Project II

Ruben Solis

Due: 26 February 2020

Contents

---

1 Handwritten Digits

1.1 Obtaining the data

We begin our project by first obtaining the data to be used for this project. Then the dimension of the training and testing data set are displayed to make sure that when merged the dimension of the merged data set is indeed 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
```

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

1.2 Exploratory data analysis

We 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 can see that `x1` and `x40` have only recorded one value as measurements, so we move to 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. Now we then plot these values with `heatmap()` to identify any potential patterns.

```
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")
```

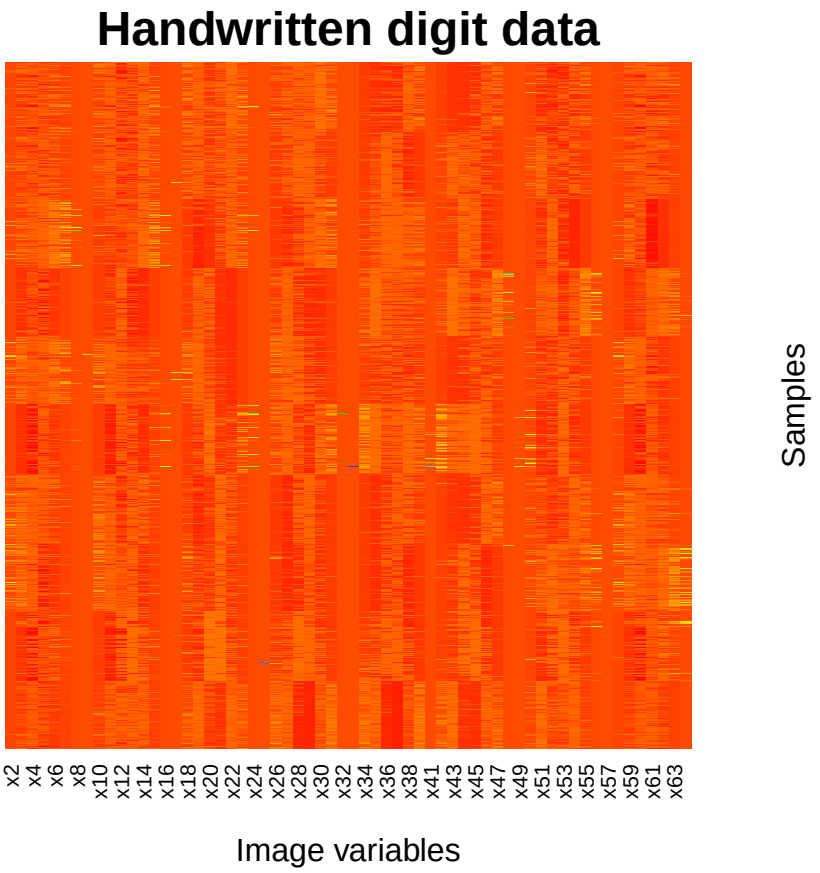

Figure 1: A heatmap of the handwritten digit data.

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

1.3 Principal Components Analysis (PCA)

Now we begin PCA with the data. First we scale the data, and then compute the Principal Components (PCs) for the data. This is accomplished as follows:

```
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.

```
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", lwd = 2)
abline(h=0.9, col='blue', lwd = 2)
```

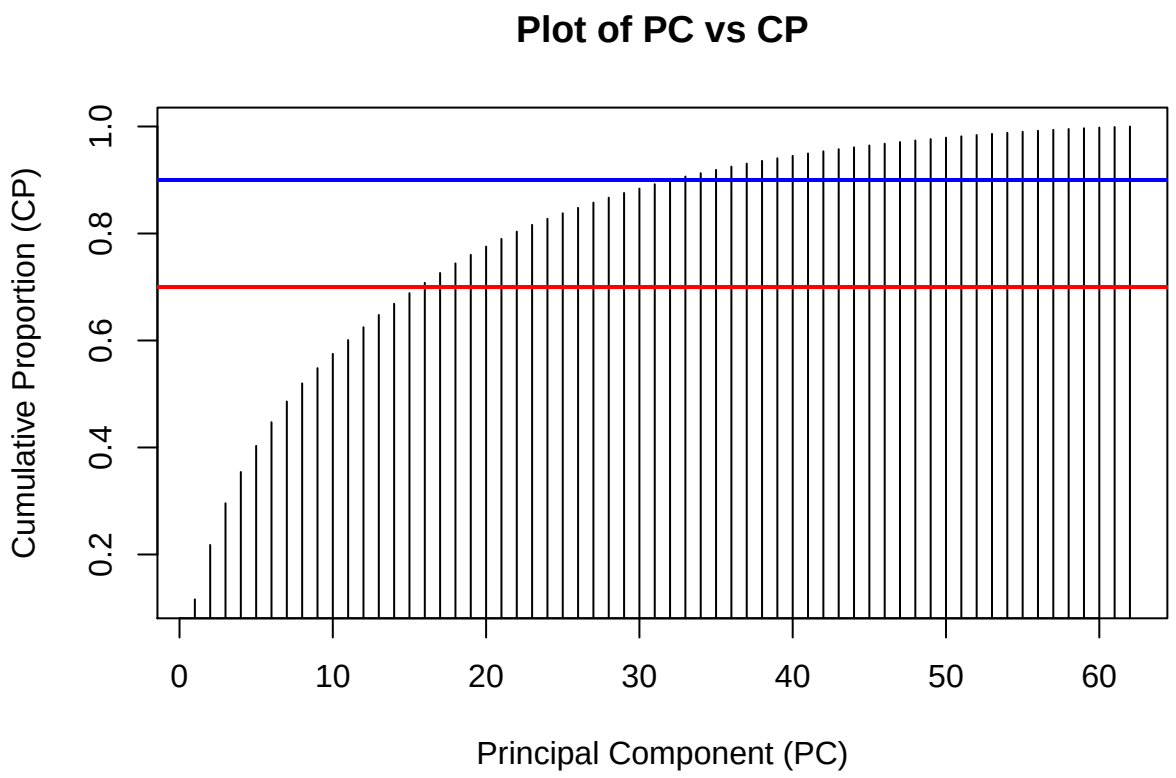

Figure 2: Plot showing CP for specific PCs. The first instance where the red line intersects with PC shows the 70% level of variation explained for that number of PCs, and likewise for the blue line except it explains 90% of the variation instead.

In Figure 2 we see that if we use the 70% and 90% rule [1], 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.

Now, 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 do that in appendix so as to not disrupt the flow of the project.

[1] See Everitt and Hothorn (2011)

Finally, we will plot PC2 versus PC1, where the 'dots' for each digit are represented with different colors and digit symbols.[2]

```
library(RColorBrewer)
palette(brewer.pal(n=length(unique(labs)), name="Set3"))
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)
```

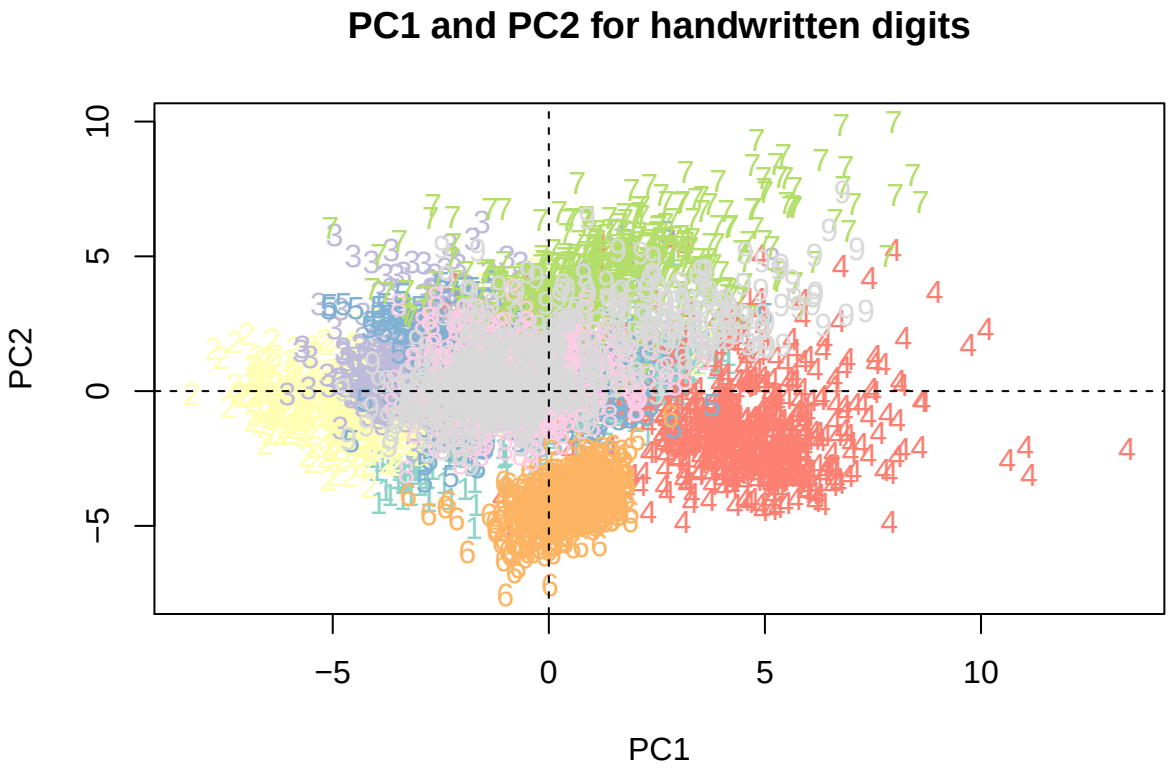

Figure 3: Plot of the handwritten digit data using PCA.

In Figure 3 there appears to be some "clustering" however there is also a seemingly amorphous blob around the origin.

1.4 Sammon's nonlinear mapping

We are instructed to use a different method, and we select to use Sammon's nonlinear mapping [3]

[2]This color scheme was chosen becuase it performs the best for colour blind personnel.
[3]The reason for this choice is detailed in the discussion section.

```
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)
```

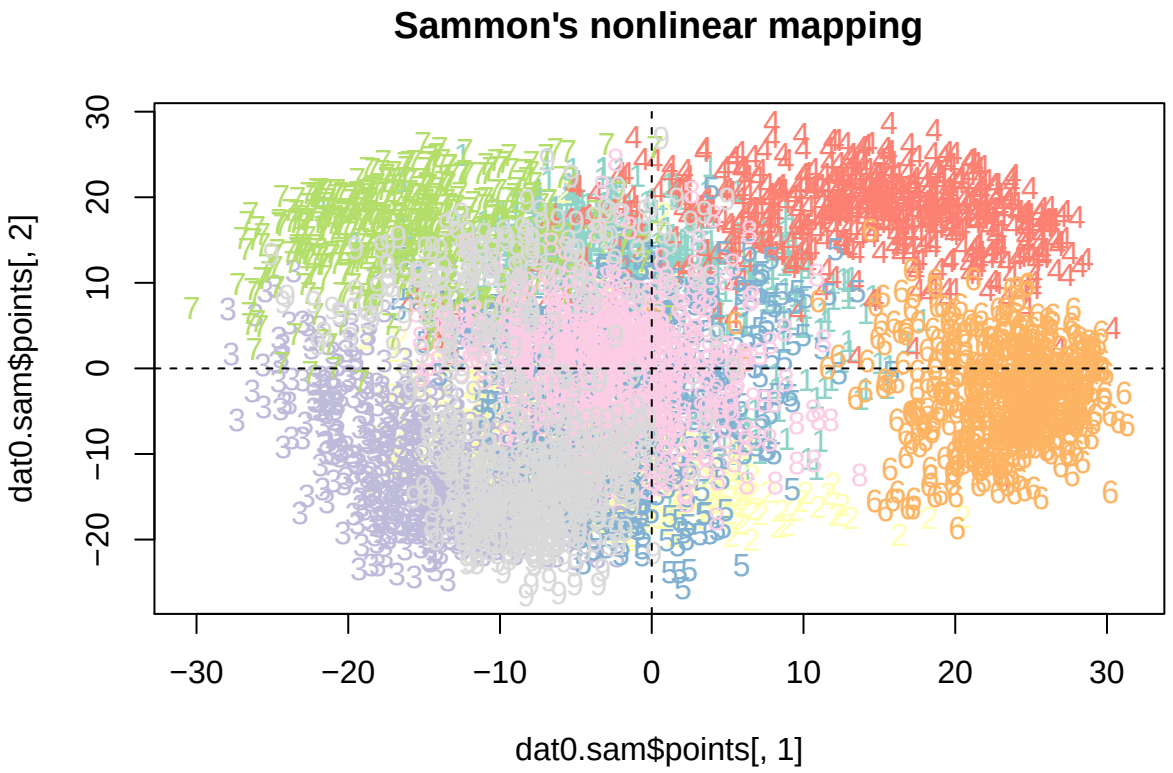

Figure 4: Plot of the handwritten digit data using Sammon's nonlinear mapping method.

In Figure 4 we again see some clustering and the amorphous blob around the origin, but it is not as dense as it was for the PCA portion. Likewise, we note that some separation is beginning to appear.

1.5 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 they 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)
```

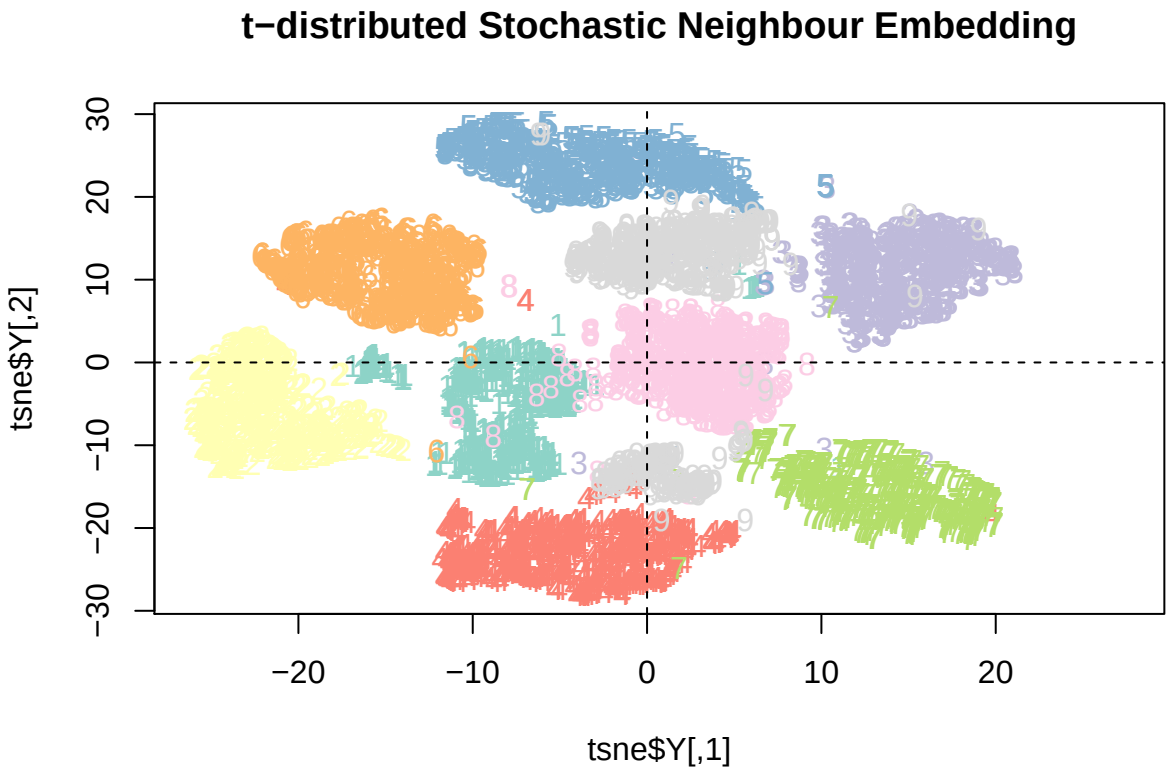

Figure 5: Plot of the handwritten digit data using tSNE.

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

2 General Comments

I did not expect the other MDS methods to take such a long time to complete. However, it should have been obvious that they would have taken so long since the size of the distance matrix. Since I am using R markdown, I frequently need to "re-knit" the project to fix some issues, so some of these other nonclassical-MDS methods were not practical on my laptop, as the time to compute the answers for them was much longer. Using `sammon()` from `MASS` was superior to the others in terms of time completion and as a result this method was used for this project.

I have a family member who happens to be color blind and I like to share with them some of the material we learn in this class. In particular, my cousin, who happens to be a business major, took data mining at a university in Tennessee, has color blindness, so I wanted to accommodate her and others who may read this project. I ran each of the plots through Colblindor (2006) 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`.

3 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`

4 Appendix

4.1 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
```

4.2 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)
```

```
## Performing PCA
## Read the 5620 x 50 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
```

```
## Computing input similarities...
## Building tree...
## Done in 4.25 seconds (sparsity = 0.021480)!
## Learning embedding...
## Iteration 50: error is 90.738635 (50 iterations in 1.22 seconds)
## Iteration 100: error is 74.468090 (50 iterations in 1.18 seconds)
## Iteration 150: error is 71.206665 (50 iterations in 1.10 seconds)
## Iteration 200: error is 70.130210 (50 iterations in 1.08 seconds)
## Iteration 250: error is 69.594381 (50 iterations in 1.08 seconds)
## Iteration 300: error is 2.329177 (50 iterations in 0.96 seconds)
## Iteration 350: error is 1.964666 (50 iterations in 0.88 seconds)
## Iteration 400: error is 1.765457 (50 iterations in 0.94 seconds)
## Iteration 450: error is 1.644011 (50 iterations in 0.96 seconds)
## Iteration 500: error is 1.562559 (50 iterations in 0.90 seconds)
## Fitting performed in 10.30 seconds.
```