Introduction to Machine Learning Classification lab for Mass Spectrometry data

Francisco Madrid, Toni Pardo

2023-03-30

Objectives

Introduce the following feature extraction and classification algorithms: • Principal Component Analysis (PCA) • k-Nearest Neighbours classifiers (kNN)

Apply those algorithms to spectra from prostate tissues measured with Surface-Enhanced Laser Desorption/Ionization (SELDI) Mass Spectrometry (from the ChemometricsWithRData package).

Dataset

We will use the prostate mass spectra from the ChemometricsWithRData package. Since this package is no longer in the CRAN repository you will need to install it directly from the ChemometricsWithR compressed file provided in the campus virtual. The Prostate2000Raw data contains 654 mass spectra, belonging to 327 subjects (two replicates per subject). Each subject belongs to one the following groups: • patients with prostate cancer • patients with benign prostatic hyperplasia • control subjects This data was made public in the following papers 1 and 2.

Procedure

In this lab we will explore several feature extraction and classification techniques applied to proteomic data. These techniques are: • Principal Component Analysis • K-Nearest Neighbours

---

1 B.L. Adam, Y.Qu, J.W. Davis, M.D. Ward, M.A. Clements, L.H. Cazares, O.J. Semmes, P.F. Schellhammer, Y. Yasui, Z. Feng, G.L. Wright, “Serum protein fingerprinting coupled with a pattern-matching algorithm distinguishes prostate cancer from benign prostate hyperplasia and healthy men”, Cancer Res. 63, 3609-3614, (2002) 2 Y. Qu, B.L. Adam, Y. Yasui, M.D. Ward, L.H. Cazares, P.F. Schellhammer, Z. Feng, O.J. Semmes, G.L. Wright , “Boosted decision tree analysis of surface-enhanced laser desorption/ionization mass spectral serum profiles discriminates prostate cancer from noncancer patients”, Clinical Chemistry, 48, 1835-1843, (2002)

We start loading all the needed packages and the data using the RStudio terminal. Optionally, you can obtain the same results using the following code lines: # run `install.packages` only if packages are not already installed!

```
install.packages(c("ChemometricsWithR", "MASS"))

install.packages(c("e1071", "sfsmisc", "class", "caret", ”lolR”))

packageurl <- "http://cran.r-project.org/src/contrib/Archive/ChemometricsWithRData/ChemometricsWithRData_0.1.3.tar.gz" install.packages(packageurl, repos=NULL, type="source")
```

```
library("ChemometricsWithR")

##
## Attaching package: 'ChemometricsWithR'

##    The    following    objects    are    masked    from    'package:stats':
##
##     loadings, screeplot

library("MASS")
library("pls")

##
## Attaching package: 'pls'

##  The  following  objects  are  masked  from  'package:ChemometricsWithR':
##
##     loadingplot, loadings, scoreplot, scores

##    The    following    object    is    masked    from    'package:stats':
##
##     loadings

library("sfsmisc")
library("e1071")
library("class")
library("caret")

## Loading required package: ggplot2

## Loading required package: lattice

##
## Attaching package: 'caret'

##    The    following    object    is    masked    from    'package:pls':
##
##     R2

library("lolR")
```

Loading the data

```
data(Prostate2000Raw, package = "ChemometricsWithRData")
mz_prost <- Prostate2000Raw$mz
intensity_with_replicates <- Prostate2000Raw$intensity
medical_cond <- Prostate2000Raw$type
```

Preprocessing

The spectra from the Prostate2000Raw dataset are already baseline corrected and normalized, according to the help page. We will perform two additional preprocessing steps: • Replicate averaging • Log transformation

Replicate averaging

As each subject is measured twice, we will average consecutive spectra (belonging to the same subject):

```
num_subjects <- ncol(intensity_with_replicates)/2
intensity_avg <- matrix(0, nrow = nrow(intensity_with_replicates), ncol = num_subjects)
for (i in seq(1, num_subjects)) {
intensity_avg[, i] <- rowMeans(intensity_with_replicates[, c(2*i - 1, 2*i)])
}
```

medical_cond has 654 class values, one for each spectrum. We take one every two types to have 327 values, one for each subject in our intensity_avg matrix:

```
subject_type <- medical_cond[seq(from = 1, to = 654, by = 2)]
```

Log transformation

Log transformation transforms the intensities to their log values. The measured intensities span a wide dynamic range of values. The same peak in some spectrum can be much larger than in other spectra. The distribution of intensities in the spectra is non-gaussian, and by using a log transform we can make it more like a gaussian distribution.

Having gaussian-like data is beneficial for PCA, as PCA is based on the covariance matrix.

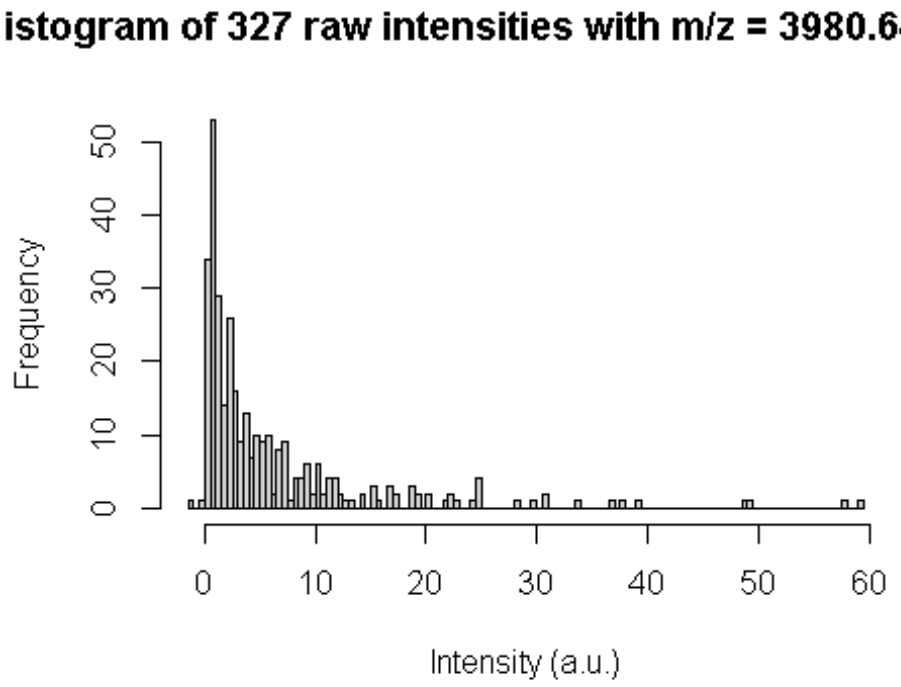
Therefore, if we create our models using the logarithm of the intensities instead of the intensities, we will be able to capture better the information of our largest peaks in the mass spectra, as their histogram will resemble more a gaussian

```
# First we transform the intensity values to a log scale. To do that, we #
create a copy of our data and then transform it:
intensity_log <- intensity_avg
```

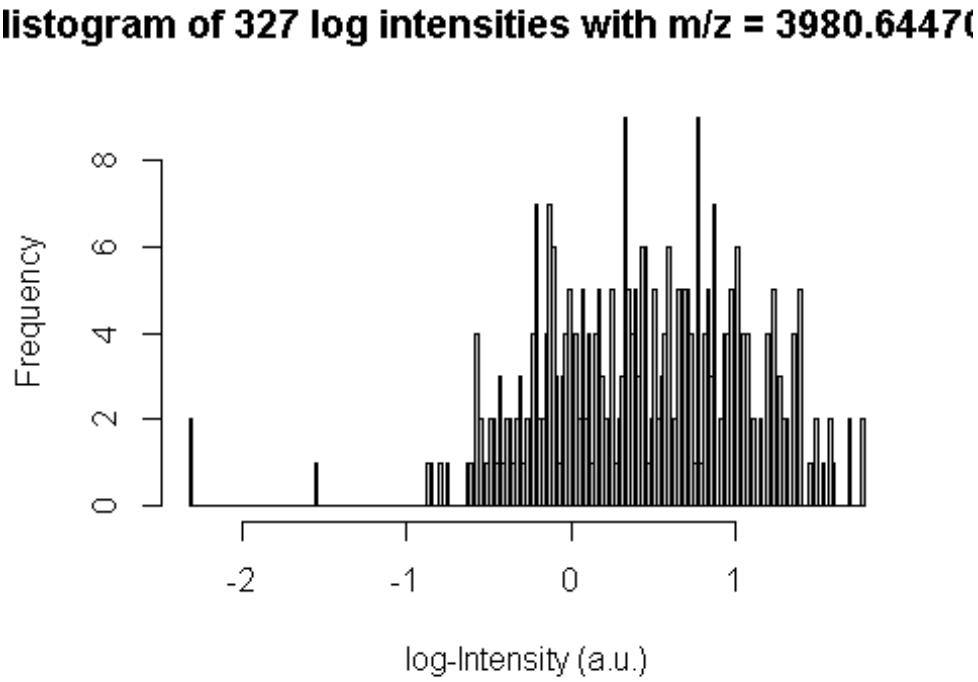
```
# values close to zero would go to -infinity. We want to avoid that. A simple
# solution is to use a threshold:
intensity_log[intensity_log < 5e-3] <- 5e-3
# log transformation:
intensity_log <- log10(intensity_log)
```

If you look at a given m/z variable and check the intensity values you can see that our variables were not normally distributed: There are few values with large intensities, the distribution is not symmetric:

```
hist(intensity_avg[2000,], breaks = 200, xlab = "Intensity (a.u.)",
main = sprintf("Histogram of 327 raw intensities with m/z = %f Da",
mz_prost[2000]))
```




```
hist(intensity_log[2000,], breaks = 200, xlab = "log-Intensity (a.u.)",
main = sprint ("Histogram of 327 log intensities with m/z = %f Da",
mz_prost[2000]))
```



The classification algorithms assume that each sample is given in a row, therefore we need to transpose the intensity matrix:

```
intensity <- t(intensity_log)
```

Let's just check the dimensionality to confirm:

```
message("Number of samples: ", nrow(intensity))

message("Number of variables: ", ncol(intensity))
```

The balance of sample types in the dataset is important to many algorithms: If we had very few samples of a particular class (for instance very few benign prostatic hyperplasia subjects), we would have to consider either (i) looking for more samples of that class, (ii) drop all the hyperplasia samples and simplify the experiment or (iii) use algorithms able to work with unbalanced datasets.

```
table(subject_type)
```

Is the dataset balanced? What is the percentage of samples of each class?

As you may see the dimensionality of the raw data is pretty high. The usual procedure to reduce this type of data consist of finding common peaks and integrating their area (aside from smoothing, binning, peak alignment, normalization and other signal processing steps to enhance signal quality). However, here to simplify we will follow a brute force strategy (not optimal but easy): We will consider every single point in the spectra as a distinctive

feature. This brute force strategy is sometimes used in bioinformatics, but generally does not provide the best results.

Train/Test division

To estimate how our trained model will perform, we need to split the dataset into a training subset and a test subset. The train subset will be used to train the model and the test subset will be used to estimate the performance of the model. We will use 60% of the samples for training and 40% of samples for test, having the training and test subsets balanced for each subject condition.

```
pca_idx <- which(subject_type == "pca")
pca_idx_train <- sample(pca_idx, round(0.6*length(pca_idx)))
pca_idx_test <- setdiff(pca_idx, pca_idx_train)

bph_idx <- which(subject_type == "bph")
bph_idx_train <- sample(bph_idx, round(0.6*length(bph_idx)))
bph_idx_test <- setdiff(bph_idx, bph_idx_train)

ctrl_idx <- which(subject_type == "control")
ctrl_idx_train <- sample(ctrl_idx, round(0.6*length(ctrl_idx)))
ctrl_idx_test <- setdiff(ctrl_idx, ctrl_idx_train)

train_idx <- c(pca_idx_train, bph_idx_train, ctrl_idx_train)
test_idx <- c(pca_idx_test, bph_idx_test, ctrl_idx_test)

# use the indexes to split the matrix into train and test
intensity_trn <- intensity[train_idx,]
intensity_tst <- intensity[test_idx,]

# use the indexes to split the labels
subject_type_trn <- subject_type[train_idx]
subject_type_tst <-subject_type[test_idx]
message("Number of samples in training: ", nrow(intensity_trn))
message("Number of samples in test: ", nrow(intensity_tst))
```

knn in full input space

Let us first implement a knn classifier in the raw input space.

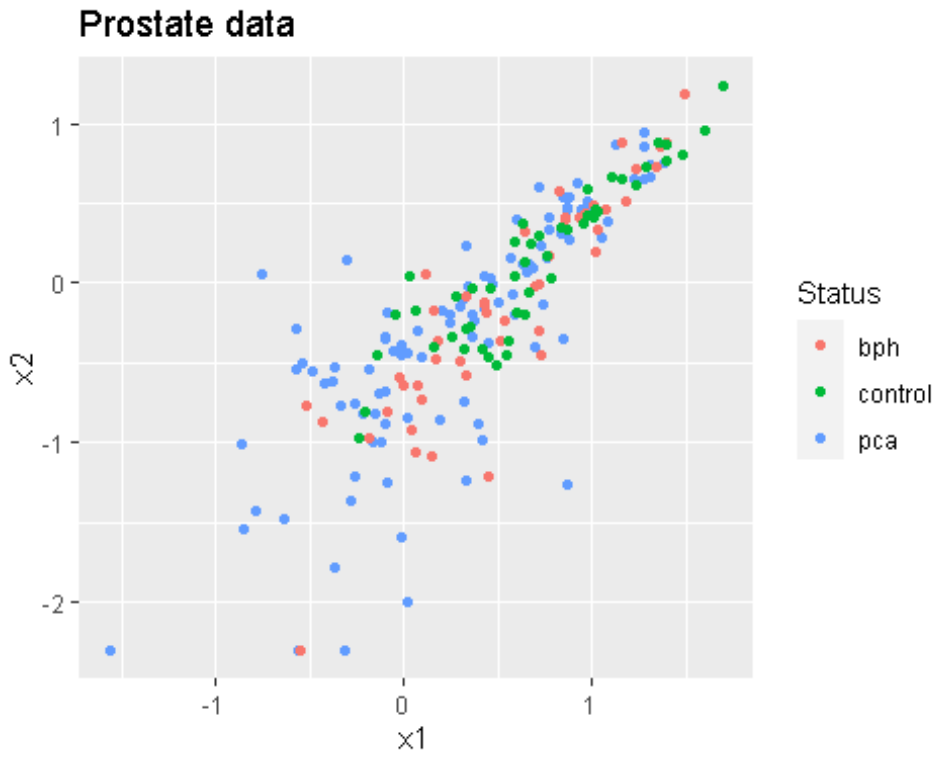
```
subject_type_tst_knn_pred <- class::knn(train = intensity_trn,
                                        test = intensity_tst,
                                        cl = subject_type_trn, k = 5)
confmat_knn <- table(subject_type_tst, subject_type_tst_knn_pred)
print(confmat_knn)
```

```
CR_knn <- sum(diag(confmat_knn))/sum(confmat_knn)
message("The classification rate for kNN is: ", 100*round(CR_knn, 2), "%")
```

Nearest Centroid Classifier

In this section we are going to use a Nearest Centroid Classifier using the 'lolR' package. First let us produce a scatter plot of two random components.

```
X <- intensity_trn
Y <- subject_type_trn

datalab1<-data.frame(x1=X[,2000],x2=X[,2010],y=Y)
datalab1$y<-factor(datalab1$y)

ggplot(datalab1, aes(x = x1, y = x2, color = y)) +
  geom_point() +
  labs(color = "Status") +
  xlab("x1") +
  ylab("x2") +
  ggtitle("Prostate data")
```

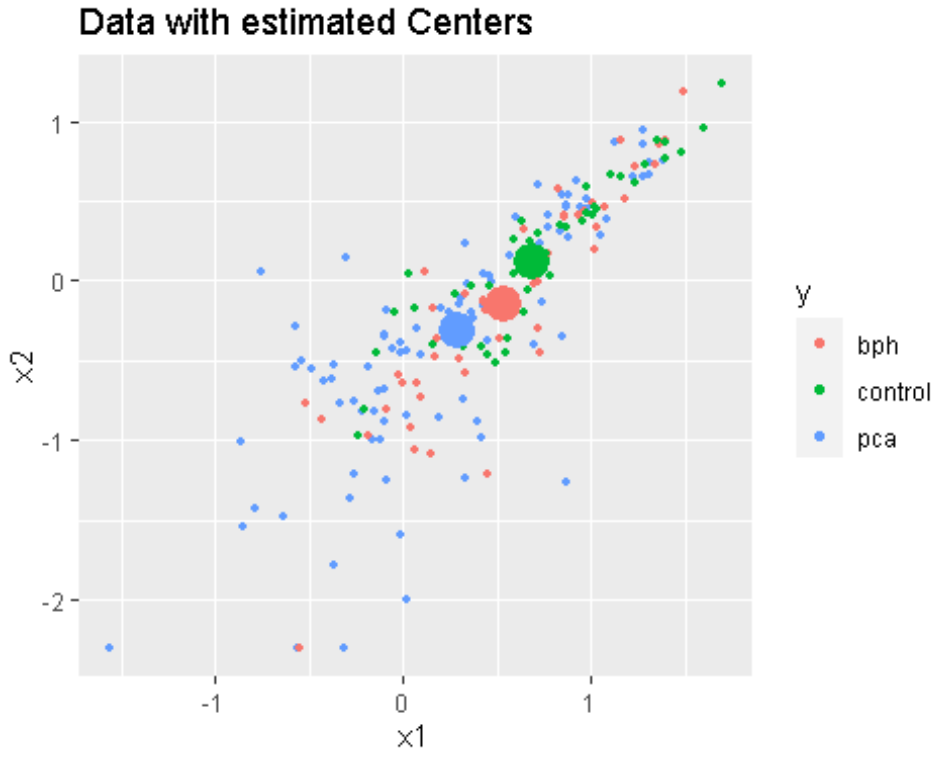


Now we estimate the class centers with the Nearest Centroid Classifier

```
classifier <- lol.classify.nearestCentroid(X,Y)
```

Now let's plot the centroids on the same scatter plot. Please take into account that this is only a partial representation of the data, since the real dimensionality is much higher.

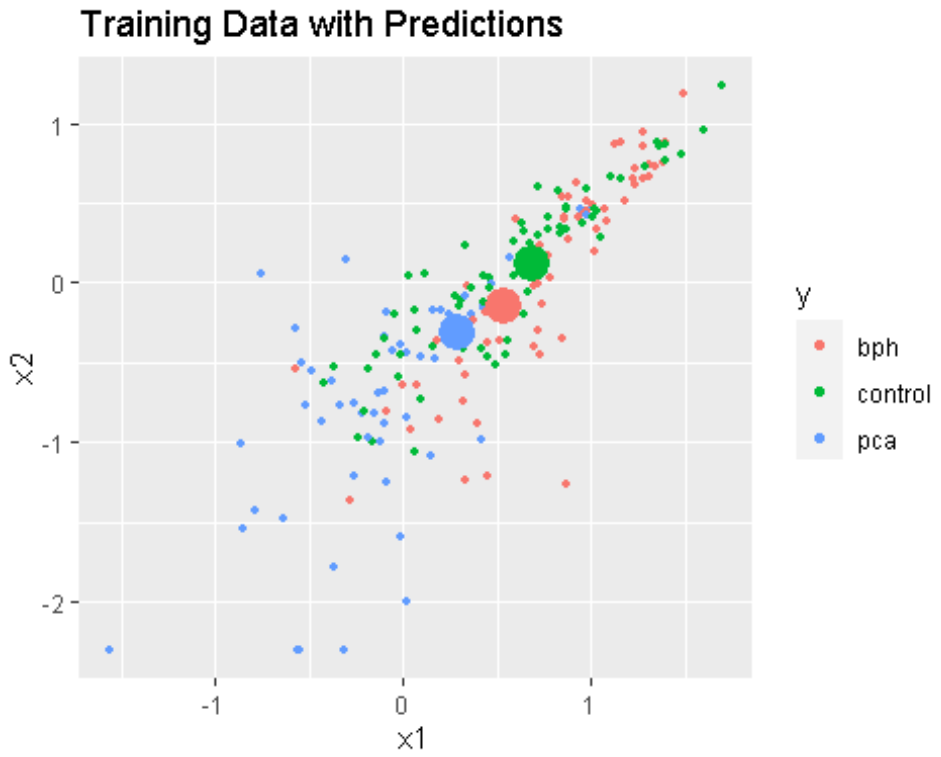
```
datalab11 <- cbind(datalab1,data.frame(size = 1))
datalab111 <- rbind(datalab11,data.frame(x1 = classifier$centroids[,2000],
                                         x2 = classifier$centroids[,2010],
                                         y = classifier$ylabs,
                                         size = 5))

ggplot(datalab111, aes(x=x1, y=x2, color=y, size=size)) +
  geom_point() +
  xlab("x1") +
  ylab("x2") +
  ggtitle("Data with estimated Centers") +
  guides(size = FALSE)
```



Let us now predict training data with the nearest centroid classifier.

```
Yhat <- predict(classifier,X)
datalab111$y[1:(length(datalab111$y)-3)] <- Yhat

ggplot(datalab111,aes(x=x1,y=x2, color=y,size=size))+
  geom_point()+
  xlab("x1")+
  ylab("x2")+
```

```
  ggtitle("Training Data with Predictions")+
  guides(size=FALSE)
```



Finally we can assess the performance of the classifier on the test set.

```
subject_type_tst_NC_pred<-predict(classifier,intensity_tst)
confmat_NC <- table(subject_type_tst, subject_type_tst_NC_pred)
print(confmat_NC)

CR_NC <- sum(diag(confmat_NC))/sum(confmat_NC)
message("The classification rate for NC is: ", 100 * round(CR_NC, 2), "%")
```

Take into account that the classifier operates in the full space, not just in the projection that we have plotted.

Feature Extraction by Principal Component Analysis

In most occasions, classifiers are not built in the raw data due to the existence of many dimensions without discriminant information.

In this section we will explore Dimensionality Reduction by Principal Component Analysis (PCA). PCA is an unsupervised technique that returns directions (eigenvectors) that explain the maximum data variance while being orthogonal among them. While PCA is optimal in compressing the data into a lower dimensionality, we have to be aware that maximum

variance does not imply maximum discriminability. In other words, PCA may not be the optimal procedure to reduce the dimensionality. It is however, the default technique for initial data exploration and pattern recognition design.

$$D = S * L^T + E$$

PCA will find a new basis for our data in which each of the directions maximize the explained variance of our data. The change of basis matrix L is called the PCA loadings, while the projection of our data in this new basis are the scores.

If we truncate the PCA decomposition to a number of principal components npc, then the product S∗LT will not be exactly as D, and the difference will be the matrix of residuals E. We will use the PCA function from the ChemometricsWithR package.

PCA needs to be applied to a matrix where each feature has zero mean.

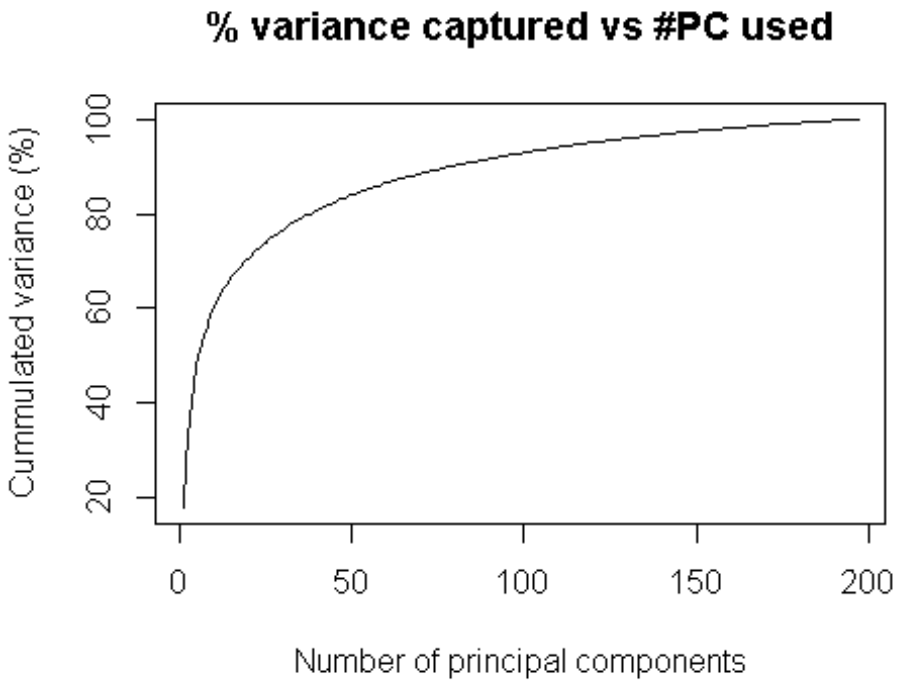
```
intensity_trn_preproc <- scale(intensity_trn, center = TRUE,
                               scale = FALSE)

mean_spectrum_trn <- attr(intensity_trn_preproc, 'scaled:center')
```

The test data is centered using the mean and standard deviation computed from the train data:

```
intensity_tst_preproc <- scale(intensity_tst,
                               center = mean_spectrum_trn,
                               scale = FALSE)
```

We can now perform the dimensionality reduction and observe how the variance is distributed among the first principal components:

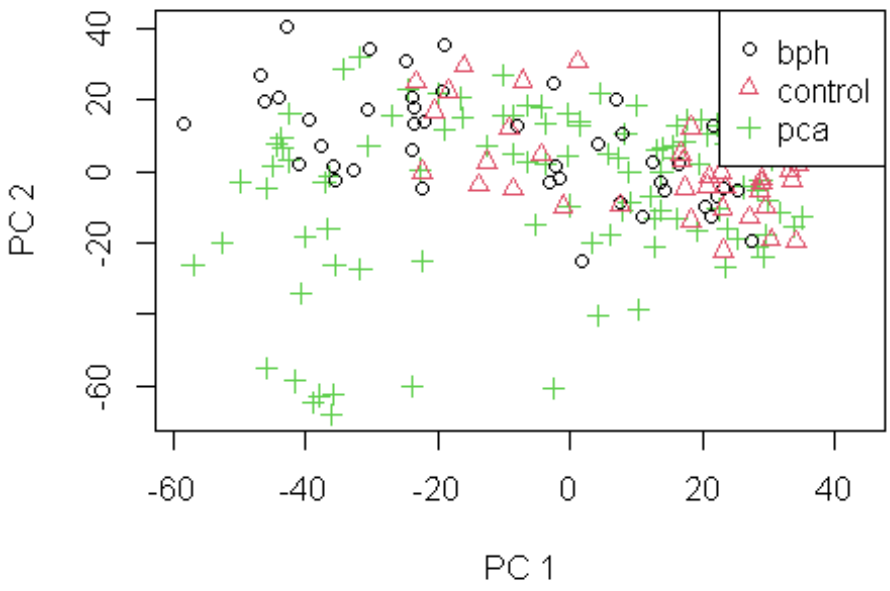
```
pca_model <- PCA(intensity_trn_preproc)
summary(pca_model)

pca_model_var <- variances(pca_model)
pca_model_var_percent <- 100 * cumsum(pca_model_var)/sum(pca_model_var)
plot(x = 1:length(pca_model_var_percent),
y = pca_model_var_percent, type = "l",
xlab = "Number of principal components",
ylab = "Cummulated variance (%)",
main = "% variance captured vs #PC used")
```



Based on the knee of the plot, we choose the number of components of the PCA space.

We can plot the projection of the date in the plane of maximum variance. **Do you think that the classes have a Gaussian distribution?**

```
scoreplot(pca_model,
          pch = as.integer(subject_type_trn),
          col = as.integer(subject_type_trn))
legend("topright",
legend = levels(subject_type_trn),
pch = 1:nlevels(subject_type_trn),
col = 1:nlevels(subject_type_trn))
```

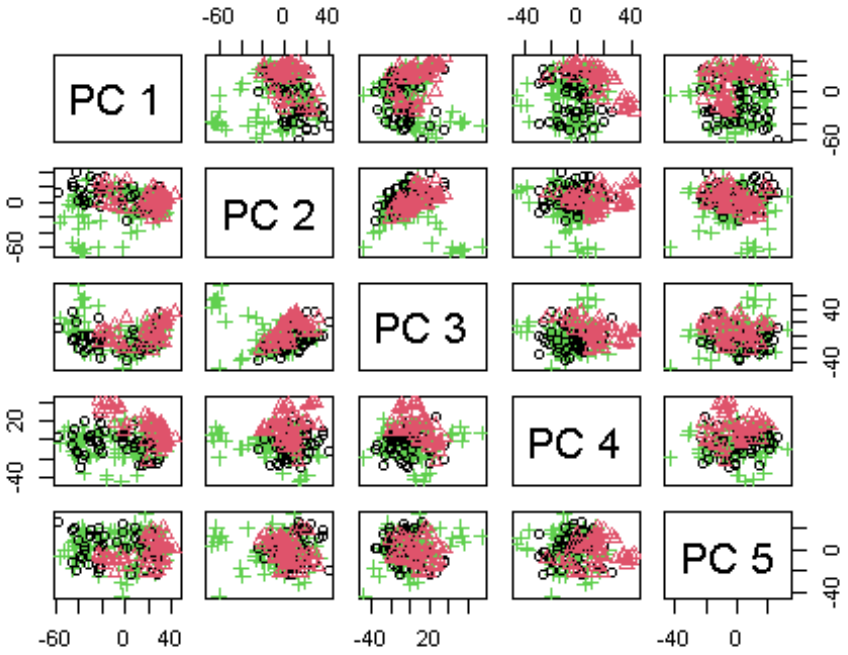


We can retrieve the new feature vectors after dimensionality reduction. In other words, these are the projections of the feature vector in the space spanned by the first eigenvectors of the covariance matrix. Those eigenvectors are now the new space basis. For classifier design this value needs to be optimized. It is always advisable to have many more samples than dimensions. Additionally we project the test data into the model computed with the train data.

```
pca_model_ncomp <- 50
intensity_scores_trn <- project(pca_model, npc = pca_model_ncomp,
intensity_trn_preproc)
intensity_scores_test <- project(pca_model, npc = pca_model_ncomp,
intensity_tst_preproc)
```

We can visualize how the training data is distributed along the different principal components (beyond the first two components):

```
pairs(intensity_scores_trn[,1:5],
labels = paste("PC", 1:5),
pch = as.integer(subject_type_trn),
col = as.integer(subject_type_trn))
```

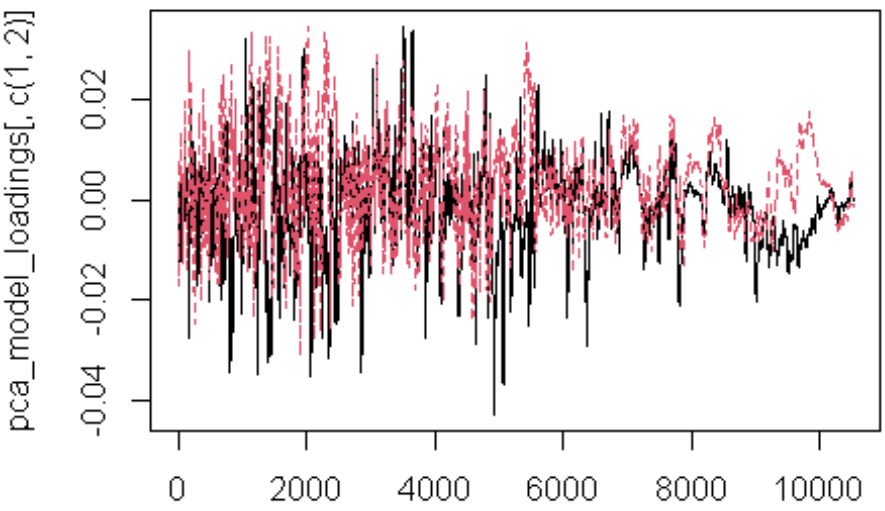


Is any of these first 5 principal components able to discriminate sample types?

We can also visualize the loadings of the PCA model. These loadings select regions in the mass spectra that co-vary and explain simultaneously most of the variance. Let's calculate the first two eigenvectors and let's plot them.

By looking at the first loading, we will see the variables that were used to compute the scores of the first principal component.

```
pca_model_loadings <- loadings(pca_model, pca_model_ncomp)

matplot(pca_model_loadings[,c(1,2)], col = c(1,2), type = 'l')
```



If there is no variable clearer than the rest, it means that the first (and second loadings) are distributed among many variables and the interpretation may be harder than what we can cover in this session.

Nearest Neighbour Classifier on the PCA space

A nearest neighbour classifier is a non-parametric method used for pattern recognition and classification. The method classifies new samples based on the classes of the k closest training samples.

The k-NN classifier is implemented in the class package.

The kNN will predict the classes of the test set, given the training samples, their classes and the number of neighbours to look.

We will apply the k-NN classifier on the PCA space.

```
library("class")
```

We can calculate the confusion matrix. Take some time to understand this table. **What groups of samples is the kNN misclassifying?**

```
subject_type_tst_pca_knn_pred <- class::knn(train = intensity_scores_trn,
                                            test = intensity_scores_test,
                                            cl = subject_type_trn,
                                            k = 5)
```

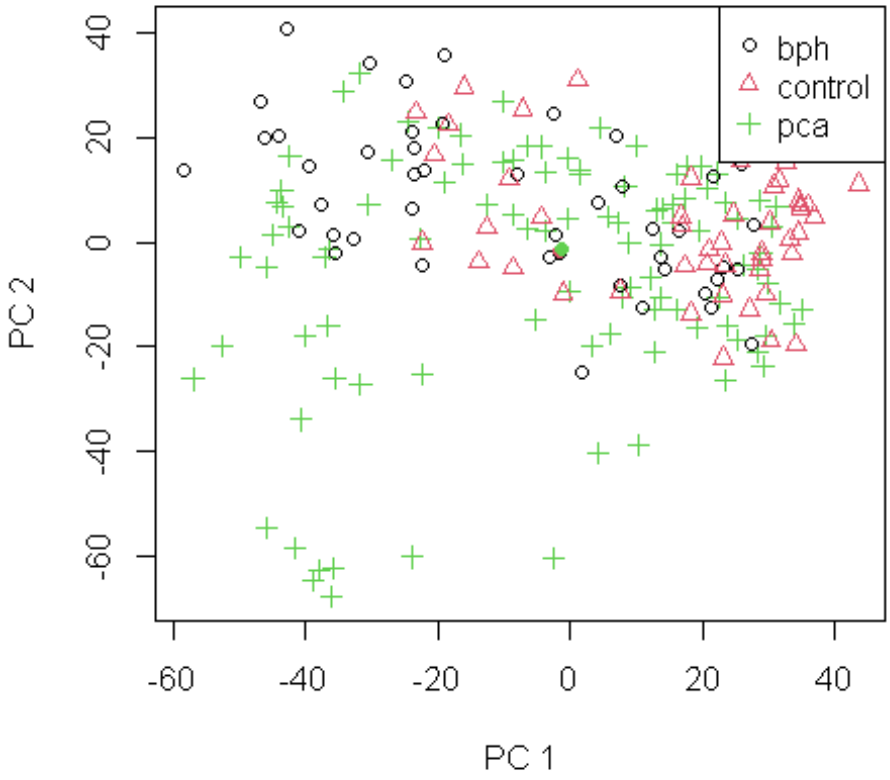
```r
confmat_pca_knn <- table(subject_type_tst, subject_type_tst_pca_knn_pred)
print(confmat_pca_knn)
```

```
##                 subject_type_tst_pca_knn_pred
## subject_type_tst bph control pca
##          bph      21       1   9
##          control   2      29   1
##          pca      15       4  48
```

```r
CR_pca_knn <- sum(diag(confmat_pca_knn))/sum(confmat_pca_knn)
message("The classification rate for PCA-kNN is: ", 100*round(CR_pca_knn, 2),
"%")
```

```
## The classification rate for PCA-kNN is: 75%
```

```r
# let us project the centroids in the scoreplot

PCA_centroids <- classifier[["centroids"]]

scoreplot(pca_model, pch = as.integer(subject_type_trn),
          col = as.integer(subject_type_trn))
legend("topright",
       legend = levels(subject_type_trn),
       pch = 1:nlevels(subject_type_trn),
       col = 1:nlevels(subject_type_trn))

points(PCA_centroids[,1],
       PCA_centroids[,2],
       pch = 20,
       col = as.factor(classifier[["ylabs"]]),
       cex = 1.5)
```



Parameter optimization

In the previous scenario we have performed the classification using the PCA projection, with 50 principal components. However we have not checked how the performance of the model (in our case we have seen the classification rate) is affected by changing the number of principal components.

```
# We will try using 1 to 100 principal components:
max_pca_ncomp <- 100
```

We can not simply change the number of principal components above and check the classification rate. If we do that, we will be using the test subset to select the best parameters of the model, so we will not have an actual blind test subset of samples.

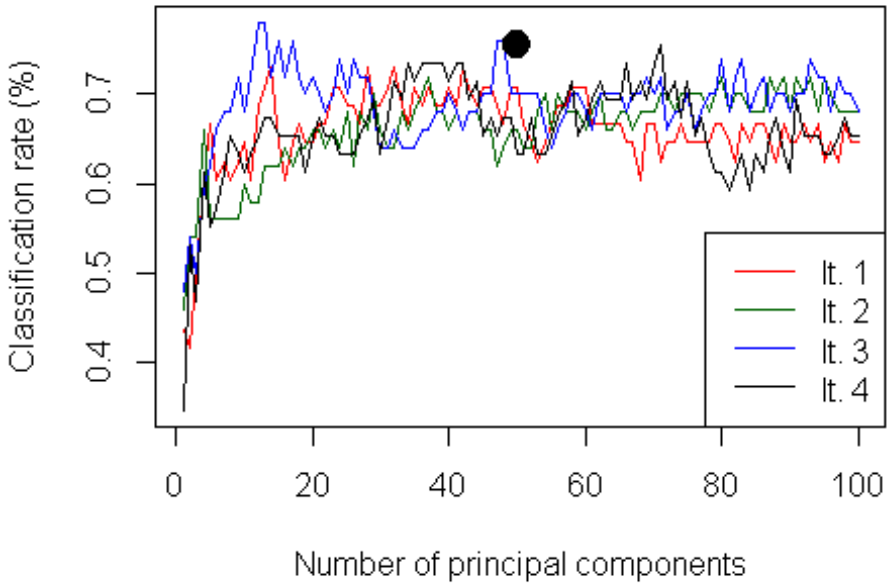
As explained in class, the right approach is to split the train subset into two groups, one used for training the models with different number of principal components, and the other used to test the models and choose the one with best classification rate. Then the best model can be used on the blind test samples to have an estimation of the best model performance.

With so many partitions, the number of samples available for training the models starts to be too small. To ensure that our results are robust to the random variability of how we split the train subset, we perform the computation several times, in several iterations, using a cross validation method.

We will use a k-fold validation and split the train subset into 4 groups. For each fold, for each number of principal components we will train the PCA-knn models and compute its classification rate. We will obtain 4 values (one for each fold) of the classification rate for each number of principal components.

```
# To perform the cross-validation we only use the TRAIN subset, that we #
divide into cv_train and cv_test subsets.
# Partition the train subset into 4 groups:
kfolds <- 4
pca_idx_cv <- caret::createFolds(y = pca_idx_train, k = kfolds, returnTrain =
TRUE)
bph_idx_cv <- caret::createFolds(y = bph_idx_train, k = kfolds, returnTrain =
TRUE)
ctrl_idx_cv <- caret::createFolds(y = ctrl_idx_train, k = kfolds, returnTrain
= TRUE)
classification_rates <- matrix(0, nrow = kfolds, ncol = max_pca_ncomp)

for (iter in 1:kfolds){
  cv_train_idx <- c(pca_idx_train[pca_idx_cv[[iter]]],
bph_idx_train[bph_idx_cv[[iter]]], ctrl_idx_train[ctrl_idx_cv[[iter]]])
cv_test_idx <- setdiff(train_idx, cv_train_idx)

# Get the cv_train and cv_test matrices, with their labels:
intensity_cv_trn <- intensity[cv_train_idx,]
intensity_cv_tst <- intensity[cv_test_idx,]
subject_type_cv_trn <- subject_type[cv_train_idx]
subject_type_cv_tst <- subject_type[cv_test_idx]

# Preprocessing, like above:
intensity_cv_trn_preproc <- scale(intensity_cv_trn, center = TRUE, scale =
FALSE)
mean_spectrum_cv_trn <- attr(intensity_cv_trn_preproc, 'scaled:center')
intensity_cv_tst_preproc <- scale(intensity_cv_tst, center =
mean_spectrum_cv_trn, scale = FALSE)

# Train PCA
pca_cv_model <- ChemometricsWithR::PCA(intensity_cv_trn_preproc)
for (pca_model_ncomp_cv in 1:100) {

# Get the PCA scores for the cv_train and cv_test subsets:
intensity_scores_cv_trn <- ChemometricsWithR::scores(pca_cv_model, npc =
pca_model_ncomp_cv)
intensity_scores_cv_test <- ChemometricsWithR::project(pca_cv_model, npc =
pca_model_ncomp_cv, intensity_cv_tst_preproc)
```

```r
# Train the knn with the PCA scores, using only the cv_train subset
## FALLA
   subject_type_tst_pca_knn_pred <- class::knn(train =
intensity_scores_cv_trn,
                                               test =
intensity_scores_cv_test,
                                               cl = subject_type_cv_trn, k =
5)

    confmat_pca_knn <- table(subject_type_cv_tst,
subject_type_tst_pca_knn_pred)

    CR_pca_knn_cv <- sum(diag(confmat_pca_knn))/sum(confmat_pca_knn)
    message("The classification rate for PCA-kNN is: ",
100*round(CR_pca_knn_cv, 2), "%")

# Store the classification rate for this k-fold iteration and this number of
principal components:
    classification_rates[iter, pca_model_ncomp_cv] <- CR_pca_knn_cv
    }
}

# Plot the classification rates obtained on each k-fold iteration for all #
the principal components tested:
matplot(x = 1:100, y = t(classification_rates), type = "l",
col = c("red", "darkgreen", "blue", "black"), lty = "solid",
xlab = "Number of principal components", ylab = "Classification rate (%)")
legend("bottomright", legend = c("It. 1", "It. 2", "It. 3", "It. 4"),
       col = c("red", "darkgreen", "blue", "black"), lty = "solid")

# From the plot above we can choose the number of principal components that
we prefer
# and build the final model again. (50 was a fine value)
# Our estimation of the model performance choosing 50 principal components
can # be represented as a single point in the plot:
points(x = pca_model_ncomp, y = CR_pca_knn, col = "black", cex = 2, pch = 21,
bg = "black")
```



Further questions

What would have happened if the researchers had not considered to include benign prostatic hypertrophy subjects in the experimental design?

What would be our prediction if we built a model without benign prostatic hypertrophy samples and later on we tried to predict a patient who suffered from that condition?

How do you compare the prediction in internal validation and in external validation. Surprisingly it seems results are better in external validation than in internal validation. Could you reason why this can happen?

In this case, dimensionality reduction does not improve the performance of the classifier. Do you think this behavior could repeat in classifiers of higher complexity?