HW 15

Ines Pancorbo

1.

```
library("ggplot2")
```

```
sequences <- as.matrix(read.csv("matrix_sequences.csv", header = F,
                           as.is = TRUE, colClasses = "factor"))
colnames(data) <- NULL
rownames(data) <- NULL

reference <- as.matrix(read.csv("matrix_reference.csv", header = F,
                                as.is = TRUE, colClasses = "factor"))
colnames(reference) <- NULL
rownames(reference) <- NULL

countries <- as.matrix(read.csv("matrix_countries.csv", header = F,
                                as.is = TRUE, colClasses = "factor"))
```

```
dim(sequences)
```

```
## [1]   241 29903
```

```
dim(reference)
```

```
## [1]     1 29903
```

```
dim(countries)
```

```
## [1] 241   1
```

```
# If a sequence has the same letter at a given position as
# the reference, set the matrix entry to 1, otherwise set it to 0
X <- t(apply(sequences, 1, function(x) as.numeric(x == reference)))
dim(X)
```

```
## [1]   241 29903
```

```
# Let us center the sequences
X <- t(apply(X, 1, function(x) x-colMeans(X)))
```

```
norm <- function(x) sqrt(sum(x^2))

# power iteration approach
power_iteration <- function(X, start) {
  start1 <- matrix(start[,1],nrow = ncol(X), ncol = 1)/norm(start[,1])
  start2 <- matrix(start[,2],nrow = ncol(X), ncol = 1)/norm(start[,2])
  # not computing X^TX
  RQ1 <- t(start1) %*% (t(X) %*% (X %*% start1))
```

```
  RQ2 <- t(start2) %*% (t(X) %*% (X %*% start2))

  repeat {

    start <- t(X) %*% (X %*% start)
    start <- qr.Q(qr(start))

    start1 <- matrix(start[,1],nrow = ncol(X), ncol = 1)/norm(start[,1])
    start2 <- matrix(start[,2],nrow = ncol(X), ncol = 1)/norm(start[,2])

    new_RQ1 <- t(start1) %*% (t(X) %*% (X %*% start1))
    new_RQ2 <- t(start2) %*% (t(X) %*% (X %*% start2))

    if (abs(new_RQ1 - RQ1) < 10^-10 && abs(new_RQ2 - RQ2) < 10^-10){
      break
    }
    else{
      RQ1 <- new_RQ1
      RQ2 <- new_RQ2
    }
  }
  return (list(principal_components = start, lambda1 = RQ1, lambda2 = RQ2))
}
```

```
# let us get the first two principal components of X, i.e.,
# two most dominant eigenvectors of X^TX
start <- matrix(runif(ncol(X)*2), nrow=ncol(X), ncol=2)
start_time <- Sys.time()
power_iteration_result <- power_iteration(X, start)
end_time <- Sys.time()
end_time  - start_time
```

```
## Time difference of 9.771214 secs
```

```
pcs <- power_iteration_result$principal_components
```

Let us check we got the correct first two principal components of X by computing the SVD of X. The column vectors of the matrix V are the eigenvectors of X^TX (from most dominant to least), and consequently the first two column vectors of V are the first two principal components of X. Also note that computing the SVD takes less than doing power iteration. Let us check that the eigenvalues calculated via power iteration are the same as the ones given by R's svd().

```
start_time <- Sys.time()
svd.X <- svd(X)
end_time <- Sys.time()
end_time  - start_time
```

```
## Time difference of 5.132273 secs
```

```
# the two most dominant eigenvalues of X^TX are (using power iteration)
power_iteration_result$lambda1
```

```
##          [,1]
## [1,] 319.9156
```

```
power_iteration_result$lambda2
```

```
##          [,1]
## [1,] 120.3299
```

```
# the two most dominant eigenvalues of X^TX are (using R's svd() function)
svd.X$d[1]^2
```

```
## [1] 319.9156
```

```
svd.X$d[2]^2
```

```
## [1] 120.3299
```

Let's determine the fraction of the variance captured by the 2-d PCA. We could do this in a couple different ways. Here are two:

```
# Let the 2-d projection of X be X2.
# We could compute the frobenius norm of X-X2 and X
# Let's get X2 first
start_time <- Sys.time()
X2 <- X %*% pcs %*% t(pcs)
fnorm <- function(x) sum(x^2)
# variance captured by the 2-d PCA is
1 - fnorm(X-X2)/fnorm(X)
```

```
## [1] 0.3214464
```

```
# the first pc captures
1 - fnorm(X-(X %*% pcs[,1] %*% t(pcs[,1])))/fnorm(X)
```

```
## [1] 0.2335872
```

```
# the second pc captures
1 - fnorm(X-(X %*% pcs[,2] %*% t(pcs[,2])))/fnorm(X)
```

```
## [1] 0.08785917
```

```
end_time <- Sys.time()
end_time  - start_time
```

```
## Time difference of 0.457526 secs
```

```
# We could use R's svd() function and look at the singular values
# We want the eigenvalues of X^TX, which are the squares of the singular values of X
# And so the variance captured by the 2-d PCA is
start_time <- Sys.time()
sum(svd.X$d[1:2]^2)/sum(svd.X$d^2)
```

```
## [1] 0.3214464
```

```
# the first pc captures
svd.X$d[1]^2/sum(svd.X$d^2)
```

```
## [1] 0.2335872
```

```
# the second pc captures
svd.X$d[2]^2/sum(svd.X$d^2)
```

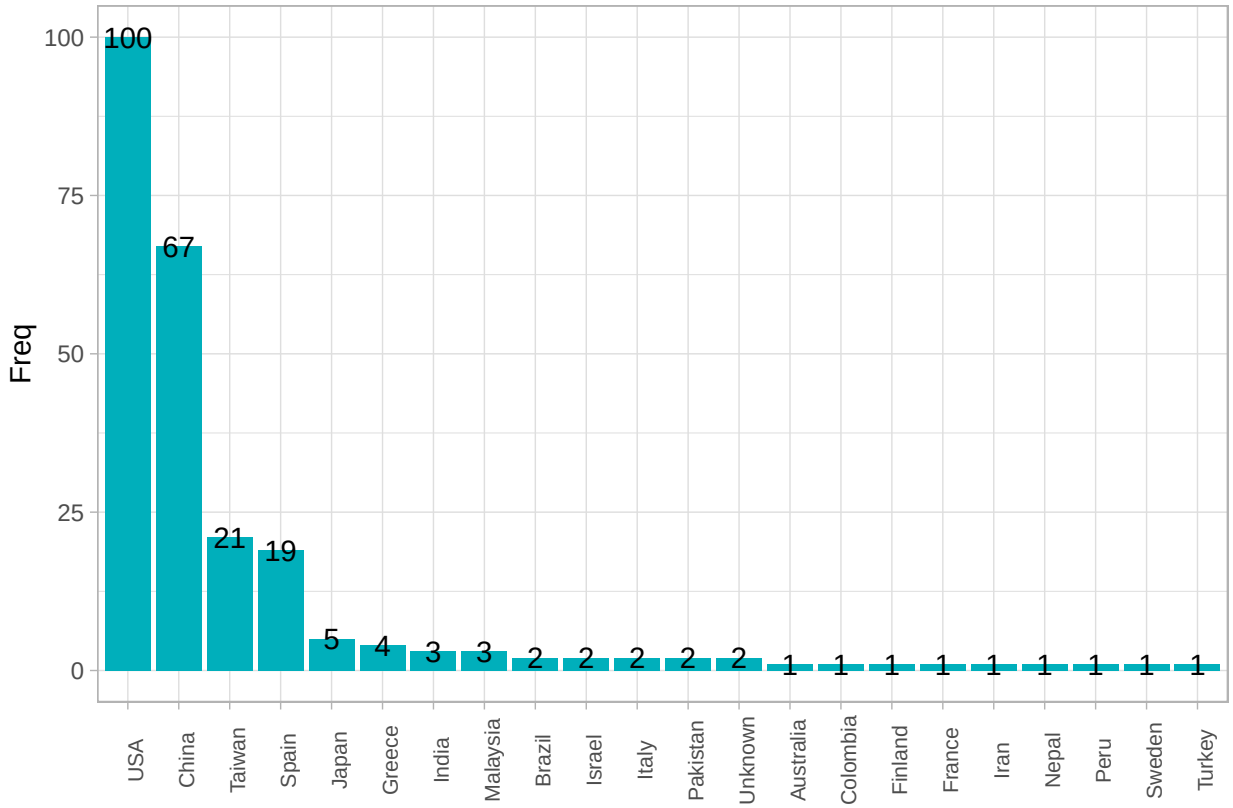
```
## [1] 0.08785917
```

```
end_time <- Sys.time()
end_time  - start_time
```

```
## Time difference of 0.003633976 secs
```

The 2-d PCA captures approx 32.14% of the variance (approx 23.36% by pc1 and approx 8.79% by pc2). It takes less time using R's svd() function.

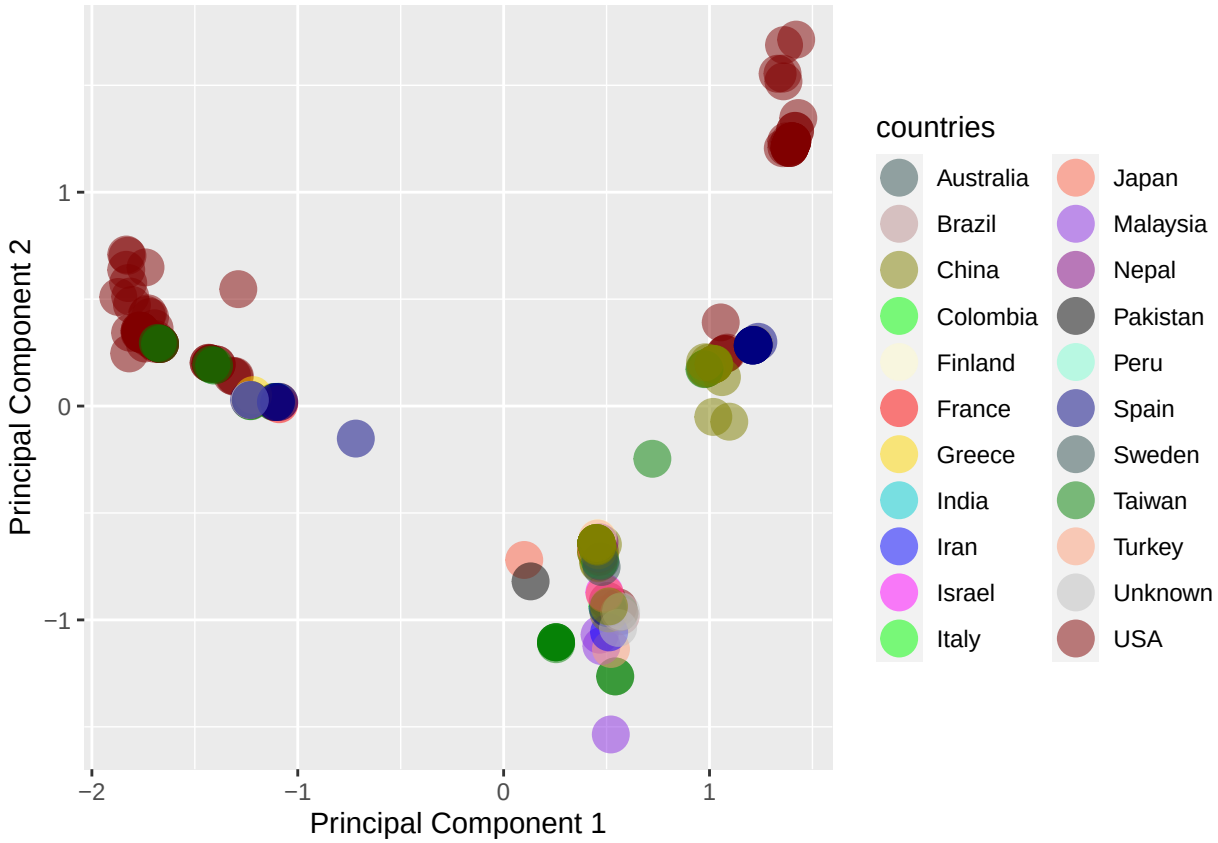
```
# Before projecting let's look at how many genome samples
# per country
freq_table <- data.frame(table(countries))
ggplot(freq_table, aes(x = reorder(countries, -Freq), y = Freq)) +
  geom_col(fill = "#00AFBB")+
  geom_text(aes(label = Freq)) +
  xlab("") +
  theme_light() +
  theme(axis.text.x = element_text(size = 8, angle = 90),
        legend.position = "none")
```




```
# projecting onto the two most dominant principal components
# getting coordinates
c <- X %*% pcs

data <- data.frame(x = c[,1], y = c[,2], country = c(countries))
```

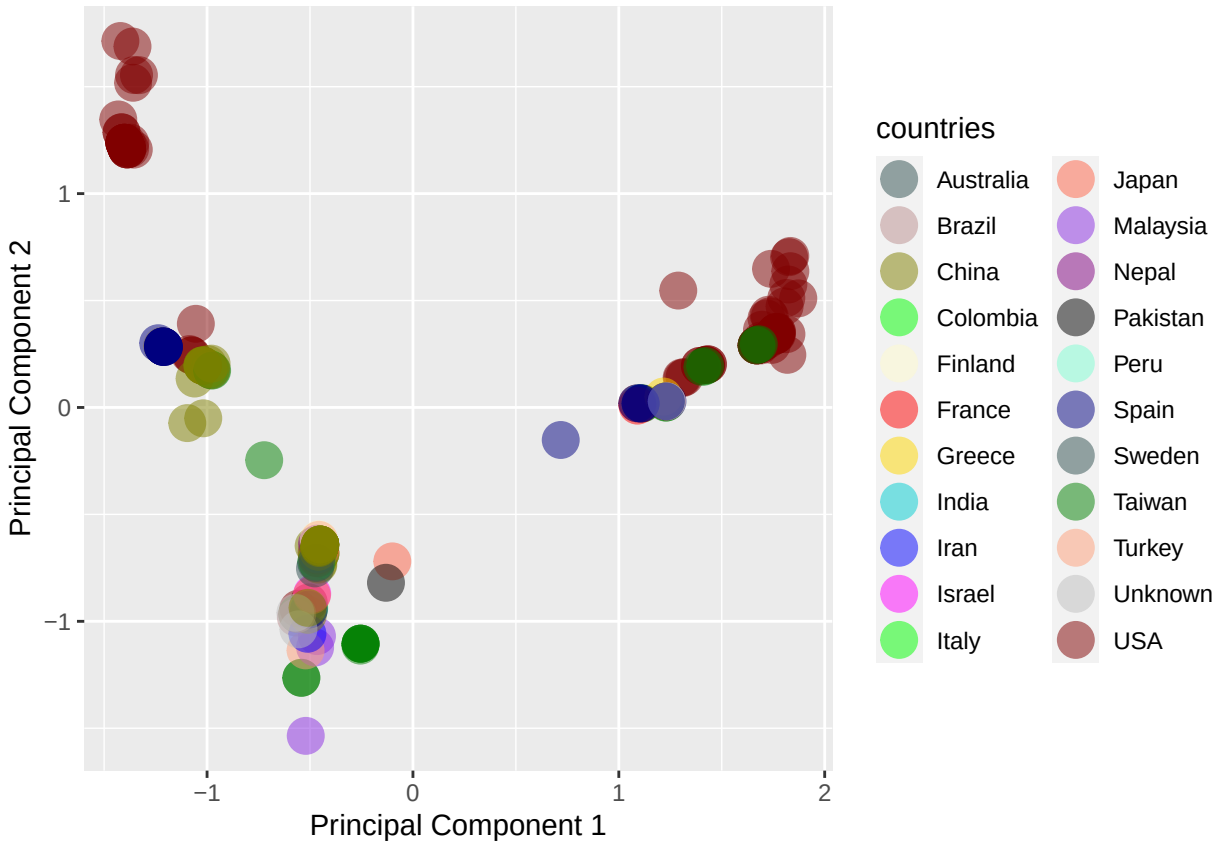
```
# let's create a color palette because R's default is too light
# make the countries with > 1 genome data point more visible
countries_sorted <- as.vector(freq_table[order(freq_table[,2], decreasing = T),][,1])
colors <- c("#800000", "#808000", "#008000", "#000080", "#FF6347",
            "#FFD700", "#00CED1", "#8A2BE2", "#BC8F8F", "#FF00FF", "#00FF00",
            "#000000", "#C0C0C0", "#2F4F4F", "#00FF00", "#FFFACD", "#FF0000",
            "#0000FF", "#800080", "#7FFFD4", "#2F4F4F", "#FFA07A")

names(colors) <- countries_sorted
```

```
ggplot(data) +
  geom_point(aes(x = x, y = y, color = country), alpha = 0.5, size = 6) +
  scale_colour_manual("countries", values = colors) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```



Let's check that we get the same plot using R's svd().

```
# gettting the two most dominant principal components
pcs_svd <- svd.X$v[,1:2]

# projecting onto the two most dominant principal components
# getting coordinates
c_svd <- X %*% pcs_svd

data_svd <- data.frame(x = c_svd[,1], y = c_svd[,2], country = c(countries))

ggplot(data_svd) +
  geom_point(aes(x = x, y = y, color = country), alpha = 0.5, size = 6) +
  scale_colour_manual("countries", values = colors) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```



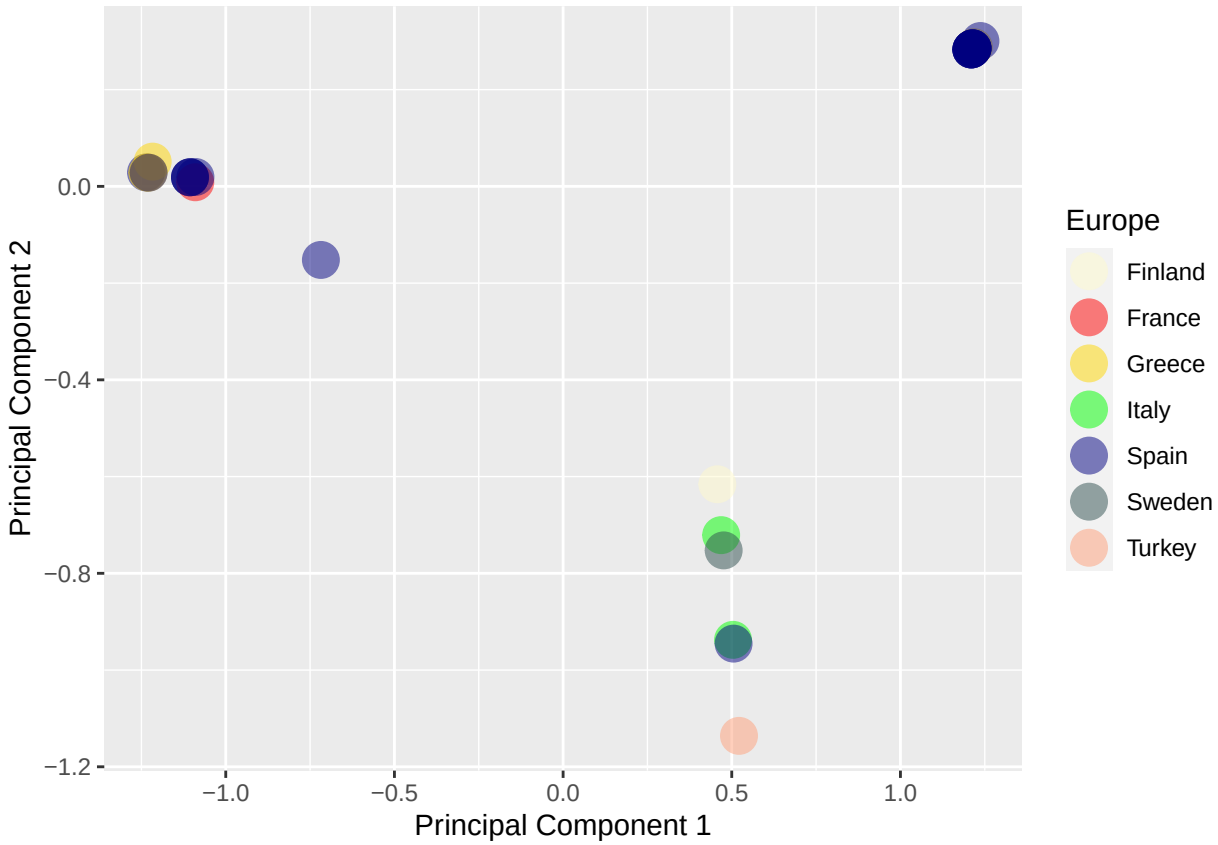
We get the same plot (reflected, but eigenvectors are not unique, they are unique up to a scalar).

Comment on the implications for how the virus spread.

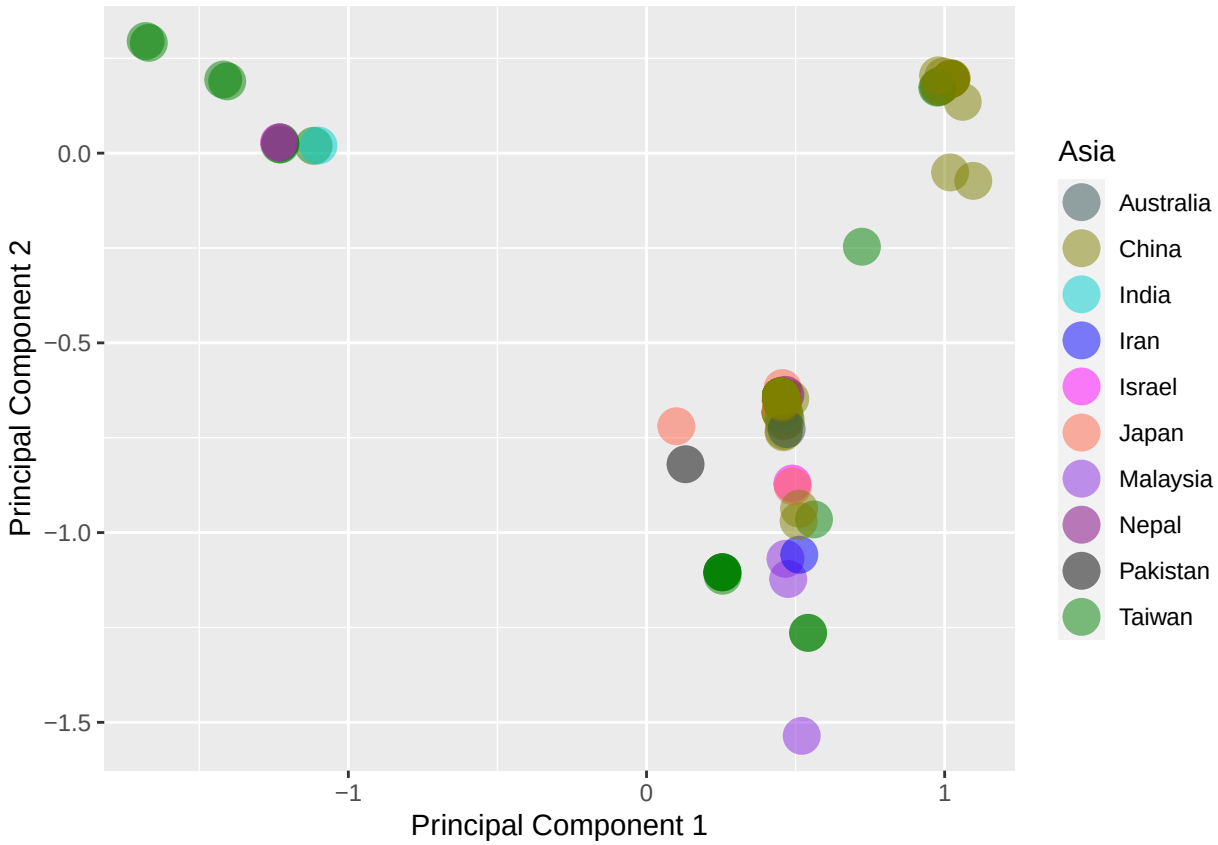
Some observations from the plot: It looks like there are four clusters. Patients from China are present in two of them. Patients from the US dominate two of them as well but are more spead out. As seen from the barplot, we have many data points corresponding to patients from Taiwan and Spain too, and in both cases they are, for the most part, spread out.

Let's plot according to geographic region.

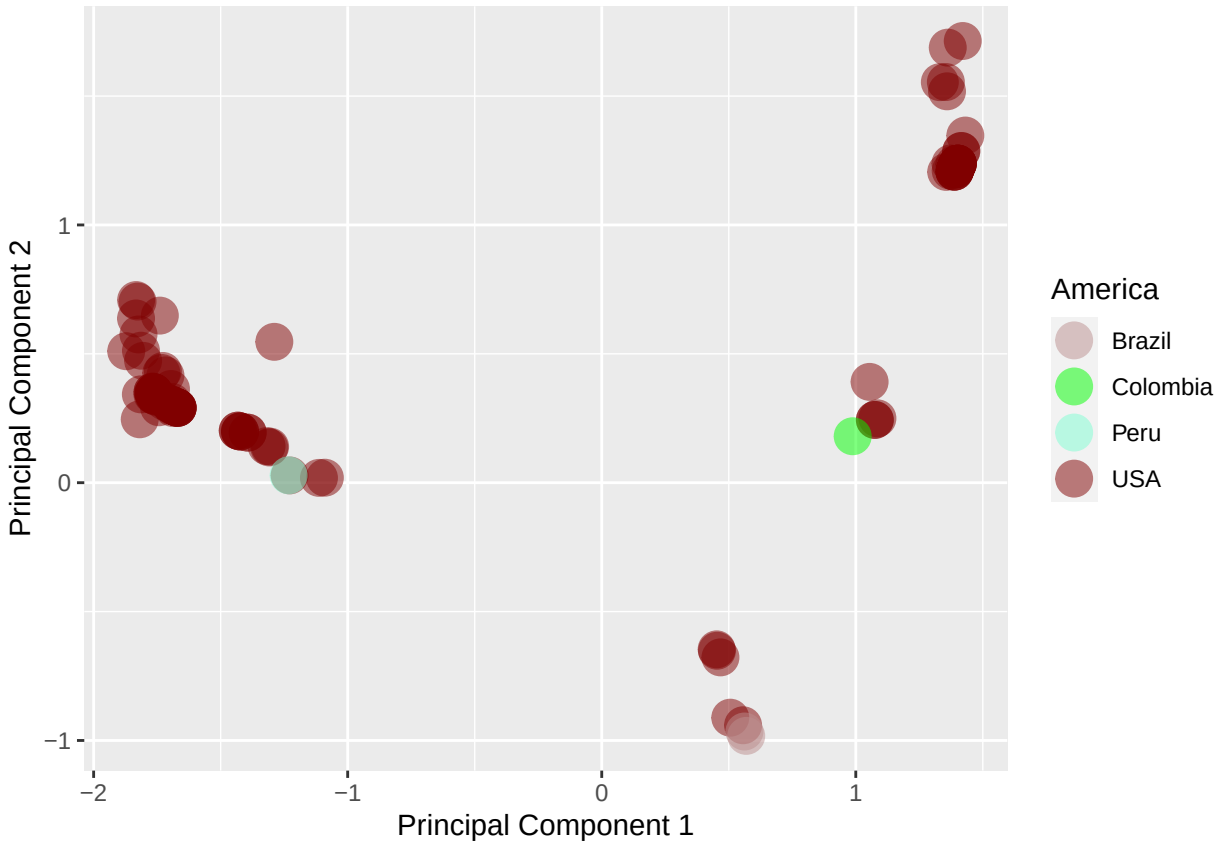
```
data_europe <- subset(data, country %in%
                        c("Finland", "France", "Greece", "Italy", "Spain",
                          "Sweden", "Turkey"))
data_asia <- subset(data, country %in%
                      c("China", "India", "Iran", "Israel", "Japan",
                        "Malaysia", "Nepal", "Pakistan", "Taiwan", "Australia"))
data_america <- subset(data, country %in% c("Brazil", "Colombia", "USA", "Peru"))

ggplot(data_europe) +
  geom_point(aes(x = x, y = y, color = country), alpha = 0.5, size = 6) +
  scale_colour_manual("Europe", values = colors) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```




```
ggplot(data_asia) +
  geom_point(aes(x = x, y = y, color = country), alpha = 0.5, size = 6) +
  scale_colour_manual("Asia", values = colors) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```




```
ggplot(data_america) +
  geom_point(aes(x = x, y = y, color = country), alpha = 0.5, size = 6) +
  scale_colour_manual("America", values = colors) +
  xlab("Principal Component 1") + ylab("Principal Component 2")
```



From the plots above: For the most part, for those countries with more than 1 datapoint, we can see that the corresponding sequences are mapped closely. This is true for Brazil, Malaysia, Japan, Italy. Sequences belonging to patients from countries present in various clusters (US, Taiwan, China, US) are generally mapped closely within the cluster (US and China are clear examples).

It doesn't look like sequences corresponding to patients from countries close geographically are mapped closely. For example, sequences from patients in Brazil are mapped closely to sequences from patients in Spain or Turkey and Taiwan or Malaysia. This is probably because of tourism and being able to fly in/out of countries.

2.

2. (a)

Since we want $w \ni w^T\phi(x)$ is conformable, it must be that $w \in R^m$. So dimension is m.

We have data $(x^{(i)}, y_i)$ where $x^{(i)} \in R^n, i = 1, 2, \cdots, N$.

We define features through the mapping $\phi(x) : R^n \to R^m$ and so we have data $(\phi(x^{(i)}), y_i)$.

We are given that $P(y_i = 1|w, \phi(x^{(i)})) = \frac{1}{1+e^{-w^T\phi(x^{(i)})}}$.

We can assume independence and we can think of each of the response y_i as conditionally Bernoulli distributed. So we have $Y_i|\phi(x^{(i)}) \sim \text{Bernoulli}(p)$ where $p = P(Y_i = 1|\phi(x^{(i)}); w) = \frac{1}{1+e^{-w^T\phi(x^{(i)})}}$

This implies

$$f_{Y_i|\phi(x^{(i)})}(y_i) = p^{y_i}(1-p)^{1-y_i}$$

$$= P(Y_i = 1|\phi(x^{(i)}); w)^{y_i}(1 - P(Y_i = 1|\phi(x^{(i)}); w)^{1-y_i}$$

$$= (\frac{1}{1+e^{-w^T\phi(x^{(i)})}})^{y_i}(1-\frac{1}{1+e^{-w^T\phi(x^{(i)})}})^{1-y_i} \quad \text{for} \quad y_i \in \{0,1\} \quad \text{and} \quad 0 \quad \text{otherwise.}$$

Consequently, the likelihood function is

$$L(w) = \prod_{i=1}^{N} p^{y_i}(1-p)^{1-y_i}$$

$$= \prod_{i=1}^{N} (\frac{1}{1+e^{-w^T\phi(x^{(i)})}})^{y_i}(1-\frac{1}{1+e^{-w^T\phi(x^{(i)})}})^{1-y_i}.$$

Thus, the log likelihood function is

$$\log L(w) = \sum_{i=1}^{N} (y_i \log(\frac{1}{1+e^{-w^T\phi(x^{(i)})}}) + (1-y_i)\log(1-\frac{1}{1+e^{-w^T\phi(x^{(i)})}}))$$

$$= \sum_{i=1}^{N} (y_i\log(1) - y_i\log(1+e^{-w^T\phi(x^{(i)})}) + (1-y_i)\log(\frac{e^{-w^T\phi(x^{(i)})}}{1+e^{-w^T\phi(x^{(i)})}}))$$

$$= \sum_{i=1}^{N} (-y_i\log(1+e^{-w^T\phi(x^{(i)})}) + (1-y_i)(-w^T\phi(x^{(i)})) - (1-y_i)\log(1+e^{-w^T\phi(x^{(i)})})$$

$$= \sum_{i=1}^{N} ((1-y_i)(-w^T\phi(x^{(i)})) + (-1+y_i-y_i)\log(1+e^{-w^T\phi(x^{(i)})}))$$

$$= \sum_{i=1}^{N} ((1-y_i)(-w^T\phi(x^{(i)})) - \log(1+e^{-w^T\phi(x^{(i)})})).$$

2. (b) i.

Consider $E = \text{span}(\phi(x^{(1)}), \phi(x^{(2)}), \cdots, \phi(x^{(N)}))$. Then $E \subset R^m$. And any $v \in R^m$ can be represented as $u + z$, where $u \in E$ and $z \in E^{\perp}$. u will be the orthogonal projection of v onto E and $z = v - u$ will be the component of v orthogonal to E.

Let B be the model matrix for the features of the sampled data. Considering the above, we can write the w in the log likelihood, which is in R^m, as $B^T a + z$, where $B^T a$ is the orthogonal projection of w onto E and z is the component of w orthogonal to E.

Let us write $w = B^T a + z$. We want to show that we can write the w that maximizes the log likelihod, w^*, as $w^* = B^T a^*$ for some $a^* \in R^N$. In other words, that $w^* \in E$.

Consider the maximization problem:

$$\max_{w \in R^m} \sum_{i=1}^{N} ((1-y_i)(-w^T\phi(x^{(i)})) - \log(1+e^{-w^T\phi(x^{(i)})}))$$

$$= \max_{w \in R^m} \sum_{i=1}^{N} ((1-y_i)(-\phi(x^{(i)})^T w) - \log(1+e^{-\phi(x^{(i)})^T w}))$$

Let us substitute w for $B^T a + z$ and consider the maximization problem again. Note $a \in R^N, z \in R^m$.

$$\max_{a\in R^N, z\in R^m} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^T(B^Ta+z)) - \log(1+e^{-\phi(x^{(i)})^T(B^Ta+z)}))$$

$$= \max_{a\in R^N, z\in R^m} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^TB^Ta - \phi(x^{(i)})^Tz) - \log(1+e^{-\phi(x^{(i)})^TB^Ta-\phi(x^{(i)})^Tz}))$$

Now, since z is orthogonal to each of the $\phi(x^{(i)})$ we get the following simplification:

$$= \max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^TB^Ta) - \log(1+e^{-\phi(x^{(i)})^TB^Ta}))$$

$$= \max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-(B\phi(x^{(i)}))^Ta) - \log(1+e^{-(B\phi(x^{(i)}))^Ta}))$$

$$= \max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-a^TB\phi(x^{(i)})) - \log(1+e^{-a^TB\phi(x^{(i)})}))$$

So the above shows that in terms of the maximization problem, z plays no role. So $w^* = B^Ta^*$ for some $a^* \in R^N$. Thus, $w^* \in E = \text{span}(\phi(x^{(1)}), \phi(x^{(2)}), \cdots, \phi(x^{(N)}))$.

Aside: More compactly you could write the summation above as,

$$-(1-y)^TBB^Ta - \log(1+e^{-BB^Ta})$$

2. (b) ii.

From above, we have the maximization problem reduced to:

$$\max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-a^TB\phi(x^{(i)})) - \log(1+e^{-a^TB\phi(x^{(i)})}))$$

Now consider the $N \times N$ matrix BB^T,

$$BB^T = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{pmatrix} \begin{pmatrix} \phi(x^{(1)}) & \phi(x^{(2)}) & \cdots & \phi(x^{(N)}) \end{pmatrix}$$

$$= \begin{pmatrix} \phi(x^{(1)})^T\phi(x^{(1)}) & \phi(x^{(1)})^T\phi(x^{(2)}) & \cdots & \phi(x^{(1)})^T\phi(x^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(x^{(N)})^T\phi(x^{(1)}) & \phi(x^{(N)})^T\phi(x^{(2)}) & \cdots & \phi(x^{(N)})^T\phi(x^{(N)}) \end{pmatrix}$$

Then from above, BB^T is a $N \times N$ kernel matrix since each entry $BB^T_{jl} = \phi(x^{(j)})^T\phi(x^{(l)})$. Denote BB^T by K. Each entry K_{jl} of K is then $\phi(x^{(j)})^T\phi(x^{(l)})$.

Now, the ith column, $k^{(i)}$, of K is the ith column, $(BB^T)_i$, of BB^T,

$$k^{(i)} = (BB^T)_i = \begin{pmatrix} \phi(x^{(1)})^T\phi(x^{(i)}) \\ \phi(x^{(2)})^T\phi(x^{(i)}) \\ \phi(x^{(3)})^T\phi(x^{(i)}) \\ \vdots \\ \phi(x^{(N)})^T\phi(x^{(i)}) \end{pmatrix} = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{pmatrix} \phi(x^{(i)}) = B\phi(x^{(i)})$$

And so substituting $B\phi(x^{(i)})$ with $k^{(i)}$ we have,

$$\max_{a \in R^N} \sum_{i=1}^{N} ((1-y_i)(-a^T B\phi(x^{(i)})) - \log(1 + e^{-a^T B\phi(x^{(i)})}))$$

$$= \max_{a \in R^N} \sum_{i=1}^{N} ((1-y_i)(-a^T k^{(i)}) - \log(1 + e^{-a^T k^{(i)}}))$$

Aside: More compactly you could write the summation above as,

$$-(1-y)^T Ka - \log(1 + e^{-Ka})$$

2. (b) iii.

Given $x \in R^n$, we know

$$P(y = 1 | w^*, \phi(x)) = \frac{1}{1 + e^{-(w^*)^T \phi(x)}}.$$

In ii. we showed that $w^* = B^T a^*$ and so,

$$P(y = 1 | a^*, \phi(x)) = \frac{1}{1 + e^{-(B^T a^*)^T \phi(x)}}$$

$$= \frac{1}{1 + e^{-(a^*)^T B\phi(x)}}$$

Now,

$$B\phi(x) = \begin{pmatrix} \phi(x^{(1)})^T \\ \phi(x^{(2)})^T \\ \phi(x^{(3)})^T \\ \vdots \\ \phi(x^{(N)})^T \end{pmatrix} \phi(x) = \begin{pmatrix} \phi(x^{(1)})^T\phi(x) \\ \phi(x^{(2)})^T\phi(x) \\ \phi(x^{(3)})^T\phi(x) \\ \vdots \\ \phi(x^{(N)})^T\phi(x) \end{pmatrix}$$

And so by denoting $B\phi(x) \in R^N$ as $\tilde{k}$ (from above one can see that the ith entry, $\tilde{k}_i$, of $\tilde{k}$ is $\phi(x^{(i)})^T\phi(x)$) we have:

$$P(y = 1 | a^*, \phi(x)) = \frac{1}{1 + e^{-(a^*)^T \tilde{k}}}$$

2. (c)

(3) and (4) would be advantageous when $m >> N$ and N is not unreasonably "large" (otherwise, we could resort to Neural Nets).

Note, we saw that we would have to solve the following maximization problem:

$$\max_{w \in R^m} \sum_{i=1}^{N} ((1-y_i)(-\phi(x^{(i)})^T w) - \log(1 + e^{-\phi(x^{(i)})^T w}))$$

which is equivalent to maximizing for w

$$-(1-y)^T Bw - \log(1 + e^{-Bw})$$

The above requires being able to store B. This implies being able to store a $N \times m$ matrix. If $m >> N$, we might not be able to store this matrix or even compute, in a reasonable amount of time, Bw which requires Nm multiplications.

So in this case, resorting to (3) (and by extension, for prediction, (4)), i.e.,

$$\max_{a \in R^N} \sum_{i=1}^{N} ((1-y_i)(-a^T k^{(i)}) - \log(1 + e^{-a^T k^{(i)}}))$$

which is equivalent to maximizing for a

$$-(1-y)^T Ka - \log(1 + e^{-Ka})$$

is advantageous since it requieres storing K, which is $N \times N$, and computing Ka which requires N^2 multiplications, which is less than Nm multiplications.

2. (d) i.

Consider,

$$g_i(w) = (1-y_i)(w^T \phi(x^{(i)})) + \log(1 + e^{-w^T \phi(x^{(i)})})$$

From HW 7, we know the Hessian of $g_i(w)$ is,

$$\text{H } g_i(w) = \phi(x^{(i)})\phi(x^{(i)})^T \big[\frac{e^{-w^T \phi(x^{(i)})}}{(1 + e^{-w^T \phi(x^{(i)})})^2}\big]$$

The above is using what we showed in HW 7, 3.b.ii), substituting α with w and the vector $\tilde{x_i}$ with the vector $\phi(x^{(i)})$.

Then given any $x \in R^m$,

$$\begin{aligned} & x^T \text{H } g_i(w) x \\ &= (\frac{e^{-w^T \phi(x^{(i)})}}{(1 + e^{-w^T \phi(x^{(i)})})^2}) x^T (\phi(x^{(i)})\phi(x^{(i)})^T) x \\ &= (\frac{e^{-w^T \phi(x^{(i)})}}{(1 + e^{-w^T \phi(x^{(i)})})^2}) (\phi(x^{(i)})^T x)(\phi(x^{(i)})^T x) \\ &= (\frac{e^{-w^T \phi(x^{(i)})}}{(1 + e^{-w^T \phi(x^{(i)})})^2}) (\phi(x^{(i)})^T x)^2 \geq 0 \end{aligned}$$

And so $Hg_i(w)$ is positive semi-definite $\implies$ $g_i(w)$ is convex. By further results of HW 7, $\sum_{i=1}^{N} g_i(w)$ is convex too since it is a finite sum of convex functions.

In addition, since $\lambda > 0$, the quadratic function of w, $\lambda||w||^2 = \lambda w^T I w \geq 0$ is convex as well.

So the sum,

$$\sum_{i=1}^{N} g_i(w) + \lambda||w||^2$$

is convex. This implies,

$$-(\sum_{i=1}^{N} g_i(w) + \lambda||w||^2)$$

$$= -(\sum_{i=1}^{N}((1-y_i)(w^T\phi(x^{(i)})) + \log(1+e^{-w^T\phi(x^{(i)})}) + \lambda||w||^2)$$

$$= \sum_{i=1}^{N}((1-y_i)(-w^T\phi(x^{(i)})) - \log(1+e^{-w^T\phi(x^{(i)})}) - \lambda||w||^2$$

is concave (by further results of HW 7, since we have the negative of a convex function).

2. (d) ii.

Consider,

$$\max_{w\in R^m} \sum_{i=1}^{N}((1-y_i)(-w^T\phi(x^{(i)})) - \log(1+e^{-w^T\phi(x^{(i)})})) - \lambda||w||^2$$

$$= \max_{w\in R^m} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^T w) - \log(1+e^{-\phi(x^{(i)})^T w})) - \lambda||w||^2$$

Let us again write $w = B^T a + z$ and make the substitution,

$$\max_{a\in R^N, z\in R^m} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^T(B^T a+z)) - \log(1+e^{-\phi(x^{(i)})^T(B^T a+z)})) - \lambda(B^T a+z)^T(B^T a+z)$$

$$\max_{a\in R^N, z\in R^m} \sum_{i=1}^{N}((1-y_i)(-\phi(x^{(i)})^T B^T a) - \log(1+e^{-\phi(x^{(i)})^T B^T a})) - \lambda(a^T BB^T a + a^T Bz + z^T B^T a + z^T z)$$

$$\max_{a\in R^N, z\in R^m} \sum_{i=1}^{N}((1-y_i)(-a^T B\phi(x^{(i)})) - \log(1+e^{-a^T B\phi(x^{(i)})})) - \lambda a^T BB^T a - \lambda z^T z$$

Now, since we are maximizing and for any z, $\lambda z^T z \geq 0$, we must choose $z^* = 0$. And so we have,

$$\max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-a^T B\phi(x^{(i)})) - \log(1+e^{-a^T B\phi(x^{(i)})})) - \lambda a^T BB^T a$$

Now, consider BB^T, shown in 2.b.ii) to be a $N \times N$ kernel matrix, and denote it K. As also shown in 2.b.ii), the ith column, $k^{(i)}$, of K is the ith column of BB^T, which is $B\phi(x^{(i)})$. And so, making the respective substitutions we get,

$$\max_{a\in R^N} \sum_{i=1}^{N}((1-y_i)(-a^T k^{(i)}) - \log(1+e^{-a^T k^{(i)}})) - \lambda a^T Ka$$

3.

3. (a)

Let $x, x' \in R^2$ be given. Consider $\phi(x) \cdot \phi(x')$,

$$\begin{aligned}
&\phi(x) \cdot \phi(x') \\
&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (1, \sqrt{2}x_1', \sqrt{2}x_2', x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \\
&= 1 + 2x_1x_1' + 2x_2x_2' + x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 \\
&= 1 + 2(x_1x_1' + x_2x_2') + (x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2) \\
&= 1 + 2(x_1x_1' + x_2x_2') + (x_1x_1' + x_2x_2')^2 \\
&= (1 + (x_1x_1' + x_2x_2'))^2 \\
&= (1 + x \cdot x')^2.
\end{aligned}$$

3. (b)

Consider the kernel $K(x, x') = (1 + x \cdot x')^3$. By the binomial expansion we have,

$$\begin{aligned}
&(1 + x \cdot x')^3 \\
&= 1 + 3x \cdot x' + 3(x \cdot x')^2 + (x \cdot x')^3 \\
&= 1 + 3(x_1x_1' + x_2x_2') + (x \cdot x')^2(3 + x \cdot x') \\
&= 1 + 3(x_1x_1' + x_2x_2') + (x_1x_1' + x_2x_2')^2(3 + x_1x_1' + x_2x_2') \\
&= 1 + 3x_1x_1' + 3x_2x_2' + (x_1^2x_1'^2 + x_2^2x_2'^2 + 2x_1x_1'x_2x_2')(3 + x_1x_1' + x_2x_2') \\
&= 1+3x_1x_1'+3x_2x_2'+3x_1^2x_1'^2+3x_2^2x_2'^2+6x_1x_1'x_2x_2'+x_1^3x_1'^3+x_2^2x_2'^2x_1x_1'+2x_1^2x_1'^2x_2x_2'+x_1^2x_1'^2x_2x_2'+x_2^3x_2'^3+2x_1x_1'x_2^2x_2'^2 \\
&= 1 + 3x_1x_1' + 3x_2x_2' + 3x_1^2x_1'^2 + 3x_2^2x_2'^2 + 6x_1x_1'x_2x_2' + x_1^3x_1'^3 + 3x_2^2x_2'^2x_1x_1' + 3x_1^2x_1'^2x_2x_2' + x_2^3x_2'^3 \\
&= (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{3}x_1^2, \sqrt{3}x_2^2, \sqrt{6}x_1x_2, x_1^3, \sqrt{3}x_1x_2^2, \sqrt{3}x_1^2x_2, x_2^3) \cdot \\
&(1, \sqrt{3}x_1', \sqrt{3}x_2', \sqrt{3}x_1'^2, \sqrt{3}x_2'^2, \sqrt{6}x_1'x_2', x_1'^3, \sqrt{3}x_1'x_2'^2, \sqrt{3}x_1'^2x_2', x_2'^3) \\
&= \phi(x) \cdot \phi(x')
\end{aligned}$$

if we define $\phi(x) = (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{3}x_1^2, \sqrt{3}x_2^2, \sqrt{6}x_1x_2, x_1^3, \sqrt{3}x_1x_2^2, \sqrt{3}x_1^2x_2, x_2^3)$.

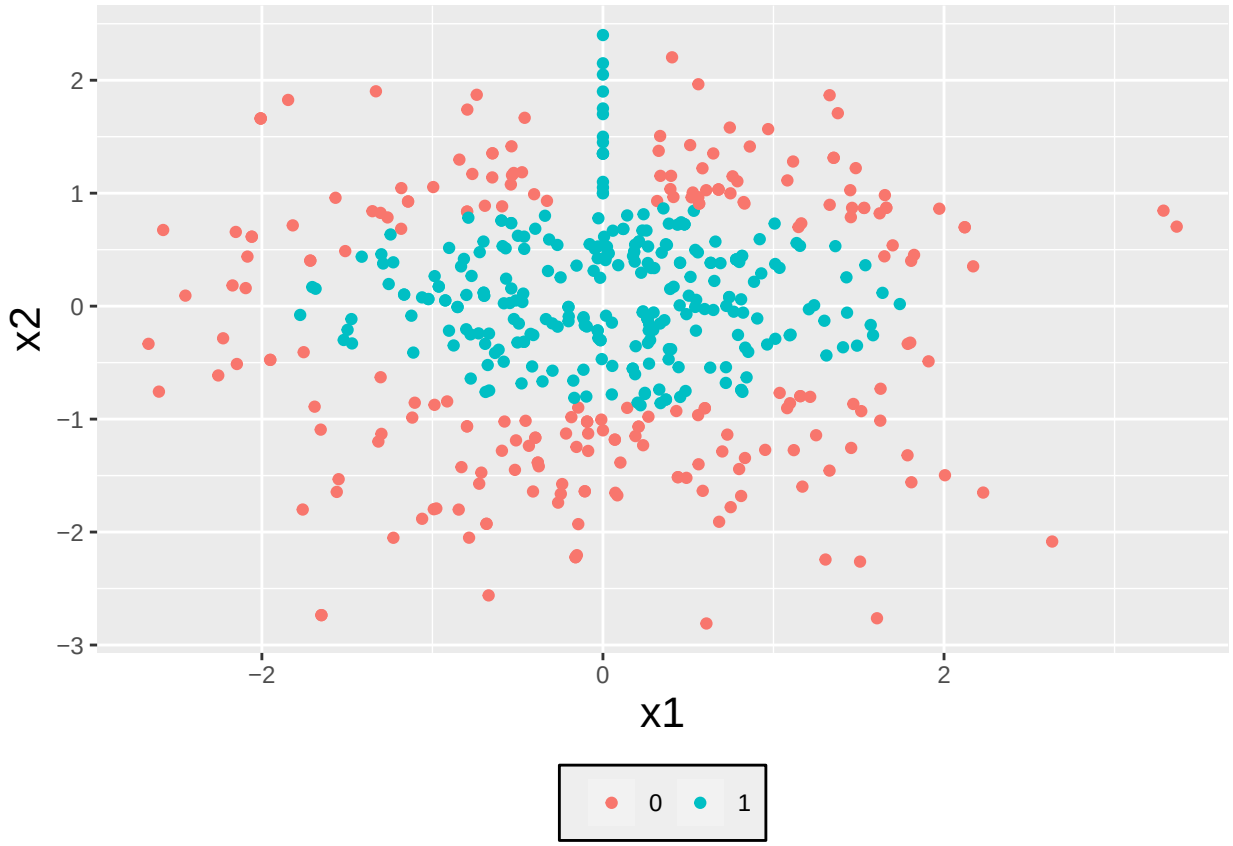
So the feature vector for any $x \in R^2$ is defined by $\phi(x) : R^2 \to R^{10}$,

$$\phi(x) = (1, \sqrt{3}x_1, \sqrt{3}x_2, \sqrt{3}x_1^2, \sqrt{3}x_2^2, \sqrt{6}x_1x_2, x_1^3, \sqrt{3}x_1x_2^2, \sqrt{3}x_1^2x_2, x_2^3)$$

3. (c) and (d)

```r
set.seed(2020)
data <- read.csv("nn.txt", header = TRUE, sep = ' ', stringsAsFactors = TRUE)

# choosing 500 data points
r <- sort(round(runif(500, min = 1, max = (nrow(data)-1))))
X <- as.matrix(data[r,1:2])
y <- as.matrix(data$y[r])
```

```r
# Let us plot our data
library(ggplot2)
ggplot() +
  geom_point(aes(x = X[,1], y = X[,2], color = as.factor(y))) +
  xlab("x1") + ylab("x2") +
  theme(legend.position = "bottom",
        legend.background = element_rect(fill = "#EEEEEE", color = "black"),
        legend.title = element_blank(),
        axis.title = element_text(size = 16))
```




```r
# construct the kernel matrix
power <- 2
K <- (1 + X %*% t(X))^power

# log likelihood function
log_L <- function(a, K, y, lambda){
  return(sum(-(1-y)*(K %*% a) - log(1 + exp(-K %*% a)))
  - lambda*t(a)%*%K%*%a)
}

# gradient of log likelihood
grad_logL <- function(a, K, y, lambda) {
  total <- K * as.vector(y - 1) +
    K * as.vector(exp(-K %*% a)/(1+exp(-K %*% a)))
  grad <- colSums(total) - lambda*2*K%*%a
  return(grad)
}
```

For the hessian calculation let us put everything in vector form. The hessian is

$$\sum_{i=1}^{N}(-k^{(i)}(k^{(i)})^T[\frac{e^{-a^Tk^{(i)}}}{(1+e^{-a^Tk^{(i)}})^2}]) - 2\lambda K$$

Which is equivalent to,

$$-KDK - 2\lambda K$$

where K is the kernel matrix mentioned above and D is a diagonal matrix with,

$$d_{ii} = \frac{e^{-a^Tk^{(i)}}}{(1+e^{-a^Tk^{(i)}})^2}$$

```
# hessian of log likelihood
hessian_logL <- function(a, K, lambda){
  d <- exp(-K %*% a)/(1+exp(-K %*% a))^2
  D <- diag(c(d))
  hessian <- -K %*% D %*% K - lambda*2*K
  return(hessian)
}

##########################################
# let's used Damped Newton's Method      #
# Damped Newton's Method will converge   #
# since the log likelihood is concave and #
# so will pick ascent directions         #
##########################################

norm <- function(x) sqrt(sum(x^2))

damped_NM <- function(a, K, y, lambda, eps = 1e-10, print = F, graph = F) {

  g <- grad_logL(a, K, y, lambda)
  i <- 1
  s <- 1
  I <- diag(length(g))
  x_values <- c()
  y_values <- c()

  while(norm(g) > eps) {
    h <- hessian_logL(a, K, lambda)

    # let's make sure we can invert the hessian
    # otherwise, we need modification
    alpha <- 0
    while (kappa(h + alpha*I) > 1e15){
      alpha <- 2*(alpha + 0.01)
    }

    h <- h + alpha*I
    d <- -solve(h,g)

    # printing option
    l <- log_L(a, K, y, lambda)
    if (print){
```

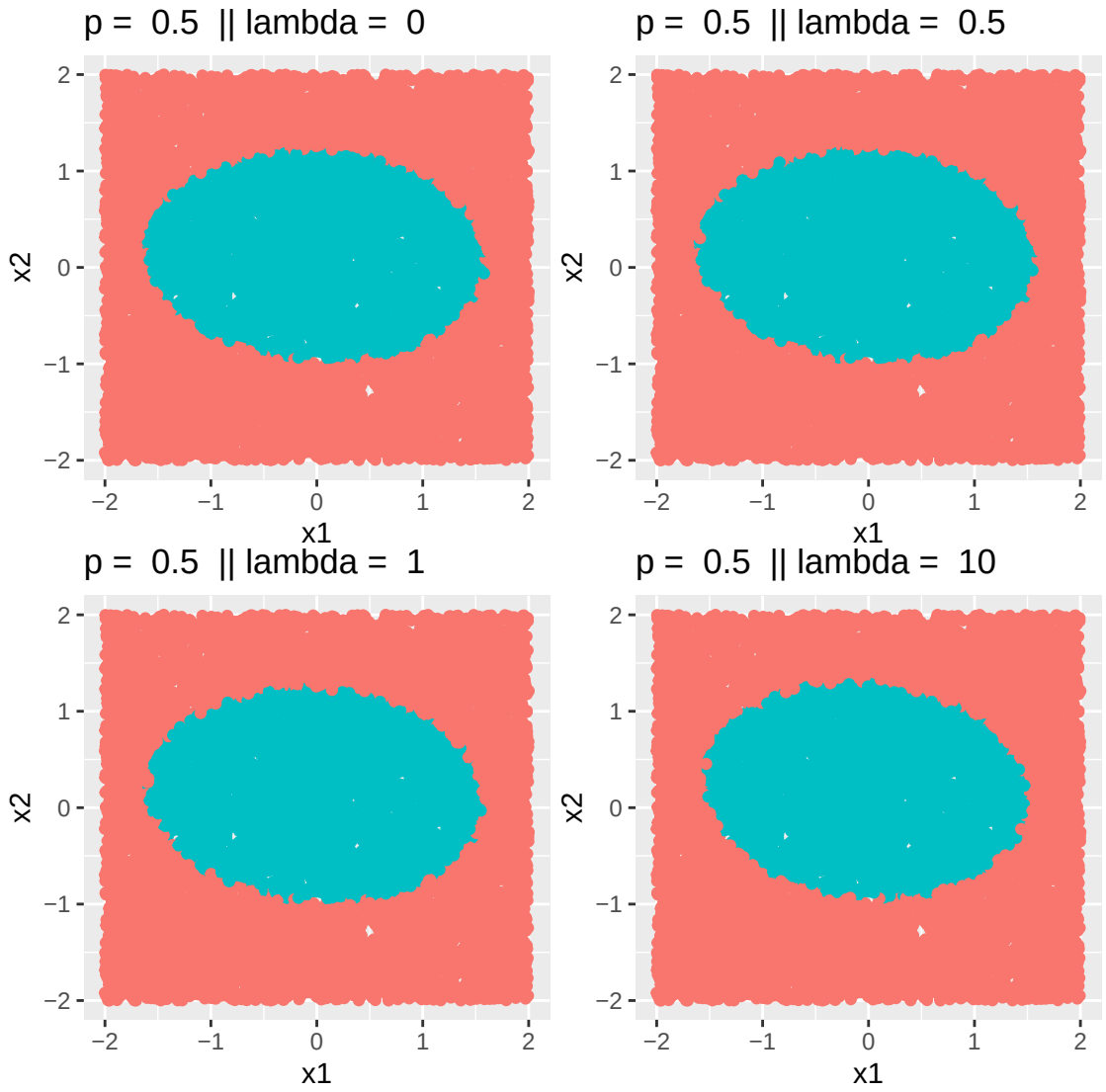
```
      cat("iteration", i, "logL =", l, "\n")
    }

    # graphing option
    if (graph){
      x_values <- append(x_values, i)
      y_values <- append(y_values, l)
    }

    # backtracking
    while (l > log_L(a + s*d, K, y, lambda)){
      s <- s/2
    }

    # update parameters
    a <- a + s*d

    if(abs(l-log_L(a + s*d, K, y, lambda)) < eps){
      break
    }

    g <- grad_logL(a, K, y, lambda)
    i <- i + 1
}

  return(list(max = a, iter = i, x_values = x_values, y_values = y_values, grad = g))
}
```

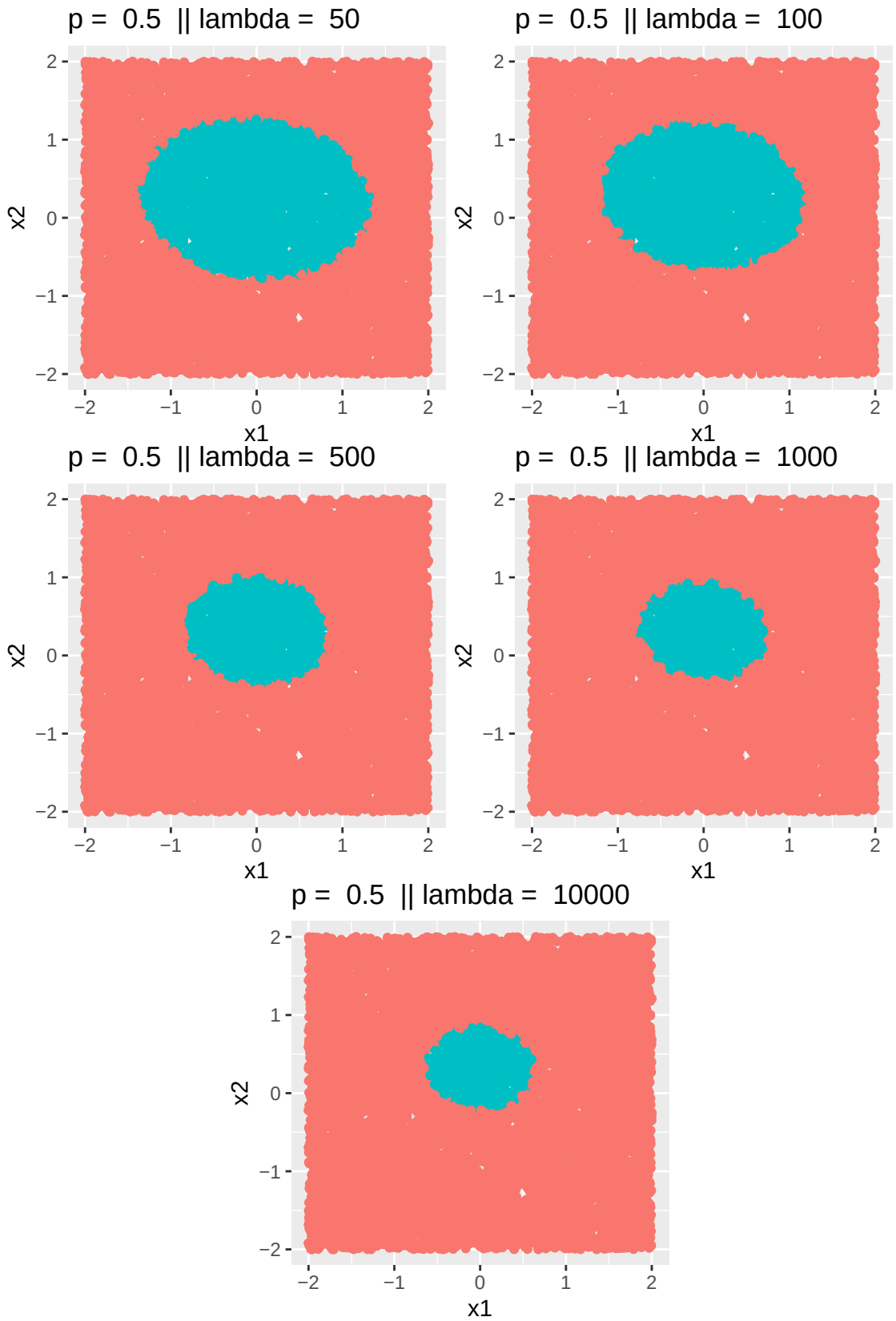
```
# defining classifier
classifier <- function(X_test, a_star, p){
  t <- (1 + X %*% t(X_test))^power
  prob <- 1/(1+exp(-t(t) %*% a_star))
  y_pred <- as.numeric(prob >= p)
  return(list(y = y_pred, p = prob))
}
```

```
# plotting function, given lambda and a probability cut-off
plotting <- function(lambda, p){
  # initializing a
  a <- rep(0,500)
  damped_NM_soln <- damped_NM(a, K, y, l)
  a_star <- damped_NM_soln$max
  pred_y <- classifier(X_test, a_star, p)$y
  print(ggplot() +
    geom_point(aes(x = X_test[,1], y = X_test[,2], color = as.factor(pred_y)), show.legend = F) +
    labs(title = paste(paste("p = ", p), paste(" || lambda = ", lambda))) +
    xlab("x1") + ylab("x2"))
}
```

```
set.seed(1)
# choose test points
X_test = cbind(4*runif(10000)-2, 4*runif(10000)-2)
# let us fit for these given values of lambda
lambda <- c(0, 0.5, 1, 10, 50, 100, 500, 1000, 10000)
# let us first choose a probability cut-off of 50%
```

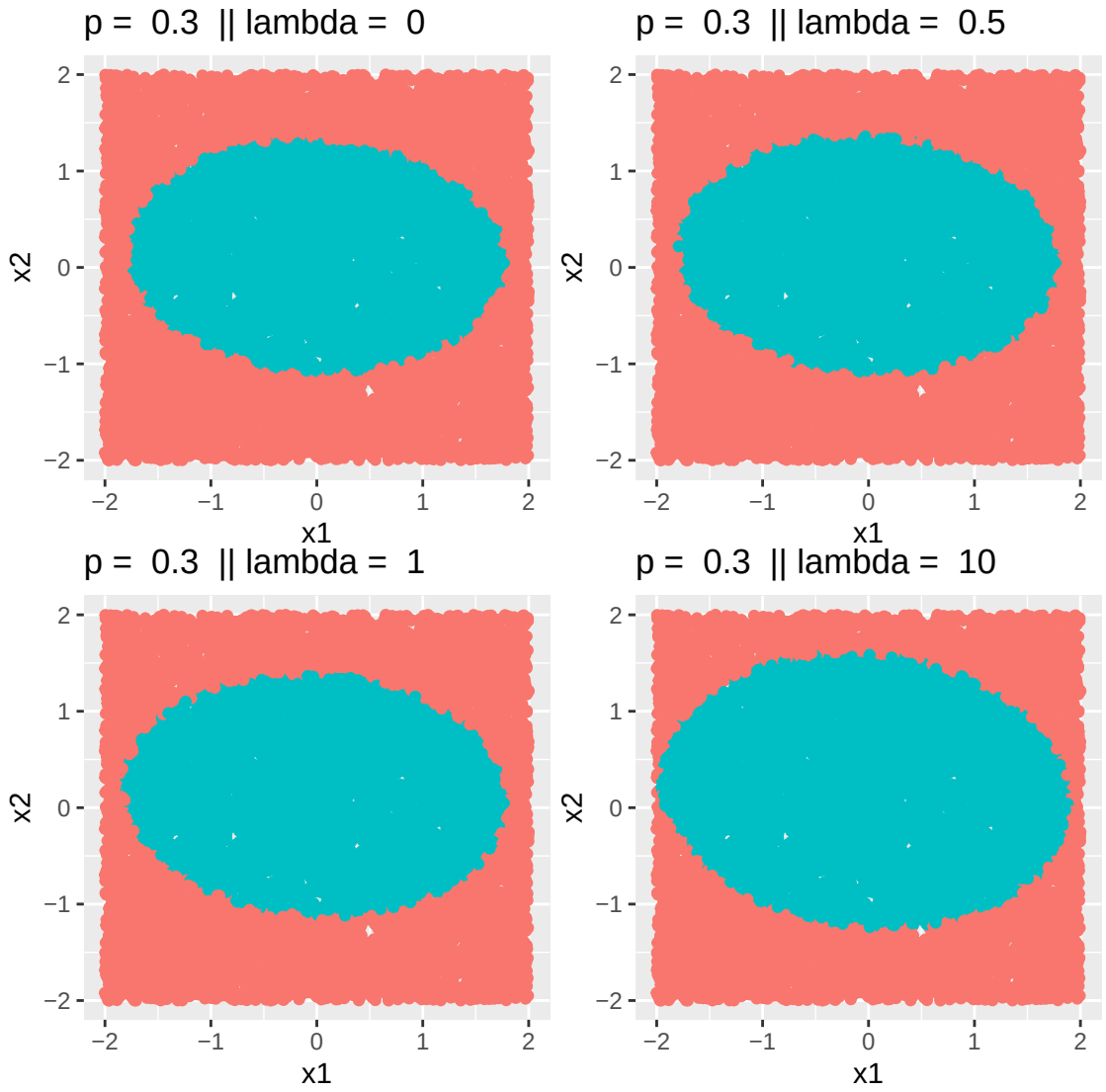
```
for(l in lambda){
  plotting(l, 0.5)
}
```

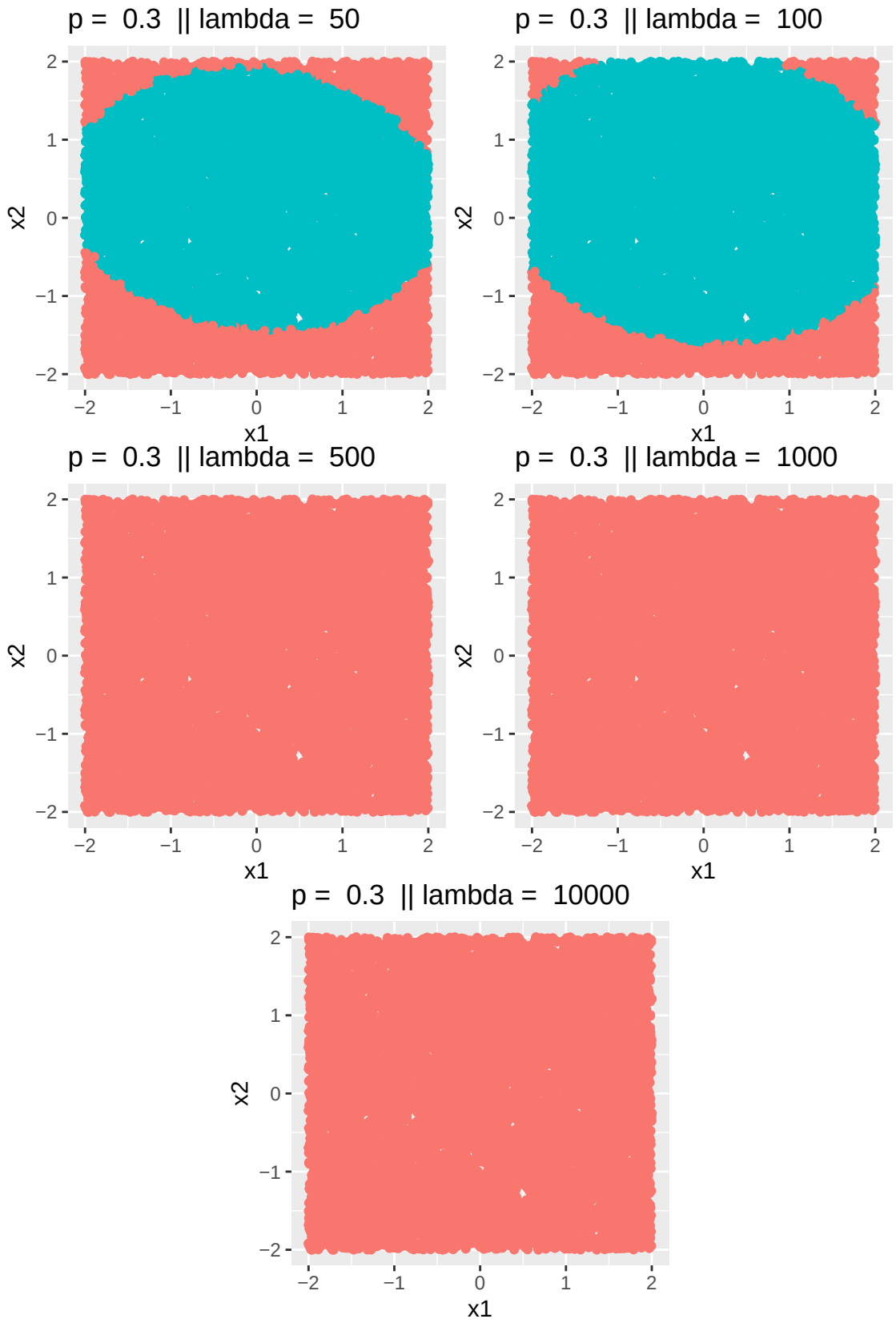




Now let us fit for the same values of lambda but try for a probability cut-off of 30% and 70%.

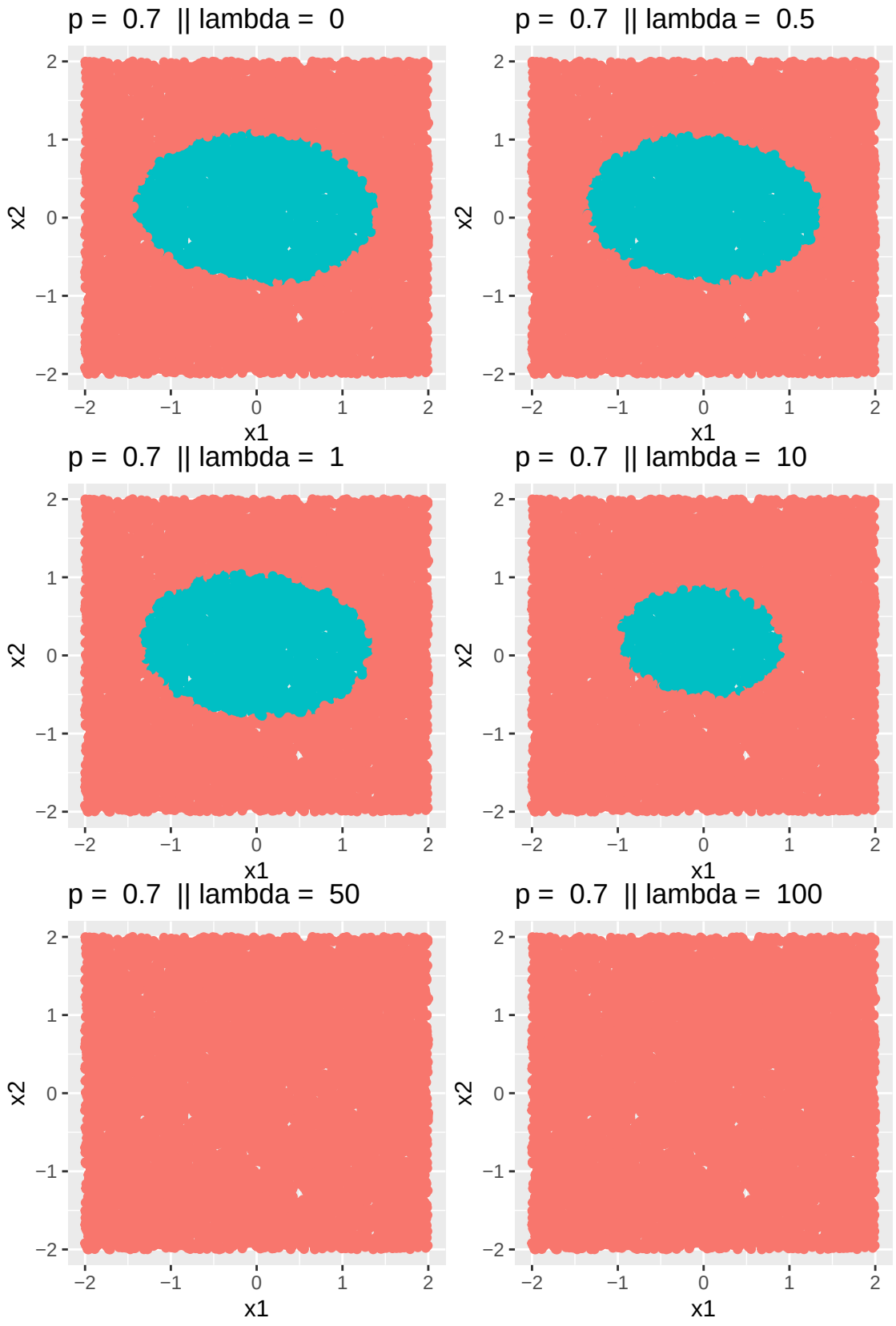
```
for(l in lambda){
  plotting(l, 0.3)
}
```

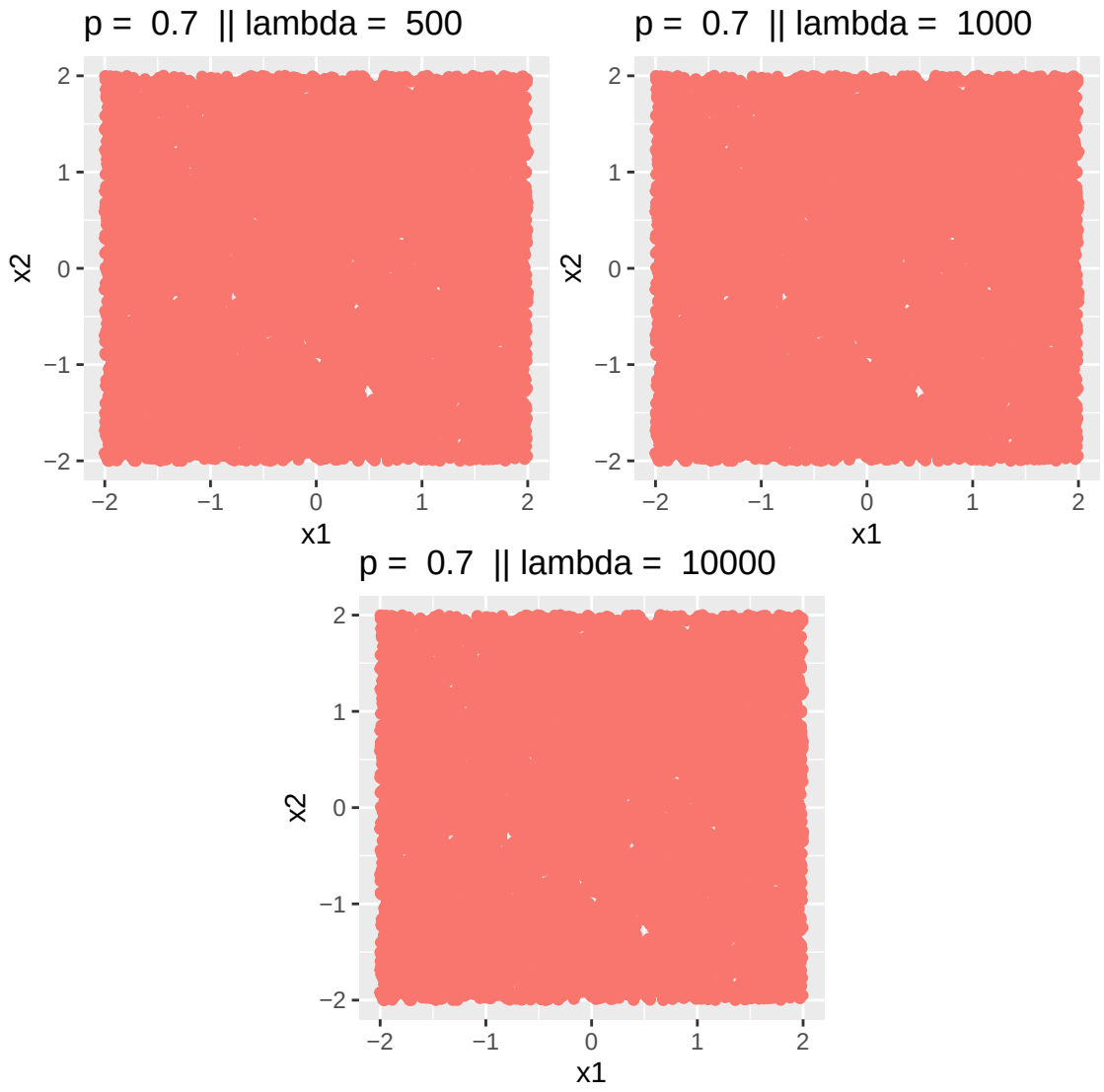





```
for(l in lambda){
  plotting(l, 0.7)
```

```
}
```


p = 0.7 || lambda = 0
p = 0.7 || lambda = 0.5
p = 0.7 || lambda = 1
p = 0.7 || lambda = 10
p = 0.7 || lambda = 50
p = 0.7 || lambda = 100
x1
x2
−2
−1
0
1
2

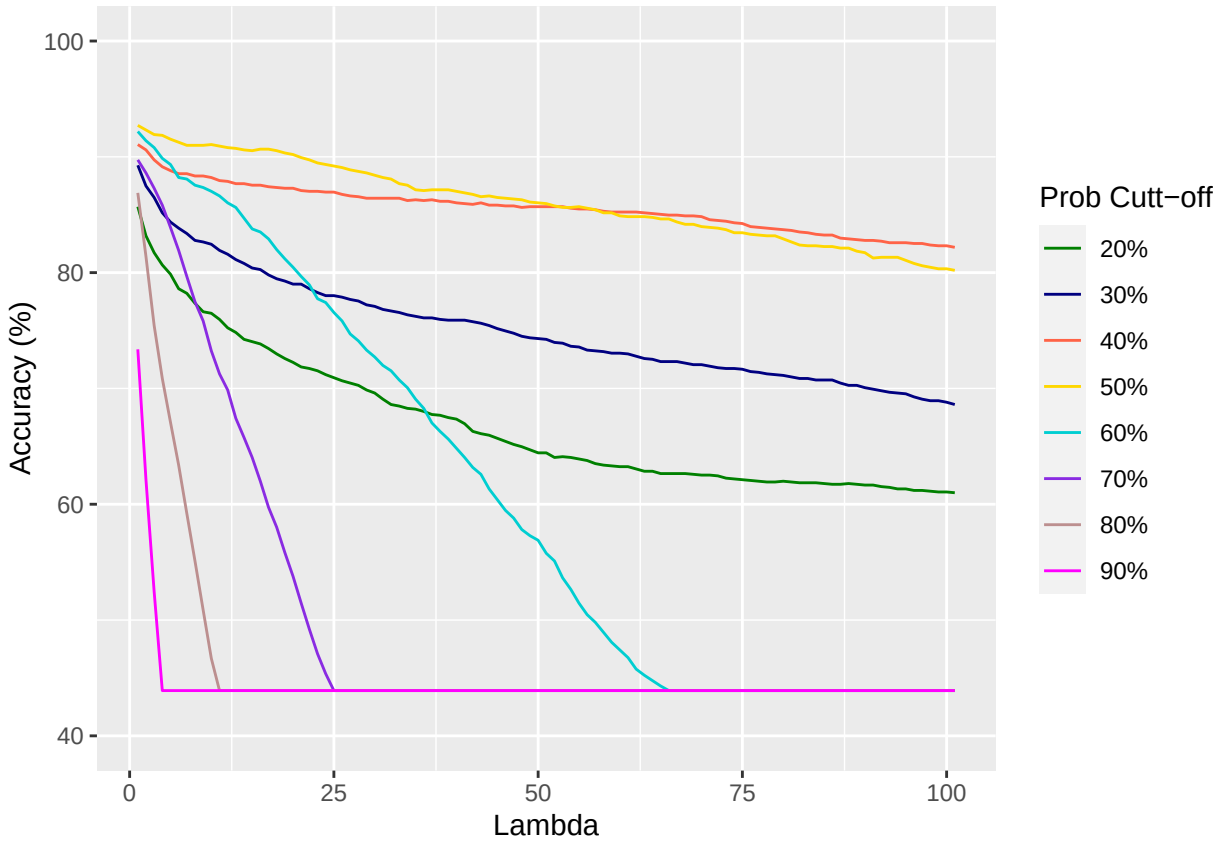


It looks like we get a better fit when lambda is very small (approx 0). This makes sense given # of parameters vs. # of data points so you should be able to see decent results without needing to add a penalty term to the regression (i.e., $\lambda = 0$). The shape of the region for which the classifier predicts 1 is an ellipse/circle (essentially, has a round shape). This also makes sense since we are using a quadratic polynomial kernel (the boundary should be a curve based on the degree of the polynomial, in this case 2).

As an aside: We could do something else, we have a dataset with close to 2000 points and we only trained on/used 500. We can take the remaning ~ 1500 points and treat it as our test dataset, and so instead of testing with visualizations we can look at accuracy (since our test dataset, ~ 1500 points, are labeled).

```
library("reshape2")
# let's look at overall accuracy for different combinations
# of lambda values and probability cut-offs
# let's use the remaining points to test
X_test <- data[setdiff(1:nrow(data),r),1:2]
y_test <- data[setdiff(1:nrow(data),r),3]

accuracy <- function(y_test, pred_y) {
  sens <- sum(pred_y==1 & y_test==1)/sum(y_test==1)
  spec <- sum(pred_y==0 & y_test==0)/sum(y_test==0)
```

```
  overall <- (sum(pred_y==1 & y_test==1) + sum(pred_y==0 & y_test==0))/length(y_test)
  return (list(sensitivity=sens, specificity=spec, overall=overall))
}

probs <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
lambda <- seq(0, 100, 1)

plotting.2 <- function(x, lambda){
  a <- rep(0,500)
  acc <- c()
  for (l in lambda){
    a <- rep(0, 500)
    damped_NM_soln <- damped_NM(a, K, y, l)
    a_star <- damped_NM_soln$max
    pred_y <- classifier(X_test, a_star, x)$y
    acc <- c(acc, accuracy(y_test, pred_y)$overall*100)
  }
  return(acc)
}

accuracies <- sapply(probs, FUN=plotting.2, lambda = lambda)
df <- data.frame(accuracies, Lambda=1:101)
df <- melt(df, id.vars = 'Lambda', variable.name = 'series')

ggplot(df, aes(Lambda, value, color = series)) + geom_line() +
  ylab("Accuracy (%)") + ylim(c(40,100)) +
  scale_color_manual("Prob Cutt-off", values = c("#008000", "#000080", "#FF6347",
            "#FFD700", "#00CED1", "#8A2BE2", "#BC8F8F", "#FF00FF"),
            labels = c("20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%"))
```



We can see from the above plot that we would get the highest accuracy on the test dataset (about 93%) if we choose $\lambda = 0$ and a probability cut-off of 50%.

3. (e)

It is reasonable to suppose that the cubic polynomial kernel would be more accurate than the quadratic polynomial kernel. With the cubic polynomial kernel we are capturing more features, and the boundary should be a curve based on a cubic polynomial and so, is less restrictive than the shapes allowed by a quadratic polynomial kernel (and so we might be able to capture some part of the antenna).