Homework 5

Basic random number generation

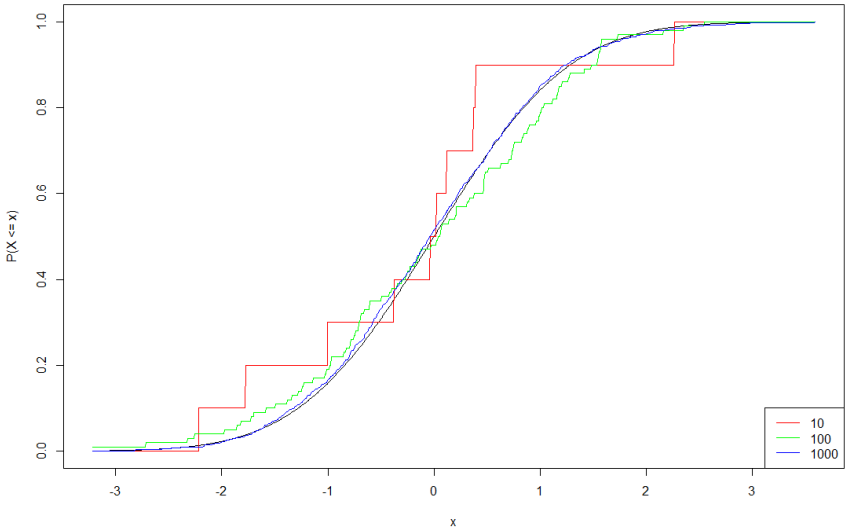
- **1a.** Generate the following objects, save them to variables (with names of your choosing), and call `head()` on those variables.
  - A vector with 1000 standard normal random variables.
  - A vector with 20 draws from Beta$(0.1, 0.1)$.
  - A vector of 2000 characters sampled uniformly from "A", "G", "C", and "T".
  - A data frame with a column `x` that contains 100 draws from Unif$(0, 1)$, and a column `y` that contains 100 draws of the form $y_i \sim \text{Unif}(0, x_i)$. Do this without using explicit iteration.
- **1b.** We've written a function `plot.cum.means()` below which plots cumulative sample mean as the sample size increases. The first argument `rfun` stands for a function which takes one argument `n` and generates this many random numbers when called as `rfun(n)`. The second argument `n.max` is an integer which tells the number samples to draw. As a side effect, the function plots the cumulative mean against the number of samples.

```
# plot.cum.means: plot cumulative sample mean as a function of sample size
# Inputs:
# - rfun: function which generates random draws
# - n.max: number of samples to draw
# Ouptut: none
plot.cum.means = function(rfun, n.max) {
  samples = rfun(n.max)
  plot(1:n.max, cumsum(samples) / 1:n.max, type = "l")
}
```

  Modify this function to include the paprameters in generating random samples, such as the arguments `mean =`, `sd =` in the `rnorm` function. Make plots for the following distributions, with `n.max=1000`. Then answer: do the sample means start concentrating around the appropriate value as the sample size increases?
  - $N(-3, 10)$
  - Exp(mean $= 5$)
  - Beta$(1, 1)$
- **1c** Find a distribution whose sample mean should not converge (in theory) as the sample size grows. Call `plot.cum.means()` with the appropriate random number generator and `n.max=1000`.
- **1d.** For the same distributions as Q1b, Run your own `plot.ecdf` function to do the following:
  - Generate 10, 100, and 1000 random samples from the distribution.
  - On a single plot, display the ECDFs (empirical cumulative distribution functions) from each set of samples, and the true CDF, with each curve being displayed in a different color.

  The function `plot.ecdf(rfun, pfun, sizes,...)` takes as its arguments the single-argument random number generating function `rfun`, the corresponding single-argument conditional density function `pfun`, a vector of sample sizes `sizes` for which to plot the ecdf, and other parameters needed to be transfered to `rfun` and `pfun`.

  Example of usage: `plot.ecdf(rnorm, pnorm, c(10,100,1000))` should return the following figure:



Acceptance-rejection and multivariate normal distribution

- **2a.** Use Acceptance-rejection method to generate samples of size 100 from beta distribution.
- **2b.** Compare its ecdf with the theoretical disitribution function.
- **2c.** Compare your samples with those generated by using `rbeta` function in `qqplot`.
- **2d.** Use Acceptance-rejection method to genrate samples of size 100 from gamma distribution by using the exponential density $g(x) = \lambda e^{-\lambda x}$, where $1/\lambda$ is the mean of the gamma distribution. Answer the same questions in Q2b-Q2c.
- **2e.** Generate a sample from a multivariate normal density $N(\mu, \Sigma)$ using SVD, calculate the sample mean and the sample covariance matrix, and compare them with μ and Σ. Check how your estimates change as the sample size changes. Example of usage: generate a sample of size 100 from $N(\mu, \Sigma)$ with $\mu = (0, 3)$ and $\Sigma_{1,1} = \Sigma_{1,1} = 1$, $\Sigma_{1,2} = \Sigma_{2,1} = 0.5$.

Root finding and optimization basic

- **3a.** Solve the following equation

$$g(y) = (x-1)^3 - 2x^2 + 10 - \sin(x)$$

  with bisection and Secant method. Start wutg $x_1 = 1$ and $x_2 = 2$.

- **3b.** Optimize the following univariate function:

$$f(x) = 4x^2 e^{-2x}$$

  with gradient descent and Newton's method. Compare the iterations and accuracy of solution with different step sizes in gradient descent method.

Optimization for logistic regression

For the logistic regression model,

$$p_i = P(Y_i = 1) = \frac{\exp(\beta x_i)}{1 + \exp(\beta x_i)}, i = 1, \ldots, n$$

where Y_i is a binary response and x_i is a predictor. Then the **log likelihood** is

$$l(\beta) = \sum_{i=1}^{n} Y_i \beta x_i - \sum_{i=1}^{n} \log(1 + \exp(\beta x_i)).$$

The Hessian can be obtained by direct differentiation:

$$H = \sum_{i=1}^{n} p_i(1 - p_i)x_i^2.$$

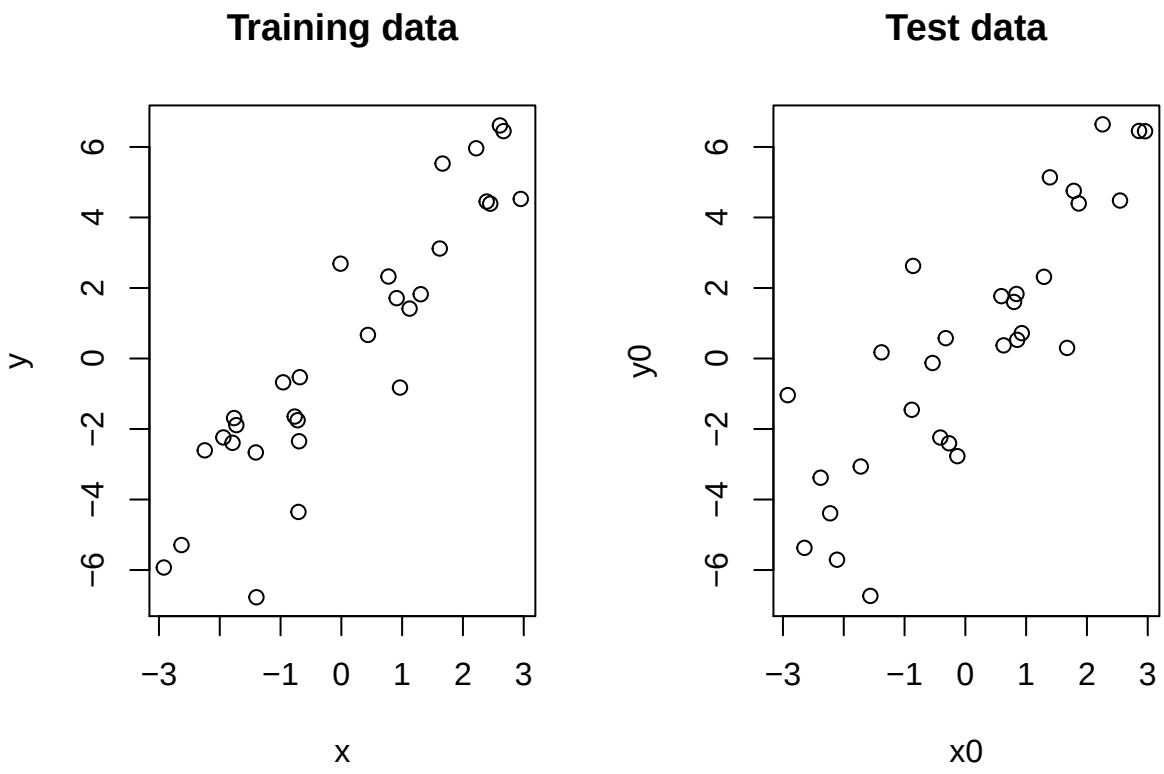
- **4a.** Outline the Newton algorithm and the Fisher Scoring algorithm. Show the relation between Newton and Fisher Scoring.
- **4b.** Implement the Newton algorithm in `R`.
- **4c.** Implement the Fisher Scoring algorithm in `R`.
- **4d.** Conduct a simulation study. For 1,000 individuals, generate the binary response from logistic regression model with $\beta = 0.3$ and $x_i \sim N(0, 1)$. Apply your Newton and Fisher scoring algorithms to this data set; select your own starting value and stopping criterion. Then report MLE $\hat{\beta}$ and number of iterations.
- **4e.** Redo Q4b with β being a 2-dimensional vector. Then generate the binary response from logistic regression model with $\beta = c(0.3, 0.1)$ and $x_i \sim N(\mu, \Sigma)$, where $\mu = (0, 0)$ and $\Sigma_{1,1} = \Sigma_{1,1} = 1$, $\Sigma_{1,2} = \Sigma_{2,1} = 0.5$. Apply your Newton algorithm to this data set and report MLE $\hat{\beta}$ and number of iterations.

Practice with training and test errors

The code below generates and plots training and test data from a simple univariate linear model, as in lecture. (You don't need to do anything yet.)

```
set.seed(1)
n = 30
x = sort(runif(n, -3, 3))
y = 2*x + 2*rnorm(n)
x0 = sort(runif(n, -3, 3))
y0 = 2*x0 + 2*rnorm(n)

par(mfrow=c(1,2))
xlim = range(c(x,x0)); ylim = range(c(y,y0))
plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
```



- **5a.** For every k in between 1 and 15, regress `y` onto a polynomial in `x` of degree k, and record the training and test error. Plot the test error and training errors curves, as functions of k, on the same plot, with properly labeled axes, and an informative legend. What do you notice about the relative magnitudes of the training and test errors? What do you notice about the shapes of the two curves? If you were going to select a regression model based on training error, which would you choose? Based on test error, which would you choose?
- **5b.** Without any programmatic implementation, answer: what would happen to the training error in the current example if we let the polynomial degree be as large as 29?
- **5c.** Modify the above code for the generating current example data so that the underlying trend between `y` and `x`, and `y0` and `x0`, is cubic (with a reasonable amount of added noise). Recompute training and test errors from regressions of `y` onto polynomials in `x` of degrees 1 up to 15. Answer the same questions as before, and notably: if you were going to select a regression model based on training error, which would you choose? Based on test error, which would you choose?

Sample-splitting with the prostate cancer data

Below, we read in data on 97 men who have prostate cancer (from the book The Elements of Statistical Learning). (You don't need to do anything yet.)

```
pros.df = read.table(
  "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data")
dim(pros.df)
```

```
## [1] 97 10
```

```
head(pros.df)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

- **6a.** As we can see, the designers of this data set already defined training and test sets for us, in the last column of `pros.ds`! Split the prostate cancer data frame into two parts according to the last column, and report the number of observations in each part. On the training set, fit a linear model of `lpsa` on `lcavol` and `lweight`. On the test set, predict `lpsa` from the `lcavol` and `lweight` measurements. What is the test error?
- **6b.** Using the same training and test set division as in the previous question, fit a linear model on the training set `lpsa` on `age`, `gleason`, and `pgg45`. Then on the test set, predict `lpsa` from the relevant predictor measurements. What is the test error?
- **6c.** How do the test errors compare in the last two questions? Based on this comparison, what regression model would you recommend to your clinician friend? What other considerations might your clinician friend have when deciding between the two models that is not captured by your test error comparison?
- **6d.** The difference between the test errors of the two linear models considered above seems significant, but we have no sense of variability of these test error estimates, since it was just performed with one training/testing split. Repeatedly, split the prostate cancer data frame randomly into training and test sets of roughly equal size, fit the two linear models on the training set, and record errors on the test set. As a final estimate of test error, average the observed test errors over this process for each of the two model types. Then, compute the standard deviation of the test errors over this process for each of the two model types. After accounting for the standard errors, do the test errors between the two linear model types still appear significantly different?

Cross-validation with the prostate cancer data

- **7a.** Let's revisit the prostate cancer data. Randomly split the prostate cancer data frame into $k = 5$ folds of roughly equal size. Report the number of observations that fall in each fold.
- **7b.** Over the folds you computed in the previous question, compute the cross-validation error of the linear model that regresses `lpsa` on `lcavol` and `lweight`.
- **7c.** Write a function `pros.cv()`, which takes three arguments: `df`, a data frame of prostate cancer measurements, with a default of `pros.df`; `k`, an integer determining the number of cross-validation folds, with a default of 5; and `seed`, an integer to be passed to `set.seed()` before defining the folds, with a default of NULL (meaning no seed shall be set). Your function should split up the given data `df` into `k` folds of roughly equal size, and using these folds, compute the cross-validation error of the linear model that regresses `lpsa` on `lcavol` and `lweight`. Its output should simply be this cross-validation error.

- **7d.** Investigate the result of `pros.cv()` for different values of `k`, specifically, for `k` equal to 2, 5, 10, and 97. For each value, run `pros.cv()` some large number of times (say, 50) and report the average of the cross-validation error estimates, and the standard deviation of these estimates. Then, plot them in an informative way (say, a box plot with `boxplot()`). What do you notice? Is this surprising?
- **7e.** In general, is 2-fold cross-validation the same as sample-splitting? Why or why not?
- **7f.** In general, what can you say about the differences in cross-validation as the number of folds varies? What is different about cross-validation with 2 folds, versus 5 folds, versus n folds (with n being the number of observations in the data set)?
- **7g.** Modify your function `pros.cv()` so that it takes another argument: `formula.str`, a string in the format of a formula specifying which linear model is to be evaluated by cross-validation, with the default being "lpsa ~ lcavol + lweight". Demonstrate the use of your function for different formulas, i.e., different linear regression models.