Assessed Coursework 2

CID 01252821

Problem 1

Part a

The quasi-likelihood function is

$$L(W,H) = \sum_{i=1}^{d}\sum_{j=1}^{n}\left(X_{i,j}\log\left((WH)_{i,j}\right) - (WH)_{i,j}\right), \tag{1}$$

where $X \in R_{\geq 0}^{d\times n}$, $W \in R_{\geq 0}^{d\times q}$, and $H \in R_{\geq 0}^{q\times n}$. $L(W,H)$ is represented as summations of elements, so it is easy to calculate its derivative with respect to the element $W_{i,k}$. Let's rewrite $L(W,H)$ using that $(WH)_{i,j} = \sum_{k=1}^{q} W_{i,k}H_{k,j}$:

$$L(W,H) = \sum_{i=1}^{d}\sum_{j=1}^{n}\left(X_{i,j}\log\left((WH)_{i,j}\right) - (WH)_{i,j}\right) =$$

$$= \sum_{i=1}^{d}\sum_{j=1}^{n}\left(X_{i,j}\log\left(\sum_{k=1}^{q}W_{i,k}H_{k,j}\right) - \sum_{k=1}^{q}W_{i,k}H_{k,j}\right) \tag{2}$$

Now, let's take partial derivative with respect to $W_{i,k}$:

$$\frac{\partial L(W,H)}{\partial W_{i,k}} = \sum_{j=1}^{n}\left(\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j} - H_{k,j}\right) = \sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j} - \sum_{j=1}^{n}H_{k,j}. \tag{3}$$

The update (1) is

$$W_{i,k} \leftarrow W_{i,k}\frac{\sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j}}{\sum_{j=1}^{n}H_{k,j}}. \tag{4}$$

- If $\frac{\partial L(W,H)}{\partial W_{i,k}} > 0$ or, from Equation 3, equivalently $\sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j} > \sum_{j=1}^{n}H_{k,j}$ which (if at least one of $H_{k,j} is > 0$ leads to $\frac{\sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j}}{\sum_{j=1}^{n}H_{k,j}} > 1$. This is actually the term used for multiplication in the update equation in Equation 4. Hence, on every update $W_{i,k}$ is updated with a bigger value than $W_{i,k}$ because it is multiplied with a number > 1, and this means $W_{i,k}$ is increasing.
- Similarly, if $\frac{\partial L(W,H)}{\partial W_{i,k}} < 0$ or equivalently $\sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j} < \sum_{j=1}^{n}H_{k,j}$ which (if at least one of $H_{k,j} is > 0$ leads to $\frac{\sum_{j=1}^{n}\frac{X_{i,j}}{(WH)_{i,j}}H_{k,j}}{\sum_{j=1}^{n}H_{k,j}} < 1$. Hence, on every update $W_{i,k}$ is updated with a smaller value than $W_{i,k}$ because it is multiplied with a number < 1, and this means $W_{i,k}$ is decreasing.

- If $\frac{\partial L(W,H)}{\partial W_{i,k}} = 0$ or equivalently $\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j} = \sum_{j=1}^{n} H_{k,j}$ which (if at least one of $H_{k,j} is > 0$ leads to $\frac{\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j}}{\sum_{j=1}^{n} H_{k,j}} = 1$. Hence, on every update $W_{i,k}$ stays unchanged because it is multiplied with 1.
- If $W_{i,k}$ increases, from Equation 4 it means the multiplication term $\frac{\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j}}{\sum_{j=1}^{n} H_{k,j}}$ is greater than 1, or $\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j} - \sum_{j=1}^{n} H_{k,j} > 0$. This in Equation 3 leads to $\frac{\partial L(W,H)}{\partial W_{i,k}} > 0$.
- If $W_{i,k}$ decreases, from Equation 4 it means the multiplication term $\frac{\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j}}{\sum_{j=1}^{n} H_{k,j}}$ is less than 1, or $\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j} - \sum_{j=1}^{n} H_{k,j} < 0$. This in Equation 3 leads to $\frac{\partial L(W,H)}{\partial W_{i,k}} < 0$.
- If $W_{i,k}$ stays unchanged, from Equation 4 it means the multiplication term $\frac{\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j}}{\sum_{j=1}^{n} H_{k,j}}$ is equal to 1, or $\sum_{j=1}^{n} \frac{X_{i,j}}{(WH)_{i,j}} H_{k,j} - \sum_{j=1}^{n} H_{k,j} = 0$. This in Equation 3 leads to $\frac{\partial L(W,H)}{\partial W_{i,k}} = 0$.

What is the advantage of multiplicative updates over additive ones (gradient ascent) in the context of NMF?

1. **Non-negativity preservation** - the multiplicative updates naturally enforce the non-negativity constraints on W and H, which is crucial for the NMF updating process. If an element starts as non-negative, it remains non-negative throughout the updating process. In contrast, additive updates like gradient ascent would require additional mechanisms to ensure non-negativity, which can be less efficient and more complex to implement.
2. **Stable Convergence** - the multiplicative update rules tend to have stable convergence properties (often monotonic), ensuring that the algorithm makes consistent progress towards finding a local optimum, which is particularly suitable for NMF problems where the objective function is non-convex.
3. **Simplicity and Efficiency** - the multiplicative updates are relatively simple to implement and computationally efficient. They are often simpler and can be more numerically stable than additive updates because they do not require choosing a learning rate, which is an important and prone to problems part of gradient-based methods. Choosing an appropriate learning rate for gradient ascent can be challenging, a rate that is too high may lead to divergence, while a rate that is too low can result in slow convergence.
4. **Scalability** - the multiplicative update rules can be more scalable for large datasets since they can be easily adapted to sparse matrix operations, because they are element-wise and do not require complex matrix operations. Moreover, for data with a lot of zeros, multiplicative updates tend to be more efficient because they implicitly handle sparsity by not involving zero elements in the computations.
5. **Intuitive Interpretation** - the multiplicative update rules can have an intuitive interpretation in terms of reinforcing positive contributions, since they adjust the factors in a way that directly corresponds to the ratio of the current approximation to the original matrix, making the updates easy to interpret. For instance, if a particular factor contributes to reducing the reconstruction error, the multiplicative update will increase its weight, and vice versa.

Part b

The bias is given by:

$$\text{Bias}\left[H_{x_0,h}(x)\right] = \sum_{k \in Z} \frac{1}{h} \int_{x_0+(k-1)h}^{x_0+kh} f(u)du \mathbf{1}_{\{x \in B_k\}} - f(x).$$

In the sum, each term corresponds to a bin of the histogram. $1_{\{x \in B_k\}}$ is an indicator function that is 1 if x is in the k-th bin and 0 otherwise.

For the bin that contains x, and the fact that f is continuous and its derivative f' exists and is bounded, the mean value theorem for integrals suggests there exists a point $x_k^* \in B_k$ such that

$$\int_{x_0+(k-1)h}^{x_0+kh} f(u)du = hf(x_k^*),$$

and for the bin containing x it follows

$$\text{Bias}\left[H_{x_0,h}(x)\right] = \sum_{k\in Z} f(x_k^*)\mathbf{1}_{\{x\in B_k\}} - f(x) = f(x_k^*) - f(x). \tag{5}$$

Therefore, by the continuity of f:

$$\text{Bias}\left[H_{x_0,h}(x)\right] \to 0,\ h \to 0.$$

Now, let f' be bounded by M. We can apply the mean value theorem for differentiation to estimate the difference between $f(x_k^*)$ and $f(x)$. For the bin containing x, we have:

$$|f(x_k^*) - f(x)| = |f'(c_k)||(x_k^* - x)| \leq Mh,$$

where, c_k is some point between x_k^* and x, and $|x_k^* - x| \leq h$ because x_k^* is in the bin of width h containing x. Now, for the Bias from Equation 5, we have that it is bounded by Mh, meaning it is at most proportional to h. Therefore, as h goes to zero, the bias at x also goes to zero at a rate of $O(h)$.

Part c

The clusters C and C' are joined under agglomerative complete linkage, meaning the dissimilarity measure between clusters is defined as

$$d_{C,C'} = \max_{i\in C, j\in C'} d_{i,j},$$

where $d_{i,j}$ is the dissimilarity between x_i and x_j. The clusters are joined at height $h > 0$ which equals their dissimilarity $d_{C,C'}$:

$$\text{height} = d_{C,C'} = \max_{i\in C, j\in C'} d_{i,j} = h > 0. \tag{6}$$

We need to show that for any pair of samples x_i, x_j in $C \cup C'$, the dissimilarity $d_{i,j}$ is $\leq h$. There are three cases:

- Both x_i and x_j are in C, or both are in C'

Since C and C' were not merged before, the maximum dissimilarity within each cluster is less than h. Therefore, $d_{i,j} \leq h$.

- The two samples are in separate clusters: x_i is in C, and x_j is in C'

Equation 6 suggests that $d_{i,j} \leq h$ for $i \in C$, $j \in C'$.

- x_i and x_j are in different sub-clusters of C or C'

Let's say x_i is in a sub-cluster C_1 of C and x_j is in a sub-cluster C_1' ofC'. Since C and C' were merged at height h, the dissimilarity between any sub-clusters of C and C' is also at most h. Hence, $d_{i,j} \leq h$.

We can conclude that in all cases $d_{i,j} \leq h$. Therefore, in agglomerative complete linkage clustering, when clusters C and C' are joined at height h, the dissimilarity between any pair of samples in the new cluster $C \cup C'$ is at most h.

Part d

The within-dispersion measure is given by

$$W_K = \sum_{k=1}^{K} \frac{1}{2 \cdot \#C_k} \sum_{(i,j):x_i,x_j \in C_k} \|x_i - x_j\|^2.$$

Let μ_k be the centroid of cluster C_k. The objective function of K-means is to minimize the sum of squared distances of each point to its cluster centroid, which can be written as:

$$E_k = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2.$$

Now, let's observe that

$$\sum_{x_i \in C_k} \|x_i - \mu_k\|^2 = \sum_{x_i \in C_k} \sum_{x_j \in C_k} \frac{\|x_i - x_j\|^2}{2|C_k|}$$

This equality holds because the term $\|x_i - x_j\|^2$ appears twice for each pair (i, j) in the summation, and the mean μ_k minimizes the sum of squared distances within its cluster.

Therefore, E_k can be rewritten as:

$$E_k = \sum_{k=1}^{K} \frac{1}{2 \cdot \#C_k} \sum_{(i,j):x_i,x_j \in C_k} \|x_i - x_j\|^2,$$

and this is exactly the definition of W_K. Thus, minimizing E_k, the K-means objective, is equivalent to minimizing W_K, the within-cluster dispersion measure, when the dissimilarity is measured using the squared Euclidean distance. This completes the proof.

Problem 2

Part a

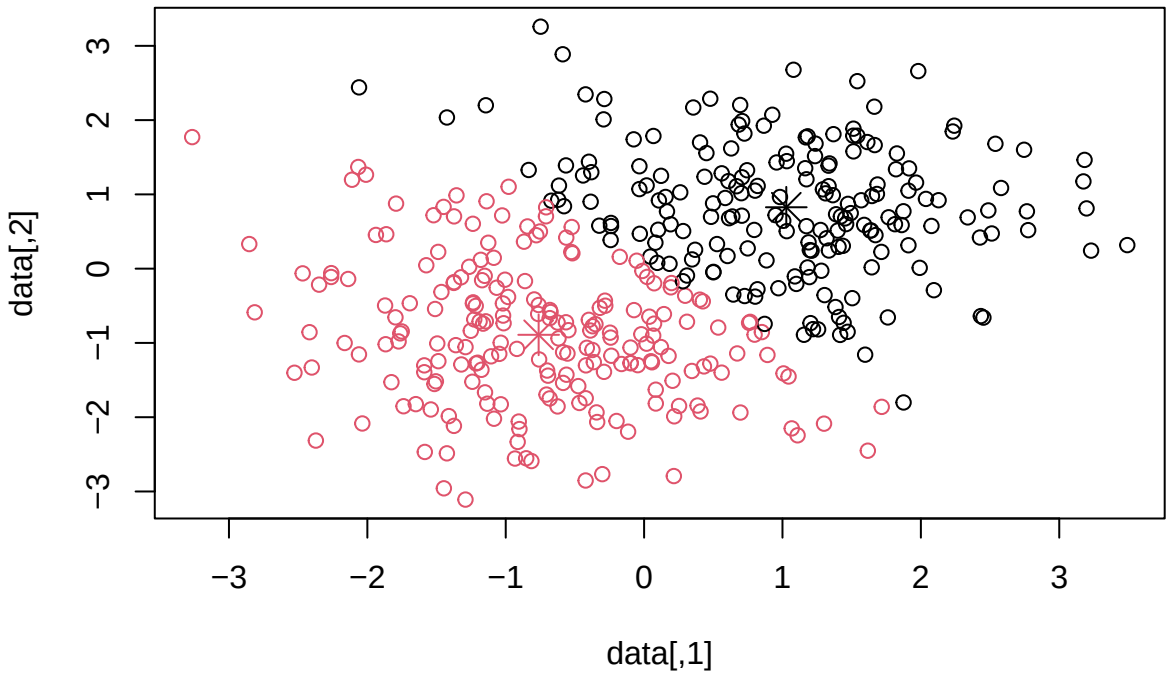
```r
library(mvtnorm)


gaussian.mixture.EM <- function(x, sigma, niter=100) {
  n <- dim(x)[1]  # number of observations
  d <- dim(x)[2]  # dimension of data (should be 2)

  # vectors storing the histories of all parameters
  pi <- numeric(niter+1)
  mu1 <- matrix(nrow=niter+1, ncol=d)
  mu2 <- matrix(nrow=niter+1, ncol=d)

  # initial values
  pi[1] <- 0.5
  mu1[1, ] <- colMeans(x) + rnorm(d)
  mu2[1, ] <- colMeans(x) + rnorm(d)
```

```
  # fixed covariance matrices
  cov1 <- sigma * diag(d)
  cov2 <- sigma * diag(d)

  for (i in 1:niter) {
    # Expectation
    x_density1 <- dmvnorm(x, mean=mu1[i,], sigma=cov1)
    x_density2 <- dmvnorm(x, mean=mu2[i,], sigma=cov2)

    resp <- 1 / (1 + ((1 / pi[i]) - 1) * exp(log(x_density2) - log(x_density1)))

    # Maximisation
    pi[i+1] <- mean(resp)
    mu1[i+1, ] <- colSums(resp * x) / sum(resp)
    mu2[i+1, ] <- colSums((1 - resp) * x) / sum(1 - resp)

  }

  final_responsibilities <- 1 / (1 + ((1 / pi[niter]) - 1) * exp(
    log(dmvnorm(x, mean=mu2[niter,], sigma=cov2)) - log(
      dmvnorm(x, mean=mu1[niter,], sigma=cov1))))

  return(
    list(pi=pi, mu1=mu1, mu2=mu2, cov1=cov1, cov2=cov2,
         responsibilities=final_responsibilities))
}
```

Part b

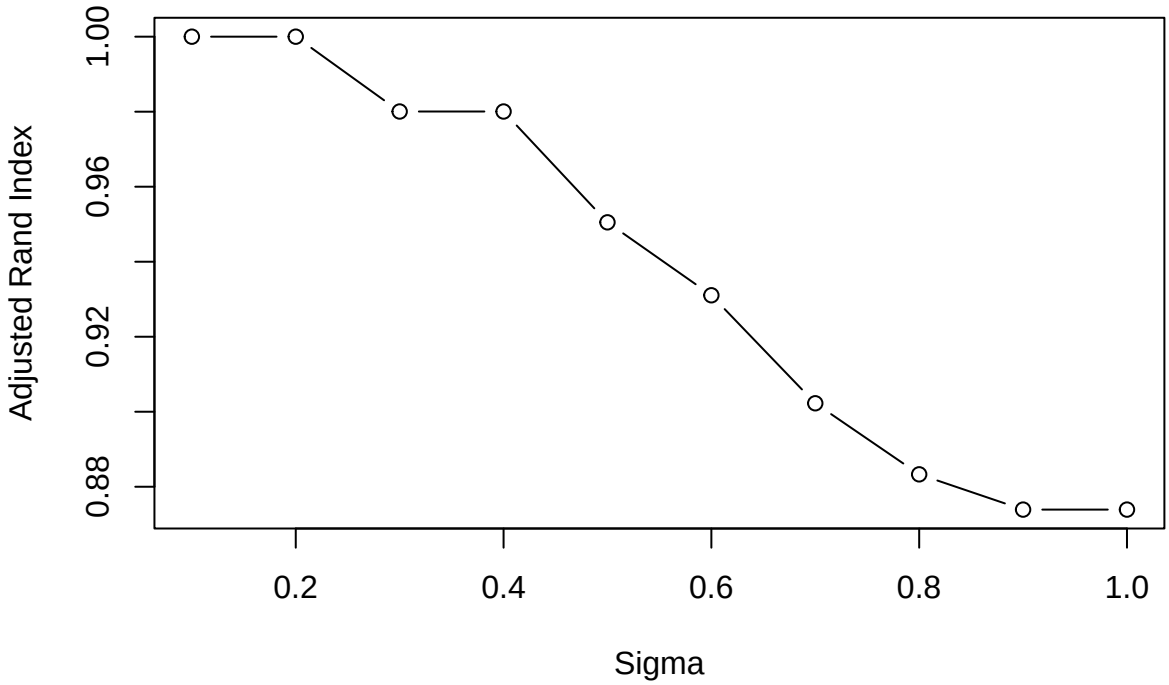
```
set.seed(123)

# generate samples
data1 <- mvtnorm::rmvnorm(200, mean=c(-0.8, -0.8), sigma=diag(2))
data2 <- mvtnorm::rmvnorm(200, mean=c(0.8, 0.8), sigma=diag(2))
data <- rbind(data1, data2)
```

```
# Apply K-means clustering
kmeans_result <- kmeans(data, centers=2)

plot(data, col=kmeans_result$cluster)
points(kmeans_result$centers, col=1:2, pch=8, cex=2)
```




```r
set.seed(123)

sigma_values <- seq(0.1, 1, by=0.1)
cluster_comparisons <- list()
for (sigma in sigma_values) {
    EM_fit <- gaussian.mixture.EM(data, sigma=sigma, niter=100)
    responsibilities <- EM_fit$responsibilities

    # to hard assignments
    EM_assignments <- ifelse(responsibilities > 0.5, 1, 2)

    # compare with K-means assignments
    comparison <- data.frame(KMeans=kmeans_result$cluster, EM=EM_assignments)
    cluster_comparisons[[as.character(sigma)]] <- comparison
}
```

```r
library(mclust)

adjusted_rand_indices <- numeric(length(sigma_values))
for (i in 1:length(sigma_values)) {
    sigma <- sigma_values[i]
    comparison <- cluster_comparisons[[as.character(sigma)]]
    adjusted_rand_indices[i] <- adjustedRandIndex(comparison$KMeans, comparison$EM)
}

plot(sigma_values, adjusted_rand_indices, type="b", xlab="Sigma", ylab="Adjusted Rand Index")
```



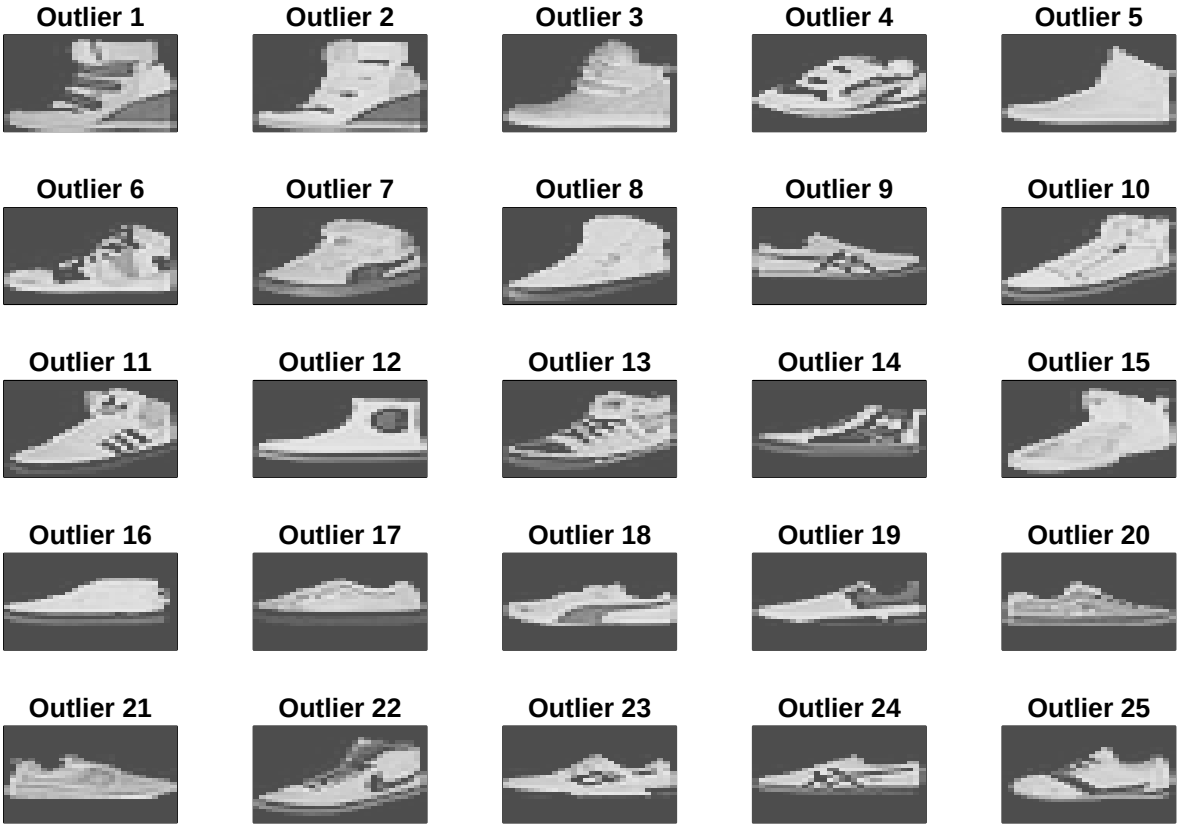
The plot shows the adjusted Rand index as a function of the σ parameter. The adjusted Rand index is a measure of the similarity between two clusterings, and a value closer to 1 indicates a higher similarity. From the plot, it appears that as σ increases, the similarity between the EM clustering and K-means clustering decreases. When σ is small, the adjusted Rand index is close to 1, suggesting that the EM responsibilities align closely with the K-means assignments. This is expected because as σ approaches 0, the clusters become more distinct and the EM algorithm behaves more like K-means with hard assignments. As σ increases, the clusters become more spread out and overlap more, leading to less agreement between the K-means and EM clustering assignments. This results in a lower adjusted Rand index. This trend is consistent with the hypothesis that as the variance of the clusters decreases (σ -> 0), the EM algorithm will produce results that are more similar to those of K-means.

Problem 3

Part a

```
library(dbscan)

sneaker_data <- read.csv('problem3a.csv', header = FALSE)

# compute LOF
k <- sqrt(nrow(sneaker_data))
lof_values <- lof(sneaker_data, minPts = ceiling(k))

# top 25 outliers
top_outliers_indices <- order(lof_values, decreasing = TRUE)[1:25]
# outlier images
outlier_images <- sneaker_data[top_outliers_indices, ]

plot_image <- function(image_vector, main = "") {
  image_vector <- as.numeric(image_vector)
```

```
  # reshape the image vector to a 28x28 matrix
  image_matrix <- matrix(image_vector, nrow = 28, byrow = TRUE)
  image_matrix <- t(apply(image_matrix, 2, rev))  # rotate the image

  image(1:28, 1:28, image_matrix, col = gray.colors(256), main = main, xaxt = 'n', yaxt = 'n')
}

outlier_images <- data.matrix(outlier_images)
par(mfrow=c(5,5), mai=c(0.2, 0.2, 0.2, 0.2))
for (i in 1:25) {
  image_data <- as.numeric(outlier_images[i, ])
  plot_image(image_data, main = paste("Outlier", i))
}
```

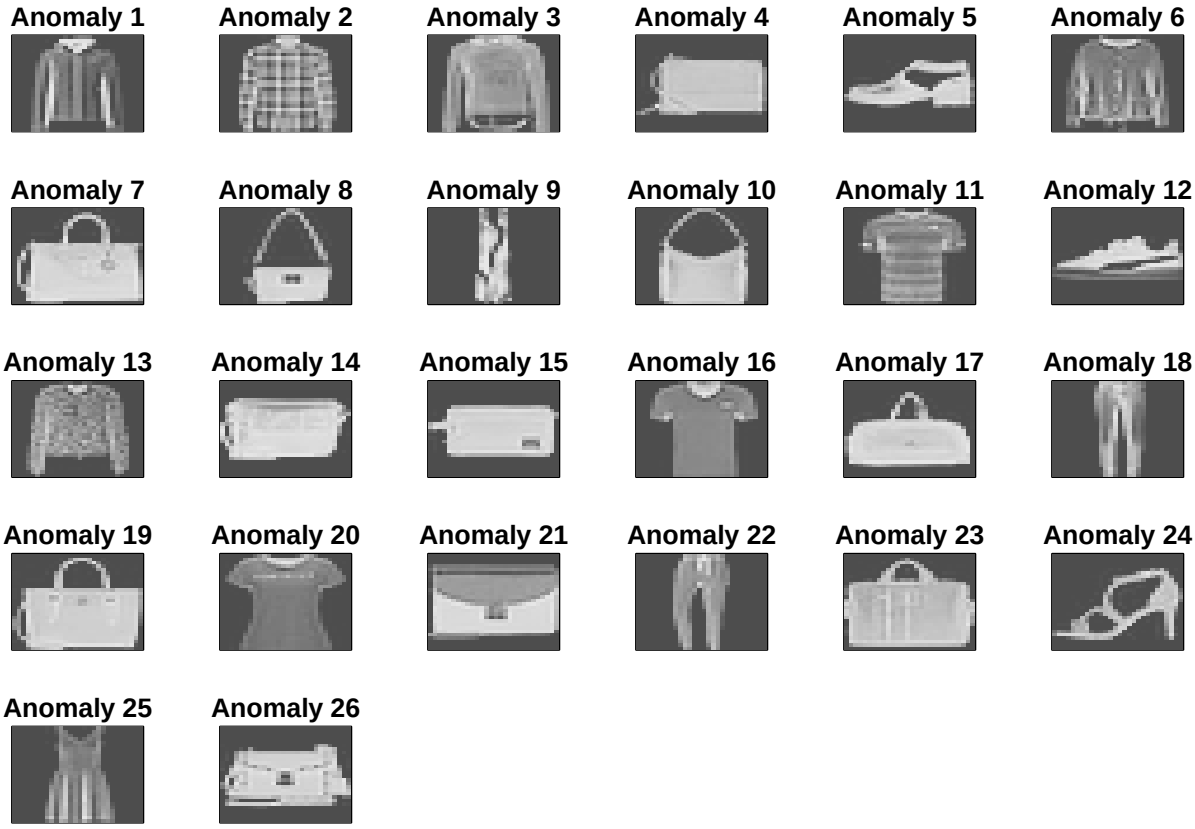


Generally, the images represent normal sneakers and I would not classify them as outliers. There are a few sneakers with a platform which might be considered not very typical.

Part b

```
sneaker_data <- read.csv('problem3a.csv', header = FALSE)
sneaker_data <- as.matrix(sneaker_data)

mixed_data <- read.csv('problem3b.csv', header = FALSE)
mixed_data <- as.matrix(mixed_data)
# combine both datasets
combined_data <- rbind(sneaker_data, mixed_data)
```

```r
# calculate LOF for the combined dataset
# with the same k value as calculated for the sneaker dataset
lof_values_combined <- lof(combined_data, minPts = ceiling(k))

# LOF scores for the mixed dataset
lof_values_mixed <- lof_values_combined[(nrow(sneaker_data) + 1):nrow(combined_data)]


threshold <- sort(lof_values, decreasing = TRUE)[25]
# identify anomalies
anomalies_indices <- which(lof_values_mixed > threshold)
anomaly_images <- mixed_data[anomalies_indices, ]

# plot a single image
plot_image <- function(image_vector, main = "") {
  image_vector <- as.numeric(image_vector)

  image_matrix <- matrix(image_vector, nrow = 28, byrow = TRUE)
  image_matrix <- t(apply(image_matrix, 2, rev))

  image(1:28, 1:28, image_matrix, col = gray.colors(256), main = main, xaxt = 'n', yaxt = 'n')
}

num_anomalies <- nrow(anomaly_images)
par(mfrow=c(5, ceiling(num_anomalies / 5)), mai=c(0.2, 0.2, 0.2, 0.2))
for (i in 1:num_anomalies) {
  plot_image(anomaly_images[i, ], main = paste("Anomaly", i))
}
```



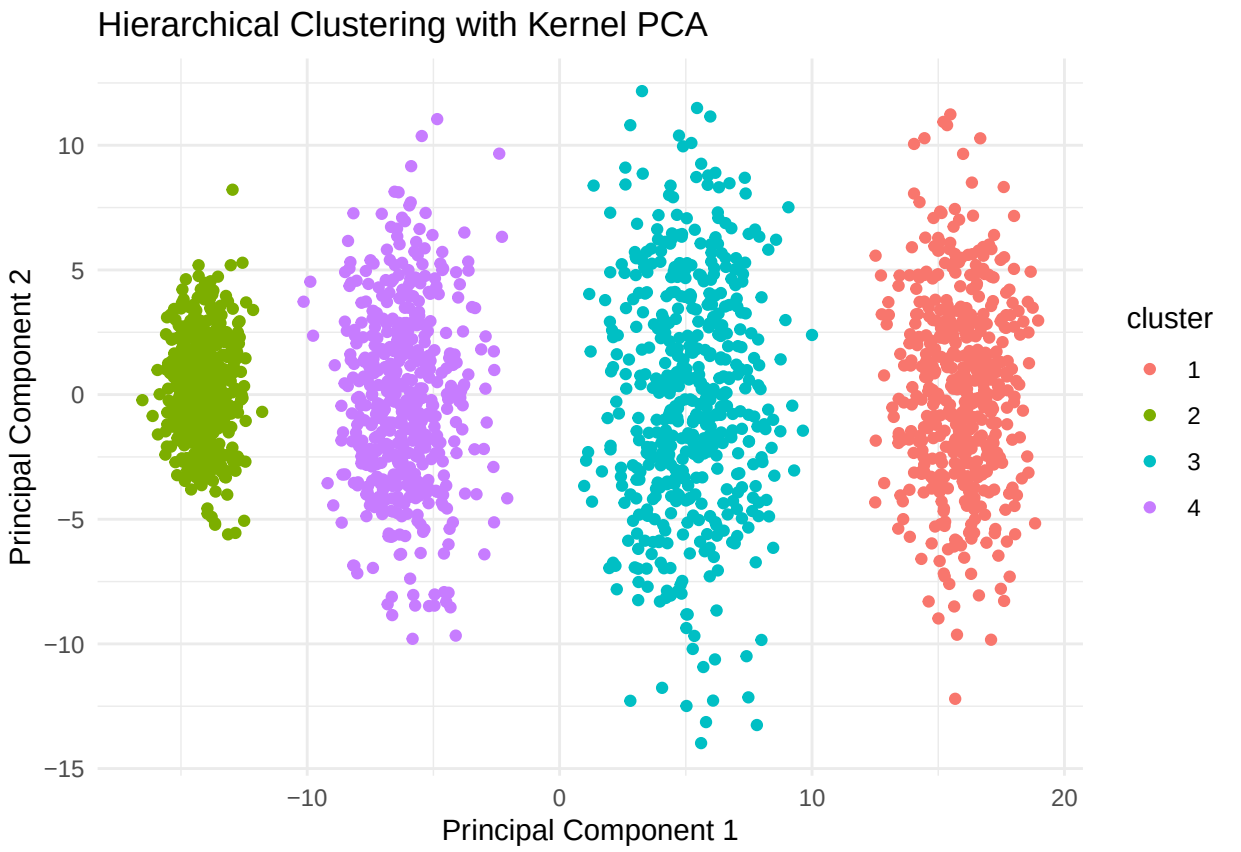
As a whole the method is able to successfully detect anomalies. There is only one case of all 26 detected anomalies, Anomaly 12, in which the method determines sneakers as an anomaly.

Problem 4

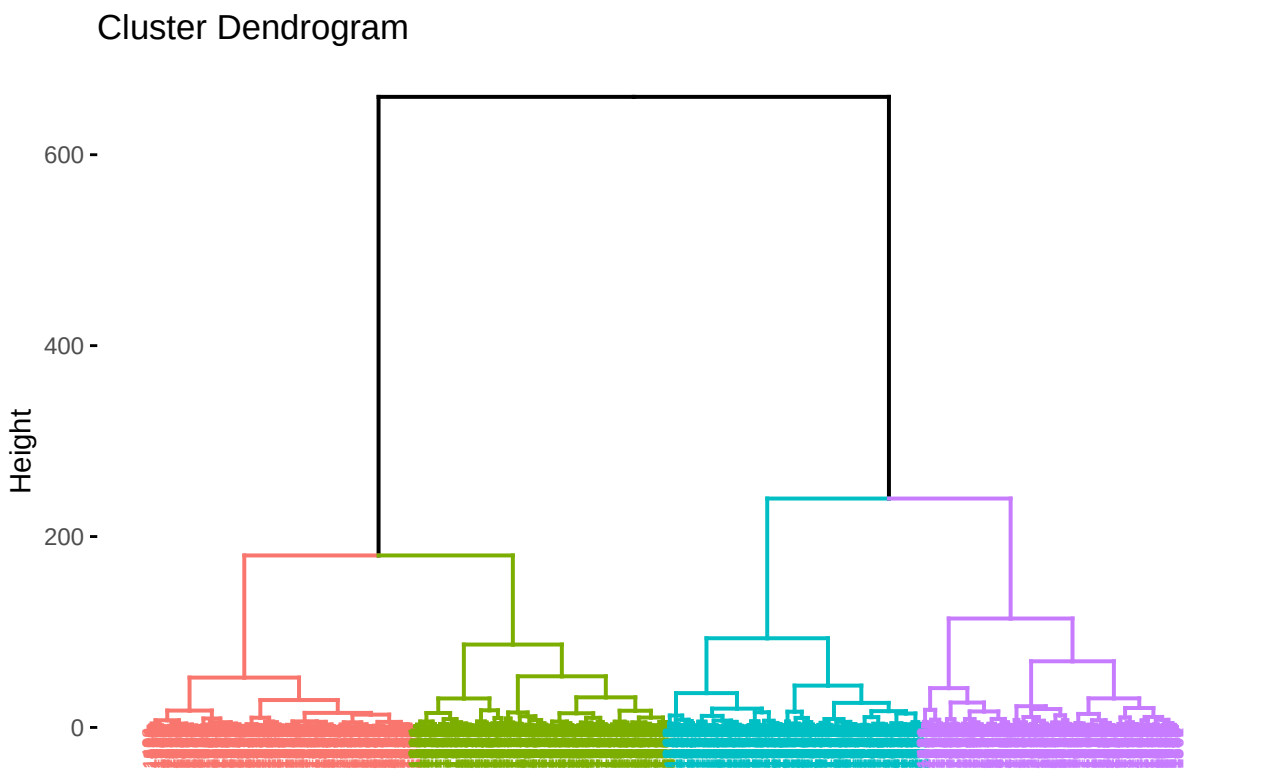
Part a

```
library(ggplot2)
library(factoextra)
library(kernlab)

data <- read.csv("problem4.csv", header = FALSE)
data <- scale(data)

# dimensionality reduction with Kernel PCA
set.seed(42)
kpca_result <- kpca(as.matrix(data), kernel = "rbfdot", features = 2, kpar = list(sigma = 1/50))
kpca_data <- as.data.frame(kpca_result@rotated)
colnames(kpca_data) <- c("PC1", "PC2")

# Hierarchical clustering
distance_matrix <- dist(kpca_data)
hc <- hclust(distance_matrix, method = "ward.D2")

# cut the dendrogram to create 4 clusters
clusters <- cutree(hc, k = 4)
```

```
kpca_data$cluster <- as.factor(clusters)
ggplot(kpca_data, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point() +
  labs(title = "Hierarchical Clustering with Kernel PCA",
       x = "Principal Component 1",
       y = "Principal Component 2") +
  theme_minimal()
```




```
# plotting dendrogram
fviz_dend(hc, k = 4, cex = 0.5)
```




```
filename <- '01252821_Levtcheva_p4.txt'

fileConn <- file(filename, open = "w")
for(label in clusters) {
  write(label, file = fileConn, ncolumns = 1)
}
close(fileConn)
```

Part b

Approach:

1. First, the data is scaled
2. Then, Kernel PCA (allows to uncover non-linear relationships within the data) is applied with `rbfdot` kernel, 2 features, and `sigma` equal to 1 divided by the value of the dimension of the data (50).

Note: In Python, `KernelPCA` from `sklearn.decomposition` has the default sigma value (called gamma there) set to this value. In R it is set to 0.1.

3. After that, agglomerative hierarchical clustering is applied to the distance matrix created by using the KernelPCA result. Finally, the dendrogram is cut such that it forms 4 clusters.