EECS 391 Written Assignment 5

William Koehrsen wjk68

December 5, 2017

2. The posterior distribution for θ is the expected distribution of θ given y, the number of heads, and n, the number of coin flips.

We can express the posterior using Bayes’ Rule as follows:

This represents the familiar formula

We already know the likelihood:

The prior is a constant because we are assuming a uniform prior distribution for the probability of a heads. The probability of a particular flip coming up heads does not change based on the number of the flip. The normalization constant is necessary to make the posterior distribution sum to 1. The normalization constant is expressed as:

Again, the prior is a constant which can be pulled out of the integral. The prior then cancels from the numerator and the denominator. From the problem statement, we are provided:

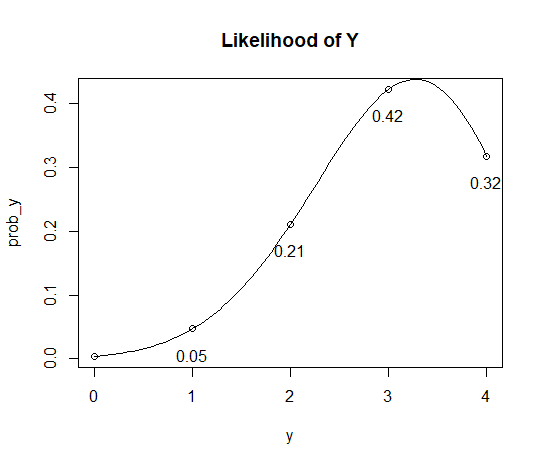
The expression for the posterior after some rearranging becomes:

Leading to the final answer:

1. The likelihood is the distribution of the possible number of heads given the probability of heads and the number of flips

Given n = 4 and θ = ¾

We can use R to plot the range of probabilities for y = 0 to y = 4.

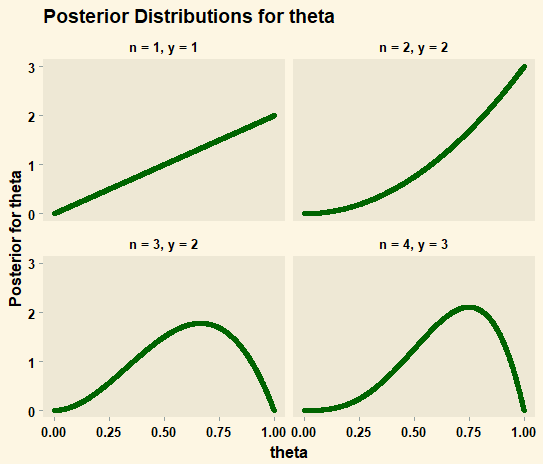


As expected, with a value for the probability of heads of 0.75, the most probable outcome for the number of heads given 4 total flips is 3.

1. From part A, the posterior distribution of θ is

The four cases are y = 1, n = 1; y = 2, n = 2; y = 2, n = 3; y = 3, n = 4

The posterior distribution of the value of θ after each flip can be plotted in R.



In each case, the maximum posterior for θ occurs at a value of y/n. For the first two plots, this is at , for the third plot at , and at , for the fourth plot.

1. The graphs in Figure 20.1 were drawn under hypothesis 5, that 100% of candies in the selected bag are lime.



There are four other hypothesis to evaluate:

Under each hypothesis, we must plot the posterior distributions for all the hypotheses and the probability that the next candy is lime over a range of 100 observations.

1. Posterior Distribution

Posterior probabilities are expressed in terms of Bayes’ rule:

Where **d** is a vector representing all of the observations. represents the likelihood of the data under the hypothesis, and is the prior for the hypothesis. The prior probabilities for each of the hypotheses are provided by the manufacturer as [0.1, 0.2, 0.4, 0.2, 0.1] for hypotheses 1-5 respectively. The normalization constant, α, represents the total probability of the observed sequence of observations.

We can treat this problem as a binomial distribution as in Problem 1. In this case, θ is the percentage of lime candies in a bag under a given hypothesis, y is the number of lime candies unwrapped, and n is the total number of candies unwrapped. Under each hypothesis, and for each value of y and n observed, we can compute the posterior probability for a given hypothesis, θ:

In this case, the normalization constant is the total chance of the observed number of limes given the total number of observations summed across all hypothesis.

This normalization constant is needed so that the posterior probability for all the hypothesis given a certain set of observations sums to one. The prior probabilities do not depend on the number of observations and are constant. The total equation for the posterior probability is thus expressed:

Where is the hypothesis (percentage of lime candies in the bag), y is the number of lime candies unwrapped, and n is the total number of candies unwrapped. is the prior for each hypothesis provided by the manufacturer, and the normalization constant appears in the denominator.

1. Probability of next candy being a lime

The Bayesian prediction for the next candy being lime is This represents the probability the next candy is a lime given all the prior observations. In math notation this is expressed as

The posterior was derived above:

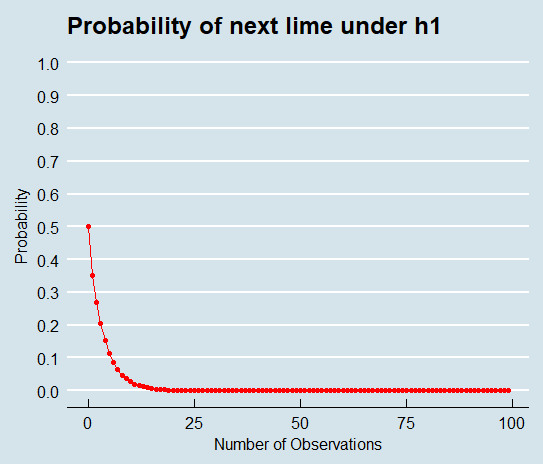
The conditional is the probability of a lime given the hypothesis. The final equation for the next candy selected being a lime is:

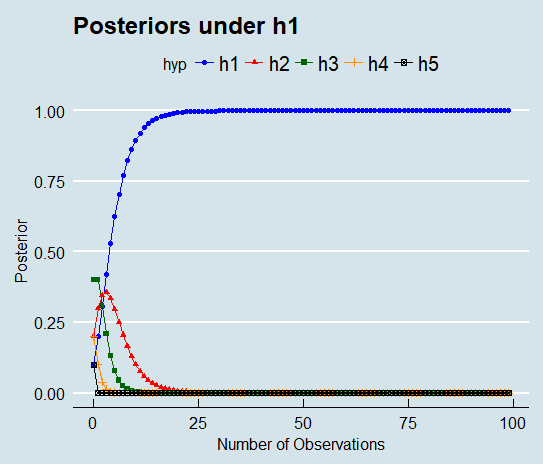
Both the posterior for each hypothesis and the probability the next candy is a lime under each of the five hypotheses ( I will recreate hypothesis 5 for completeness) can be calculated using straightforward code. The basic procedure is to assume one hypothesis as true, generate a distribution of observations corresponding to that hypothesis, and then iterate through the observations each time calculating the posterior for all five hypotheses and the probability the next candy is a lime. This task is repeated five times, each time assuming a different one of the five hypotheses is true. This is implemented in R code.

The results are then graphed in R based on the style of Figure 20.1

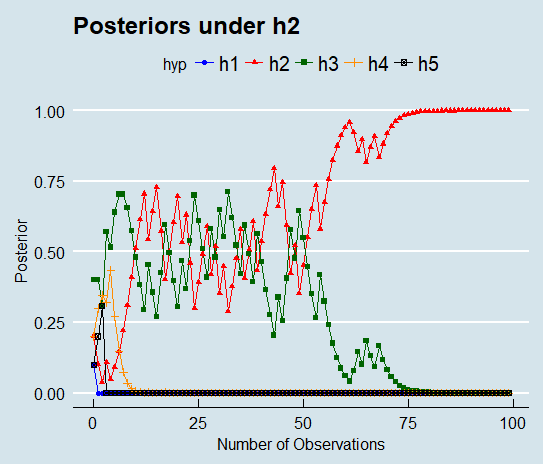
**Graphical Results**

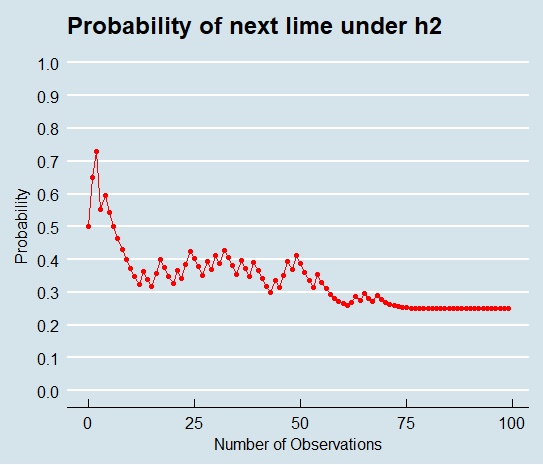
**Hypothesis 1: 0% Lime**



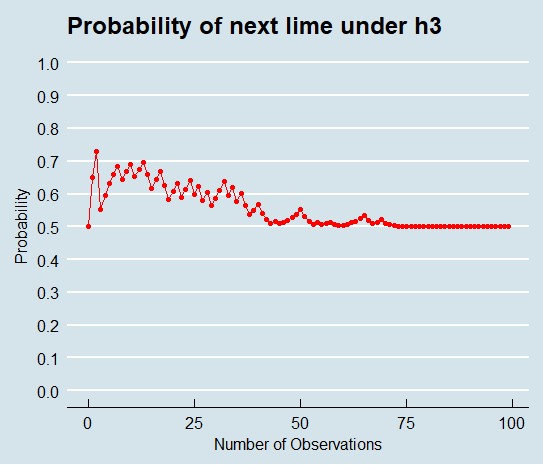


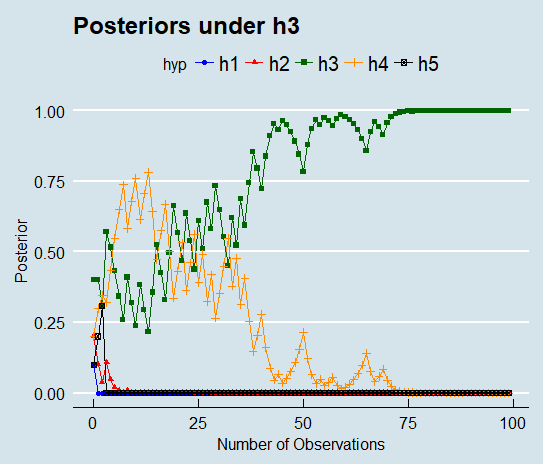
**Hypothesis 2: 25% Lime**

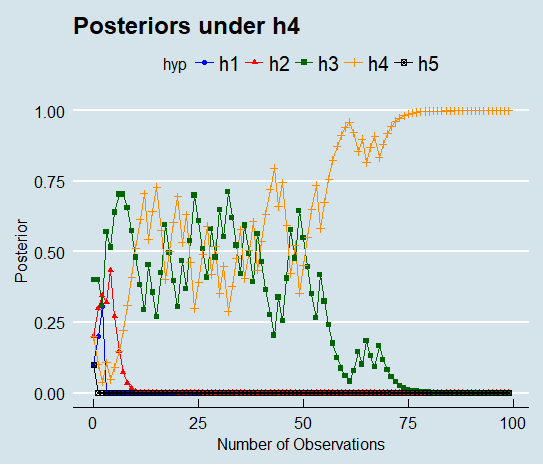
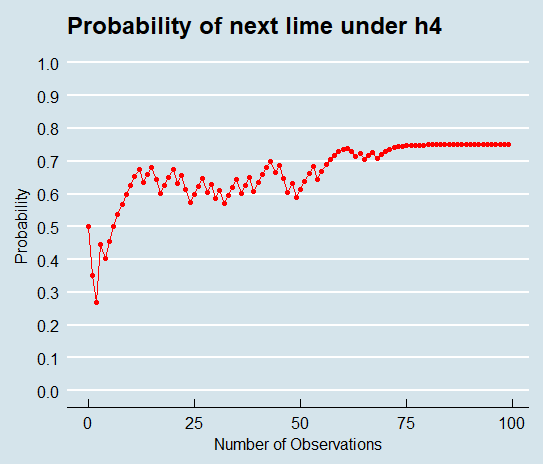




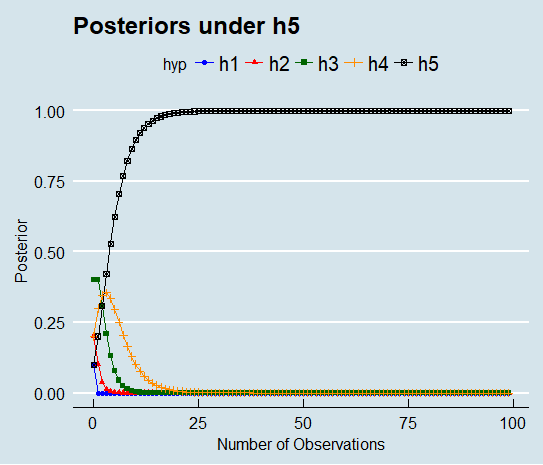
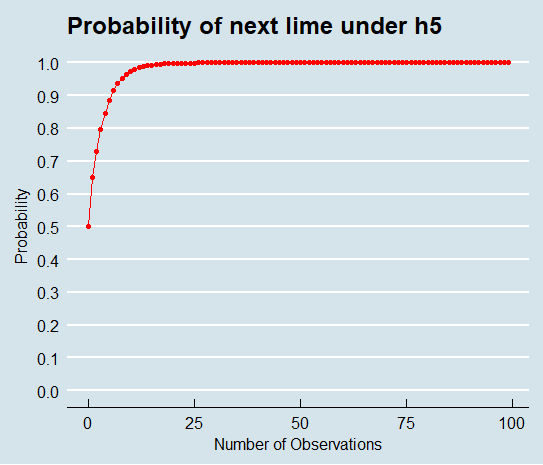
**Hypothesis 3: 50% Lime**





H**ypothesis 4: 75% Lime**

**Hypothesis 5: 100% Lime**



**Discussion**

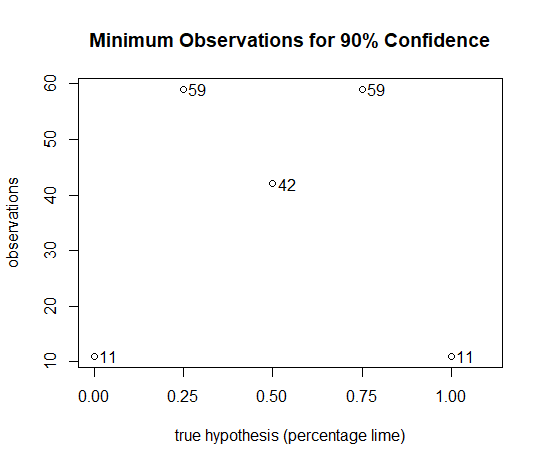
These graphs are insightful as the demonstrate how the posterior probability for each hypothesis and the probability the next candy is a lime converge on the true values over enough observations. In all graphs, the results are exactly as would be expected from Bayesian Inference. Hypotheses 1 and 5 are mirror images of each other and they correctly show the probability of the next candy being a lime converges to 0% and 100% respectively. Hypotheses 2 and 4 are also mirror images of each other, with the probability the next candy is a lime converging to 25% and 75% respectively as indicated by the hypotheses. H3 shows the probability of the next candy being a lime converges to 50%. In all cases, by 50 observations, the most likely hypothesis as indicated by the posteriors is the correct hypothesis. In H1 and H5 this is arrived at after 4 observations. The exact results under H2, H3, and H4 will vary because the observations are randomly generated from a binomial distribution with the probability of success set by the true hypothesis.

B. The posterior probability for a hypothesis is given by the following equation. θ is the hypothesis (probability of a lime candy being drawn next), y is the number of successes (lime candies), and n is the total number of observations.

To solve for the minimum number of observations needed to be more than 90% sure of the hypothesis, the posterior expression should be set equal to 0.9 and then solved for n. To solve for n, the hypothesis must be known as well as the sequence of observations.

The exact number of observations at which this will occur depends on the true hypothesis, which affects the actual sequence of observations. For example, we can use code to determine the number of observations at which the posterior probability for the correct hypothesis exceeds 90%.

**Plot Showing Minimum Number of Observations for 90% Confidence**



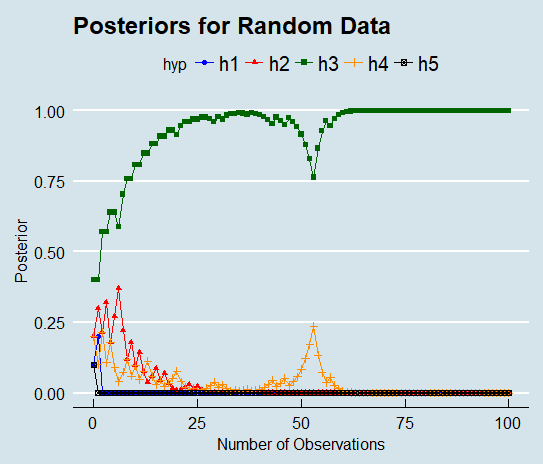
As is expected, with H1 and H5, the confidence reaches 90% extremely quickly while it takes the most number of observations in H2 and H4.

C. First, I need to create a random dataset of 100 observations. To do this, I will generate 100 datasets of length 100, with the distributions as determined by the priors from the manufacturer. The resulting observations can then be averaged to arrive at a random dataset of 100 observations weighted by the prior probability (as specified by the manufacturer) for each hypothesis. Under this procedure, we would expect the most likely number of lime candies would be 50% because the distribution for the hypotheses is symmetrical about 50%.

I will define the uncertainty for a hypothesis as:

Where is the posterior for the hypothesis given an unlimited number of observations. This is the steady-state value for the posterior of the hypothesis, or in other words, it is the value to which the posterior for a given hypothesis will converge over an infinite number of observations. In order to find the steady-state value for each of the five hypotheses, we can use the randomly generated data and plot the posteriors for all hypotheses versus the number of observations. Then, based on this plot, we can determine the posterior for the hypothesis if we were to perform an infinite number of experiments. We can then use this steady-state value to calculate the uncertainty in each hypothesis with each observation and find the difference from one observation to the next. All of these values will be plotted to provide an intuitive explanation.

As the data has been randomly generated and the most likely hypothesis is h3 at 0.4, we would expect the final posteriors would go to 100% for h3 and 0% for all other hypotheses. The plot below shows this to be true with a dataset of length 100.



The random data and the plot was created in R.

To summarize thus far, the steady-state posteriors are as follows:

|  |  |
| --- | --- |
| **Hypothesis** | **with random data** |
|  | 0.0 |
|  | 0.0 |
|  | 1.0 |
|  | 0.0 |
|  | 0.0 |

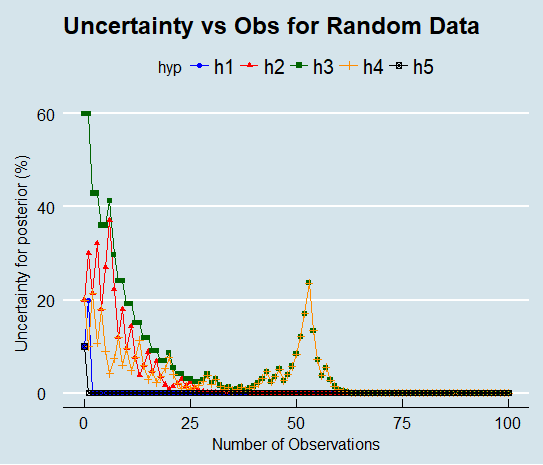
Now, I need to calculate the uncertainty at each observation for each hypothesis and the change in uncertainty from the previous observation.

Again, this is accomplished in R.

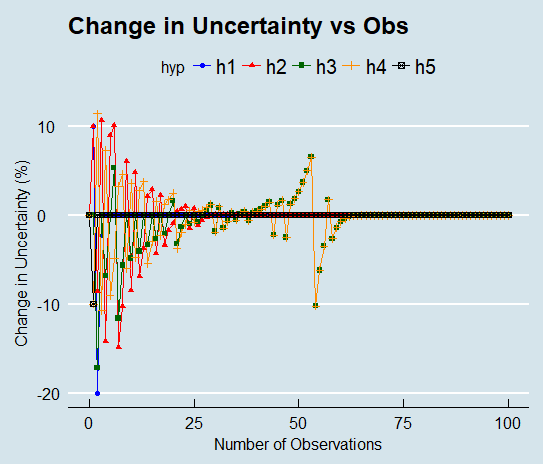
**Visualization of Results**

This first plot shows the total uncertainty for each posterior versus the number of observations.

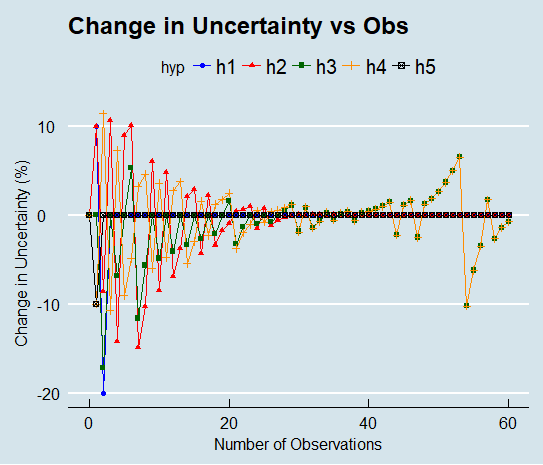
As can be seen, the uncertainty in each observation eventually drops to near zero with enough observations. However, not all observations decrease the uncertainty for some hypotheses, and instead increase the uncertainty.



The next graph shows the final answer, the change in uncertainty versus the number of observations for each type of bag with the randomly generated data. A negative value indicates the observation decreased the uncertainty for that bag, while a positive value indicated the observation increased the uncertainty for that type of bag.



Most of the reduction in uncertainty has been accomplished by the 60th observation, so we can segment to that part of the plot to better visualize what is occurring.



**Discussion**

After enough observations, the posteriors converge on the steady-state value. The initial observations have a much larger effect on the uncertainty of the bag than do later observations because the earlier observations change the proportion of limes more than do later observations as greater amounts of data are collected. Eventually, after 60 observations, each observation no longer has an effect on the uncertainty for the bag because the posteriors have converged to their steady state values. This plot illustrates the pitfalls of reasoning with small datasets: the fewer total observations, the greater the effect of any single observation. In a random process, this can lead to erroneous conclusions because randomly generated data will only converge to the true distribution with enough trials. Some observations can be seen to actually increase the uncertainty in a type of bag. For instance around observation 50, there is a long string of limes in a row, which makes h4 more likely and h3 correspondingly less likely. There are several other stretches such as which show that even random data appears to exhibit systematic trends over a small enough sample size. Nonetheless, with enough observations, the posteriors converge on the true value which in this case in H3, a 50% split.

A. The probability of x when the class is unknown is the sum of the likelihood of x under each of the components of the Gaussian Mixture Model:

Where is the likelihood of the evidence given the class, and is the component weight (the prior for the class). In this case, we are told that the prior probability of class 1 is twice that of class 2. Therefore, and as the component weights must sum to 1. The expression for the probability of x becomes:

As each of the component distributions are Gaussian, the likelihood for the observations can be expressed using the definition of the Gaussian. The final probability for x becomes:

B. The total classification error is the number of errors where the wrong classification is made given the decision boundary. This is illustrated by the following image from Professor Lewicki’s class slides:



In the current problem, 2 should be replaced by 1 and 3 replaced by 2. The expression for the total error then becomes:

Where is the area in which class 1 is incorrectly classified as class 2 and is the area in which class 2 is incorrectly classified as class 1. is the posterior probability for Class 1 given the observation. The posterior can be expressed using Bayes’ Rule:

The likelihood of the observation x given the Class j has already been calculated in part A in addition to the total probability of x, . The prior in this case is just the component weight provided in the assignment. The posterior probabilities for each class are thus:

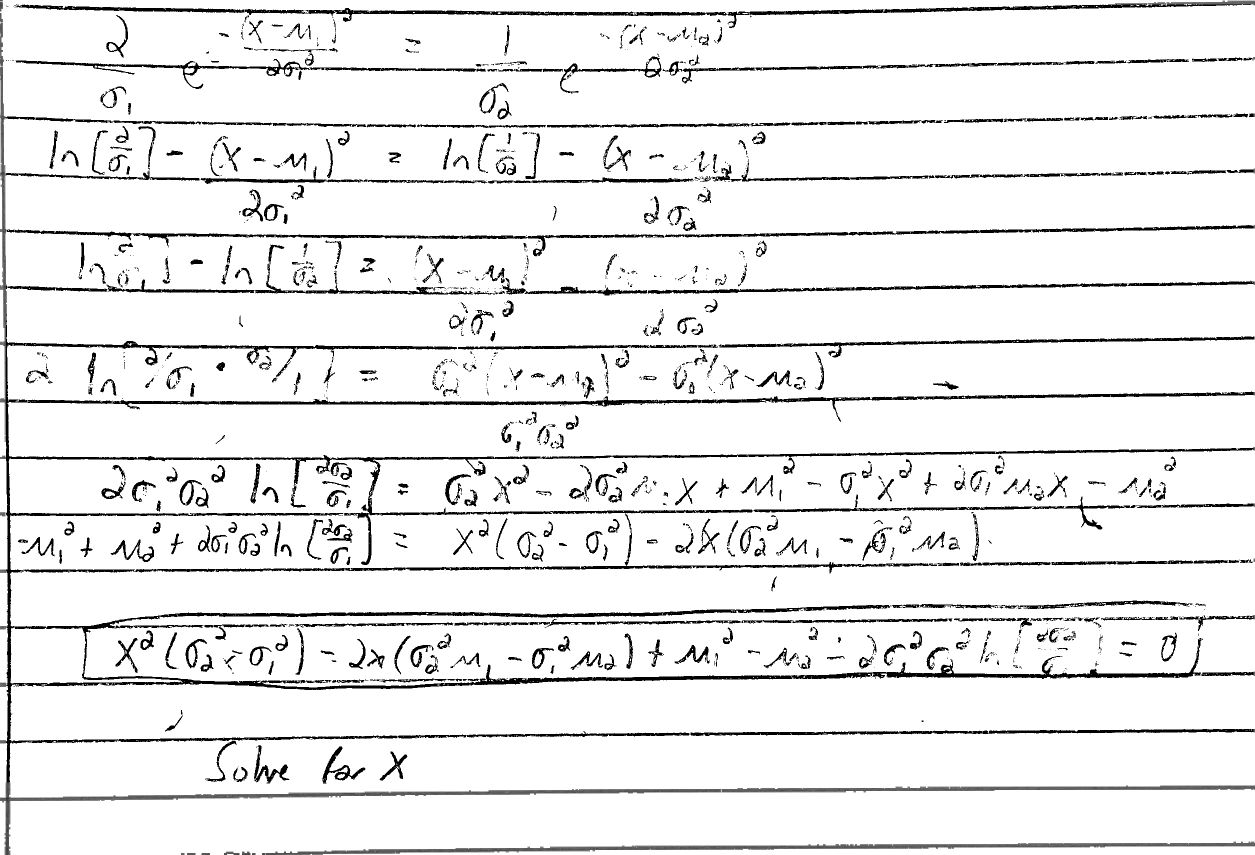
Given that the boundary occurs at , the errors where x is in Class 1 and is classified as Class 2 occur in the region . The errors where x is in Class 2 and is classified as Class 1 occur in the region . The final expression for the error is therefore:

C. The minimum probability of an error occurs where .

Using the Gaussian function and the equations for the posteriors derived previously, the expression becomes:

This can be simplified to the following:

This equation can then be solved for the value of x that minimizes the classification error. The following picture shows my attempt to simplify the equation on paper. I started by taking the log of both sides but eventually got lost in all the algebra. The end result is a quadratic equation in terms of x. The value of x that satisfies the equation is the location of the optimal decision boundary. In order to solve for the optimal value of the decision boundary, the paramters, σ and µ, for each Gaussian distribution must be known.



1. A. The objective function for vector and vector is

We are first asked to solve for the scalar form of the update rule. The update rule is found by taking the derivative of the objective function with respect to the parameter we wish to update, which in this case is the ith component of , setting the result equal to zero, and solving for the value of the parameter that minimizes the gradient.

This can be easily solved for the ith feature of the cluster center vector.

The value of is 1 if x is in cluster k, and 0 otherwise. Therefore, the update rule for the ith component of the cluster mean is the average of the ith component for all observations assigned to the cluster. This can be generalized to the update rule for the vector cluster mean

The update for the position of the cluster mean is the average position of all the data points assigned to the cluster. The general algorithm for implementing kmeans is

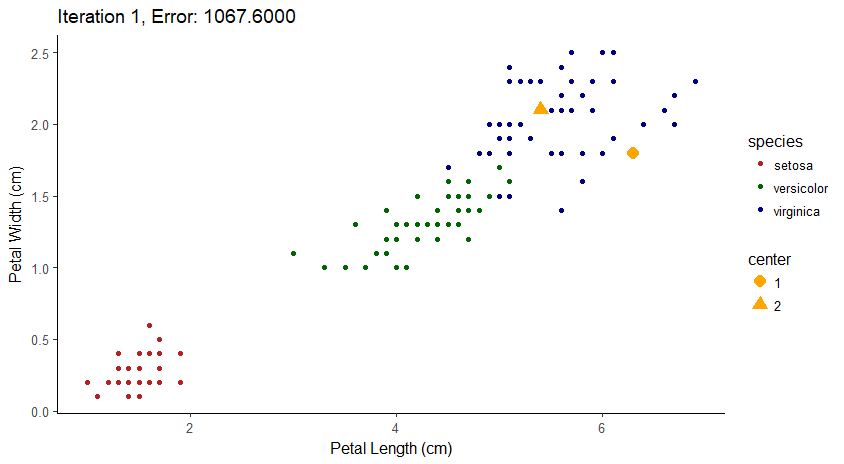
1. Define initial cluster centers at random (or using smart initialization such as [kmeans++ in Scikit-Learn](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.k_means.html))
2. Assign data points to the nearest cluster center as defined by Euclidean distance (Manhattan or [Minkowski distances with different values for the power](http://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/minkdist.htm) can [also be used](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.4030&rep=rep1&type=pdf))
3. Update cluster centers to the mean location of all points assigned to the cluster.
4. Repeat until the [algorithm converges which occurs when the cluster centers do not change](http://math.mit.edu/~rothvoss/18.304.1PM/Presentations/1-Yihui-Talk1.pdf) position.

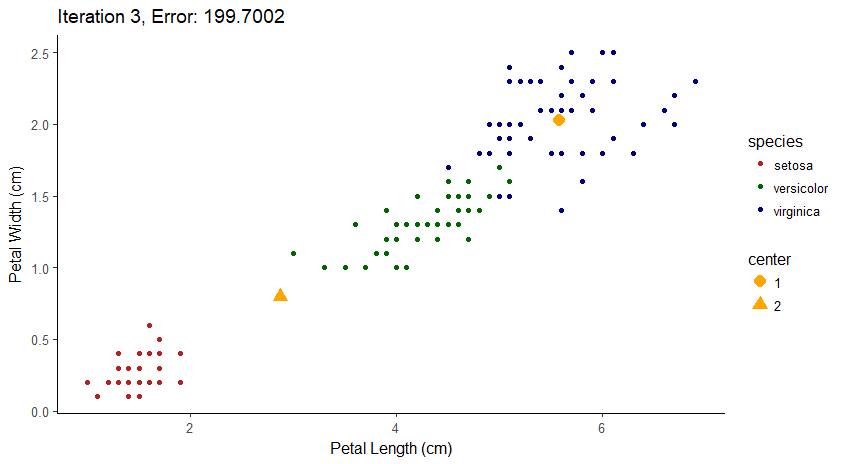
B. Kmeans R Implementation on Iris Data

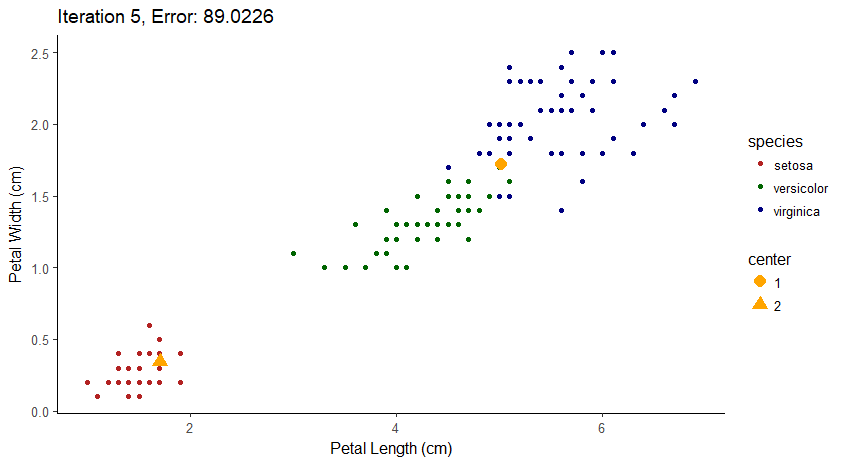
The Kmeans algorithm is relatively straightforward to implement in many languages (or better yet, simply use an existing framework). I will use R because of the graphing properties and not because it is the most efficient. Full code is presented in the Appendices.

Graphical Results of Convergence

Results with k = 2 (Converged in 7 iterations)

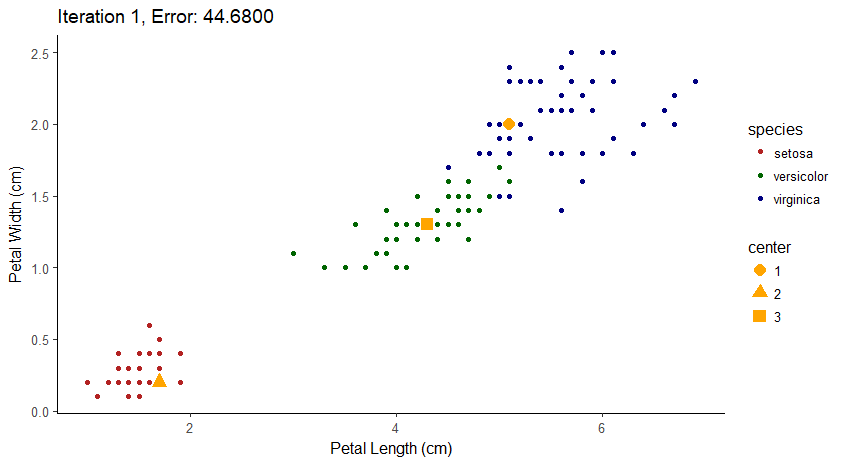


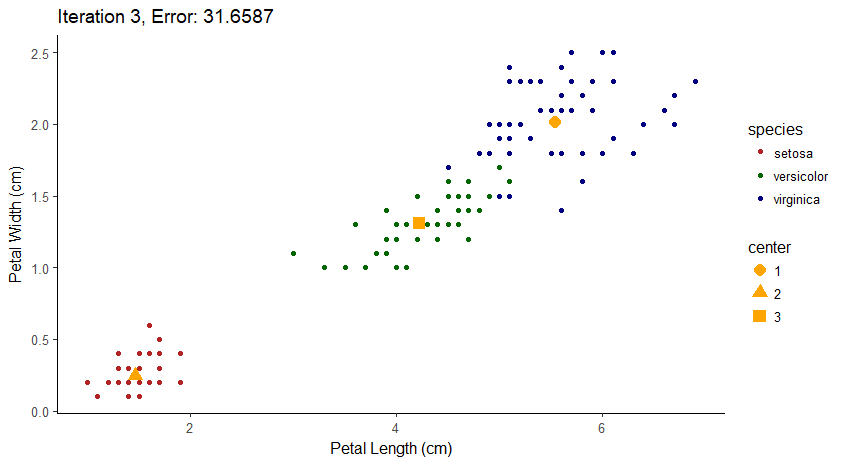


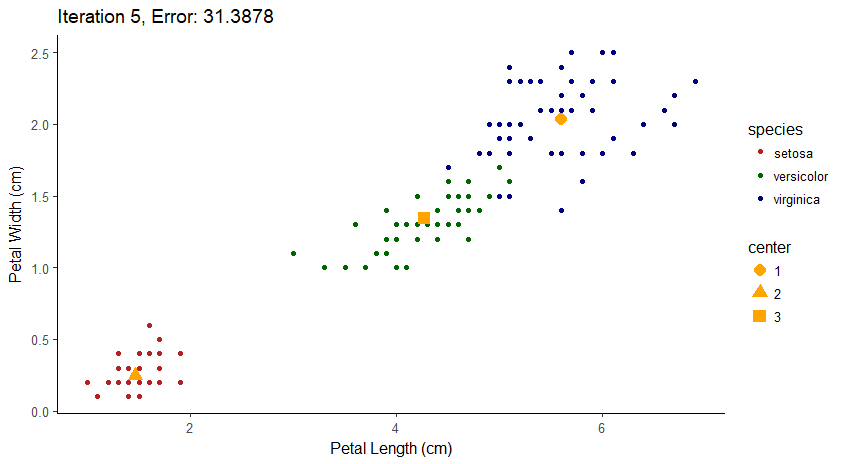


Results with k = 3 (Converged in 5 iterations)

In the following example, the algorithm was extremely fortunate it its selection of initial cluster centers. One cluster center was selected from each species (clusters were initialized at random and were not set to come from each species). Therefore, the algorithm reached convergence in a remarkably short time period. This behavior was not observed in all tests of the algorithm, and these results demonstrate one of the shortcomings of kmeans clustering; it is very susceptible to starting location of cluster centers. Approaches to reduce the chance of overfitting include doing multiple random restarts and taking the result with the lowest overall error or choosing smart initializations for the cluster centers. One implementation of this is the kmeans++ initialization in Scikit-Learn. This approach chooses initial cluster centers one at a time where the probability of a point being a cluster center is directly proportional to its distance from the nearest already assigned cluster resulting in initial assignments of cluster centers that are spread apart from one another. No such scheme was used in this implemented, but it would make a nice project to try!







Overall, the time for convergence and the final positioning of the cluster centers is highly dependent on the initial locations of the clusters. The need for a smart initialization is evident.

Appendices R Code

# EECS 391 HW5

# 1. B. Plot the Likelihood for n = 4 and theta = 0.75 --------------------

library(tidyverse)

library(ggthemes)

# Define possible values of y

y <- seq(0, 4, 1)

# Function that takes in a value of y and returns the probability of y

# given n = 4 and theta = 0.74

prob <- function(y) {

first\_term <- (factorial(4) / (factorial((4 - y)) \* factorial(y)))

second\_term <- (0.75 ^ y) \* (0.25 ^ (4 - y))

return(first\_term \* second\_term)

}

# The probability distribution of y

prob\_y <- sapply(y, prob)

# Created y for smooth lines

y\_smooth <- seq(0, 4, by = 4/100)

prob\_y\_smooth <- sapply(y\_smooth, prob)

# Plot the likelihood of y under the assumption theta = 0.75 and n = 4

plot(y, prob\_y, main = 'Likelihood of y') + lines(y\_smooth, prob\_y\_smooth) +

text(y, prob\_y - 0.04, labels = round(prob\_y, 2))

# 1. C.

# Calculates posterior probability for theta given n and y

calc\_posterior <- function(theta, y, n) {

first\_term <- factorial(n + 1) / (factorial(n - y) \* factorial(y))

second\_term <- (theta ^ y) \* ((1 - theta) ^ (n - y))

return(first\_term \* second\_term)

}

y <- c(1, 2, 2, 3)

n <- c(1, 2, 3, 4)

theta <- seq(0, 1, 1/1000)

posterior\_df <- as.data.frame(matrix(nrow = length(theta) \* 4, ncol = 4))

names(posterior\_df) <- c('n', 'y', 'theta', 'posterior')

i <- 1

all\_posteriors <- c()

for (head\_count in y) {

posteriors <- sapply(theta, calc\_posterior, y = head\_count, n = i)

all\_posteriors <- c(all\_posteriors, posteriors)

i <- i + 1

}

posterior\_df$n <- c(rep(1, 1001), rep(2, 1001),

rep(3, 1001), rep(4, 1001))

posterior\_df$y <- c(rep(1, 1001), rep(2, 2002),

rep(3, 1001))

posterior\_df$n\_y <- c(rep(c('n = 1, y = 1'), 1001), rep(c('n = 2, y = 2'), 1001),

rep(c('n = 3, y = 2'), 1001), rep(c('n = 4, y = 3'), 1001))

posterior\_df$theta <- c(rep(theta, 4))

posterior\_df$posterior <- all\_posteriors

ggplot(posterior\_df, aes(theta, posterior)) + geom\_point(col = 'darkgreen') +

geom\_line(col = 'darkgreen') +

facet\_wrap(~n\_y) + ylab('Posterior for theta') +

ggtitle('Posterior Distributions for theta') + theme\_solarized\_2() +

theme(panel.grid = element\_blank(), text = element\_text(face = 'bold', color = 'black'),

title = element\_text(face = 'bold', color = 'black'))

# 2. A. Plot the posterior distribution and the next lime probability --------

# Prior probabilities and list of hypotheses expressed as probability of lime candy

priors <- list('h1' = 0.1, 'h2' = 0.2, 'h3' = 0.4, 'h4' = 0.2, 'h5' = 0.1)

hypotheses <- list('h1' = 0.0, 'h2' = 0.25, 'h3' = 0.5, 'h4' = 0.75, 'h5' = 1.0)

# The likelihood of a hypothesis (theta) given observations of successes, y,

# and total number of trials

calc\_likelihood <- function(theta, y, n) {

first\_term <- factorial(n) / (factorial(n - y) \* factorial(y))

second\_term <- (theta ^ y) \* ((1 - theta) ^ (n - y))

return(first\_term \* second\_term)

}

# Calculates the posterior for each of the five hypothesis under one true hypothesis

calc\_posteriors <- function(true\_hyp) {

# Set seed to ensure consistent results

set.seed(42)

# Create a dataframe to hold results

posts\_df <- as.data.frame(matrix(ncol = 8))

names(posts\_df) <- c('n', 'y', 'h1', 'h2', 'h3', 'h4', 'h5', 'next\_lime')

# Vector of observations under the true hypothesis

limes <- c(0, rbinom(n = 99, size = 1, prob = true\_hyp))

total\_limes <- cumsum(limes)

# Priors for each hypothesis and the conditional probability of a lime candy

# for each hypothesis

priors <- list('h1' = 0.1, 'h2' = 0.2, 'h3' = 0.4, 'h4' = 0.2, 'h5' = 0.1)

lime\_pct <- list('h1' = 0.0, 'h2' = 0.25, 'h3' = 0.5, 'h4' = 0.75, 'h5' = 1.0)

# Iterate through the observations and calculate posteriors and

# chance of next lime

for (i in seq(1, 100, 1)) {

n\_lime <- total\_limes[i]

# Calculate the unnormalized posterior for each hypothesis

h1\_post <- calc\_likelihood(lime\_pct$h1, n\_lime, i - 1) \* priors$h1

h2\_post <- calc\_likelihood(lime\_pct$h2, n\_lime, i - 1) \* priors$h2

h3\_post <- calc\_likelihood(lime\_pct$h3, n\_lime, i - 1) \* priors$h3

h4\_post <- calc\_likelihood(lime\_pct$h4, n\_lime, i - 1) \* priors$h4

h5\_post <- calc\_likelihood(lime\_pct$h5, n\_lime, i - 1) \* priors$h5

# Calculate the normalization constant and normalize the posteriors

posts <- c(h1\_post, h2\_post, h3\_post, h4\_post, h5\_post)

norm\_constant <- sum(posts)

norm\_posts <- posts / norm\_constant

# The probability of a lime next

next\_lime <- sum(unlist(lime\_pct) \* norm\_posts)

# Explicitly add results to a dataframe

posts\_df <- add\_row(posts\_df, n = i - 1, y = n\_lime, h1 = norm\_posts[1], h2 = norm\_posts[2],

h3 = norm\_posts[3], h4 = norm\_posts[4], h5 = norm\_posts[5],

next\_lime = next\_lime)

}

return(posts\_df[-1, ])

}

# Assume each hypothesis is true and calculate the corresponding distributions

h1\_true <- calc\_posteriors(0.0)

h2\_true <- calc\_posteriors(0.25)

h3\_true <- calc\_posteriors(0.5)

h4\_true <- calc\_posteriors(0.75)

h5\_true <- calc\_posteriors(1.0)

# Function to graph posterior distributions for each hypothesis and the

# probability the next candy is a lime

# Takes in a dataframe with posteriors and the string of the true hypothesis

# that was used to calculate the posteriors

graph\_posts <- function(post\_df, true\_hyp\_str) {

# Titles for figures

main1 <- sprintf('Posteriors under %s', true\_hyp\_str)

main2 <- sprintf('Probability of next lime under %s', true\_hyp\_str)

# Put results in long format with each row and observation and each column

# a variable

long\_post <- gather(post\_df, key = 'hyp', value = 'post', -n, -y, -next\_lime)

color\_vector <- c('blue', 'red', 'darkgreen', 'darkorange', 'black')

th <- theme(axis.text = element\_text(color = 'black'))

# Graph the posteriors for each hypothesis under the true hypothesis

p1 <- ggplot(long\_post, aes(n, post, col = hyp, group = hyp, shape = hyp)) + geom\_line() +

geom\_point() + xlab('Number of Observations') + ylab('Posterior') +

ggtitle(main1) + theme\_economist(12) + th + scale\_color\_manual(values = color\_vector)

# Graph the probability of the next candy being a lime

p2 <- ggplot(post\_df, aes(n, next\_lime)) + geom\_line(col = 'red') + geom\_point(col = 'red') +

xlab('Number of Observations') + ylab('Probability') + ggtitle(main2) +

coord\_cartesian(ylim = c(0, 1)) + scale\_y\_continuous(breaks = seq(0, 1, 0.1)) +

theme\_economist(12) + th

# Display the graphs

print(p1)

print(p2)

}

# Graph results for each assumed hypothesis

graph\_posts(h1\_true, 'h1')

graph\_posts(h2\_true, 'h2')

graph\_posts(h3\_true, 'h3')

graph\_posts(h4\_true, 'h4')

graph\_posts(h5\_true, 'h5')

# 2. B. Find the minimum number of observations for 90% Confidence --------

# Find the minimum number of observations for a given confidence level

# Posterior df is generated from the specified true hypothesis (lime pct)

find\_minimum\_obs <- function(confidence\_level, post\_df, true\_hyp\_str) {

# Filter dataframe to observations with true posterior greater than confidence level

above\_cl <- post\_df[post\_df[true\_hyp\_str] > confidence\_level, ]

minimum\_obs <- above\_cl[[which.min(above\_cl$n), 'n']]

# Return the minimum observations for the specified confidence level

return(minimum\_obs)

}

# Find minimum observations required under each hypothesis

h1\_min <- find\_minimum\_obs(0.9, h1\_true, 'h1')

h2\_min <- find\_minimum\_obs(0.9, h2\_true, 'h2')

h3\_min <- find\_minimum\_obs(0.9, h3\_true, 'h3')

h4\_min <- find\_minimum\_obs(0.9, h4\_true, 'h4')

h5\_min <- find\_minimum\_obs(0.9, h5\_true, 'h5')

# Vectors for plotting results

hyps <- c(0.0, 0.25, 0.5, 0.75, 1.0)

min\_obs <- c(h1\_min, h2\_min, h3\_min, h4\_min, h5\_min)

# Plot the results

plot(x = hyps, y = min\_obs, xaxt = 'n', xlim = c(0, 1.1),

xlab = 'true hypothesis (percentage lime)',

ylab = 'observations', main = 'Minimum Observations for 90% Confidence') +

text(x = hyps + 0.04, y = min\_obs, labels = min\_obs) + axis(side = 1, at = hyps)

# 2. C. Illustrate the reduction in uncertainty over the observati --------

# Generate 100 random datasets of 100 observations and average

generate\_random <- function() {

observations <- c(rep(0, 100))

priors <- c(0.1, 0.2, 0.4, 0.2, 0.1)

i <- 1

total <- 0

for (prob in c(0.0, 0.25, 0.5, 0.75, 1.0)) {

iterations <- priors[i]

for (m in 1:(iterations \* 100)) {

observations <- (observations + rbinom(n = 100, size = 1, prob = prob))

total <- total + 1

}

i <- i + 1

}

# Take mean and round to either a 0 or 1

observations <- (observations / total)

observations[which(observations >= 0.5)] = 1L

observations[which(observations < 0.5)] = 0L

return(observations)

}

# Generate the random dataset

random\_obs <- generate\_random()

# Calculates the posterior for each of the five hypothesis given a set of observations

calc\_posteriors\_given\_obs <- function(obs) {

# Create a dataframe to hold results

posts\_df <- as.data.frame(matrix(ncol = 8))

names(posts\_df) <- c('n', 'y', 'h1', 'h2', 'h3', 'h4', 'h5', 'next\_lime')

# Vector of observations provided

limes <- c(0, obs)

total\_limes <- cumsum(limes)

# Priors for each hypothesis and the conditional probability of a lime candy

# for each hypothesis

priors <- list('h1' = 0.1, 'h2' = 0.2, 'h3' = 0.4, 'h4' = 0.2, 'h5' = 0.1)

lime\_pct <- list('h1' = 0.0, 'h2' = 0.25, 'h3' = 0.5, 'h4' = 0.75, 'h5' = 1.0)

# Iterate through the observations and calculate posteriors and

# chance of next lime

for (i in seq(1, 101, 1)) {

n\_lime <- total\_limes[i]

# Calculate the unnormalized posterior for each hypothesis

h1\_post <- calc\_likelihood(lime\_pct$h1, n\_lime, i - 1) \* priors$h1

h2\_post <- calc\_likelihood(lime\_pct$h2, n\_lime, i - 1) \* priors$h2

h3\_post <- calc\_likelihood(lime\_pct$h3, n\_lime, i - 1) \* priors$h3

h4\_post <- calc\_likelihood(lime\_pct$h4, n\_lime, i - 1) \* priors$h4

h5\_post <- calc\_likelihood(lime\_pct$h5, n\_lime, i - 1) \* priors$h5

# Calculate the normalization constant and normalize the posteriors

posts <- c(h1\_post, h2\_post, h3\_post, h4\_post, h5\_post)

norm\_constant <- sum(posts)

norm\_posts <- posts / norm\_constant

# The probability of a lime next

next\_lime <- sum(unlist(lime\_pct) \* norm\_posts)

# Explicitly add results to a dataframe

posts\_df <- add\_row(posts\_df, n = i - 1, y = n\_lime, h1 = norm\_posts[1], h2 = norm\_posts[2],

h3 = norm\_posts[3], h4 = norm\_posts[4], h5 = norm\_posts[5],

next\_lime = next\_lime)

}

return(posts\_df[-1, ])

}

# Calculate the posterior odds given the randomly generated data

random\_post <- calc\_posteriors\_given\_obs(random\_obs)

# Put dataframe in long format

random\_long <- gather(random\_post, key = 'hyp', value = 'post', -n, -y, -next\_lime)

# Theme for axis text and color vector

th <- theme(axis.text = element\_text(color = 'black'))

color\_vector <- c('blue', 'red', 'darkgreen', 'darkorange', 'black')

# Graph the posteriors for each hypothesis with the random data

p1 <- ggplot(random\_long, aes(n, post, col = hyp, group = hyp, shape = hyp)) + geom\_line() +

geom\_point() + xlab('Number of Observations') + ylab('Posterior') +

ggtitle('Posteriors for Random Data') + theme\_economist(12) + th +

scale\_color\_manual(values = color\_vector)

# Graph the probability of the next candy being a lime with the random data

p2 <- ggplot(random\_post, aes(n, next\_lime)) + geom\_line(col = 'red') + geom\_point(col = 'red') +

xlab('Number of Observations') + ylab('Probability') +

ggtitle("Probability of Next Lime with Random Data") +

coord\_cartesian(ylim = c(0, 1)) + scale\_y\_continuous(breaks = seq(0, 1, 0.1)) +

theme\_economist(12) + th

print(p1)

print(p2)

# Create uncertainty columns for each hypothesis

random\_unc <- random\_post %>% mutate(h1\_un = h1 - 0,

h2\_un = h2 - 0,

h3\_un = 1 - h3,

h4\_un = h4 - 0,

h5\_un = h5 - 0)

# Create a long format dataframe for plotting

random\_unc\_long <- random\_unc %>% dplyr::select(n, h1\_un:h5\_un) %>%

gather(key = 'hyp', value = 'unc', -n)

# Clean up the names for plotting

random\_unc\_long <- random\_unc\_long %>%

mutate(hyp = stringr::str\_split\_fixed(hyp, pattern = '\_', n = 2)[, 1])

# Plot the uncertainties

ggplot(random\_unc\_long, aes(n, unc \* 100, col = hyp, shape = hyp, group = hyp)) +

geom\_line() + geom\_point() + xlab('Number of Observations') +

ylab('Uncertainty for posterior (%)') + ggtitle('Uncertainty vs Obs for Random Data') +

scale\_color\_manual(values = color\_vector) + theme\_economist(12) + th

# Create columns with reduction in uncertainty

random\_reduc <- random\_unc %>% mutate(h1\_reduc = c(0, diff(h1\_un)),

h2\_reduc = c(0, diff(h2\_un)),

h3\_reduc = c(0, diff(h3\_un)),

h4\_reduc = c(0, diff(h4\_un)),

h5\_reduc = c(0, diff(h5\_un)))

# Put into long format

random\_reduc\_long <- random\_reduc %>% dplyr::select(n, h1\_reduc:h5\_reduc) %>%

gather(key = 'hyp', value = 'reduc', -n)

# Clean up the names for plotting

random\_reduc\_long <- random\_reduc\_long %>%

mutate(hyp = stringr::str\_split\_fixed(hyp, pattern = '\_', n = 2)[, 1])

# Plot the reductions in uncertainty

ggplot(random\_reduc\_long, aes(n, (reduc \* 100), col = hyp, shape = hyp, group = hyp)) +

geom\_line() + geom\_point() + xlab('Number of Observations') +

ylab('Change in Uncertainty (%)') + ggtitle('Change in Uncertainty vs Obs') +

scale\_color\_manual(values = color\_vector) + theme\_economist(12) + th

# Plot for the first 50 observations

ggplot(dplyr::filter(random\_reduc\_long, n <= 60),

aes(n, (reduc \* 100), col = hyp, shape = hyp, group = hyp)) +

geom\_line() + geom\_point() + xlab('Number of Observations') +

ylab('Change in Uncertainty (%)') + ggtitle('Change in Uncertainty vs Obs') +

scale\_color\_manual(values = color\_vector) + theme\_economist(12) + th

# 4. B. KMeans Implementation on Iris Dataset -----------------------------

# Read in Iris Data

iris\_data <- read\_csv('irisdata.csv')

# Takes the number of cluster centers, the iris dataframe, and max iterations

kmeans <- function(k, iris\_df, max\_iter = 5) {

# Select the features used for clustering

iris\_df <- dplyr::select(iris\_df, petal\_length, petal\_width, species)

# Choose initial cluster centers at random from all data points

initial\_indices <- sample(1:length(iris\_df$species), size = k)

cluster\_centers <- data.matrix((iris\_df[initial\_indices, c(1,2)]))

# Dataframe to track progression of cluster centers

track\_centers <- as.data.frame(matrix(ncol = 2))

names(track\_centers) <- c('petal\_length', 'petal\_width')

# Used for keeping track of iterations

index <- c()

center\_index <- c()

# Vector with all errors

total\_errors <- c()

# Iterate for the max iterations or until convergence

for (n in 1:max\_iter) {

# Needed for convergence check

previous\_cluster\_centers <- cluster\_centers

# Holds the class assignments

classes <- c()

# Track progression of clusters

track\_centers <- rbind(track\_centers, cluster\_centers)

# Iterate through the irises

for (i in 1:nrow(iris\_df)) {

# Select an iris

iris <- as.numeric((dplyr::select(

iris\_df, petal\_length, petal\_width))[i, ])

distances <- c()

# Iterate through the cluster centers

for (j in 1:k) {

center <- cluster\_centers[j, ]

# Calculate the Euclidean distance between each point

# and the cluster center

distance <- dist(matrix(data = c(center, iris),

ncol = 2, byrow = TRUE),

method = 'euclidean')

distances <- c(distances, distance)

}

# The class is the cluster center with the

# minimum distance from the iris

class <- which.min(distances)

classes <- c(classes, class)

}

# Assign classes to all the irises (irisi perhaps?)

iris\_df$class <- classes

# Used to record errors for assignment of cluster centers

track\_errors <- c()

# Loop to update all the cluster centers

for (class\_num in unique(iris\_df$class)) {

# Extract only those points assigned to the cluster

class\_df <- dplyr::filter(iris\_df, class == class\_num)

# Calculate error associated with cluster center

class\_center <- cluster\_centers[class\_num, ]

# Error associated with each feature

class\_df$length\_error <- class\_df$petal\_length - class\_center[1]

class\_df$width\_error <- class\_df$petal\_width - class\_center[2]

# Total error is the sum of the individual feature squared errors

class\_df <- dplyr::mutate(class\_df, total\_error =

length\_error ^ 2 + width\_error ^ 2)

# Keep track of the errors

total\_cluster\_error <- sum(class\_df$total\_error)

track\_errors <- c(track\_errors, total\_cluster\_error)

# Update rule for cluster center

cluster\_centers[class\_num, ] = c(mean(class\_df$petal\_length),

mean(class\_df$petal\_width))

}

# The total error is the sum of the errors for each cluster

total\_errors <- c(total\_errors, sum(track\_errors))

# Used for tracking the changes in cluster centers

index <- c(index, rep(n, k))

center\_index <- c(center\_index, seq(1, k, by = 1))

# Convergence condition

if (all(previous\_cluster\_centers == cluster\_centers)) {

print('Convergence Achieved')

track\_centers <- track\_centers[complete.cases(track\_centers), ]

track\_centers$iter <- index

track\_centers$center <- center\_index

# Return the cluster centers and errors for plotting

return(list('cluster\_centers' = track\_centers,

'errors' = total\_errors))

# If convergence not achieved continue iteration

} else {

previous\_cluster\_centers <- cluster\_centers

}

# Provide feedback

print(sprintf('Iteration: %0.0f total error: %0.2f', n,

sum(track\_errors)))

}

# Return the position of clusters over iterations

track\_centers <- track\_centers[complete.cases(track\_centers), ]

track\_centers$iter <- index

track\_centers$center <- center\_index

print(sprintf('Max iterations: %0.0f reached with error: %0.4f',

max\_iter, sum(track\_errors)))

return(list('cluster\_centers' = track\_centers, 'errors' = total\_errors))

}

# Example with 3 cluster centers

results <- kmeans(3, iris\_data, max\_iter = 12)

cluster\_df <- results$cluster\_centers

total\_error <- results$errors

iterations <- unique(cluster\_df$iter)

# Display plot at beginning, middle, and end of iterations

for (iteration in seq(1, max(iterations),

by = floor(length(cluster\_df)/2))) {

# Subset the data to the relevent iteration

cluster\_subset <- dplyr::filter(cluster\_df, iter == iteration)

cluster\_subset$center <- as.factor(cluster\_subset$center)

# Plot the iris data and the cluster centers

p <- ggplot(iris\_data, aes(x = petal\_length, y = petal\_width,

color = species)) +

geom\_point() + xlab('Petal Length (cm)') +

ylab('Petal Width (cm)') +

ggtitle('Cluster Centers with Iris Data') + theme\_classic(12) +

scale\_color\_manual(values = c('firebrick', 'darkgreen', 'navy')) +

geom\_point(data = cluster\_subset, aes(x = petal\_length, y = petal\_width,

shape = center),

color = 'orange', size = 4) +

ggtitle(sprintf('Iteration %0.0f, Error: %0.4f', iteration,

total\_error[iteration]))

print(p)

}