ST 790 Assignment 2

David Elsheimer and Jimmy Hickey

9/29/2020

Instruction

This assignment consists of 4 problems. The assignment is due on **Tuesday, September 29** at 11:59pm EDT. Please submit your assignment electronically through the **Moodle** webpage. The assignment can be done as a group with at most 3 members per group (please include the name of the group members on the front page of the assignment).

Problem 1.

Let $\mathbf{D}$ be a $n \times n$ symmetric dissimilarity matrix, i.e., the entries d_{ij} of $\mathbf{D}$ are non-negative for all i, j, $d_{ii} = 0$ for all i. Let $\mathbf{B} = -\frac{1}{2}(\mathbf{I} - 11^\top/n)\mathbf{D}(\mathbf{I} - 11^\top/n)$ where $\mathbf{I}$ is the $n \times n$ identity matrix and $1 \in \mathbb{R}^n$ is the vector of all ones.

Given an integer $r \leq n$, the classical multidimensional scaling (CMDS) procedure produces a configuration $\mathbf{Z}_* \in \mathbb{R}^{n \times r}$ representing n points in $\mathbb{R}^r$ that best approximate $\mathbf{D}$ with respect to the STRAIN criterion, i.e.,

$$\mathbf{Z}_* = \arg \min_{\mathbf{Z} \in \mathbb{R}^{n \times r}} \|\mathbf{B} - \mathbf{Z}\mathbf{Z}^\top\|_F. \tag{1}$$

Consider the following eigendecomposition of $\mathbf{B}$.

$$\mathbf{B} = \sum_{i=1}^{n} \lambda_i u_i u_i^\top; \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

Then the CMDS procedure returns the $n \times r$ configuration

$$\mathbf{Z}_* = \left[(\lambda_1)_+^{1/2} u_1, (\lambda_2)_+^{1/2} u_2, \ldots, (\lambda_r)_+^{1/2} u_r\right] \tag{2}$$

where $(\lambda_i)_+ = \max\{\lambda_i, 0\}$.

Show that the expression $\mathbf{Z}_*$ in Eq.(2) is really the minimizer of the STRAIN criterion for $\mathbf{B}$. More specifically, show the following result.

Proposition Let $\mathbf{B}$ be a $n \times n$ symmetric matrix with eigendecomposition

$$\mathbf{B} = \sum_{i=1}^{n} \lambda_i u_i u_i^\top; \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

Then the best rank r, positive semidefinite approximation to $\mathbf{B}$, with respect to the Frobenius norm, is the matrix

$$\mathbf{B}_r = \sum_{i=1}^{r} (\lambda_i)_+ u_i u_i^\top.$$

Hint: Let $\mathbf{M}$ be any $n \times n$ matrix. Then

$$\|\mathbf{B} - \mathbf{M}\|_F^2 = \|\mathbf{U}\Lambda\mathbf{U}^\top - \mathbf{V}\Sigma\mathbf{V}^\top\|_F^2 = \|\Lambda - \mathbf{W}\Sigma\mathbf{W}^\top\|_F^2$$

where $\mathbf{U}\Lambda\mathbf{U}^\top$ and $\mathbf{V}\Sigma\mathbf{V}^\top$ are the eigendecomposition of $\mathbf{B}$ and $\mathbf{M}$, respectively, and $\mathbf{W}$ is a $n \times n$ orthogonal matrix.

Now, for fixed diagonal matrices Λ and Σ, show that a minimizer of

$$\min_{\mathbf{W}} \|\Lambda - \mathbf{W}\Sigma\mathbf{W}^\top\|_F^2$$

over the set of orthogonal matrices $\mathbf{W}$ is given by a permutation matrix. Using the rearrangement inequality, show that this permutation matrix will rearrange the diagonal entries of Σ to coincide with the ordering of the diagonal entries of Λ.

We will start by looking at the hint.

Notice that

$$\begin{aligned}
\|\mathbf{B} - \mathbf{M}\|_F^2 &= \|\boldsymbol{U\Lambda U}^T - \boldsymbol{V\Sigma V}^T\|_F^2 \\
&= \|\boldsymbol{UU}^T(\boldsymbol{U\Lambda U}^T - \boldsymbol{V\Sigma V}^T)\|_F^2 && \text{multiply by identity} \\
&= \|\boldsymbol{U}^T(\boldsymbol{U\Lambda U}^T - \boldsymbol{V\Sigma V}^T)\boldsymbol{U}\|_F^2 && \text{rearrange via trace} \\
&= \|\boldsymbol{U}^T\boldsymbol{U\Lambda U}^T\boldsymbol{U} - \boldsymbol{U}^T\boldsymbol{V\Sigma V}^T\boldsymbol{U}\|_F^2 \\
&= \|\boldsymbol{\Lambda} - \boldsymbol{U}^T\boldsymbol{V\Sigma V}^T\boldsymbol{U}\|_F^2.
\end{aligned}$$

Thus, take $\boldsymbol{W} = \boldsymbol{U}^T\boldsymbol{V}$. Notice that $\boldsymbol{W}$ is orthogonal.

$$\boldsymbol{W}^T\boldsymbol{W} = (\boldsymbol{U}^T\boldsymbol{V})^T(\boldsymbol{U}^T\boldsymbol{V}) = \boldsymbol{V}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{V} = \boldsymbol{V}^T\boldsymbol{I}\boldsymbol{V} = \boldsymbol{I}.$$

Now we want to find $\boldsymbol{W}$ such that

$$\min_{\mathbf{W}} \|\Lambda - \mathbf{W}\Sigma\mathbf{W}^\top\|_F^2.$$

Recall that $\boldsymbol{\Sigma}$ is the diagonal matrix of eigenvalues of $\boldsymbol{M}$. Intuitively, we want $\boldsymbol{W}$ to be a permutation matrix. Thus, $\boldsymbol{W\Sigma W}^T$ would result in rearranged, but still diagonal, matrix with the eigenvalues of $\boldsymbol{M}$ on the diagonal. Now, in order to minimize the Frobenius norm, we want to rearrange these elements in such a way that the ordering of eigenvalues matches that of $\boldsymbol{\Lambda}$. For example, if

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_{(1)} & 0 & 0 & 0 \\ 0 & \lambda_{(4)} & 0 & 0 \\ 0 & 0 & \lambda_{(3)} & 0 \\ 0 & 0 & 0 & \lambda_{(2)} \end{bmatrix},$$

then we would want

$$\boldsymbol{W\Sigma W}^T = \begin{bmatrix} \sigma_{(1)} & 0 & 0 & 0 \\ 0 & \sigma_{(4)} & 0 & 0 \\ 0 & 0 & \sigma_{(3)} & 0 \\ 0 & 0 & 0 & \sigma_{(2)} \end{bmatrix}.$$

This will minimize the Frobenius norm.

Thus, we need to show that $\boldsymbol{W}$ can be a permutation matrix and that the *correct* permutation matrix will minimize the Frobenius norm.

Take $\boldsymbol{V}$ to be the columns of $\boldsymbol{U}$ in an arbitrary order. For example,

$$\boldsymbol{W} = \boldsymbol{U}^T\boldsymbol{V} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_3 \\ \boldsymbol{u}_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_3 & \boldsymbol{u}_2 & \boldsymbol{u}_4 & \boldsymbol{u}_1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_1 \cdot \boldsymbol{u}_3 & \boldsymbol{u}_1 \cdot \boldsymbol{u}_2 & \boldsymbol{u}_1 \cdot \boldsymbol{u}_4 & \boldsymbol{u}_1 \cdot \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \cdot \boldsymbol{u}_3 & \boldsymbol{u}_2 \cdot \boldsymbol{u}_2 & \boldsymbol{u}_2 \cdot \boldsymbol{u}_4 & \boldsymbol{u}_2 \cdot \boldsymbol{u}_1 \\ \boldsymbol{u}_3 \cdot \boldsymbol{u}_3 & \boldsymbol{u}_3 \cdot \boldsymbol{u}_2 & \boldsymbol{u}_3 \cdot \boldsymbol{u}_4 & \boldsymbol{u}_3 \cdot \boldsymbol{u}_1 \\ \boldsymbol{u}_4 \cdot \boldsymbol{u}_3 & \boldsymbol{u}_4 \cdot \boldsymbol{u}_2 & \boldsymbol{u}_4 \cdot \boldsymbol{u}_4 & \boldsymbol{u}_4 \cdot \boldsymbol{u}_1 \end{bmatrix}$$

However, recall that $\boldsymbol{U}$ is the matrix of orthogonal eigenvectors. So,

$$\boldsymbol{u}_i \cdot \boldsymbol{u}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Thus, $\boldsymbol{W}$ will be a permutation matrix. In our example,

$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Now we need to show that the right choice of permutation (that orders the values of $\boldsymbol{\Sigma}$ the same as in $\boldsymbol{\Lambda}$) will minimize the Frobenius norm.

We can to this by reframing our question. Minimizing our Frobenius norm is equivalent to

$$\max_{\boldsymbol{W}} \operatorname{tr}\Big(W\Sigma W^T\Lambda\Big) = \sum_i \sum_j w_{ij}^2 \lambda_i \sigma_j.$$

We can rewrite this once more. Take $h_{ij} = w_{ij}^2$.

$$\sum_i \sum_j h_{ij}\lambda_i\sigma_j$$
$$\text{subject to } \sum_i h_{ij} = \sum_j h_{ij} = 1$$

The constraint can be written in this form because W is orthogonal. Since $WW^T = I$, we get $\sum_j w_{ij}^2 = 1$ for all i. Similarly, since $W^TW = I$ we get $\sum_i w_{ij}^2 = 1$ for all j.

This is a linear programming question with the constraint $\sum_i h_{ij} = \sum_j h_{ij} = 1$. We know that for a linear programming question, the solution exists on the boundary of the feasible set. Further, the Birkhoof-von Neumann Theorem states that the boundary of this constraint (the set of doubly stochastic matrices) is the set of $n \times n$ permutation matrices. Thus, a permutation matrix will minimize our objective.

Further, the rearrangement inequality states that the maximum reorderring will match the order of λ_i and σ_j in our rewritten objective.

Our proposition is an application of this result. We want the best rank r approximation of B. To obtain this, we can use W to strip off the smallest $n - r$ eigenvalues of B to maximize the Frobenius norm.

Problem 2

Download the dataset of games between American College Football teams available here. Now perform community detection using (1) a spectral clustering algorithm using normalized cut (2) spectral clustering using modularity and (3) Louvain algorithm. Evaluate the performance of your clustering (compared to the given ground truth).

```
g<-read.graph("football.gml",format=c("gml"))

g2 <- as.undirected(g)
## Largest connected component
lcc <- decompose.graph(g2, min.vertices = max(components(g2)$csize))[[1]]
Xhat2 <- embed_laplacian_matrix(lcc, type = "I-DAD", which = "sa", no = 12, scaled = FALSE)

Xhat2$D ## The three smallest eigenvalue of the normalized Laplacian
```

```
##  [1] 0.0000000 0.1368043 0.1829191 0.2250875 0.2396260 0.2823248 0.2998659
##  [8] 0.3247005 0.3773143 0.4099849 0.4581212 0.5512367
```

```
table(vertex_attr(lcc)$value) ## Ground truth
```

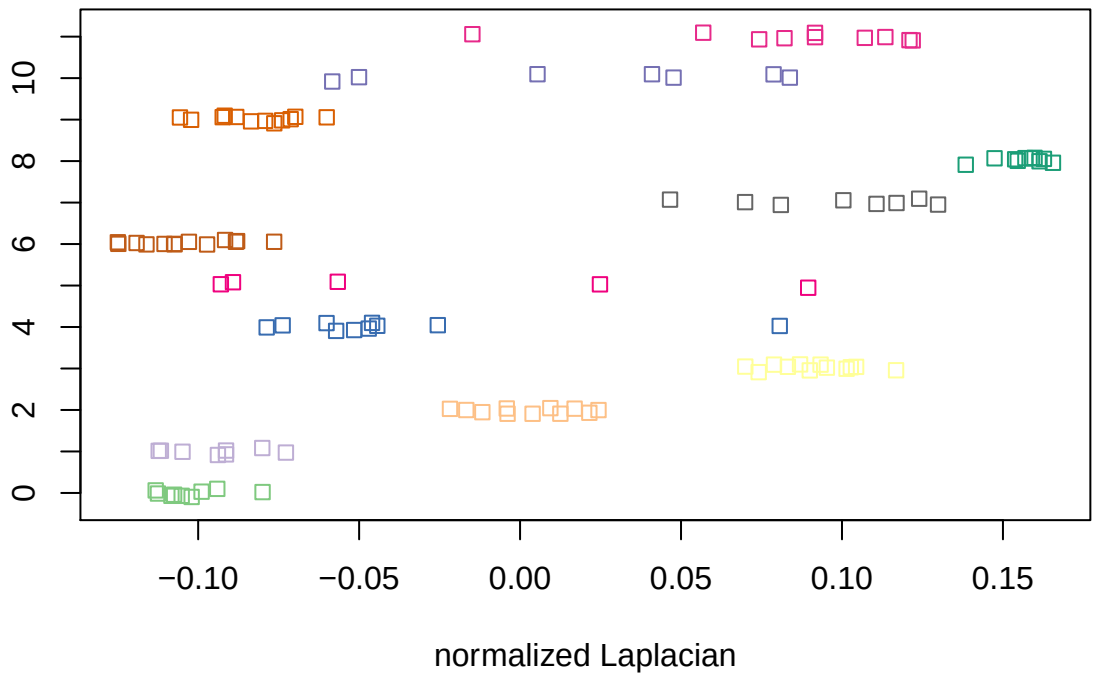
```
##
##  0  1  2  3  4  5  6  7  8  9 10 11
##  9  8 11 12 10  5 13  8 10 12  7 10
```

```
gt <- vertex_attr(lcc)$value
gt
```

```
##   [1]  7  0  2  3  7  3  2  8  8  7  3 10  6  2  6  2  7  9  6  1  9  8  8  7 10
##  [26]  0  6  9 11  1  1  6  2  0  6  1  5  0  6  2  3  7  5  6  4  0 11  2  4 11
##  [51] 10  8  3 11  6  1  9  4 11 10  2  6  9 10  2  9  4 11  8 10  9  6  3 11  3
##  [76]  4  9  8  8  1  5  3  5 11  3  6  4  9 11  0  5  4  4  7  1  9  9 10  3  6
## [101]  2  1  3  0  7  0  2  3  8  0  4  8  4  9 11
```

```
### Getting a vector of distinct colors for use in normalized cut
n <- 12
qual_col_pals = brewer.pal.info[brewer.pal.info$category == 'qual',]
col_vector = unlist(mapply(brewer.pal, qual_col_pals$maxcolors, rownames(qual_col_pals)))
col_vector<- col_vector[1:12]


stripchart(Xhat2$X[,2] ~ vertex_attr(lcc)$value, method = "jitter",
col = col_vector[1:12], xlab = "normalized Laplacian")
```




```
clusters2 <- kmeans(Xhat2$X, centers = 12)

table(clusters2$cluster, vertex_attr(lcc)$value)
```

```
##
##       0  1  2  3  4  5  6  7  8  9 10 11
##    1  0  0  0  0  0  1  0  0  0  0  4  1
##    2  0  0  0  0  0  1  0  0  0  0  3  1
##    3  0  0  3  0  0  0  0  0  0  0  0  0
##    4  0  0  8  0  0  0  0  0  0  0  0  0
##    5  0  0  0  0  1  0  0  0  0  0  0  8
##    6  0  0  0  0  0  1 13  0  0  0  0  0
##    7  0  0  0  0  0  0  0  8  0  0  0  0
##    8  0  8  0  0  0  2  0  0  0  0  0  0
##    9  9  0  0  0  9  0  0  0  0  0  0  0
##    10 0  0  0  0  0  0  0  0  0 12  0  0
##    11 0  0  0  0  0  0  0  0 10  0  0  0
##    12 0  0  0 12  0  0  0  0  0  0  0  0
```

```
###spectral with modularity
c2 = cluster_leading_eigen(g2)
# as per the documentation this performs clustering by calculating the
#leading non-negative eigenvector of the modularity matrix of the graph.
# c2$membership is the membership based on the results of modularity spectral clustering
```

```
###louvain
cc <- cluster_louvain(g2)

table(cc$membership)
```

```
##
##  1  2  3  4  5  6  7  8  9 10
## 12 11  9 15  9 14 10 16  9 10
```

```
###Normal cut comparison
## Compare the clustering using normalized mutual information
compare(clusters2$cluster, gt, method = c("nmi"))
```

```
## [1] 0.8894889
```

```
## Compare the clustering using Rand index
compare(clusters2$cluster, gt, method = c("rand"))
```

```
## [1] 0.9687262
```

```
## Compare the clustering using adjusted Rand index
compare(clusters2$cluster, gt, method = c("adjusted.rand"))
```

```
## [1] 0.7981271
```

```
###modularity comparison
## Compare the clustering using normalized mutual information
compare(c2$membership, gt, method = c("nmi"))
```

```
## [1] 0.6986702
```

```
## Compare the clustering using Rand index
compare(c2$membership, gt, method = c("rand"))
```

```
## [1] 0.8930587
```

```
## Compare the clustering using adjusted Rand index
compare(c2$membership, gt, method = c("adjusted.rand"))
```

```
## [1] 0.4640505
```

```
###Louvain comparison
## Compare the clustering using normalized mutual information
compare(cc$membership, gt, method = c("nmi"))
```

```
## [1] 0.8903166
```

```
## Compare the clustering using Rand index
compare(cc$membership, gt, method = c("rand"))
```

```
## [1] 0.9688787
```

```
## Compare the clustering using adjusted Rand index
compare(cc$membership, gt, method = c("adjusted.rand"))
```

```
## [1] 0.8069409
```

For normalized mutual information, Louvain was the largest, followed by normal cut and modularity. For Rand index, normal cut was the largest, followed by modularity and Louvain. For adjusted Rand index, Louvain was largest, followed by normal cut and modularity. Based on this, it appears to be the case that Louvain is doing the best job of correctly assesisng the data, as for adjusted Rand and NMI, Louvain appears to be closest to the truth. In the Rand index caclulation, all 3 comparison measures are pretty large, indicating that for this metric all 3 methods of clustering perform well.

Problem 2.5 (optional)

Now try to do community deteciton on the Youtube network dataset available here using any algorithm you want. Now try to evaluate the performance of your clustering compared to the given ground truth (it is not going to be easy :-))

Problem 3.

Download the USPS digits dataset from here. Next do the following steps.

- Choose the digits for class $4, 7$ and 9.
- Randomly sample 800 digits from each class.
- Do an embedding of these 2400 data point into $\mathbb{R}^d$ for some d, say $d = 10$, using **Laplacian eigenmaps**. You are free to choose your choice of ϵ or k-NN or weights.
- Run a 3-class SVM classifier on the embeddings for these 2400 datapoints. Use a linear kernel for your SVM.
- With the remaining 900 data points (300 from each class), evaluate the performance of your SVM classifier on this holdout data.
- Compare this performance with say SVM classification in the original 256 dimension data using say a linear kernel or a Gaussian kernel.

Hint To evaluate the performance on the hold-out data, you will need to do an out-of-sample embedding. Read this paper to see how to construct an out-of-sample embedding for Laplacian eigenmaps.

```
usps<-readMat(file.path("usps_all.mat"))$data
usps_sub <- usps[,,c(4,7,9)]
sample_size = 800
n_eigenvec = 10
```

```
usps_4 <- usps_sub[,,1]
usps_7 <- usps_sub[,,2]
usps_9 <- usps_sub[,,3]

holdout_size = dim(usps_4)[2] - sample_size

# sample data for into training and test
sample_4 = sample(1:ncol(usps_4), sample_size)
sample_7 = sample(1:ncol(usps_7), sample_size)
sample_9 = sample(1:ncol(usps_9), sample_size)

insample_4 = usps_4[, sample_4]
outsample_4 = usps_4[,-sample_4]
insample_7 = usps_7[, sample_7]
outsample_7 = usps_4[,-sample_7]
insample_9 = usps_9[, sample_9]
outsample_9 = usps_4[,-sample_9]

insample = t(cbind(insample_4, insample_7, insample_9))
outsample = t(cbind(outsample_4, outsample_7, outsample_9))

inlabels = c(rep(4, sample_size), rep(7, sample_size), rep(9, sample_size))
outlabels = c(rep(4, holdout_size ), rep(7, holdout_size ), rep(9, holdout_size ))


eigenprocessing = function(inmat) {
  # distance matrix
  distmat = as.matrix(dist(inmat))
  # guassian kernel with bandwidth sigma^2
  sigma = 1000
  W <- exp(-distmat^2/sigma^2)
  d = rowSums(W)


  norm_laplacian <- W/outer(sqrt(d),sqrt(d)) ## Normalized Laplacian matrix
  Xhat <- eigen(norm_laplacian, symmetric = TRUE)

  # pairs(Xhat$vectors[,1:n_eigenvec], col = rep(c(4,7,9), each = 800))

  return(Xhat)
}

# implementing equation 11 from the paper
out_embedding = function(new_data, full_data, eigen_vals, eigen_vecs)
{

  embedded = matrix(data = NA, nrow = dim(new_data)[1], ncol = length(eigen_vals))

  for( j in 1:dim(new_data)[1] )
  {
    y= rep(NA, length(eigen_vals))

    sigma=1000
```

```
    distmat = as.matrix(dist(rbind(new_data[j, ], full_data)))
    ktilde = exp(-distmat^2/sigma^2)
    d = rowSums(ktilde)
    norm_laplacian <- ktilde/outer(sqrt(d),sqrt(d))



    for(k in 1:length(y))
    {

      sum_val = 0
      for(i in 1:dim(full_data)[1])
      {
        sum_val = sum_val + eigen_vecs[i,k] * norm_laplacian[1,i]
      }

      y[k] = 1 / eigen_vals[k] * sum_val

    }

    embedded[j,] = y
  }

  return(embedded)
}

insample_eigen = eigenprocessing(insample)
in_vecs = insample_eigen$vectors[,1:n_eigenvec]
in_vals = insample_eigen$values[1:n_eigenvec]

# fit eigen decomp svm
eigen = data.frame(in_vecs, y = as.factor(inlabels))
eigenfit = svm(y ~ ., data = eigen, kernel = "linear", cost = 0.1, scale = FALSE)

## training
confusionMatrix(eigen$y, predict(eigenfit))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   4   7   9
##          4 575   0 225
##          7  25 697  78
##          9  41  48 711
##
## Overall Statistics
##
##                Accuracy : 0.8262
##                  95% CI : (0.8105, 0.8412)
##     No Information Rate : 0.4225
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7394
```

```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 4 Class: 7 Class: 9
## Sensitivity            0.8970   0.9356   0.7012
## Specificity            0.8721   0.9378   0.9358
## Pos Pred Value         0.7188   0.8713   0.8888
## Neg Pred Value         0.9587   0.9700   0.8106
## Prevalence             0.2671   0.3104   0.4225
## Detection Rate         0.2396   0.2904   0.2963
## Detection Prevalence   0.3333   0.3333   0.3333
## Balanced Accuracy      0.8846   0.9367   0.8185
```

```
## test
ptm <- proc.time()
outembed = data.frame(out_embedding(new_data = outsample[1:4,], full_data = insample, eigen_vals = in_va
proc.time() - ptm
```

```
##    user  system elapsed
##   17.98    0.17   18.16
```

```
colnames(outembed) = c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")

confusionMatrix(as.factor(outlabels[1:4]),
                predict(eigenfit, outembed))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 4 7 9
##          4 3 0 1
##          7 0 0 0
##          9 0 0 0
##
## Overall Statistics
##
##                Accuracy : 0.75
##                  95% CI : (0.1941, 0.9937)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 0.7383
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 4 Class: 7 Class: 9
## Sensitivity              1.00       NA     0.00
## Specificity              0.00        1     1.00
## Pos Pred Value           0.75       NA      NaN
```

```
## Neg Pred Value            NaN       NA     0.75
## Prevalence               0.75        0     0.25
## Detection Rate           0.75        0     0.00
## Detection Prevalence     1.00        0     0.00
## Balanced Accuracy        0.50       NA     0.50
```

```
# fit full data svm
fulldata = data.frame(insample, y = as.factor(inlabels))
fullfit = svm(y ~ ., data = fulldata, kernel = "linear", cost = 0.1, scale = FALSE)

## training
confusionMatrix(fulldata$y, predict(fullfit))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   4   7   9
##          4 800   0   0
##          7   0 800   0
##          9   0   0 800
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 4 Class: 7 Class: 9
## Sensitivity            1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.3333   0.3333   0.3333
## Detection Prevalence   0.3333   0.3333   0.3333
## Balanced Accuracy      1.0000   1.0000   1.0000
```

```
# 100 percent accuracy

## test
confusionMatrix(as.factor(outlabels),
                predict(fullfit, data.frame(outsample)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   4   7   9
##          4 294   0   6
##          7 299   0   1
##          9 299   0   1
##
## Overall Statistics
##
##                Accuracy : 0.3278
##                  95% CI : (0.2972, 0.3595)
##     No Information Rate : 0.9911
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0083
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 4 Class: 7 Class: 9
## Sensitivity          0.329596       NA 0.125000
## Specificity          0.250000   0.6667 0.664798
## Pos Pred Value       0.980000       NA 0.003333
## Neg Pred Value       0.003333       NA 0.988333
## Prevalence           0.991111   0.0000 0.008889
## Detection Rate       0.326667   0.0000 0.001111
## Detection Prevalence 0.333333   0.3333 0.333333
## Balanced Accuracy    0.289798       NA 0.394899
```

```
# 33% accuracy
```

Problem 4.

Complete the proof of Theorem 7 on page 55 of the dimension reduction lecture slides. In particular, show that if $x_1, x_2, \ldots, x_n$ are **distinct** then the matrix $\mathbf{H} = f(\|x_i - x_j\|^2)$ is positive definite.
