, `sample(100, 10)` is enough for indexing. Actually the other kinds of sampling are also based on this useful function.

If we keep on sampling from a population, the samples will also change randomly, so this is the base of animations.

`http://r.yihui.name/stat/sampling_survey/simple_random/index.htm`

5.2.2 Stratified Sampling

Stratified Sampling is commonly used probability method that is superior to random sampling because it reduces sampling error. A stratum is a subset of the population that share at least one common characteristic. Examples of stratums might be males and females, or managers and non-managers. The researcher first identifies the relevant stratums and their actual representation in the population. Random sampling is then used to select a sufficient number of subjects from each stratum. "Sufficient" refers to a sample size large enough for us to be reasonably confident that the stratum represents the population. Stratified sampling is often used when one or more of the stratums in the population have a low incidence relative to the other stratums.

`http://r.yihui.name/stat/sampling_survey/stratified/index.htm`

5.2.3 Cluster Sampling

Sometimes it is cheaper to "cluster" the sample in some way e.g. by selecting respondents from certain areas only, or certain time-periods only. (Nearly all

samples are in some sense "clustered" in time – although this is rarely taken into account in the analysis.)

`http://r.yihui.name/stat/sampling_survey/cluster/index.htm`

5.2.4 Systematic Sampling

Systematic Sampling is often used instead of random sampling. It is also called an Nth name selection technique. After the required sample size has been calculated, every Nth record is selected from a list of population members. As long as the list does not contain any hidden order, this sampling method is as good as the random sampling method. Its only advantage over the random sampling technique is simplicity. Systematic sampling is frequently used to select a specified number of records from a computer file.

`http://r.yihui.name/stat/sampling_survey/systematic/index.htm`

5.2.5 CLT in Sampling Survey

TODO...

Central Limit Theorem in sampling survey for estimation and inference.

5.3 Mathematical Statistics

5.3.1 Confidence Intervals

5.4 Linear Models

5.4.1 Subset Selection

5.5 Multivariate Statistics

5.5.1 K-Means Cluster Analysis

This algorithm has already been discussed in section 3.1.

Usage

```
kmeans.ani(saveANI = FALSE, x, centers = 2, interval = 2,
    nmax = 30)
```

You may try several examples for yourself.

```
> x = matrix(runif(100), ncol = 2)
> kmeans.ani(saveANI = FALSE, x, centers = 2, interval = 1)
> x = matrix(runif(300), ncol = 2)
> kmeans.ani(FALSE, x, 6, 0.5)
```

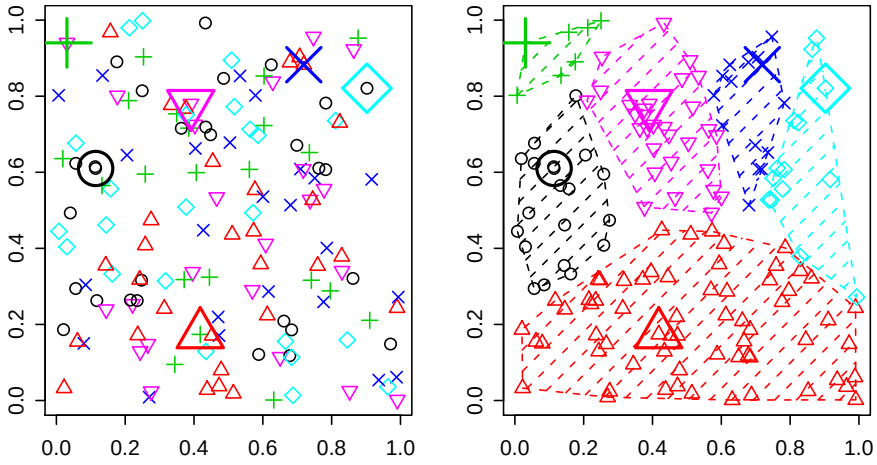

Figure 5.3: The first iteration for K-Means cluster analysis.

Figure 5.3 shows the first iteration in the K-Means algorithm: random centers are selected in the left plot, then distances are computed to determine the cluster membership; next we shall calculate the cluster centers again and repeat the steps till the maximum number of iterations is reached or the cluster membership is stable.

5.6 Nonparametric Statsitics

5.6.1 Kernal Density Estimation

5.7 Computational Statistics

5.7.1 Gradient Descent Algorithm

5.8 Data Mining

5.9 Machine Learning

5.9.1 Bootstrapping

What I will introduce here is rather superficial; for further knowledge about bootstrapping, please refer to [2] for theories, and [7] can also be a simple guide to implementations in S language.

The two critical points for bootstrapping are: (1) data generating mechanism; (2) plug-in principle. The first tells us how to re-generate data from a sample, while the latter point tells us how to make estimations. The idea of bootstrapping is based on the method of resampling to a large degree. In the

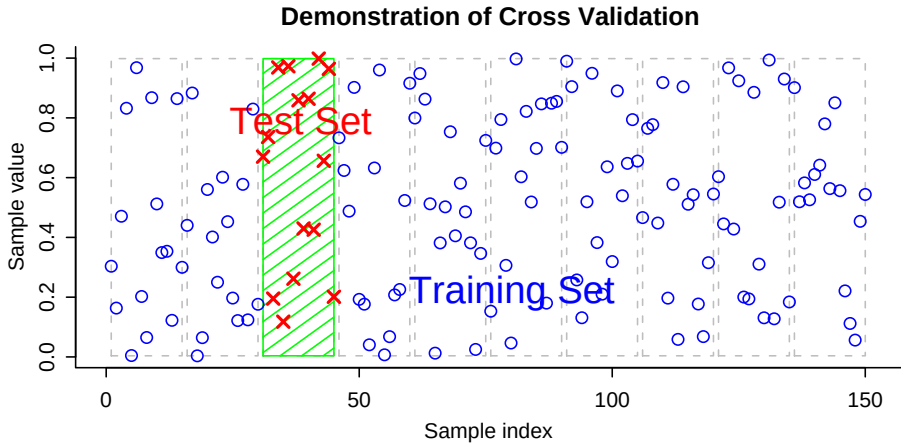

Figure 5.4: An illustration of 10-fold cross-validation.

real world, we only have one sample, say, n sample points $x_1, x_2, \ldots, x_n$, then the problems we must face (when making inferences) are:

- How to guarantee the population distribution which we have assumed is correct?
- How to derive the expression of the point estimate or confidence interval of a parameter if the population distribution is tooooooo complicated?
- Or how can we obtain the distribution of a statistic when the population distribution is complicated?

We have always been deriving mathematical formulae... for this statistic... for that statistic... under perfect but unwarranted assumptions...

Why not re-generate some samples (resample the original sample with replacement) and re-compute the values of our statistic of interest? Then we can get a series of estimations of a certain parameter and in a result, we are able to make inferences based on these numbers using the plug-in principle, e.g. we may compute the standard error of a parameter by compute the corresponding standard error of that series of numbers (please do note the factual computation is not exactly so; read the references to learn the details), and estimate the quantiles of a statistic just by computing the corresponding quantiles of that series of numbers, etc. If you are confused by my description here, just keep on to the below animation example.

Function *boot.iid()* is available now.

`http://r.yihui.name/stat/machine_learning/bootstrapping/index.htm`.

5.9.2 k-fold / Leave-one-out Cross-validation

Cross-validation, sometimes called rotation estimation ([3]), is the statistical practice of partitioning a sample of data into subsets such that the analysis is

initially performed on a single subset, while the other subset(s) are retained for subsequent use in confirming and validating the initial analysis.

The initial subset of data is called the *training set*; the other subset(s) are called *validation sets* or *testing sets*.

The theory of cross-validation was inaugurated by Seymour Geisser. It is important in guarding against testing hypotheses suggested by the data ("Type III error"), especially where further samples are hazardous, costly or impossible (uncomfortable science) to collect.

The function *cv.ani()* provides an illustration for k-fold cross-validation. Computation of the sizes of subsets is based on the function *kfcv()*. When `k` is specified as `length(x)`, the k-fold cross-validation will become "leave-one-out cross-validation".

Usage

```
cv.ani(saveANI = FALSE, x = runif(150), k = 10, interval = 2,
    nmax = 50)
```

Figure 5.4 shows a possible partition of the whole data set into a training set and test set (10-fold cross-validation). The test test can move from the first part to the last part, and this is the base for animations.

5.9.3 *k*-Nearest Neighbor Classification

The k-nearest neighbor algorithm is amongst the simplest of all machine learning algorithms. It is a supervised learning algorithm where the result of new instance query is classified based on majority of k-nearest neighbor category. The purpose of this algorithm is to classify a new object based on attributes and training samples. The classifiers do not use any model to fit and only based on memory. Given a query point, we find K number of objects or (training points) closest to the query point. The classification is using majority vote among the classification of the K objects. Any ties can be broken at random. K-nearest neighbor algorithm used neighborhood classification as the prediction value of the new query instance.

The function *knn.ani()* provides the animated demonstration for kNN algorithm in the 2D case.

Usage

```
knn.ani(saveANI = FALSE, train, test, cl, k = 1, interval = 1,
    nmax = 100, interact = FALSE)
```

You may either provide a test set or specify `interact = TRUE` so that you can simply use mouse-click to decide the test set. Figure 5.5 is the eventual result of the whole classification process.

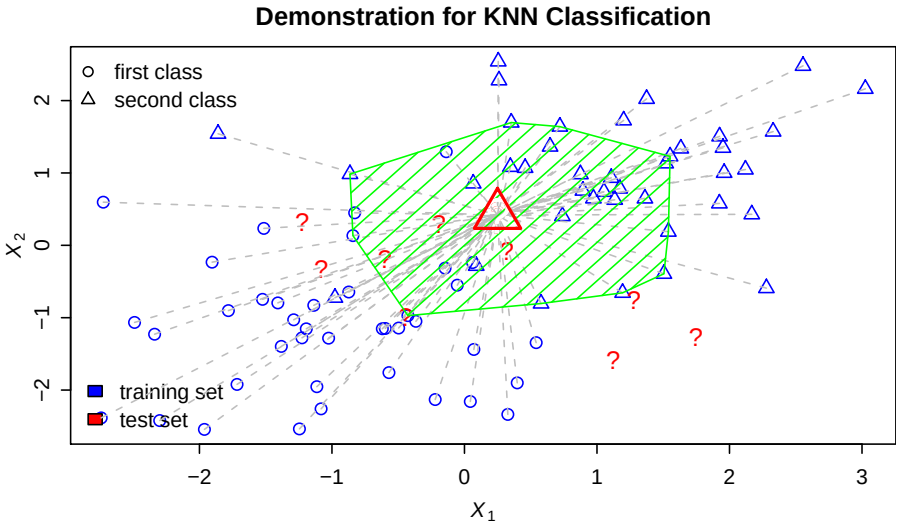

Figure 5.5: kNN algorithm in the 2D plane: gray dashed lines stand for "distances" so that neighbors can be decided; green polygon means the k nearest neighbors; at last let these neighbors vote for the classification, and the symbol (classification) was changed according to the majority vote.

Acknowledgements

I'm grateful to Dr Paul Murrell for his instructions and suggestions on the initial idea of my animation package.

Appendix A

Introduction to R

Appendix B

R Graphics

Appendix C

Misc Functions in **animation**

C.1 Functions for R

tidy up source code: *tidy.source()*

generate R definition file for the software Highlight *highlight.def()*

C.2 Functions for Systems

rename a sequence of files *rename.seq()*

C.3 Functions for Web (HTML/XML/RSS)

create RSS feed *write.rss()*

Bibliography

[1] Rudolf Beran. The impact of the bootstrap on statistical algorithms and theory. *Statistical Science*, 18(2):175–184, 2003.

[2] Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1994.

[3] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 2(12):1137–1143, 1995.

[4] Paul Murrell. *R Graphics*. Chapman & Hall/CRC, 2005.

[5] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[6] Robert A. Meyer, Jr. Estimating coefficients that change over time. *International Economic Review*, 13(3):705–710, 1972.

[7] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, 4th edition, 2002.