Multivariate Statistics - Assignment 2

Andrea Osmani Paolo Rinaldi Tri Duc Nguyen Ward Vandecruys

January 2026

***important disclaimer: due to space constraints, we only report the code for scenario 1 in Task 1. Including the full code would have caused the report to exceed the 15-page limit. The complete code can be found in the accompanying R script.**

1 Task 1

We name the different scenarios, create the train and test sets and center the data

```r
# Scenario s1 (large training set)
X_train_s1 <- train.data.s1
y_train_s1 <- train.target.s1
# Test set
X_test <- test.data
y_test <- test.target
# Centered data (covariance PCA)
Xc_train_s1 <- scale(X_train_s1, center = TRUE, scale = FALSE)
mu_s1 <- attr(Xc_train_s1, "scaled:center")
Xc_test_s1 <- scale(X_test, center = mu_s1, scale = FALSE)
```

1.1 For each scenario, conduct principal components analysis on the covariance matrix of the training data (i.e. centered variables) and select the number of components so that the components account for 90% of the variance in the training data

```r
#PCA>90% scenario 1
## PCA: Scenario s1 (covariance matrix, centered data)
pca_s1 <- prcomp(X_train_s1)
cumpropvar_s1 <- summary(pca_s1)$importance[3, ]
K_s1 <- min(which(cumpropvar_s1 >= 0.90))
K_s1
```

Output:

Table 1: Number of selected Principal Components (K) to reach 90% of explained variance.

Scenario	**Description**	**Components (K)**
Scenario s_1	Large training set	69
Scenario s_2	Small training set	65

1.2 Compute the training and test error of the following classifiers

1.2.1 LDA conducted on unstandardized principal components

```r
## Unstandardized PCA scores: Scenario s1
Z_train_s1 <- pca_s1$x[, 1:K_s1]
Z_test_s1  <- predict(pca_s1, newdata = X_test)[, 1:K_s1]
## LDA: Scenario s1
lda_s1 <- lda(Z_train_s1, grouping = y_train_s1)
## Training error
pred_train_s1 <- predict(lda_s1, Z_train_s1)
tab_train_s1 <- table(y_train_s1, pred_train_s1$class)
train_err_s1 <- 1 - sum(diag(tab_train_s1)) / sum(tab_train_s1)
## Test error
pred_test_s1 <- predict(lda_s1, Z_test_s1)
tab_test_s1 <- table(y_test, pred_test_s1$class)
test_err_s1 <- 1 - sum(diag(tab_test_s1)) / sum(tab_test_s1)
tab_train_s1
train_err_s1
tab_test_s1
test_err_s1
```

Output:

Table 2: LDA Performance Summary: Training and Test Error Rates for Scenarios s_1 and s_2.

Scenario	**Training Error**	**Test Error**
Scenario s_1 (Large set)	0.0958	0.1094
Scenario s_2 (Small set)	0.0863	0.1269

Table 3: LDA Confusion Matrix - Scenario s_1 (Test Set).

	Predicted			
Actual	D	G	O	Q
D	349	10	37	4
G	6	348	5	41
O	24	3	367	6
Q	6	15	18	361

Interpretation of results:

Results show a stable but limited behavior of LDA calssifier. On the large set s_1 the gap between training and test error is around 1.3%, which shows that the model does not suffer from overfitting. Anyway, the model shows a consistent bias, probably given by the fact that the assumption of linearity is too restrictive in order to capture the true boundaries of different classes.
On the small set s_1 we see that the error test grows around only 2 percentage points despite the drastic reduction in sample size. This behavior confirms one of the most important properties of LDA: being a low variance model, since it works with only one covariance matrix, it tends to be robust even when data is scarce. From the confusion matrix we can observe that the main number of errors is concentrated between two specific classes, D and O. On the contrary, G and Q classes appear to be much more linearly separable.

1.2.2 QDA conducted on the unstandardized principal components

```
## QDA-Scenario s1
## Fit QDA
qda_s1 <- qda(Z_train_s1, grouping = y_train_s1)
## Training error
pred_train_s1_qda <- predict(qda_s1, Z_train_s1)
tab_train_s1_qda <- table(y_train_s1, pred_train_s1_qda$class)
train_err_s1_qda <- 1 - sum(diag(tab_train_s1_qda)) / sum(tab_train_s1_qda)
tab_train_s1_qda
train_err_s1_qda
## Test error
pred_test_s1_qda <- predict(qda_s1, Z_test_s1)
tab_test_s1_qda <- table(y_test, pred_test_s1_qda$class)
test_err_s1_qda <- 1 - sum(diag(tab_test_s1_qda)) / sum(tab_test_s1_qda)
tab_test_s1_qda
test_err_s1_qda
```

Output:

Table 4: QDA Performance Summary: Training and Test Error Rates.

Scenario	Training Error	Test Error
Scenario s_1 (Large set)	0.0460	0.0575
Scenario s_2 (Small set)	0.0263	0.0675

Table 5: QDA Confusion Matrix - Scenario s_1 (Test Set).

	Predicted			
Actual	D	G	O	Q
D	353	3	37	7
G	0	391	0	9
O	16	2	372	10
Q	2	5	1	392

Interpretation of results:

The most evident result about QDA classification is the drastic improvement of train and test errors in the large set s_1 with respect to LDA analysis. This result can be interpreted due to the adaptiveness of quadratic decision boundaries rather than linear ones.
We can also state that there are signals of overfitting for what it concern the small set s_2, since train set is extremely low and test set grows around 1 percentage point. This is the "cost of complexity" as we saw during registered lessons, since QDA needs to estimate a covariance matrix for each class: this is a signal that the model is becoming too complex for the amount of given data. By observing the confusion matrix we can state that classification errors for class D towards class O remain the dame, while for class O towards class D there's an improvement. Moreover, QDA classification shows to be almost perfect in recognising class G elements.

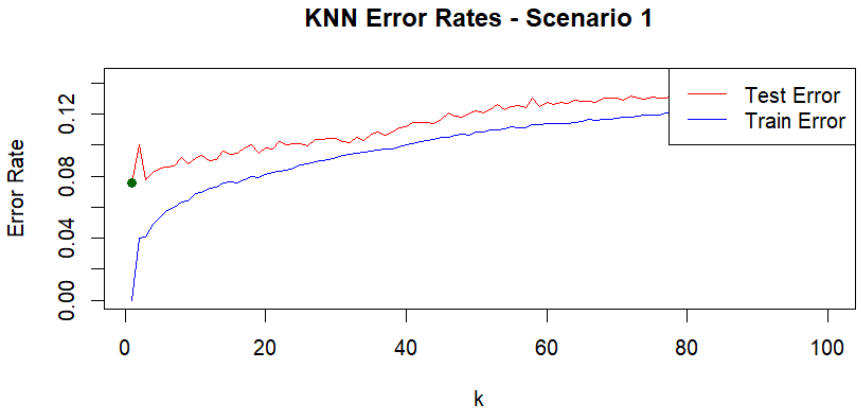
1.2.3 KNN conducted on the unstandardized principal components

```
## KNN tuning-Scenario s1
knnmax_s1 <- 100  # come nelle slide (puoi aumentare dopo)
err_knn_s1 <- matrix(0, nrow = knnmax_s1, ncol = 2)
colnames(err_knn_s1) <- c("train_err", "test_err")
```

```r
for (k in 1:knnmax_s1) {
  # training predictions (train vs train)
  pred_train_knn_s1 <- knn(train = Z_train_s1,
                           test  = Z_train_s1,
                           cl    = y_train_s1,
                           k     = k)

  tab_train_knn_s1 <- table(y_train_s1, pred_train_knn_s1)
  err_knn_s1[k, "train_err"] <- 1 - sum(diag(tab_train_knn_s1)) / sum(tab_
      train_knn_s1)
  # test predictions (train vs test)
  pred_test_knn_s1 <- knn(train = Z_train_s1,
                          test  = Z_test_s1,
                          cl    = y_train_s1,
                          k     = k)
  tab_test_knn_s1 <- table(y_test, pred_test_knn_s1)
  err_knn_s1[k, "test_err"] <- 1 - sum(diag(tab_test_knn_s1)) / sum(tab_
      test_knn_s1)
}
## best k minimizing test error
k_star_s1 <- which.min(err_knn_s1[, "test_err"])
k_star_s1
## final errors at k*
train_err_s1_knn <- err_knn_s1[k_star_s1, "train_err"]
test_err_s1_knn  <- err_knn_s1[k_star_s1, "test_err"]
train_err_s1_knn
test_err_s1_knn
# Plot of errors
plot(1:knnmax_s1, err_knn_s1[, "test_err"], type = "l", col = "red",
     ylim = c(0, max(err_knn_s1)), xlab = "k", ylab = "Error Rate",
     main = "KNN Error Rates - Scenario 1")
lines(1:knnmax_s1, err_knn_s1[, "train_err"], col = "blue")
legend("topright", legend = c("Test Error", "Train Error"),
       col = c("red", "blue"), lty = 1)
# Best K
points(k_star_s1, test_err_s1_knn, col = "darkgreen", pch = 19)
```

Plot 1:



Output:

Interpretation of results:

This method is strongly different from the previous ones since it is not parametric, but has a "local" approach. We start by saying that persistently having the K=1 result for the large set s_1 made us double

Table 6: KNN Tuning Results: Optimal k and associated error rates for Scenario s_1 and s_2.

Scenario	**Best k (k^*)**	**Training Error**	**Test Error**
Scenario s_1 (Large set)	1	0.0000	0.0756
Scenario s_2 (Small set)	3	0.0626	0.1256

check various times. In the end, we convinced ourselves that this is due to the fact that the decision boundary is extremely jagged, and that's why the model assigns every point to the class of its own nearest neighbour. This is why the train set is zero, while the result of test set is anyway better that the LDA classification one.
On the small set s_2 we received K=3 as a result, and we think that this is due to the fact that having less data, spacing of points is more scattered. Test errors grows at more that 10% for the small set, which demonstrates that KNN classification method strongly suffers data scarcity.

1.2.4 Multinomial logistic regression on the unstandardized principal components

```
## Multinomial logistic regression: Scenario s1
train_df_s1 <- data.frame(y_train_s1 = y_train_s1, Z_train_s1)
test_df_s1  <- data.frame(Z_test_s1)
# Fit multinomial logit on training PC scores
mlogit_s1 <- multinom(y_train_s1 ~ ., data = train_df_s1, maxit = 1000,
    Hess = TRUE)
# Training predictions + error
pred_train_s1_mlogit <- predict(mlogit_s1, newdata = train_df_s1)
tab_train_s1_mlogit <- table(y_train_s1, pred_train_s1_mlogit)
train_err_s1_mlogit <- 1 - sum(diag(tab_train_s1_mlogit)) / sum(tab_train_
    s1_mlogit)
tab_train_s1_mlogit
train_err_s1_mlogit
# Test predictions + error
pred_test_s1_mlogit <- predict(mlogit_s1, newdata = test_df_s1)
tab_test_s1_mlogit <- table(y_test, pred_test_s1_mlogit)
test_err_s1_mlogit <- 1 - sum(diag(tab_test_s1_mlogit)) / sum(tab_test_s1_
    mlogit)
tab_test_s1_mlogit
test_err_s1_mlogit
```

Output:

Table 7: Multinomial Logistic Regression Performance: Training and Test Error Rates.

Scenario	**Training Error**	**Test Error**
Scenario s_1 (Large set)	0.0805	0.0925
Scenario s_2 (Small set)	0.0519	0.1181

Interpretation of results:
This discriminant method showed to be superior to LDA but not to QDA and KNN for large set s_1. We can interpret this result by saying that logistic regression is more relaxed than LDA, since does not need to have data with identical covariance. This probably is the reason why logistic regression can adapt better to irregular data, by avoiding too-strong assumptions.
For small set s_2, the error grows at almost 12%, which indicates an intermediate result between LDA and KNN.

This method is strongly different from the previous ones since it is not parametric, but has a "local" approach. We start by saying that persistently having the K=1 result for the large set s_1 ma

1.2.5 Multinomial logistic regression on the unstandardized principal components and the squared unstandardized principal components

```r
## Multinomial logit + squared PCs: Scenario s1
# squared principal components
Z_train_s1_sq <- Z_train_s1^2
Z_test_s1_sq  <- Z_test_s1^2
# training and test data frames
train_df_s1_quad <- data.frame(
  y_train_s1 = y_train_s1,
  Z_train_s1,
  Z_train_s1_sq
)
test_df_s1_quad <- data.frame(
  Z_test_s1,
  Z_test_s1_sq
)
#model fit
mlogit_s1_quad <- multinom(y_train_s1 ~ ., data = train_df_s1_quad,  maxit
    = 2000,Hess = TRUE)
# Training predictions + error
pred_train_s1_quad <- predict(mlogit_s1_quad, newdata = train_df_s1_quad)
tab_train_s1_quad <- table(y_train_s1, pred_train_s1_quad)
train_err_s1_mlogit_quad <-
  1 - sum(diag(tab_train_s1_quad)) / sum(tab_train_s1_quad)
tab_train_s1_quad
train_err_s1_mlogit_quad
## Test predictions + error
pred_test_s1_quad <- predict(mlogit_s1_quad, newdata = test_df_s1_quad)
tab_test_s1_quad <- table(y_test, pred_test_s1_quad)
test_err_s1_mlogit_quad <-
  1 - sum(diag(tab_test_s1_quad)) / sum(tab_test_s1_quad)
tab_test_s1_quad
test_err_s1_mlogit_quad
```

Output:

Table 8: Quadratic Multinomial Logistic Regression (Squared PCs): Error Rates for Scenario s_1 and s_2.

Scenario	Training Error	Test Error
Scenario s_1 (Large set)	0.0552	0.0838
Scenario s_2 (Small set)	0.0000	0.1644

Interpretation of results:

In order to capture data curvature, shown to be important by the QDA classifier, we expected quadratic multinomial logistic regression to be more efficient than linear one. For the large set s_1, our intuition was right: test error is reduced by almost 1 percentage point.
For what it concerns the small set s_2, we witnessed an explosion of parameters, since training error is 0% while test error jumps to more than 16%. This is because the number of parameters approaches the sample size, since we had only few hundred observations and the model needed to estimate over 500 coefficients. The result is unnatural decision boundaries in order to perfectly classify train points, but at the cost of losing ability to generalize to new data.

1.2.6 Gradient boosting conducted on unstandardized principal components

```
## Gradient Boosting: Scenario s1
err2<-function(observed,predicted)
{tab<-table(observed,predicted)
err2<-1-sum(diag(tab))/sum(tab)
return(err2)
}
# target must be 0,1,2,3
train_target_s1_xgb <- as.integer(y_train_s1) - 1
test_target_xgb     <- as.integer(y_test) - 1
# predictors matrix
dtrain_s1 <- xgb.DMatrix(data = as.matrix(Z_train_s1), label = train_target
    _s1_xgb)
dtest_s1  <- xgb.DMatrix(data = as.matrix(Z_test_s1),  label = test_target_
    xgb)
# specify parameters
params_s1 <- list(
  objective = "multi:softprob",
  eval_metric = "mlogloss",
  num_class = 4,
  eta = 0.1,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.5
)
set.seed(1)
# cross validation
cv_s1 <- xgb.cv(
  params = params_s1,
  data = dtrain_s1,
  nrounds = 1000,
  nfold = 10,
  early_stopping_rounds = 20,
  verbose = 1
)
best_nrounds_s1 <- cv_s1$early_stop$best_iteration
cat("Best number of trees (s1):", best_nrounds_s1, "\n")
# estimate final model
final_model_s1 <- xgb.train(
  params = params_s1,
  data = dtrain_s1,
  nrounds = best_nrounds_s1
)
# TRAIN predictions
pred_vec_train_s1 <- predict(final_model_s1, newdata = as.matrix(Z_train_s1
    ))
pred_mat_train_s1 <- matrix(pred_vec_train_s1, ncol = 4, byrow = FALSE)
class_train_s1 <- apply(pred_mat_train_s1, 1, which.max) - 1
table(train_target_s1_xgb, class_train_s1)
train_err_s1_xgb <- err2(train_target_s1_xgb, class_train_s1)
train_err_s1_xgb
#   TEST predictions
pred_vec_test_s1 <- predict(final_model_s1, newdata = as.matrix(Z_test_s1))
pred_mat_test_s1 <- matrix(pred_vec_test_s1, ncol = 4, byrow = FALSE)
class_test_s1 <- max.col(pred_mat_test_s1) - 1
table(test_target_xgb, class_test_s1)
test_err_s1_xgb <- err2(test_target_xgb, class_test_s1)
test_err_s1_xgb
```

Output:

Table 9: XGBoost Results: Optimal number of trees and error rates for both scenarios.

Scenario	Optimal Trees (n_{rounds})	Training Error	Test Error
Scenario s_1 (Large set)	383	0.0000	0.0606
Scenario s_2 (Small set)	236	0.0000	0.1000

Interpretation of results:

XGBoost shows to be one of the most efficient classification methods used in this analysis. From the results, we can see that for both large and small sets we have a 0% train error, which means that the model correctly classified every training point. From the good results also in test errors, especially for s_1, we can say that that the model generalized well, avoiding the risk of overfitting.
We stress the fact that, differently from what happened with quadratic logistic regression, the test error for s_2 remained around acceptable levels (10%), which in this case confirms the superiority of boosted decisional trees in complex but small datasets.

1.2.7 HDDA conducted on all the centered variables in the training set and using the common dimension model "AKJBKQKD" in which you select the number of components using the method of Cattell with threshold=0.05

```
#Fit HDDA with common dimension model + Cattell (threshold = 0.05)
hdda_s1 <- hdda(Xc_train_s1, y_train_s1, model = "AKJBKQKD", d_select = "
    Cattell", threshold = 0.05, scaling = FALSE)
#Predict on training + error
pred_train_s1_hdda <- predict(hdda_s1, Xc_train_s1, y_train_s1)
# pred_train_s1_hdda$class contains predicted classes
table(y_train_s1, pred_train_s1_hdda$class)
train_err_s1_hdda <- err2(y_train_s1, pred_train_s1_hdda$class)
train_err_s1_hdda
#Predict on test + error
pred_test_s1_hdda <- predict(hdda_s1, Xc_test_s1, y_test)
table(y_test, pred_test_s1_hdda$class)
test_err_s1_hdda <- err2(y_test, pred_test_s1_hdda$class)
test_err_s1_hdda
```

Output:

Table 10: HDDA Performance Summary: Training and Test Error Rates (d selected via Cattell threshold 0.05).

Scenario	Training Error	Test Error
Scenario s_1 (Large set)	0.0666	0.0831
Scenario s_2 (Small set)	0.0388	0.0863

Interpretation of results:

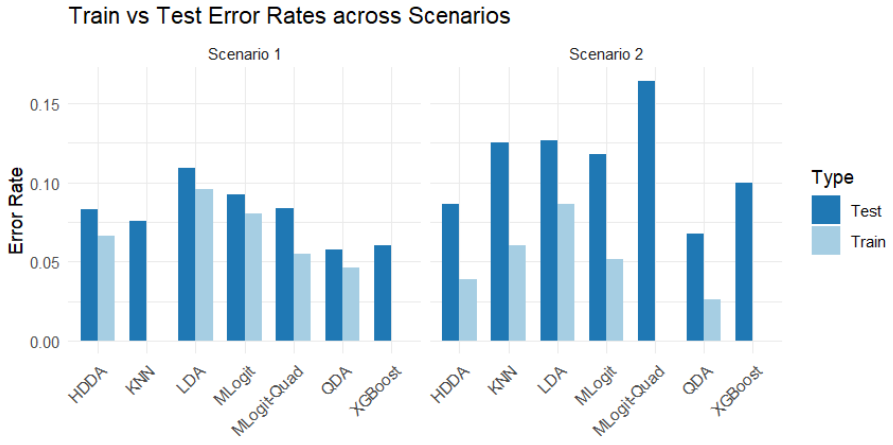
The main result about this classification method is the barely noticeable difference between test errors of large set and small set (around 0.32%). This shows how HDDA method remains stable as the sample size strongly varies, "protecting" the model from overfitting even when there is data scarcity. Anyway, even in this case we already saw different methods which obtained a much more precise performance, which may also be attributed to the fact that setting the Cattell treshold set at 0.05 is a conservative choice.
In any case, can say that HDDA classification is by far the most robust classification method seen in this analysis.

1.3 Make an overview table that includes the training and test error of each classifier for the two scenarios and visualize the results. Discuss the results of the analysis

Table 11: Overview of Training and Test Error Rates for all Classifiers in Scenarios s_1 and s_2.

	Scenario s_1 (Large Set)		**Scenario s_2 (Small Set)**	
Classifier	**Training Error**	**Test Error**	**Training Error**	**Test Error**
LDA	0.0958	0.1094	0.0863	0.1269
QDA	0.0460	**0.0575**	0.0263	**0.0675**
KNN	0.0000	0.0756	0.0626	0.1256
Multinomial Logit (Linear)	0.0805	0.0925	0.0519	0.1181
Multinomial Logit (Quadratic)	0.0552	0.0838	0.0000	0.1644
Gradient Boosting (XGBoost)	0.0000	0.0606	0.0000	0.1000
HDDA (Cattell)	0.0666	0.0831	0.0388	0.0863

Plot 2:



Final comparative analysis:

The analysis conducted on seven different classifiers based on two distinct scenarios (large set s_1 with N = 9600, small set s_2 with N = 1600) has allowed us to outline the nature of the data and robustness of each statistical approach. Our results show how performance is not only given by the computational power of the models, but above all from coherence between geometric assumptions made by classifiers and the actual distribution of classes.

The goal of the analysis was to discriminate between images of four distinct but similar letters: D, G, O, Q. A constant pattern through all models was the systematic confusion between the class of D and O, which differently from G and Q do not have unique topological characteristics (the horizontal bar for the G, the "tail" for the Q).

The role of covariance matrix: Generative Methods

The comparison between LDA and QDA has been the most revealing on data structure.

LDA showed an elevated bias, with test error $\approx 11\%$, which means that homoscedasticity assumption is too restrictive for this dataset. The high error rate confirms that the space of images of letters is not linearly separable, and LDA, which imposes linear boundaries, was not able to follow the curvature which characterizes an O differently from a D or a Q.

QDA instead emerged as the absolute best classifier, with test error $\approx 5.75\%$. This result shows that the heteroscedasticity assumption is correct: each class possess distinct covariance matrices. Quadratic boundaries were the most efficient in separating different classes, capturing an important amount of heterogeneity in the data.

Complexity and overfitting: Quadratic logistic regression and XGBoost

We observed two opposed approaches in modelling of non-linearity, with divergent results on the scenario with data scarcity, s_2.
For the quadratic logistic regression, the insertion of quadratic terms Z squared caused the explosion of parameters. With more than 500 weights to estimate on few hundreds of observations, the model lost every generalization capacity (Training Error 0% $\rightarrow$ Test Error 16.44%). This is a perfect example of violation of principle of parsimony: without structural constraints, the model interpolated the noise.
Differently, the XGBoost method, despite being a highly complex model, mantained a contained test error of 10%. The difference lies in the fact that with gradient boosting, decisional threes and shrinkage, approximates the decisional function in a incremental way, avoiding the catastrophic distortions suffered by quadratic logistic regression.

Curse of dimensionality also for KNN

Not only the Quadratic logistic regression suffered from the curse of dimensionality as we saw above, but also the KNN method did, in a different way.
While the Quadratic logistic regression "exhausted" the degrees of freedom in the s_2 set, KNN method suffered from sparcity and distance of data points. When proximity is lost, KNN fails because it does not have a theoretic structure, as the covariance matrix, to use in order to fill the gaps between data points.

Robustness and parsimony: HDDA

The High Dimensional Discriminant Analysis provided the most evident proof of robustness. While complex models suffered the transition between s_1 and s_2, HDDA mantained the test error almost constant ($\approx 8.3\% \rightarrow 8.6\%$). The use of Cattell criterion for dimension selection allowed the model to operate an efficient noise reduction, modelling each class into its specific subspace. Although slightly less precise than QDA, HDDA is confirmed to be the model with less variance.

Conclusions

In conclusion, we can say that the overall analysis shows that the classification of these characters (D, G, O, Q) in pixeled images requires models which are able to capture **non-linearity** and **heteroscedasticity**.

1. QDA confirms itself as the optimal choice, balancing the necessary flexibility to model morphological variations with the structural robustness of Gaussian distributions.
2. HDDA represents the most secure alternative in data scarcity conditions, thanks to its capacity to extract the signal of the class from dimensional noise.

2 Task 2

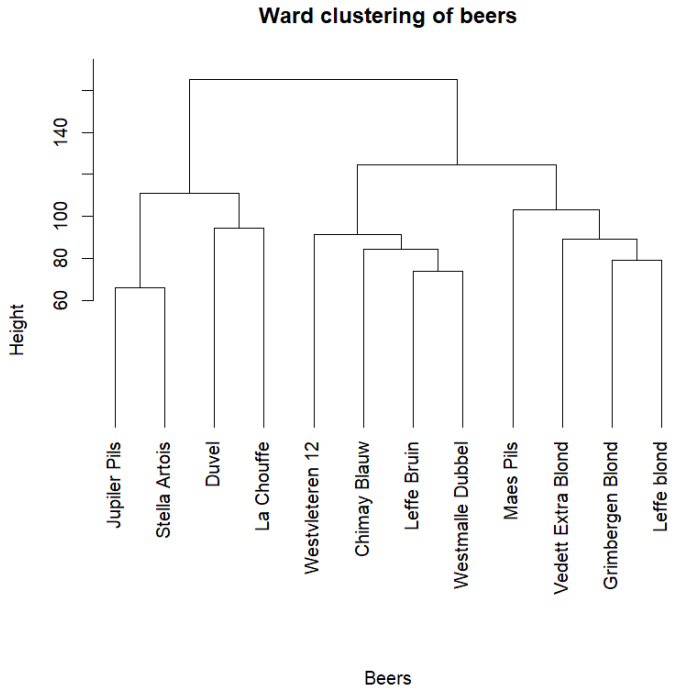
The dataset is a rank matrix with rows representing respondents (people) and columns representing 12 beers. Entries are ordinal ranks (1 = best, 12 = worst) for each person across all beers. The goal is to (i) identify groups of beers that are ranked similarly across people, (ii) assess how stable the discovered groups are with respect to the sampled respondents, and (iii) visualise the preference structure in two dimensions using unfolding.

2.1 Use a hierarchical clustering on squared Euclidean distances using the Method of Ward to cluster the 12 beers. Draw the dendrogram and discuss the results.

```
# Ward needs to be applied to euclidean distances
d_beers <- dist(t(beer), method = "euclidean")
# Cluster using Ward's method
hc_beer <- hclust(d_beers, method = "ward.D2")
```

```r
# Beers that have splits closer at the bottom are more similar so Jupiler 'very' similar to Stella
# First big cut at 165, then at 125, 105 and so on
# Same can be said about Grimbergen Blond and Leffe Blond, but also Leffe Bruin and Westmalle Dubbel (not from personal experience :)
plot(hc_beer, main="Ward clustering of beers", xlab="Beers", sub="", hang=-1)
```

Plot 3:



Discussion:

The dendrogram is the hierarchical picture behind the beer clustering: it shows how beers get merged step by step based on how similarly people rank them. The key thing to read is the height of the merges—beers that join at very low height have very similar ranking profiles, while merges at high height indicate that two groups are quite different and are only being forced together late in the algorithm.

A few strong patterns stand out. First, Jupiler Pils and Stella Artois merge extremely early, which suggests that respondents treat them almost interchangeably in their rankings; they form the clearest "pils/mainstream" pair in the data. Similarly, Grimbergen Blond and Leffe blond merge early, indicating a second tight similarity pair on the blond/abbey side. On the stronger/darker side, beers like Chimay Blauw, Leffe Bruin, Westmalle Dubbel, and Westvleteren 12 cluster on the same branch, meaning there is a subset of respondents who tend to rank these beers in a consistent way relative to the others.

The dendrogram also helps motivate why three clusters make sense. We can see a natural separation into (i) the mainstream/pils branch, (ii) the heavier Trappist/dubbel branch, and (iii) an intermediate blond/special branch. Cutting the tree to force four clusters would require splitting one of these branches further at a lower height, which is a more fragile decision—this matches what the ARI table showed: is less stable across train/validation splits. So the dendrogram provides the intuitive "shape" of the data, while the ARI results justify the final cut level statistically.

2.2 Split the data set in a training set and a validation set by assigning odd-numbered observations to the training set and by assigning even-numbered observations to the validation set.

2.2.1 Compute, for the training set and for the validation set, the following clustering solutions with 3 and 4 clusters:

1. Hierarchical clustering on squared Euclidean distances using the method of Ward
2. The best k-means solution using 100 random starting points
3. Hierarchical clustering on squared Euclidean distances using the method of Ward followed by k-means using the centroids of the hierarchical clustering as a starting point

```r
# Splitting odd data points -> training data end even data points-> test data
train <- beer[seq(1, nrow(beer), by = 2), , drop = FALSE]
valid <- beer[seq(2, nrow(beer), by = 2), , drop = FALSE]
# Function for clustering using Ward's method, this keeps the for loop cleaner
ward_hc <- function(data, k){
  hc <- hclust(dist(t(data), "euclidean"), method = "ward.D2")
  cutree(hc, k)}
# Same function logic applies to this k-means clustering function
kmeans_best <- function(data, k){
  set.seed( 1425)
  kmeans(t(data), centers = k, nstart = 100)$cluster}
# Ward clusters -> centroids for k-means
ward_kmeans <- function(data, k){
  X <- t(data)
  hc <- hclust(dist(X, "euclidean"), method = "ward.D2")
  cl0 <- cutree(hc, k)
  centers0 <- rowsum(X, cl0) / as.vector(table(cl0))
  kmeans(X, centers = centers0)$cluster}
```

2.2.2 Next compute, for each of the 6 solutions (i.e., 3 methods x 2 values for the number of clusters), the stability of the cluster solution in a split half-approach using the adjusted rand index as criterion.

```r
# Define the 3 methodologies: Ward with cutree, k-means and Ward as a kickstart for k-means
methods <- list(Ward = ward_hc, KMeans = kmeans_best, WardKMeans = ward_kmeans)
# Store the results in a table [Method used, number of clusters used, ARI]
results <- data.frame(Method = character(), K = integer(), ARI = numeric())
# Loop over the methods, number of clusters for each combination see how similar train and test is
for(m in names(methods)){
  for(k in c(3,4)){
    cl_tr <- methods[[m]](train, k)
    cl_va <- methods[[m]](valid, k)
    results <- rbind(results, data.frame(Method = m, K = k, ARI = adjustedRandIndex(cl_tr, cl_va)))
  }
}
# Both KMeans and WardKMeans with 3 clusters are tied
# We could opt to use more complex strategies such as K-fold CV using a longer of cluster amounts
# But this is outside of the scope of the course we there for use regular K-Means due to less complexity.
```

```r
print(results)
```

Output:

Table 12: Split-half stability (train vs validation) using Adjusted Rand Index (ARI) for $K \in \{3, 4\}$.

Method	K	ARI
Ward	3	0.7372
Ward	4	0.6024
KMeans	**3**	**1.0000**
KMeans	4	0.8189
WardKMeans	**3**	**1.0000**
WardKMeans	4	0.6024

Interpretation of results:

This table is the stability argument for choosing the clustering method and the number of clusters. Across the split (odd people vs even people), KMeans with and WardKMeans with both achieve ARI = 1.000, meaning the beer partition is exactly the same in train and validation (up to relabelling). That's very strong evidence that a 3-cluster structure is genuinely present in the data and is not sensitive to which respondents used.
When moving to , ARI drops for all methods (even though KMeans stays relatively high at 0.819). This suggests that forcing an extra cluster starts to split beers in a way that is less reproducible across halves, i.e. it may be capturing more sample-specific noise rather than a robust additional segment.
Finally, Ward's method is only moderately stable (0.737 at , 0.602 at). That doesn't mean Ward is "wrong", but it indicates that the specific hierarchical cut is more sensitive to the sampled people than the k-means based solutions here. Overall, the table supports selecting as the most defensible choice, and it also shows that the resulting partition is so strong that both plain k-means and Ward-initialised k-means converge to the same stable solution.

2.2.3 Apply the clustering method with the highest value of the adjusted rand index to the entire 399 x 12 matrix to obtain a final clustering solution.

```r
# Highest ARI = most stable cluster with respect to the people used
# Here K-Means is the best by default since it iss the first appearance of
    the max
best <- results[which.max(results$ARI), ]
print(best)
# Using k-Means with k = 3 we refit on the full dataset
final_beer_clusters <- methods[[best$Method]](beer, best$K)
# Extra fit on the WardKMeans k = 3
final_beer_clusters2 <- methods[["WardKMeans"]](beer, 3)
# Both have the exact same clusters just different labelling orders
# Maes seems to be in the wrong cluster though, it should swap with duvel
# If we were to cut at 3 in (a) we would almost get the same clusters
    outside of Lachouffe switching
print(final_beer_clusters)
print(final_beer_clusters2)
```

Output:

Interpretation of results:

This table is the final output of the analysis: after selecting the most stable configuration (from the ARI table), we refit the chosen clustering on the full dataset to get the definitive segmentation of the 12 beers. A clear interpretation is that the data support three distinct preference profiles:

Table 13: Final beer clusters ($K = 3$) refit on the full dataset.

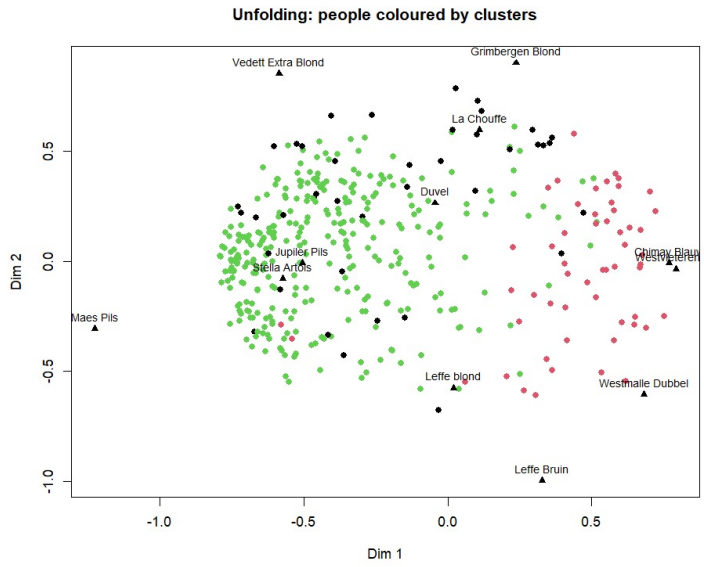
Cluster label	Beers
1	Leffe Bruin; Chimay Blauw; Westvleteren 12; Westmalle Dubbel
2	Maes Pils; Grimbergen Blond; Leffe blond; La Chouffe; Vedett Extra Blond
3	Jupiler Pils; Stella Artois; Duvel

Cluster 3 (Jupiler, Stella, Duvel) groups beers that respondents tend to treat similarly. The Jupiler–Stella pairing is very intuitive from the dendrogram, and Duvel joining them suggests that, in this sample, Duvel's ranking pattern is closer to the "mainstream/pils side" than to the darker Trappist group.
Cluster 1 (Leffe Bruin, Chimay, Westvleteren 12, Westmalle Dubbel) is a very coherent "strong/dark/-Trappist–dubbel" block. These beers are consistently ranked in a similar way by respondents, which is why they remain together across methods and splits.
Cluster 2 (Maes, Grimbergen Blond, Leffe blond, La Chouffe, Vedett) forms the remaining segment, which we can think of as an intermediate "blond/special" cluster. It contains the tight blond pair (Grimbergen–Leffe blond), plus Vedett and La Chouffe, and it also absorbs Maes. The fact that Maes doesn't fall into the Jupiler–Stella cluster is actually informative: it means respondents do not rank Maes like the other pils beers in this dataset, so its "market position" in terms of preferences is closer to the blond/special group.
Overall, this table gives a compact way to describe the structure discovered: three stable beer categories, each reflecting a different pattern of how people rank the beers, and these categories are what we then use to colour respondents in the unfolding map.

2.3 Conduct two-dimensional ordinal row-conditional unfolding on the rankings of the beers. Visualize the final clustering solution in the obtained unfolding configuration by using colored dots for the persons. Discuss what you can conclude about the preferences of the clusters from the configuration plot and from the centroids of the clusters.

```r
# 2D ordinal unfolding to minimize the distance between each person and
    their higher ranking beers
unf <- unfolding(beer, ndim = 2, type = "ordinal", conditionality = "row")
# Person coordinates P
P <- unf$conf.row
# Beer coordinates B
B <- unf$conf.col
# Turn the beer clusters into groups by split the beers by their label
beer_groups <- split(1:ncol(beer), final_beer_clusters)
# For each person we calculate the mean rank per cluster -> lower rank
    means they like the beer
person_cl <- apply(beer, 1, function(r){
  m <- sapply(beer_groups, function(idx) mean(r[idx]))
  as.integer(names(which.min(m)))
})
# The green cluster seem to more strongly prefer pils beers
# The red cluster in turn more strongly prefer trappist beers
# The black cluster is more uniform and has mixed preferences
plot(P, type="n", main="Unfolding: people coloured by clusters", xlab="Dim
    1", ylab="Dim 2", xlim=xlim, ylim=ylim)
points(P, pch=16, col=person_cl)
points(B, pch=17)
text(B, labels=colnames(beer), pos=3, cex=0.8)
```

Plot 4:



Interpretation of results:

The unfolding plot is a 2D map of the ranking data: it places beers (triangles) and people (dots) in the same space so that, roughly, a person is positioned closer to the beers they rank better (because we used ordinal, row-conditional unfolding). Compared to clustering, which gives discrete groups, unfolding is useful because it shows the preference structure as a continuous geometry.

The most important feature of the map is the strong separation along Dimension 1. On one side we see the pils/mainstream beers (notably Jupiler and Stella, which also appeared as the tightest pair in the dendrogram). On the opposite side we see the strong/darker Trappist–dubbel beers (such as Chimay, Westvleteren 12, and Westmalle Dubbel). This suggests that the dominant latent preference axis in the sample is essentially a contrast between "pils-like" preferences and "Trappist/dubbel" preferences. Beers like Duvel and La Chouffe tend to sit more centrally/top-central, which fits their role as "bridge" beers—respondents who do not strongly prefer one extreme can still rank these relatively well.

Colouring people by the beer-cluster they rank best adds a clear interpretation: people coloured with the cluster containing Jupiler/Stella/Duvel concentrate toward the pils side of the map, while people coloured with the Trappist/dubbel cluster concentrate toward the opposite side. The third cluster (the blond/special group) typically appears more spread out, which is expected: it represents an intermediate preference segment and can contain respondents with more mixed rankings, so it does not form one tight cloud in 2D. Overall, the unfolding plot confirms that the three clusters found earlier correspond to a real preference structure, and it makes the main preference dimension visually explicit.

Conclusions

In conclusion, we can say that the overall analysis shows ordinal beer rankings using hierarchical clustering, k-means clustering, split-half stability assessment, and two-dimensional ordinal unfolding to identify and validate preference structures among 12 beers.

1. Ward's hierarchical clustering revealed clear similarity patterns and suggested a natural separation into three groups, corresponding to mainstream pils beers, intermediate blond/special beers, and stronger Trappist–dubbel beers. This was formally confirmed by the split-half stability analysis: for $K = 3$, both k-means and Ward-initialised k-means achieved perfect agreement between training and validation sets (ARI = 1.000), whereas solutions with four clusters were consistently less stable. Refitting the selected model on the full dataset yielded a robust three-cluster solution.

2. The ordinal row-conditional unfolding provided a complementary low-dimensional representation of the data. The configuration was dominated by a single latent dimension contrasting pils-type beers with Trappist–dubbel beers, with blond and special beers occupying an intermediate position. Respondents coloured by their preferred cluster were spatially consistent with the beer clusters, supporting the validity of the clustering solution.