

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 $\mathbf{B} = -\frac{1}{2}(\mathbf{I} - \mathbf{1}\mathbf{1}^\top/n)\mathbf{D}(\mathbf{I} - \mathbf{1}\mathbf{1}^\top/n)$ where \mathbf{I} is the $n \times n$ identity matrix and $\mathbf{1} \in \mathbb{R}^n$ 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.,

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

Consider the following eigendecomposition of \mathbf{B} .

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

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

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

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

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

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

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

where $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ and $\mathbf{V}\mathbf{\Sigma}\mathbf{V}^\top$ 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 $\mathbf{\Lambda}$ and $\mathbf{\Sigma}$, show that a minimizer of

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

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 $\mathbf{\Sigma}$ to coincide with the ordering of the diagonal entries of $\mathbf{\Lambda}$.

We will start by looking at the hint.

Notice that

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

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

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

Now we want to find \mathbf{W} such that

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

Recall that $\mathbf{\Sigma}$ is the diagonal matrix of eigenvalues of \mathbf{M} . Intuitively, we want \mathbf{W} to be a permutation matrix. Thus, $\mathbf{W}\mathbf{\Sigma}\mathbf{W}^T$ would result in rearranged, but still diagonal, matrix with the eigenvalues of \mathbf{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 $\mathbf{\Lambda}$. For example, if

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

then we would want

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

This will minimize the Frobenius norm.

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned} & \sum_i \sum_j h_{ij} \lambda_i \sigma_j \\ \text{subject to } & \sum_i h_{ij} = \sum_j h_{ij} = 1 \end{aligned}$$

The constraint can be written in this form because \mathbf{W} is orthogonal. Since $\mathbf{W} \mathbf{W}^T = \mathbf{I}$, we get $\sum_j w_{ij}^2 = 1$ for all i . Similarly, since $\mathbf{W}^T \mathbf{W} = \mathbf{I}$ 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 Birkhoff-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 reordering 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 \mathbf{B} . To obtain this, we can use \mathbf{W} to strip off the smallest $n - r$ eigenvalues of \mathbf{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

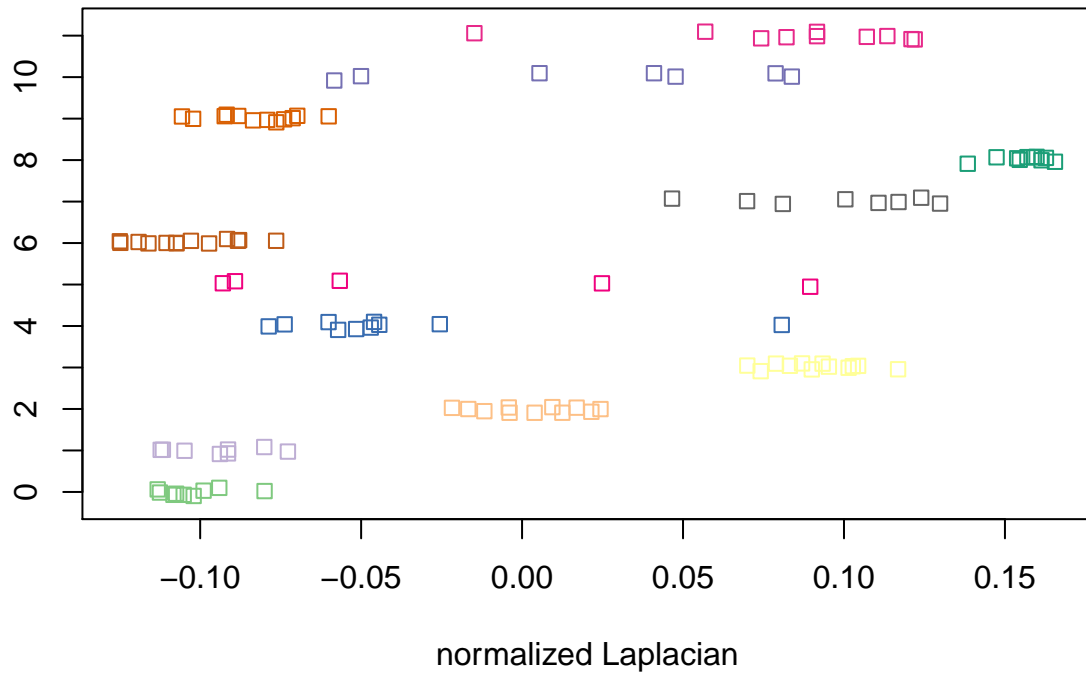
##
## 0  1  2  3  4  5  6  7  8  9 10 11
## 9  8 11 12 10  5 13  8 10 12  7 10

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 assessing the data, as for adjusted Rand and NMI, Louvain appears to be closest to the truth. In the Rand index calculation, 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 detection 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_7[, -sample_7]
insample_9 = usps_9[, sample_9]
outsample_9 = usps_9[, -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))
  # gaussian 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
cat("-----\n
training: 10 eigenvalues\n
-----")

## -----
##
## training: 10 eigenvalues
##
## -----

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
outembed = data.frame(out_embedding(new_data = outsample, full_data = insample, eigen_vals = in_vals, e

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

cat("-----\n
test: embedding\n
-----")

## -----
##
## test: embedding
##
## -----

confusionMatrix(as.factor(outlabels),
                predict(eigenfit, outembed))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  4   7   9
##           4 238   0  62
##           7 244   0  56
##           9 251   0  49
##
## Overall Statistics
##
##           Accuracy : 0.3189

```

```
##          95% CI : (0.2885, 0.3504)
##      No Information Rate : 0.8144
##      P-Value [Acc > NIR] : 1
##
##          Kappa : -0.0217
##
##      McNemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##          Class: 4 Class: 7 Class: 9
## Sensitivity      0.3247      NA 0.29341
## Specificity      0.6287      NA 0.65757
## Pos Pred Value   0.7933      NA 0.16333
## Neg Pred Value   0.1750      NA 0.80333
## Prevalence       0.8144      NA 0.18556
## Detection Rate   0.2644      NA 0.05444
## Detection Prevalence 0.3333      NA 0.33333
## Balanced Accuracy 0.4767      NA 0.47549
```

```
# 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
cat("-----\n
training: full training data\n
-----")
```

```
## -----
##
## training: full training data
##
## -----
```

```
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
```

```
## test
cat("-----\n
test: full training data\n
-----")
```

```
## -----
##
## test: full training data
##
## -----
```

```
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, \dots, x_n are **distinct** then the matrix $\mathbf{H} = f(\|x_i - x_j\|^2)$ is positive definite.

Answer

As per the notes, showing this claim is equivalent to showing the following claim:

Let $\mathbf{M} = (m_{ij})$ be a $n \times n$ s.d. matrix with distinct rows. Then $(e^{m_{ij}})$ is positive definite.

Some of this proof is already laid out in the notes, and will be restated below. The key parts which require more rigor will be covered in detail.

As \mathbf{M} is p.s.d., $\mathbf{M} = \mathbf{B}\mathbf{B}^\top$ for some $n \times n$ matrix \mathbf{B} . Note that

$$\mathbf{M}^{(2)} = (\mathbf{B}\mathbf{B}^\top) \circ (\mathbf{B}\mathbf{B}^\top) = (\mathbf{B} * \mathbf{B})(\mathbf{B} * \mathbf{B})^\top$$

Here, for $\mathbf{X}_{n \times p}, \mathbf{Y}_{n \times q}$, $\mathbf{X} * \mathbf{Y}$ is the $n \times pq$ matrix obtained by taking the Hadamard product of every column of \mathbf{X} with each column of \mathbf{Y} .

Going further, for each integer $k \geq 1$, $\mathbf{B}^{*(k)} = \mathbf{B} * \mathbf{B} * \dots * \mathbf{B}$ k times. As such, $\mathbf{M}^{(k)} = (\mathbf{B}^{*(k)})(\mathbf{B}^{*(k)})^\top$. Because

$$(e^{m_{ij}}) = \mathbf{1}\mathbf{1}^\top + \mathbf{B}\mathbf{B}^\top + \frac{1}{2!}\mathbf{B}^{*(2)}(\mathbf{B}^{*(2)})^\top + \frac{1}{3!}\mathbf{B}^{*(3)}(\mathbf{B}^{*(3)})^\top + \dots$$

it is therefore sufficient to show that the collection of column vectors of the infinite series of matrices $\mathcal{S} = \{\mathbf{1}, \mathbf{B}, \mathbf{B}^{*(2)}, \dots\}$ has n linearly independent columns.

As \mathbf{M} has distinct rows and columns, \mathbf{B} also has distinct rows. We want to show that if \mathbf{B} has distinct rows, there exists some vector $\boldsymbol{\pi}$ such that $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$.

Consider for the sake of contradiction that \mathbf{B} has distinct rows, but that there does not exist a $\boldsymbol{\pi}$ such that $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$ has distinct elements.

Then $\mathbf{B}\boldsymbol{\pi} = \begin{bmatrix} \sum_{j=1}^n B_{1j}\pi_j \\ \vdots \\ \sum_{j=1}^n B_{nj}\pi_j \end{bmatrix}$ such that $\sum_{j=1}^n B_{1j}\pi_j = \dots = \sum_{j=1}^n B_{nj}\pi_j$. This is equivalent to stating that for all j , $B_{1j} = \dots = B_{nj}$. This means that the rows are not distinct, a contradiction.

Next, consider the $n \times n$ Vandermonde matrix $\mathbf{V} = (v_{ij})$ such that $v_{ij} = z_i^{j-1}$. This matrix is non-singular.

Next we want to demonstrate that $\det(\mathbf{V})$ being a multilinear function of the columns of \mathbf{V} implies that \mathcal{S} has n linearly independent columns. This can be shown by showing that any column of \mathbf{V} can be written as linear combinations of some column of \mathcal{S} .

Note that

$$\mathbf{V} = [1 \mid \mathbf{z} \mid \mathbf{z}^2 \mid \dots \mid \mathbf{z}^{n-1}]$$

The first column of \mathbf{V} is trivial, as this is also one of \mathcal{S} . $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$. For columns $\mathbf{B}_1, \dots, \mathbf{B}_n, \mathbf{z} = \pi_1 \mathbf{B}_1 + \dots + \pi_n \mathbf{B}_n$. This is a linear combination of columns of \mathcal{S} .

Note that for an element of \mathbf{z}_i , it can be expressed as $\mathbf{z}_i = \sum_k B_{ik} \pi_k \sum_\ell B_{i\ell} \pi_\ell = \sum_{(k,\ell)} B_{ik} B_{i\ell} \pi_k \pi_\ell$. Consider the structure of the Hadamard product of \mathbf{B} with itself, $\mathbf{B} * \mathbf{B}$. A given row of this matrix will consist of the n B_{ik}^2 elements for $1 \leq k \leq n$, as well as the $n(n-1)$ elements $B_{ik} B_{i\ell}$ for $\ell \neq k$. As such, if we consider the vector $\boldsymbol{\pi}^{(2)}$ is of length n^2 , and contains all combinations of $\pi_k \pi_\ell$. Note then that a linear combination of the columns of \mathcal{S} , $\mathbf{B}^{*(2)} \boldsymbol{\pi}^{(2)} = \mathbf{z}^2$. This can be generalized to $n-1$ case by considering the linear combination of elements in \mathcal{S} $\mathbf{B}^{*(n-1)} \boldsymbol{\pi}^{(n-1)}$, where $\boldsymbol{\pi}^{(n-1)}$ is defined in a similar way as previously.

As $\det(\mathbf{V}) > 0$, the columns in \mathcal{S} contains n linearly independent columns as desired. As this has been shown, $(e^{m_{ij}})$ is positive definite, which completes the claim equivalent to the one we are trying to prove. Here, for $\mathbf{X}_{n \times p}, \mathbf{Y}_{n \times q}$, $\mathbf{X} * \mathbf{Y}$ is the $n \times pq$ matrix obtained by taking the Hadamard product of every column of \mathbf{X} with each column of \mathbf{Y} .

Going further, for each integer $k \geq 1$, $\mathbf{B}^{*(k)} = \mathbf{B} * \mathbf{B} * \dots * \mathbf{B}$ k times. As such, $\mathbf{M}^{(k)} = (\mathbf{B}^{*(k)})(\mathbf{B}^{*(k)})^\top$. Because

$$(e^{m_{ij}}) = 11^\top + \mathbf{B}\mathbf{B}^\top + \frac{1}{2!} \mathbf{B}^{*(2)}(\mathbf{B}^{*(2)})^\top + \frac{1}{3!} \mathbf{B}^{*(3)}(\mathbf{B}^{*(3)})^\top + \dots$$

it is therefore sufficient to show that the collection of column vectors of the infinite series of matrices $\mathcal{S} = \{1, \mathbf{B}, \mathbf{B}^{*(2)}, \dots\}$ has n linearly independent columns.

As \mathbf{M} has distinct rows and columns, \mathbf{B} also has distinct rows. We want to show that if \mathbf{B} has distinct rows, there exists some vector $\boldsymbol{\pi}$ such that $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$.

Consider for the sake of contradiction that \mathbf{B} has distinct rows, but that there does not exist a $\boldsymbol{\pi}$ such that $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$ has distinct elements.

Then $\mathbf{B}\boldsymbol{\pi} = \begin{bmatrix} \sum_{j=1}^n B_{1j} \pi_j \\ \vdots \\ \sum_{j=1}^n B_{nj} \pi_j \end{bmatrix}$ such that $\sum_{j=1}^n B_{1j} \pi_j = \dots = \sum_{j=1}^n B_{nj} \pi_j$. This is equivalent to stating that for all j , $B_{1j} = \dots = B_{nj}$. This means that the rows are not distinct, a contradiction.

Next, consider the $n \times n$ Vandermonde matrix $\mathbf{V} = (v_{ij})$ such that $v_{ij} = z_i^{j-1}$. This matrix is non-singular.

Next we want to demonstrate that $\det(\mathbf{V})$ being a multilinear function of the columns of \mathbf{V} implies that \mathcal{S} has n linearly independent columns. This can be shown by showing that any column of \mathbf{V} can be written as linear combinations of some column of \mathcal{S} .

Note that

$$\mathbf{V} = [1 \mid \mathbf{z} \mid \mathbf{z}^2 \mid \dots \mid \mathbf{z}^{n-1}]$$

The first column of \mathbf{V} is trivial, as this is also one of \mathcal{S} . $\mathbf{z} = \mathbf{B}\boldsymbol{\pi}$. For columns $\mathbf{B}_1, \dots, \mathbf{B}_n, \mathbf{z} = \pi_1 \mathbf{B}_1 + \dots + \pi_n \mathbf{B}_n$. This is a linear combination of columns of \mathcal{S} .

$$\mathbf{z}^2 = \mathbf{z} * \mathbf{z} = \mathbf{B}\boldsymbol{\pi} * \mathbf{B}\boldsymbol{\pi}.$$

As $\det(\mathbf{V}) > 0$, the columns in \mathcal{S} contains n linearly independent columns as desired. As this has been shown, $(e^{m_{ij}})$ is positive definite, which completes the claim equivalent to the one we are trying to prove.