Assignment 5

Thijs van der Knaap (s2752077)
Hatim Alsayahani (s3183696)
Sebastian Wehkamp (s2589907)
Dimitris Laskaratos (s3463702)

Group 15

October 9, 2017

Contents

1 Example Questions

1.1

The K Nearest Neighbour algorithm first requires the input of the training data: $\{x_i, y_i\}_{i=1..n}$, the test data: $\{\hat{x}_j\}_{j=1..m}$ and the k parameter. This corresponds to line number **4** (where $\xi_j = \hat{x}_j$).

Next, it needs a loop to iterate through $\hat{x}_j$ items, that would be line number **8**.

For each $\hat{x}_j$, we need to compute the Euclidean distance of ξ_j from $x_i, \forall i = 1..n$. That corresponds to line **1**.

Next, from the set of Euclidean distances from the previous step, we need to determine the subset of k-nearest neighbours of ξ_j from the superset $N(\xi_j)$. That is done with line **3**.

Lastly, we assign label $c(\xi_j)$ to set of nearest neighbours x_K in line number **11**

The loop ends at line **6**.

So we have:

Line	Command
4	**Input** $\{x_i, y_i\}_{i=1..n}, \{\hat{x}_j\}_{j=1..m}, k$
8	**for** $j = 1...m$
1	compute $d(\xi_j, x_i)$ $\forall x_i$ with $i = 1..n$
3	find $\{x_i, y_i\}_{i=1}^{k} \epsilon N(\xi_j)$
11	assign $c(\xi_j)$ to closest labelled point to ξ_j
6	**end**

1.2

1.2.1 a

The Entropy is calculated as $Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t) = -\frac{4}{9} log_2 \frac{4}{9} - \frac{5}{9} log_2 \frac{5}{9} = 0,99107606$

1.2.2 b

Using the entropy, I_E, from the previous question, the information gain I_G for a_1 and a_2 is calculated as: $I_G = I_E - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$, where: N: total number of records at parent
$I(v_j)$: impurity of child
$N(v_j)$: number of records at child
k: number of attributes

Consequently, we have: $I_G = 0,99107606 - \frac{4}{9}\frac{4}{9} - \frac{5}{9}\frac{5}{9} = 0,99107606 - 0,50617283 = 0,48490322$

1.2.3 c

To calculate the gain we need the function

$$gain = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

For the impurity we use the Error(t) measure.

$$Error(t) = 1 - max_i p(i|t)$$

split	left	right	gain
1.5	{1}	{2,3,4,5,6,7,8,9}	0.111
3.5	{1,6}	{2,3,4,5,7,8,9}	0
4.5	{1,4,6}	{2,3,5,7,8,9}	0.111
5.5	{1,3,4,6,9}	{2,5,7,8}	0
6.5	{1,2,3,4,6,9}	{5,7,8}	0
7.5	{1,2,3,4,5,6,8,9}	{7}	0

As can be seen in the table the gain is zero for a large amount of splits and only very small for two splits. This implies that splitting on this attribute will give you little result if you want to create a decision tree that classifies + and - elements.

1.2.4 d

The classification error rate for the data without partitioning any attribute is calculated as: $Error(t) = 1 - max_i p(i|t) = 1 - max(\frac{4}{9}, \frac{5}{9}) = 0.444$.

Next, we calculate the contingency table for each of the attributes a_1 and a_2.

Table 1: Attribute a_1

	$a_1 = T$	$a_1 = F$
+	3	1
-	1	4

Table 2: Attribute a_2

	$a_2 = T$	$a_2 = F$
+	2	2
-	3	2

The gain in error rate for attribute a_1 is:

$$E_{a_1=T} = 1 - max(\frac{3}{4}, \frac{1}{4}) = \frac{1}{4} = 0,25$$
$$E_{a_1=F} = 1 - max(\frac{1}{5}, \frac{4}{5}) = \frac{1}{5} = 0,20$$
$$G = Error(t) - \frac{4}{9}E_{a_1=T} - \frac{5}{9}E_{a_1=F} = 0,321$$

The gain in error rate for attribute a_2 is:

$$E_{a_2=T} = 1 - max(\frac{2}{5}, \frac{3}{5}) = \frac{2}{5} = 0,40$$
$$E_{a_2=F} = 1 - max(\frac{2}{4}, \frac{2}{4}) = \frac{2}{4} = 0,50$$
$$G = Error(t) - \frac{5}{9}E_{a_2=T} - \frac{4}{9}E_{a_2=F} = 0$$

Consequently, the best split between a_1 and a_2 is a_1 since it has the highest gain of the two.

1.2.5 e

As above, we calculate individual gains:

The impurity of the parent node using Gini index is: $Gini(t) = 1 - \sum_{i=0}^{c-1} p(i|t)^2 = 1 - (\frac{4}{9})^2 - (\frac{5}{9})^2 = 0,493$.

From the above tables we get the following gains for each attribute:

The gain for attribute a_1 is:

$$G_{a_1=T} = 1 - \sum_{i=0}^{c-1} p(i|t)^2 = 1 - (\frac{3}{4})^2 - (\frac{1}{4})^2 = 0,375$$
$$G_{a_1=F} = 1 - \sum_{i=0}^{c-1} p(i|t)^2 = 1 - (\frac{1}{5})^2 - (\frac{4}{5})^2 = 0,32$$
$$G_{a_1} = Gini(t) - \frac{4}{9}G_{a_1=T} - \frac{5}{9}G_{a_1=F} = 0,148$$

The gain for attribute a_1 is:

$$G_{a_2=T} = 1 - \sum_{i=0}^{c-1} p(i|t)^2 = 1 - (\frac{2}{5})^2 - (\frac{3}{5})^2 = 0,48$$
$$G_{a_2=F} = 1 - \sum_{i=0}^{c-1} p(i|t)^2 = 1 - (\frac{2}{4})^2 - (\frac{2}{4})^2 = 0$$

$$G_{a_1} = Gini(t) - \frac{5}{9}G_{a_1=T} - \frac{4}{9}G_{a_1=F} = 0,226$$

The gain is maximised for attribute a_2 which is chosen as the best split in this case.

2 Classification in practice

The first step is to merge the data from both datasets into a single dataset. This can be done using the code from Listing 1. Throughout the assignment we have 3 datasets; labels containing the separate labels, features containing all of the feature data, and merged containing a merge of both files.

```python
import pandas as pd

#Read the provided CSV file lastFM.tsv
labels = pd.read_csv('labelsFlowCapAnalysis2017.csv')

features = pd.read_csv('featuresFlowCapAnalysis2017.csv', header=0)

#Merge them
merged = labels.join(features, how='outer')

#Export to CSV
del merged.index.name
merged.to_csv('merged.csv')
```

Listing 1: Merge two datasets

2.1

2.1.1 Investigate the features

First we load the data in R and run the principal components function on just the feature data. Plotting the results of the principal components function produces Figure 1. In the plotted "elbow" from Figure 1 we can see that we need about five or six principal components to explain approximately 60% of the data.

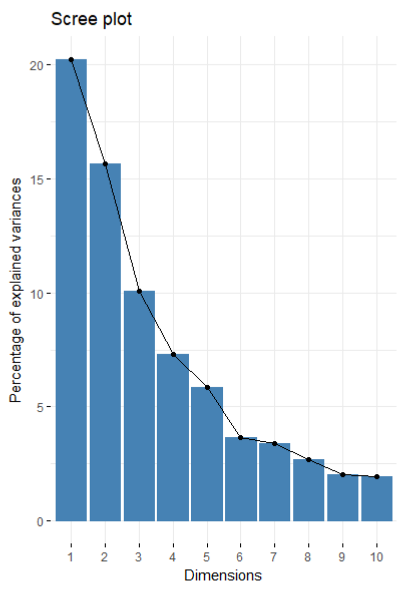


Figure 1: Principal Component Analysis on the feature data

We now want to list the features that contributed the most to the first five components. The code to do this is located in Listing 2 which produces the plot located in Figure 2. The red dashed line on the graph above indicates the expected average contribution. For a given component, a variable with a contribution larger than this cutoff could be considered as important in contributing to the component. To make the graph clearer only the first 10 are shown. The contribution of all features is shown in Figure 3 however the feature names are indistinguishable in the figure. From this image however we can conclude that the first 104 are important for the dataset since they are above the red dashed line.

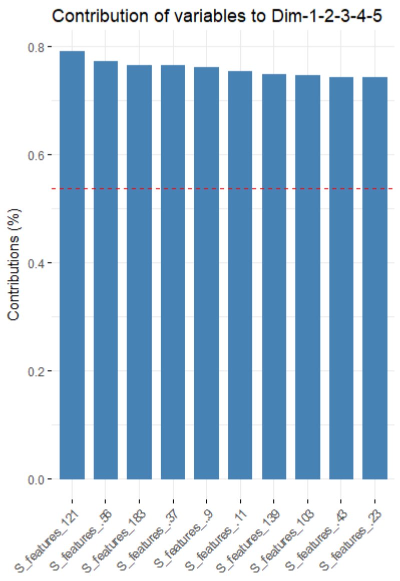


Figure 2: Top 10 features that contributed the most to the first five principal components

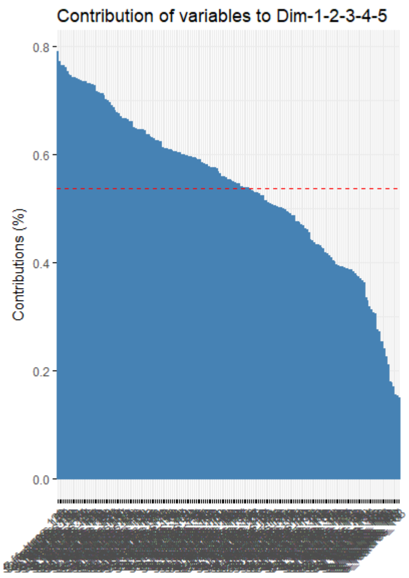


Figure 3: Contribution of all features to 5 principle components

```
library("FactoMineR")
library("factoextra")

features <- read.csv(file.choose(new = FALSE), header=TRUE,
    stringsAsFactors=TRUE)

```

```r
#Run PCA
res.pca <- PCA(features, graph = FALSE)

#Store eigen values
eigenvalues <- res.pca$eig

#Create a nice plot
fviz_screeplot(res.pca, ncp=10)

# Coordinates of variables
head(res.pca$var$contrib)

#Create a plot showing the sum of contributions of the first five
    components.
fviz_contrib(res.pca, choice = "var", axes = 1:5, top = 10)
```

Listing 2: Run principal component analysis creating 6 groups

All of the features below the red dashed line are considered unimportant since they do no contribute enough to the 5 principle components listed there.

The next step is to do an ANOVA on a number of features we previously identified as important like feature: 121, 56, 183, 37, and 9. The first feature we will do an Anova on is feature 121. The R code to this can be seen in Listing 3.

```r
#Read CSV
Merged <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Do anova
fit <- aov(S_labels ~ S_features_121, data=Merged)
```

Listing 3: Do an anova on feature 121 and the label

The results of the anova can be seen in Listing 2.1.1 in which you can see that "S_feature_121" is highly significant with respect to S_labels. This is as expected from the results of the principal component analysis.

```
                Df Sum Sq Mean Sq F value Pr(>F)
S_features_121   1  7.041   7.041   95.85 <2e-16 ***
Residuals      177 13.003   0.073
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

2.1.2 Get a holistic view on the data

To do this we created a scatterplot of the first two PCA's and a plot using TSNE. The code to create the TSNE plot is located in Listing 4 and the code to create the PCA scatter plot is located in Listing 5. Note that the function RTsne automatically uses the prcomp() function to reduce the feature space based on the dims parameter. As the assignment stated we used dims = 2 to reduce it to two dimensions.

```r
library(Rtsne)

train <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Setting fixed seed
set.seed(42)

Labels<-train$S_labels
train$S_labels<-as.factor(train$S_labels)
## for plotting
colors = rainbow(length(unique(train$S_labels)))
names(colors) = unique(train$S_labels)
```

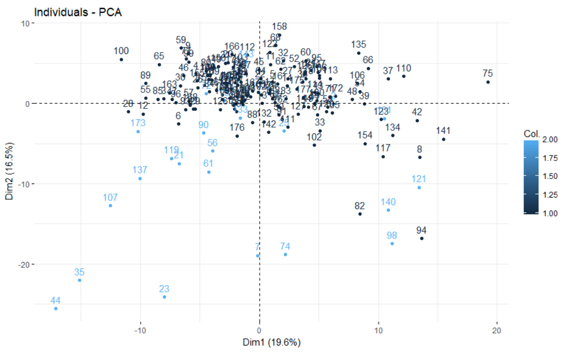
```r

tsne <- Rtsne(train[,-1], dims = 2, perplexity=30, verbose=TRUE, max_iter =
     500)

## Plotting
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=train$S_labels, col=colors[train$S_labels])
```

Listing 4: Create TSNE Plot

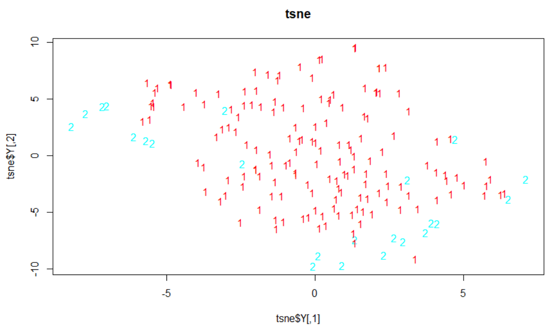
```r
library("FactoMineR")
library("factoextra")

merged <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Run PCA
res.pca <- PCA(merged, graph = FALSE)

#Plot based on labels
fviz_pca_ind(res.pca, col.ind=merged$S_labels)
```

Listing 5: Create scatter plot of the first to PC's.

The plots created with above code can be seen in Figure 4.



(a) PCA Scatterplot



(b) TSNE scatter plot

Figure 4: Scatterplots of both TSNE and PCA

In the TSNE plot you can see that the sick ones are not actually located near each other. The healthy ones form a cluster in the centre while the sick ones are located all around it. There are even two outliers located in the middle of the healthy ones. Although that is suspicious because there are two "fakes" in the dataset. So that might explain those two. From the tsne plot it would be hard to draw an ellipse which circles only the two's without surrounding a one.

In the PCA scatter plot shows that most of the healthy persons are located in a single large cluster. However there are a couple of outliers like 94 and 82 which distort the data a little. As well as with the TSNE plot in this plot there are also some sick persons located in the healthy cluster. However in the PCA showcase it would be easier to draw an elipse covering most of the sick persons while not surrounding a healthy person. This might make the classification harder.

While looking at the data and working with it we were not able to find considerable differences between the testing dataset and the training dataset.

2.1.3 General impact of preprocessing

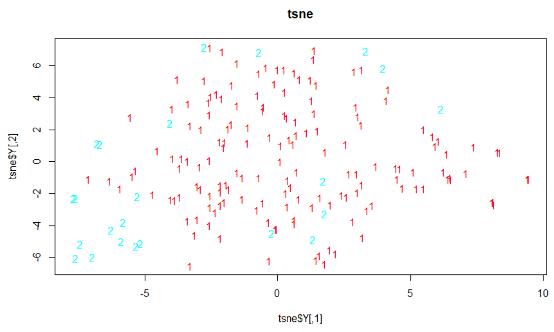
Since this section needs some experimentation with the parameters of the function calls we provided a single listing containing all of the calls used. This can be seen in Listing 6.

```r
library(Rtsne)

```

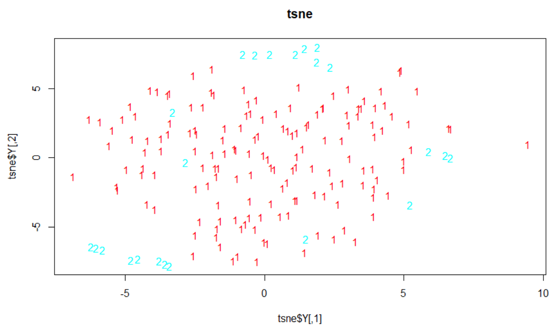
```
train <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Setting fixed seed
set.seed(42)

Labels<-train$S_labels
train$S_labels<-as.factor(train$S_labels)
## for plotting
colors = rainbow(length(unique(train$S_labels)))
names(colors) = unique(train$S_labels)

#Execute TSNE with one of the below settings

## Plotting
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=train$S_labels, col=colors[train$S_labels])

####Settings####
#No PCA
tsne <- Rtsne(train[,-1], pca=FALSE, perplexity=30, verbose=TRUE, max_iter
    = 500)

#TSNE Scaling Center settings
tsne <- Rtsne(train[,-1], dims=2, perplexity=30, verbose=TRUE, max_iter =
    500, pca_center = TRUE, pca_scale=FALSE)
tsne <- Rtsne(train[,-1], dims=2, perplexity=30, verbose=TRUE, max_iter =
    500, pca_center = FALSE, pca_scale=FALSE)
tsne <- Rtsne(train[,-1], dims=2, perplexity=30, verbose=TRUE, max_iter =
    500, pca_center = FALSE, pca_scale=TRUE)
tsne <- Rtsne(train[,-1], dims=2, perplexity=30, verbose=TRUE, max_iter =
    500, pca_center = TRUE, pca_scale=TRUE)
```

Listing 6: Experimentation with TSNE parameters

The first thing to test is if using PCA is very influential on the results. The results of not using PCA can be seen in Figure 5. Both with and without PCA most of the twos are on the out side of the cluster, using the PCA clusters them much more. This makes it easier to identify and makes clustering in a later stage easier. Thus we conclude that using PCA is useful to do as part of preprocessing in this case.



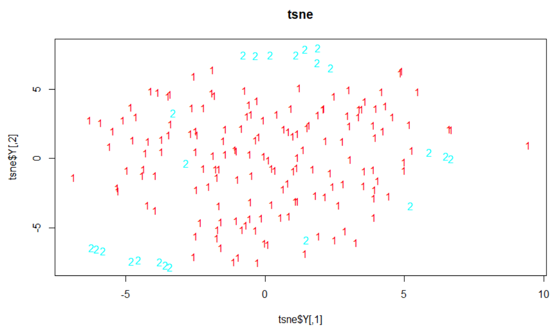
(a) Without PCA



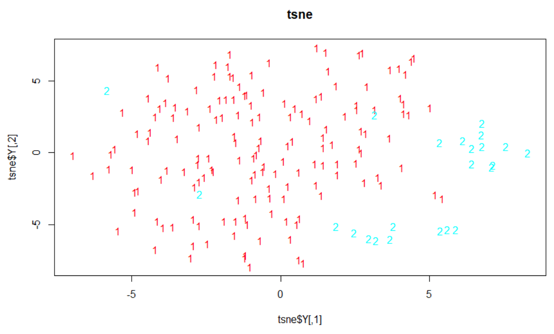
(b) With PCA

Figure 5: Influence of PCA on TSNE results

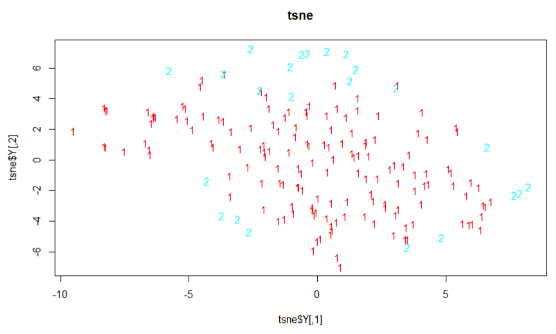
The second experiment will be to modify the parameters of the principle component analysis. There are two interesting parameters which we want to modify; pca_scale and pca_center. They are both logical values which state whether the data should be scaled and centered during the PCA. The default value of center is TRUE while scale is FALSE. Thus we will do three experiments, both False, both true, and only scale True. The results can be seen in Figure 7.



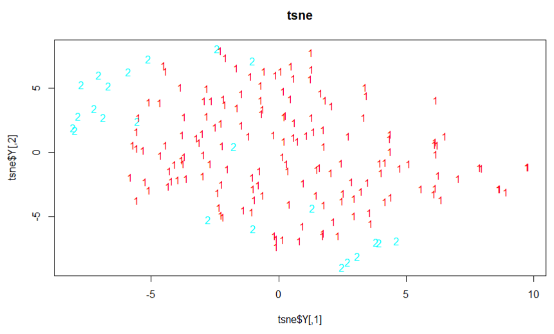
(a) Center True Scale False



(b) Center False Scale True



(a) Center False Scale False



(b) Center True Scale True

Figure 7: Results of TSNE with different PCA parameters

All settings produce rather different results but we think the best results are accomplished by using either center or scaling and not both. If both are true or false at the same time the results seem to get worse and there are more twos located in the cluster of ones. Using this specific seed (42) produces better results with Center false and Scale true but this might not be the case for other seeds.

2.2 Experimentation to find the best prediction

2.2.1 a

To start this experiment off we did a 10-fold cross validation using kNN without any preprocessing. The R code for all the kNN related classifications can be found on page 16. Although we set the seed to a static number, the results where changing a bit each time. Therefore we show 3 plots of running kNN on the basic dataset. We run kNN with k from 1 till 25. These result can be seen in figure 8

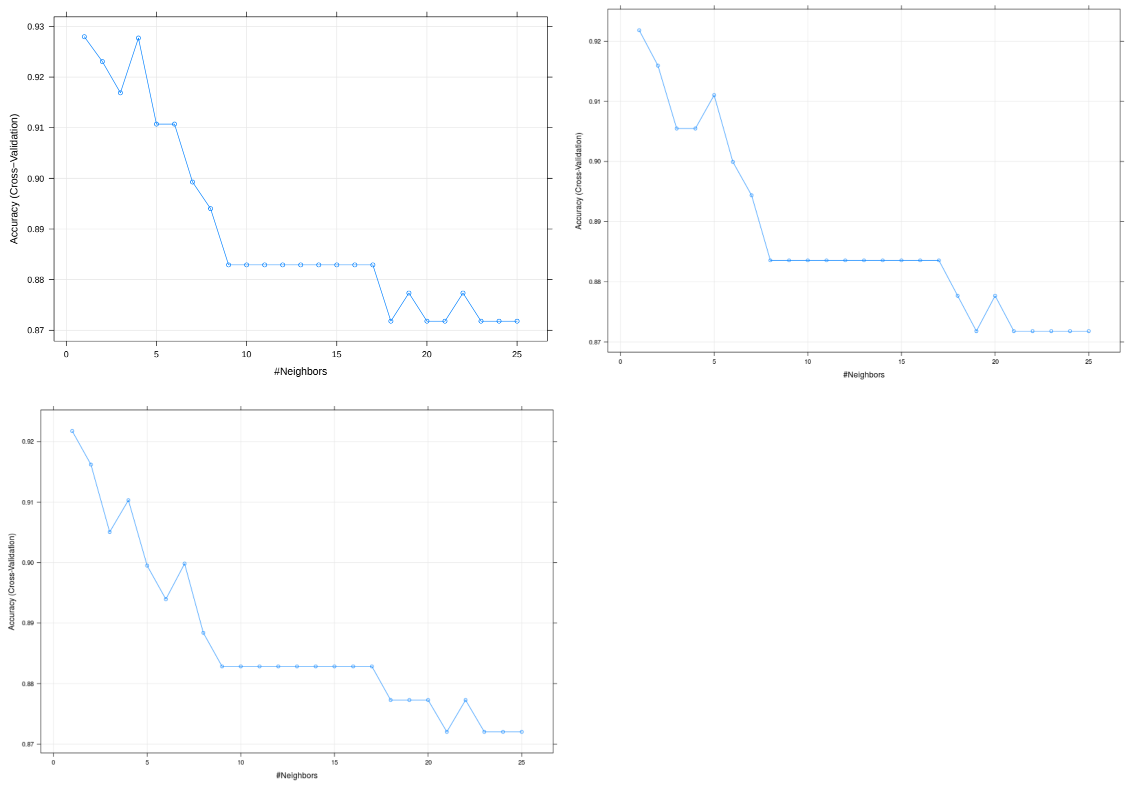


Figure 8: kNN with unfilterd set

Each time the shape is a bit different, but the general trend if very similar. The graph stabilises at around 10 neighbours which is in line with the general statement that k should be the square root of the number of variables which would be 13 in our case. Increasing the K decreases over fitting in this case.

When we only use the 5 principle components found we see very different behaviour. This can be seen in figure 9

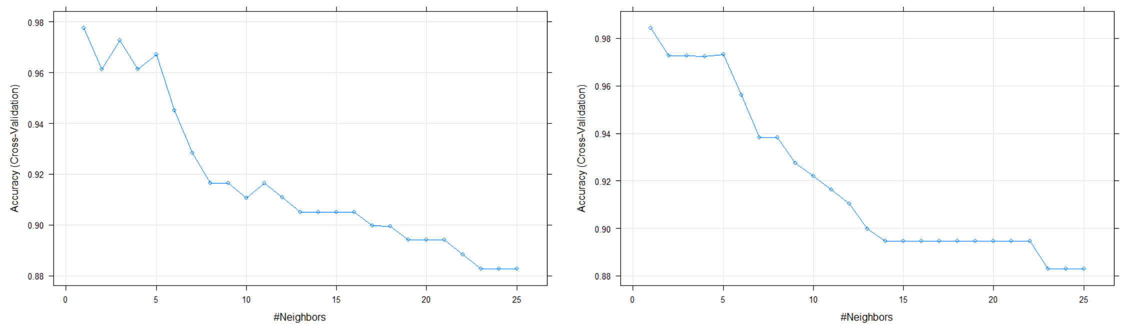


Figure 9: kNN with 5 principle components

In these images you can see that not choosing a k too high is very important. Since the type ones are in such a majority it will start identifying the type 2's wrongly. As long as we use less than 5 neighbours we are accurate in 97% of the cases. This is also in line with the statement that k should be equal to the squareroot of the variables since we reduced the number of variables to 5 principle components. The optimal number of k should be around 2 or 3.

Using Undersampling doesn't change the results compared to the normal set. This can be seen in figure 10. We undersampled with all 23 positive data rows and 23 negative data rows.

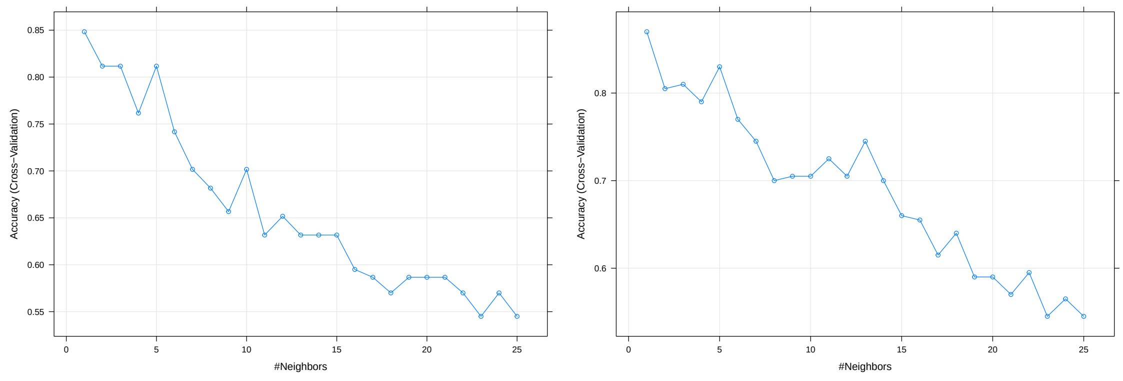


Figure 10: kNN undersampling

After a lot of experimenting, we found that kNN with 5 principle components yields the best classification result of 97%. In the following table we show the best k values and their accuracy for the best acquired k value.

Method	Accuracy
kNN 5 principle components k=4	0.97
kNN k=1	0.93
kNN undersampled k=1	0.84

In the code below is the implementation of a Decision Tree in python using third-party library *sklearn*. By running the script, the prediction for each of the test subjects is shown below:

```python
import numpy as np
from sklearn import tree
import itertools
from sklearn.decomposition import PCA

#the trainign samples
trainingData=np.genfromtxt("merged.csv", delimiter=",", skip_header=1, usecols=range(2, 187), max_rows=179 )


#the class of each of the training data
classes=np.genfromtxt("merged.csv", delimiter=",", skip_header=1, usecols=(1), max_rows=179 )

#the data we want to predict
testData=np.loadtxt("merged.csv", delimiter=",", skiprows=180, usecols=range(2, 187), )

#Initialize
pca = PCA(n_components = 5)
classifier = tree.DecisionTreeClassifier()

#transform /fit
X_t_train = pca.fit_transform(trainingData)
classifier.fit(X_t_train, classes)
X_t_test = pca.transform(testData)

#make the prediction
prediction=classifier.predict(X_t_test)


#print each sample and prediction
print("Sample    Prediction")

for i, j in itertools.zip_longest(range(len(prediction)), range(179, 359)):
    print(j, "         ", prediction[i])
```

Listing 7: Decision Tree implementation in Python

Subsequently, we ran several test cases to test the algorithm's accuracy by using part of the training data as test data. For each test case we changed the amount of training data accordingly. In the table below, an overview with each test and its accuracy is provided. Two sets of tests were conducted, with PCA and without PCA. As you can observe from the results, PCA does not really improve DT accuracy. The conclusion we can draw from these tests, is that, the bigger the training sample the, the higher the accuracy.

Table 3: Decision Tree test cases w/o PCA

Test#	Training Data	Test Data	Accuracy
1	20	50	94-96%
2	40	50	86%
3	60	50	90-96%
4	90	50	88-94%
5	100	50	92-94%
6	120	50	94-98%

Table 4: Decision Tree test cases with PCA

Test#	Training Data	Test Data	Accuracy
1	20	50	80%
2	40	50	86%
3	60	50	84-86%
4	90	50	86-90%
5	100	50	90-92%
6	120	50	90-94%

```python
import numpy as np
from sklearn import tree
from sklearn.decomposition import PCA
import itertools


#the trainign samples
trainingData=np.genfromtxt("merged.csv", delimiter=",", skip_header=1,
    usecols=range(2, 187), max_rows=160)

#the class of each of the training data
classes=np.genfromtxt("merged.csv", delimiter=",", skip_header=1, usecols
    =(1), max_rows=160)

test=np.genfromtxt("merged.csv", delimiter=",", skip_header=161, usecols=
    range(2, 187), max_rows=19)
testClass=np.genfromtxt("merged.csv", delimiter=",", skip_header=161,
    usecols=(1), max_rows=19)


pca = PCA(n_components = 5)
classifier = tree.DecisionTreeClassifier ()
X_t_train = pca.fit_transform(trainingData)
classifier.fit(X_t_train , classes)
X_t_test = pca.transform(test)


#make the prediction
prediction=classifier.predict(X_t_test)
```

```

x=0

for i in range(len(prediction)):
    if testClass[i]==prediction[i]:
        x+=1

print (x/len(prediction))
```

Listing 8: Decision Tree Test Case

2.2.2 b

As described above we had very good results using principle component analysis first and reduce it to 5 principle components as described in section 2.1. After which we executed kNN with 4 nearest neighbour but it doesn't really matter how much you choose as long as it is below five as can be seen in Figure 9. Using this configuration of kNN and 5 principle components we found that there were 162 people of group one, and 18 people of group two. Since we had to find 20 persons of group 2, we concluded that we missed two cases. However we think that finding 18 out of 20 is a very decent result in this case. The code to generate our results can be seen in Listing 9, of course you have to change the parameters of the finalpredict to make the final prediction based on the model you want. The resulting csv file created using this classifier can be found on Github.

We also did a test with using only the 5 most important features (found using Figure 2 which wasn't a very big success. When we analysed the given feature set without labels and found that there where 140 people of group one, and 40 people of group two. Since we know that there where supposed to be only 21 of group 2, therefore we can conclude that we have over-fitted the model in this case. This makes sense since we only looked at a very small number of features which aren't able to classify Type 1 and Type 2 correctly. On the test set however the results were kind of decent with 94% accuracy. We might get better performance if we had used all features above the red dashed line (first 104). That way there might have been enough variation to correctly classify Type 1 and Type 2. However we think the performance will be similar to using a PCA and have not actually tested it on the dataset.

We might be able to improve our results if we were to use TSNE instead of PCA. However since none of us has previous experience with TSNE and we were a bit short on time already we didnt do this. However we did lookup some literature about TSNE in combination with K nearest neighbour and most articles seemed to conclude that Knn does not have much added value to TSNE since uses perplexity already.

Additionally, when using Decision Tree as a classifier, we get higher accuracy, of about 98% on the training data, as can be seen in the tables above. That, however, does not necessarily mean that the Decision Tree is is more accurate than kNN since it predicts a much higher number (26) of subjects in group 2 than it's supposed to. So apparently the decision tree overfits. According to some literature we found online this might be solved by doing a principle component analysis first however in our case the results did not improve or even worsened.

2.2.3 c

The general structure of the experiments done in the next section looks like Figure 11

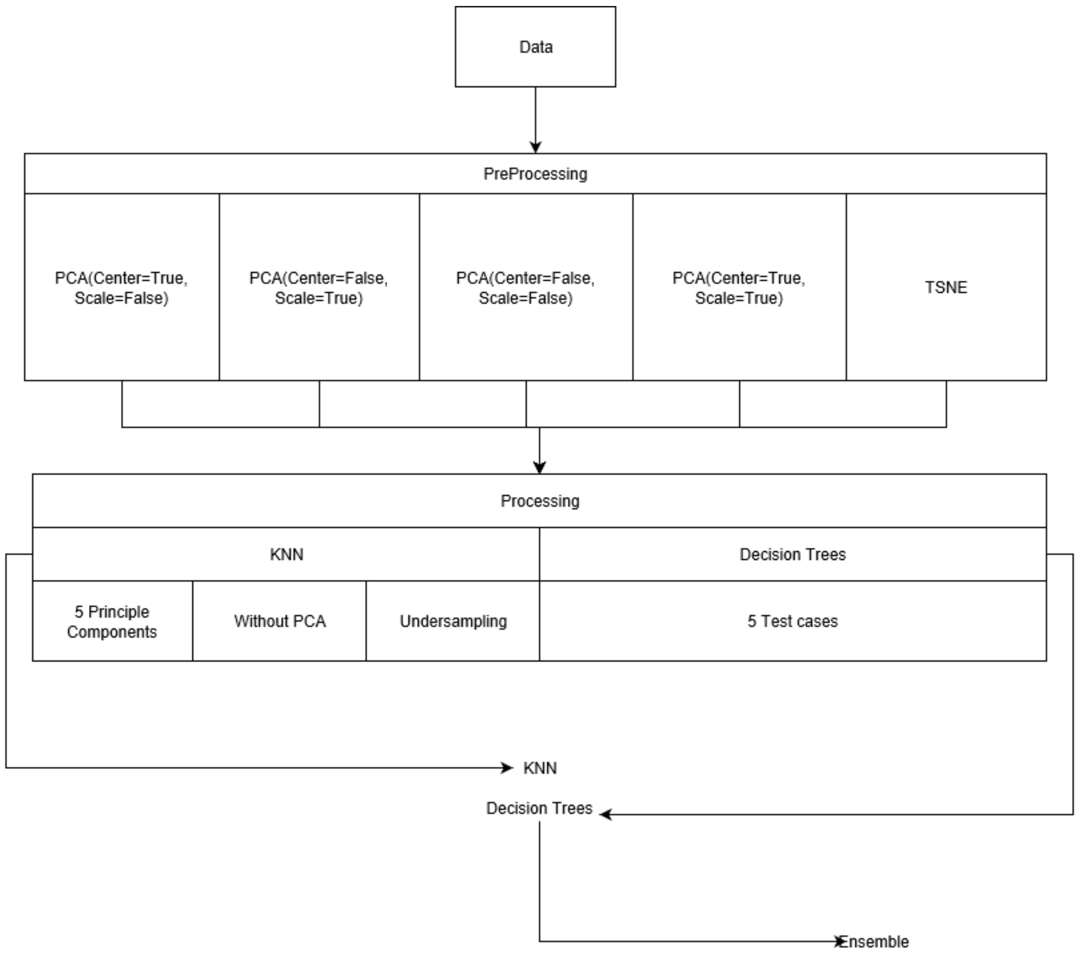


Figure 11: Structure of the document

For obvious reasons we started our research process by looking at which ways of preprocessing are best suited for this case. Since there were a lot of variables, not all of which are relevant, we were interested in ways of reducing this. This could be done using principle component analysis. When we plotted the components we noticed that the optimal point is at about 5 principle components. The next step was to look into TSNE which already did a principle component analysis by default in R. The results looked quite promising because there were clearly some clusters present of type twos. We also tested whether centering and scaling were important when using PCA and concluded we better us either as long as we don't use them at the same time.

The next step was to do an N-fold cross validation. We could do this using Decision trees or using K nearest neighbour. When doing the K-nearest neighbour we checked both with and without PCA but the results with a PCA were better. We already knew from preprocessing that 5 principle components were needed and implementation of PCA in the Knn algorithm was easily done. Using the combination we achieved an accuracy of +95% sometimes even hitting 98% which are promising results. We also tested undersampling however that did not produce results better than using the combination of PCA and Knn.l

Using Decision Trees we conducted two sets of experiments with varying training data numbers, each set having 6 iterations. The sample was quite limited, with many features for each subject. For that reason, the first set of experiments was done without the use of PCA, taking all features into account, while the second was done with PCA. The results are somewhat not expected, as PCA is supposed to produce higher accuracy based on the most promising features. That can be explained, though, when considering the limited amount of training and test subject used.

Finally we conclude that we achieved the best results using K nearest neighbour with PCA as preprocessing and k=4. We achieved an accuracy of 97% on the train data set and found 18 out of 20 Type 2 patients in the test data. In Table 5 you can find our top five experiments. You can see our final results on github with the file name FinalPredictionKNN.csv.

Table 5: Top 5 experiments

Experiment	Accuracy
Knn 5 principle components k=4	97%
Decision tree w/o PCA	97%
Knn k=1 no pre processing	94%
With PCA	~92%

```r
require(caret)

#setting seed so that random behaviour is consisitent
set.seed(38)

#set the working directory to the directory that contains the files using
    setwd()
features <-  read.csv(file.choose(new = FALSE), header=TRUE,
    stringsAsFactors=TRUE)
labels <-  read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors
    =TRUE)

practiceFeatures <- features[c(1:nrow(labels)),]
testFeatures <- features[-c(1:nrow(labels)),]

#For some reason it demands make.names
labelgood <- make.names(labels$S_labels)

#Create model without preprocessing
knnModel <- train(practiceFeatures,
                  labelgood,
                  method='knn',
                  trControl=trainControl(method='cv', number = 10,
                      classProbs=TRUE, savePredictions="final"),
                  tuneGrid=expand.grid(k=c(1:25)))

predict <- predict(knnModel, features)

#Create model using 5 principle components
knnModel <- train(practiceFeatures,
                  labelgood,
                  method='knn',
                  preProcess = c("pca", "center"),
                  trControl=trainControl(method='cv', number = 10,
                      classProbs=TRUE, savePredictions="final",
                      preProcOptions = list(pcaComp = 5)),
                  tuneGrid=expand.grid(k=c(1:25)))

top5predict <- predict(top5knnModel, features)

underSample <- features[c
    (1:29,35,44,56,61,72,74,90,98,107,119,121,131,133,137,140,143,173),]
underLabels <- labels[c
    (1:29,35,44,56,61,72,74,90,98,107,119,121,131,133,137,140,143,173),]

underLabelsGood <- make.names(underLabels)

underknnModel <- train(underSample,
                       underLabelsGood,
                       method='knn',
                       trControl=trainControl(method='cv', number = 10,
                           classProbs=TRUE, savePredictions="final"),
                       tuneGrid=expand.grid(k=c(1:25)))

```

```r
underpredict <- predict(underknnModel, features)

finalPredict <- predict(knnModel, testFeatures)
write.csv(finalPredict, "FinalPrediction.csv")
```

Listing 9: kNN implementation in R