Project Two for Data Mining

Raymond Anthony Ford
raford2@miners.utep.edu

Due: 20 February 2018

Contents

1 Theoretical problem

Given two variables $\mathbf{x} = (x_1, ..., x_n)^T$ and $\mathbf{x}' = (x'_1, ..., x'_n)^T$ we need to show that if the observations in both $\mathbf{x}$ and $\mathbf{x}'$ have been standardized so that observations in $\mathbf{x}$ now have sample mean $\bar{x} = 0$ and sample standard deviation $s_x = \sqrt{\sum_{i=1}^n (x_i - \bar{x})/(n-1)} = 1$, and similarly for $\mathbf{x}'$, then

$$\|\mathbf{x} - \mathbf{x}'\|_2^2 = \sum_{i=1}^n (x_i - x'_i)^2 \propto \{1 - r(\mathbf{x}, \mathbf{x}')\},$$

where $r(\mathbf{x}, \mathbf{x}')$ denotes the sample Pearson's correlation coefficient.

Proof: We are given

$$r(\mathbf{x}, \mathbf{x}') = \frac{\sum_{i=1}^n (x_i - \bar{x})(x'_i - \bar{x}')}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (x'_i - \bar{x}')^2}}$$

and know that $\bar{x} = \bar{x}' = 0$, $s_x = s_{x'} = 1$, and that if $s_x^2 = \sum_{i=1}^n x_i^2/(n-1) = 1$ then $(n-1) = \sum_{i=1}^n x_i^2$, so we can rewrite $r(\mathbf{x}, \mathbf{x}')$ as follows

$$r(\mathbf{x}, \mathbf{x}') = \frac{\sum_{i=1}^n x_i x_i'}{\sqrt{\sum_{i=1}^n x_i^2}\sqrt{\sum_{i=1}^n x_i'^2}} = \frac{\sum_{i=1}^n x_i x_i'}{\sqrt{(n-1)}\sqrt{(n-1)}} = \frac{\sum_{i=1}^n x_i x_i'}{n-1}.$$

From this we see that

$$\sum_{i=1}^n x_i x_i' = (n-1) r(\mathbf{x}, \mathbf{x}'). \tag{1}$$

Using the above information we have the following.

$$\begin{aligned}
\|\mathbf{x} - \mathbf{x}'\|_2^2 &= \sum_{i=1}^n (x_i - x_i')^2 \\
&= \sum_{i=1}^n x_i^2 + \sum_{i=1}^n x_i'^2 - 2\sum_{i=1}^n x_i x_i' \\
&= (n-1) + (n-1) - 2(n-1) r(\mathbf{x}, \mathbf{x}') && \text{From (1)} \\
&= 2(n-1) - 2(n-1) r(\mathbf{x}, \mathbf{x}') \\
&= 2(n-1)[1 - r(\mathbf{x}, \mathbf{x}')] \\
&\propto [1 - r(\mathbf{x}, \mathbf{x}')].
\end{aligned}$$

2 Computer problems

2.1 Obtaining the data

We begin our venture by first obtaining the data to be used for this project. The dimension of the training and testing data set are displayed to ensure that when merged the dimension of the merged data set is correct.

```
train <- read.table(file=
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tra",
                    sep=",", header = FALSE, na.strings = c("NA", "", " "),
                    col.names = c(paste("x", 1:64, sep=""), "digit"))
test <- read.table(file=
```

```
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tes",
                    sep=",", header = FALSE, na.strings = c("NA", "", " "),
                    col.names = c(paste("x", 1:64, sep=""), "digit"))
dim(train); dim(test)
```

```
## [1] 3823   65
```

```
## [1] 1797   65
```

```
dat <- rbind(train, test); dim(dat)
```

```
## [1] 5620   65
```

We see that we have 64 measurements for a total of 5620 handwritten digits. The 65th "measurement" is not actually a measurement, but rather the actual (true) digit given the values from the 64 other measurements.

2.2 Exploratory data analysis

We next explore the data by first ordering `dat` based on `dat$digit` (the true digit), placing the true digits into a vector to be used later for labeling, removing the known digit in the 65th column, and then obtaining the number of unique values for each measurement. Measurements that do not have more than one unique value will be removed from the data set, as they contribute no additional information and thus their inclusion is not necessary for analysis.

```
dat0 <- data.matrix(dat[order(dat$digit),])
labs <- dat0[, c(65)] # Labels for plotting needed later
dat0 <- dat0[,-c(65)] # Remove the known digits
uniq <- apply(dat0, 2, unique)
for (k in 1:length(uniq)){
    if (length(unique(dat0[,k])) == 1)
        print(paste("x", k))
}
```

```
## [1] "x 1"
## [1] "x 40"
```

We see that `x1` and `x40` have only recorded one value as measurements, so we delete them from the data.

```
dat0 <- dat0[, -c(1, 40)]
```

Now we have measurements for each of the variables for which the value recorded in the region observed is not constant. We then plot these values with `heatmap()` to identify any potential patterns.

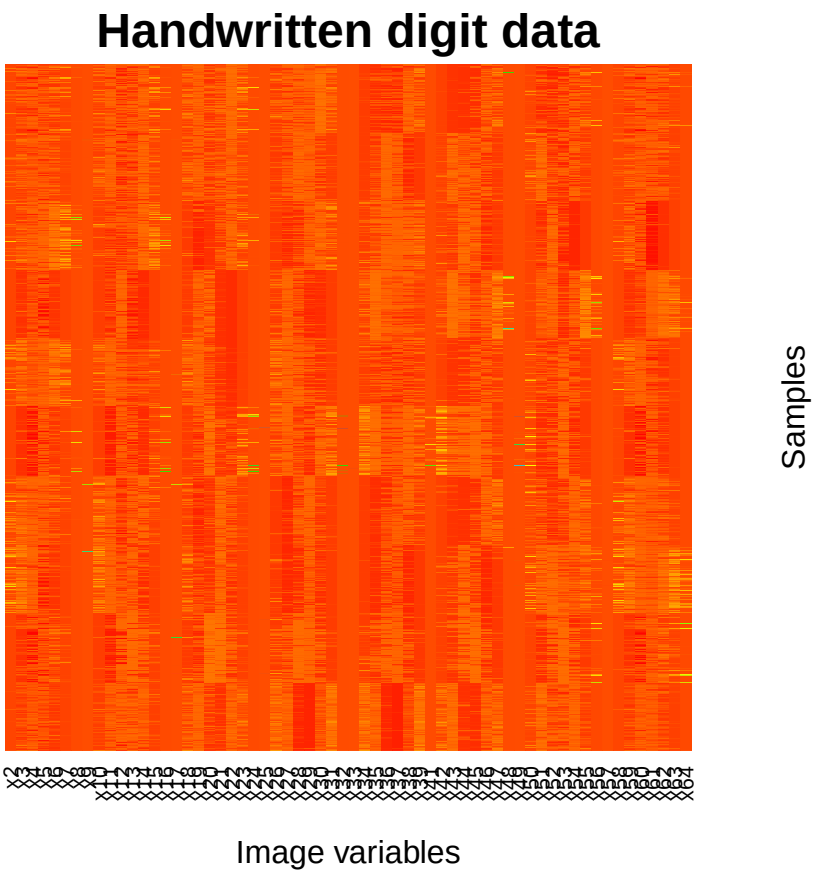

Figure 1: A heatmap of the handwritten digit data.

```
n <- NROW(dat0)
color <- rainbow(n, alpha=0.8)
heatmap(dat0, col=color, scale="column", Rowv=NA, Colv=NA,
        labRow=FALSE, margins=c(4,4), xlab="Image variables", ylab="Samples",
        main="Handwritten digit data")
```

While not initially visible, there seems to be approximately ten distinct regions when looking at the plot in Figure 1. This makes sense as the values were sorted prior to plotting, and we are looking at the handwritten digits of $0, 1, ..., 9$.

2.3 Principal Components Analysis (PCA)

Now we begin to perform PCA with the data. First we scale the data, and then compute the Principal Components (PCs) for the data.

```
dat0.scaled <- data.frame(apply(dat0, 2, scale))
pca.res <- prcomp(dat0.scaled, retx=TRUE)
```

Next, we construct a screeplot showing the cumulative proportions of variation for all of the

62 PCs. The code for this plot is adapted from Su (2018).

```
sd.pc <- pca.res$sdev
var.pc <- sd.pc**2
prop.pc <- var.pc/sum(var.pc)
plot(cumsum(prop.pc), type="h", xlab="Principal Component (PC)",
     ylab="Cumulative Proportion (CP)", main="Plot of PC vs CP")
abline(h=0.7, col='red')
abline(h=0.9, col='blue')
```

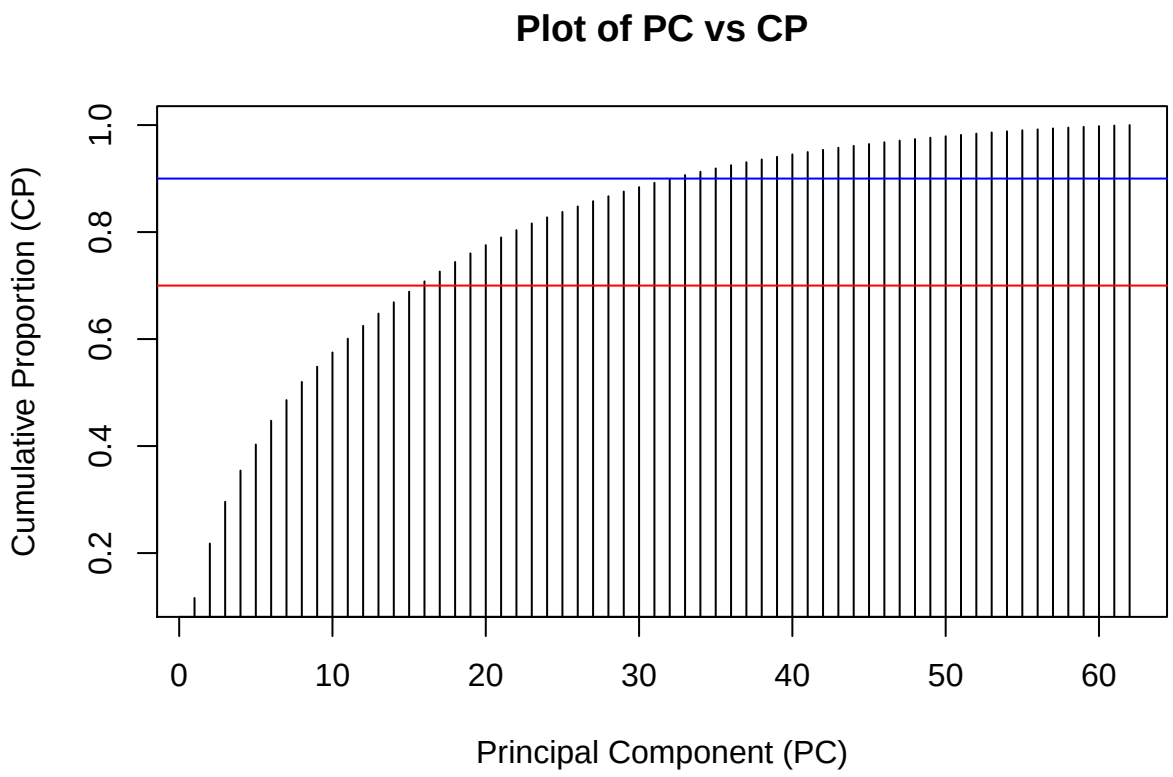

Figure 2: Plot showing CP for specific PCs. The first instance where the red line intersects with PC shows the 70% level of variation explained for that number of PCs, and likewise for the blue line except it explains 90% of the variation instead.

In Figure 2 we see that if we use the 70% and 90% rule as given in Everitt and Hothorn (2011), then we should select either the first 16 PCs to explain 70% of the variation, or select the first 33 PCs to explain 90% of the variation.

While the project assignment states that we need to print the estimated first two PC directions $(\widehat{\mathbf{a_1}}, \widehat{\mathbf{a_2}})$, we will not do that here but instead direct the reader to the appendix where these values are printed out.

Finally, we will plot PC2 versus PC1, where the 'dots' for each digit are represented with different colors and digit symbols.

```
library(RColorBrewer)
palette(brewer.pal(n=length(unique(labs)), name="Set3"))
# This colour scheme performs the best for colour blind personnel and has
# length greater than ten so we do not need to worry about a colour being
# used twice in the plot.
plot(pca.res$x[,1:2], pch="", main="PC1 and PC2 for handwritten digits")
text(pca.res$x[,1:2], labels=labs, col=labs)
abline(h=0, v=0, lty=2)
```

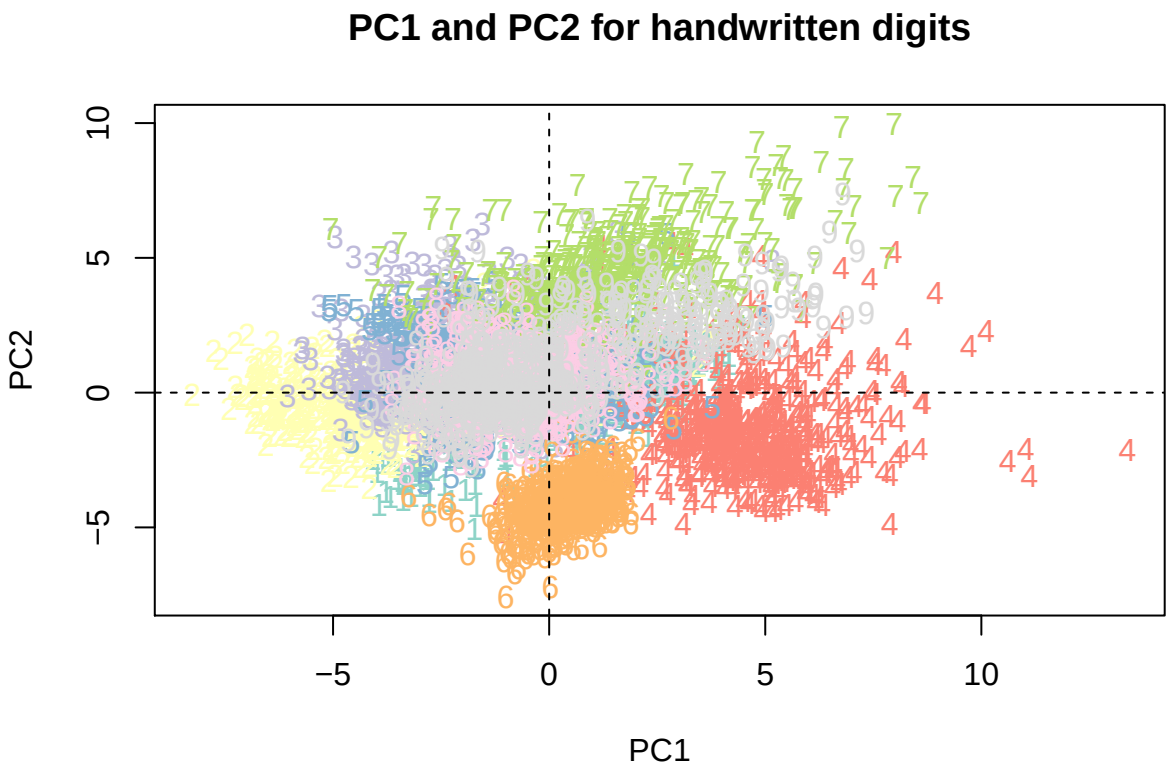

Figure 3: Plot of the handwritten digit data using PCA.

In Figure 3 we see that there appears to be some "clustering" but there is also a seemingly amorphous blob around the origin. This is similar to what was demonstrated in the laboratory where we used a different data set of handwritten digits and compared the plots of PCA and tSNE.

2.4 Sammon's nonlinear mapping

We are instructed to use a different method, and we elect to use Sammon's nonlinear mapping—the reason for this choice is detailed in the discussion section. The code for this method is adapted from Prabhakaran (2016).

```
library(MASS)
dat0.dist <- dist(dat0)
dat0.sam <- sammon(dat0.dist)
```

```
## Initial stress        : 0.30671
## stress after   0 iters: 0.30671
```

```
plot(dat0.sam$points[,1], dat0.sam$points[,2], t='n',
     main="Sammon's nonlinear mapping")
text(dat0.sam$points[,1], dat0.sam$points[,2], labels=labs, col=labs)
abline(h=0, v=0, lty=2)
```

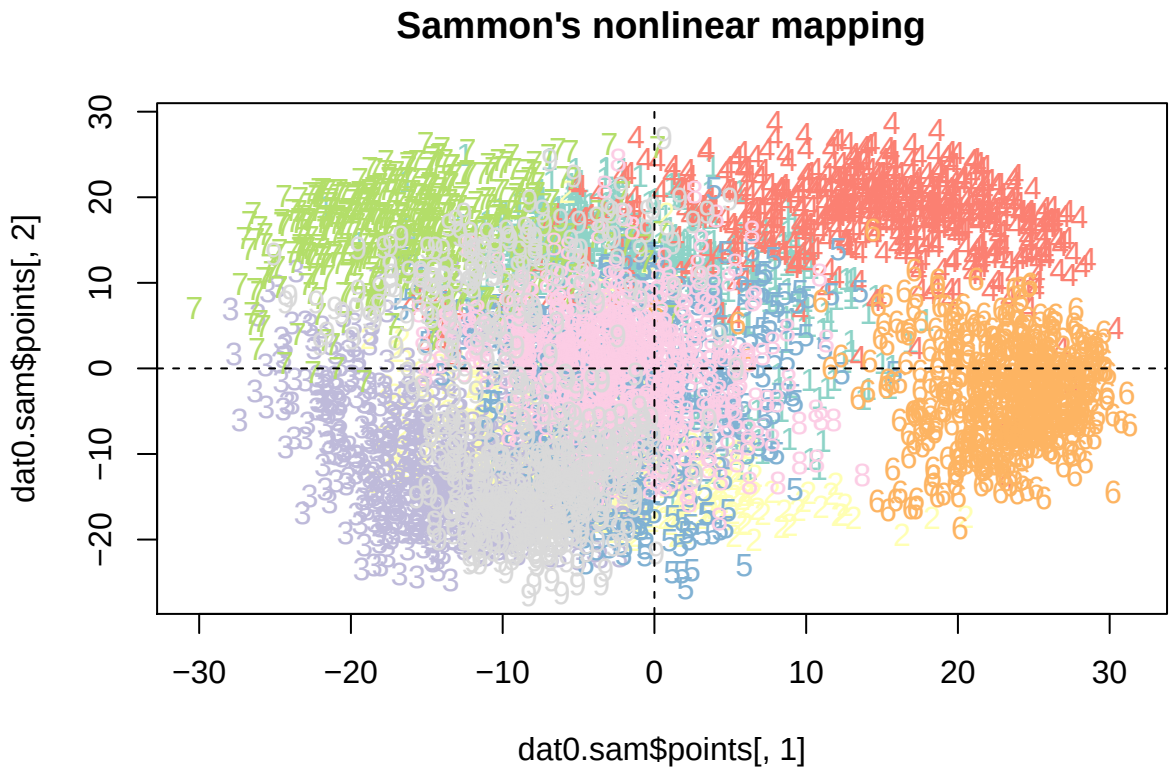

Figure 4: Plot of the handwritten digit data using Sammon's nonlinear mapping method.

In Figure 4 we again see some clustering and the amorphous blob around the origin, but that blob is not as dense as it was for the PCA portion. Moreover, we note that some separation is beginning to appear (i.e. digits are beginning to fall into more distinct regions).

2.5 Using t-distributed Stochastic Neighbor Embedding (tSNE)

We conclude this project by using tSNE. We make a plot with the following code, and direct the reader to the appendix should he wish to view the output from `Rtsne()`.

```
library(Rtsne)
set.seed(5474)
tsne <- Rtsne(dat0, dims=2, perplexity=30, max_iter=500)
plot(tsne$Y, t='n', main="t-distributed Stochastic Neighbour Embedding")
text(tsne$Y, labels=labs, col=labs)
abline(h=0, v=0, lty=2)
```

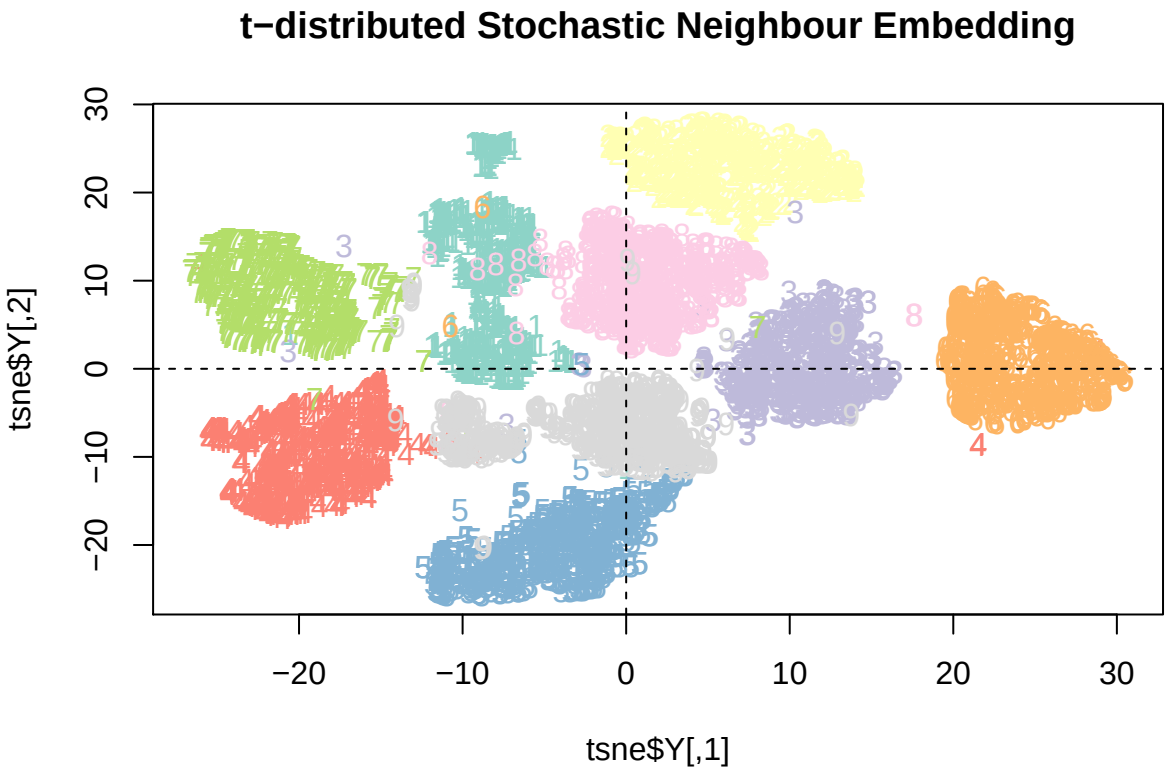

Figure 5: Plot of the handwritten digit data using tSNE.

From Figure 5, we see that the separation of the numbers is much greater than the two previous methods. That is, there are more distinct clusters in the plot compared to classical-MDS and Sammon's nonlinear mapping. Mainly, all of the digits seem to have distinct clusters for which most of them fall into with some errors

3 Discussion

When beginning this project, I did not expect the other MDS methods to take so long to complete. Looking back, it should have been obvious that they would have taken so long given the size of the distance matrix. Since I am using r markdown, I frequently need to "re-knit" my project to fix spelling, grammatical errors, or even comment on the results obtained, so some of these other nonclassical-MDS methods were not practical on my primary machine at home, as the time to compute the answers for them was much longer. Using

`sammon()` from `MASS` beat the others in terms of time completion and hence this method was used for this project.

I have also run each of the plots through Colblindor (2006) to ensure that should anyone seeing this report have some form of color blindness, then he can view the plots without too many issues. There still are some issues with the colors, but overall they are much better than had I relied on the default options available in `R`.[1]

One finding that I found especially interesting deals with the numbers one and seven. I learned to write characters using the "continental method" and frequently have people mistake what I write for something else, but in particular within the context of this project: ones for sevens, assuming a seven is not included in the same piece of writing as a one. It would be interesting to see how much error there was if digits written by North Americans were combined with digits written by Europeans.

4 References

- Colblindor (2006), *Coblis - Color Blindness Simulator*. Retrieved from: `http://www.color-blindness.com/coblis-color-blindness-simulator/`
- Everitt, B. and Hothorn, T. (2011), *An Introduction to Applied Multivariate Analysis with R*, New York: Springer.
- Prabhakaran, S. (2016), *Multi-Dimensional Scaling*. Retrieved from: `http://r-statistics.co/Multi-Dimensional-Scaling-With-R.html`
- Su, X. (2018), *Multidimensional Scaling (MDS)*. Retrieved from: `https://sites.google.com/site/xgsu00/`

[1] Do you have any recommendations on how to accommodate individuals that may have this disability? Many of the solutions that I found online only deal with palette's having a length of eight.

5 Appendix

5.1 Coefficients forming principal components $(\widehat{\mathbf{a_1}}, \widehat{\mathbf{a_2}})$

While we were instructed to print these values in the PCA portion of this project, they were not printed in that portion so as to not interrupt the flow of the project (in terms of readability). They are printed here instead.

```
a1.a2 <- pca.res$rotation[,1:2]
a1.a2
```

```
##                PC1           PC2
## x2  -0.1763830024  0.0728577623
## x3  -0.2734245087  0.1129612160
## x4  -0.2209468641  0.0487279236
## x5   0.0752759385  0.1494396336
## x6   0.0792154278  0.2592103761
## x7   0.0993651396  0.2341232238
## x8   0.0820671515  0.1305884295
## x9  -0.0166136180  0.0009636991
## x10 -0.2316687445  0.0597010278
## x11 -0.2344257865  0.0476143709
## x12  0.0487205172 -0.1107548574
## x13 -0.0323696309  0.0448630038
## x14  0.0106011572  0.2339568511
## x15  0.1339823614  0.2540618708
## x16  0.1106767724  0.1227564498
## x17 -0.0072261532  0.0012173433
## x18 -0.1307089266 -0.0110976294
## x19  0.0139352561 -0.1498939354
## x20  0.1028141044 -0.1303848680
## x21 -0.1350793517  0.0896619720
## x22  0.0513328096  0.1763545618
## x23  0.1982744014  0.1326815292
## x24  0.1104294319  0.0477815762
## x25  0.0011422714 -0.0034131301
## x26  0.0935003006 -0.0885620515
## x27  0.1536241765 -0.1622675852
## x28 -0.0341370518  0.0626460533
## x29 -0.1205278945  0.2073896146
## x30  0.1510813447  0.1411890225
## x31  0.1990106156  0.0286887778
## x32  0.0520138954 -0.0002946510
## x33  0.0425264003 -0.0094125674
## x34  0.2070957086 -0.1555447063
```

```
## x35  0.2041037004 -0.1594890006
## x36  0.0106290737  0.0582146262
## x37  0.0160802055  0.1188153939
## x38  0.1622148659  0.0169835695
## x39  0.1164965316 -0.0516177078
## x41  0.0749821530 -0.0363819281
## x42  0.1660570698 -0.1481337573
## x43  0.0973822629 -0.2244268078
## x44  0.0319245378 -0.0181996524
## x45  0.1393003227  0.0764626672
## x46  0.0990177042 -0.1080810277
## x47 -0.0498658377 -0.1991298601
## x48 -0.0070638388 -0.0722234632
## x49  0.0458057204 -0.0268616298
## x50 -0.0358002125 -0.0494186961
## x51 -0.1600216621 -0.1406474242
## x52 -0.0565329685 -0.0222883993
## x53  0.0856499249 -0.0084724788
## x54 -0.0963534345 -0.1726462188
## x55 -0.1546803893 -0.2104566855
## x56 -0.0512583389 -0.1063358310
## x57  0.0006071096 -0.0008331468
## x58 -0.1542220769  0.0625119548
## x59 -0.2601618391  0.1380498529
## x60 -0.1952201615  0.0354772522
## x61 -0.0588716253 -0.2449018435
## x62 -0.1423460486 -0.2051089694
## x63 -0.1544929093 -0.1361040639
## x64 -0.0779169359 -0.0542957373
```

5.2 tSNE output

We removed the `verbose=TRUE` option in `Rtsne()` so that the output would not be printed in the project, thus reverting to the default `verbose=FALSE`. In the event that the reader wanted to see the output then that output is below. A seed was set in the main project portion and the same seed is set here.

```
set.seed(5474)
tsne <- Rtsne(dat0, dims = 2, perplexity=30, max_iter = 500, verbose=TRUE)
```

```
## Read the 5620 x 50 data matrix successfully!
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Normalizing input...
```

```
## Building tree...
##  - point 0 of 5620
## Done in 4.36 seconds (sparsity = 0.021480)!
## Learning embedding...
## Iteration 50: error is 90.728891 (50 iterations in 3.34 seconds)
## Iteration 100: error is 74.180374 (50 iterations in 2.69 seconds)
## Iteration 150: error is 71.230723 (50 iterations in 2.68 seconds)
## Iteration 200: error is 70.262203 (50 iterations in 2.86 seconds)
## Iteration 250: error is 69.766214 (50 iterations in 2.73 seconds)
## Iteration 300: error is 2.333780 (50 iterations in 2.64 seconds)
## Iteration 350: error is 1.961396 (50 iterations in 2.65 seconds)
## Iteration 400: error is 1.763605 (50 iterations in 2.75 seconds)
## Iteration 450: error is 1.641470 (50 iterations in 2.78 seconds)
## Iteration 500: error is 1.559075 (50 iterations in 2.73 seconds)
## Fitting performed in 27.85 seconds.
```