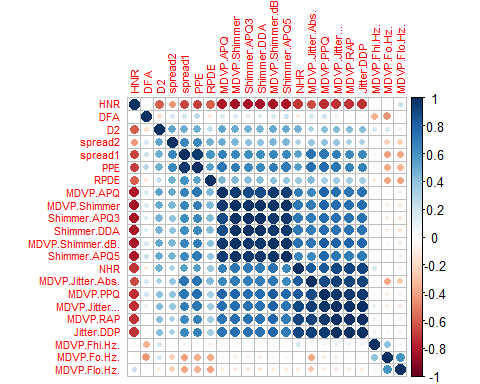
R Notebook

Steffen Lehmann - 20153945

cordata <- data %>% dplyr::select(-name, -status)  
cordata <- cor(cordata)  
corrplot(cordata, order = "hclust", tl.cex = 0.7)

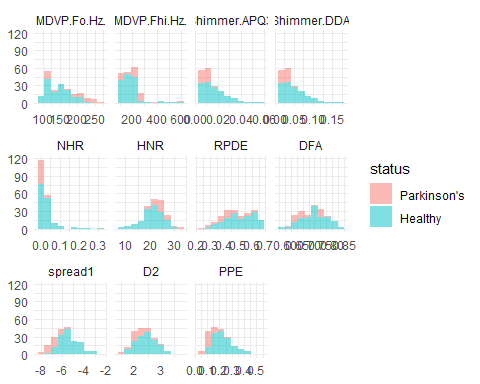


highlyCor <- colnames(data)[findCorrelation(cordata, cutoff = 0.9, verbose = TRUE)]

## Compare row 10 and column 9 with corr 0.987   
## Means: 0.642 vs 0.495 so flagging column 10   
## Compare row 9 and column 13 with corr 0.95   
## Means: 0.621 vs 0.482 so flagging column 9   
## Compare row 7 and column 4 with corr 0.974   
## Means: 0.611 vs 0.467 so flagging column 7   
## Compare row 4 and column 5 with corr 0.936   
## Means: 0.588 vs 0.452 so flagging column 4   
## Compare row 13 and column 12 with corr 0.949   
## Means: 0.564 vs 0.436 so flagging column 13   
## Compare row 14 and column 11 with corr 1   
## Means: 0.541 vs 0.421 so flagging column 14   
## Compare row 11 and column 12 with corr 0.96   
## Means: 0.51 vs 0.407 so flagging column 11   
## Compare row 5 and column 6 with corr 0.923   
## Means: 0.534 vs 0.392 so flagging column 5   
## Compare row 6 and column 8 with corr 1   
## Means: 0.469 vs 0.371 so flagging column 6   
## Compare row 8 and column 15 with corr 0.92   
## Means: 0.425 vs 0.359 so flagging column 8   
## Compare row 22 and column 19 with corr 0.962   
## Means: 0.512 vs 0.337 so flagging column 22   
## All correlations <= 0.9

corcheck <- data[, which(!colnames(data) %in% highlyCor)] # by pair comparation  
  
corcheck <- corcheck %>% dplyr::select(-name)

#Plot histograms of "\_mean" variables group by diagnosis  
ggplot(data = melt(corcheck, id.var = "status"), mapping = aes(x = value)) + theme\_minimal() +   
 geom\_histogram(bins = 10, aes(fill=as.factor(status)), alpha=0.5) + facet\_wrap(~variable, scales = 'free\_x') + labs(fill="status") +   
 scale\_fill\_discrete(labels = c("Parkinson's", "Healthy")) + theme(axis.title.x = element\_blank(), axis.title.y = element\_blank())



#converting the data frame from R to Python  
#py\_data = r.data  
py\_data = r.corcheck  
  
#Pre-processing  
# dropping / removing the labels from the data frame   
X = py\_data.drop(columns=['status'], axis=1)  
# data frame that is only containing the label  
Y = py\_data['status']  
  
  
# splits the data into a test and training test, with 80% of the data being used for training. Shuffles the data twice before spiting   
#https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html?highlight=train\_test\_split#sklearn.model\_selection.train\_test\_split  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=2)  
  
#Scaling the data.   
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html?highlight=standardscaler#sklearn.preprocessing.StandardScaler   
  
scaler = StandardScaler()  
#Only fit on the training set  
scaler.fit(X\_train)  
#Transform both sets

## StandardScaler()

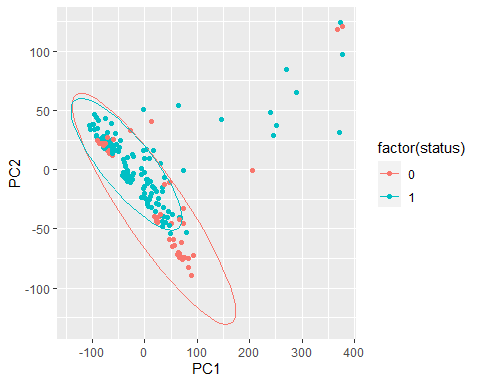
X\_train\_scaled = scaler.transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
k = 5  
kf = KFold(n\_splits=k, random\_state=None)

#pca with two principal components  
pca = PCA(n\_components=2)  
principalComponents = pca.fit\_transform(X)  
#first principal component contains 72.89% of the variance and the second principal component contains 21.82% of the variance.   
PCA\_variance\_n2 = pca.explained\_variance\_ratio\_   
  
principalDf = pd.DataFrame(data = principalComponents, columns = ['PC1', 'PC2'])  
  
#Concatenating DataFrames  
plotDf = pd.concat([principalDf, Y], axis = 1)  
  
#Scaling the data.   
scaler = StandardScaler()  
scaler.fit(plotDf)

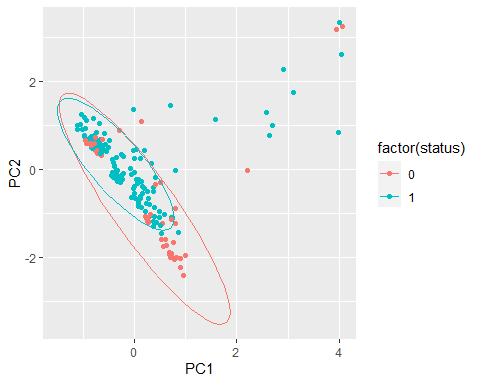
## StandardScaler()

X\_PCA\_scaled = scaler.transform(plotDf)

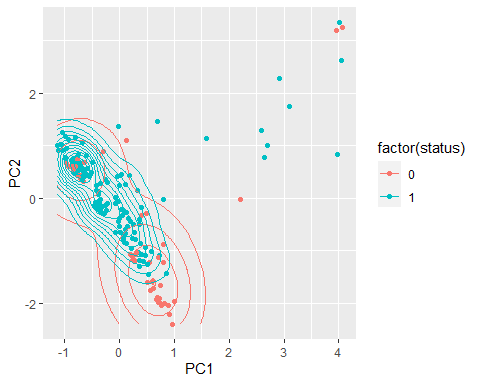
PCA2 <- py$plotDf  
scal\_PCA2 <- py$X\_PCA\_scaled  
  
scal\_PCA2Df <- as.data.frame(scal\_PCA2)  
  
scal\_PCA2Df['V3'][scal\_PCA2Df['V3'] > 0] <- 1  
scal\_PCA2Df['V3'][scal\_PCA2Df['V3'] < 0] <- 0  
scal\_PCA2Df <- scal\_PCA2Df %>% rename(PC1 = V1, PC2 = V2, status = V3)  
  
# Scatter plot of PCA(n=2)  
ggplot(PCA2, aes(x=PC1, y=PC2, group=factor(status), colour=factor(status))) + geom\_point() + stat\_ellipse()



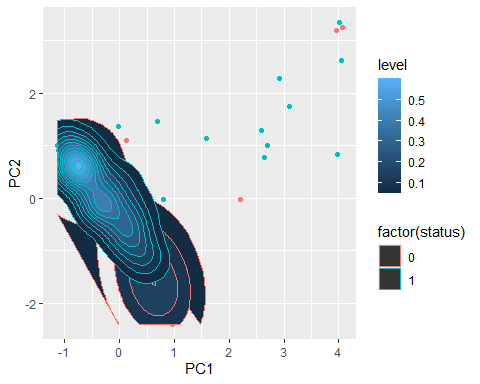
# Shows scatter of scaled dataPCA(n=2)  
ggplot(scal\_PCA2Df, aes(x=PC1, y=PC2, group=factor(status), colour=factor(status))) + geom\_point() + stat\_ellipse()



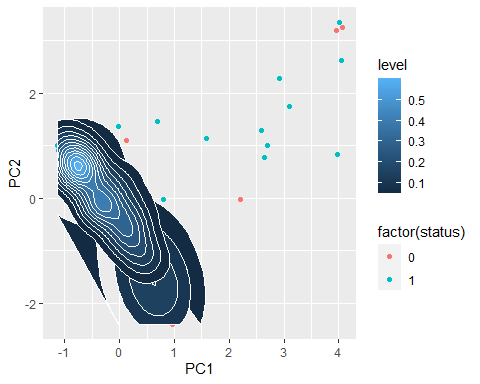
# Show the contour only PCA(n=2)  
ggplot(scal\_PCA2Df, aes(x=PC1, y=PC2, group=factor(status), colour=factor(status))) + geom\_point() + geom\_density\_2d()



# Show the area only PCA(n=2)  
ggplot(scal\_PCA2Df, aes(x=PC1, y=PC2, group=factor(status), colour=factor(status))) + geom\_point() + stat\_density\_2d(aes(fill = ..level..), geom="polygon")



# shows area + contour  
ggplot(scal\_PCA2Df, aes(x=PC1, y=PC2, group=factor(status), colour=factor(status))) + geom\_point() + stat\_density\_2d(aes(fill = ..level..), geom="polygon", colour="white")



# 0.95 are the components parameter. It means that scikit-learn choose the minimum number of principal components such that 95% of the variance is retained.  
PCA\_SVC = PCA(.95)  
PCA\_SVC.fit(X\_train\_scaled)  
#Number of PC's with 95% variance.

## PCA(n\_components=0.95)

Number\_of\_PC = PCA\_SVC.n\_components\_  
#Apply the PCA fit  
PCA\_train = PCA\_SVC.transform(X\_train\_scaled)  
PCA\_test = PCA\_SVC.transform(X\_test\_scaled)

#Training the Support Vector classifier   
SVM\_model = svm.SVC(kernel='linear')  
SVM\_model.fit(X\_train\_scaled, Y\_train)  
  
#Training the PCA Support Vector classifier

## SVC(kernel='linear')

SVM\_PCA\_model = svm.SVC(kernel='linear')  
SVM\_PCA\_model.fit(PCA\_train, Y\_train)

## SVC(kernel='linear')

model=svm.SVC(kernel='linear')  
kfold\_validation=KFold(10)  
  
results=cross\_val\_score(model,X\_train\_scaled,Y\_train,cv=kfold\_validation)  
print(np.mean(results))

## 0.8583333333333334

#TODO model is over fitted, so the attributes of the model needs to be designed better  
rf\_Model = RandomForestClassifier()  
rf\_Model.fit(X\_train\_scaled, Y\_train)  
  
  
#With PCA

## RandomForestClassifier()

rf\_PCA\_Model = RandomForestClassifier()  
rf\_PCA\_Model.fit(PCA\_train, Y\_train)

## RandomForestClassifier()

NN = MLPClassifier(activation = 'logistic', solver = 'sgd', hidden\_layer\_sizes=(100), random\_state=1, max\_iter=3000)  
NN.fit(X\_train\_scaled, Y\_train)

## MLPClassifier(activation='logistic', hidden\_layer\_sizes=100, max\_iter=3000,  
## random\_state=1, solver='sgd')

NN\_PCA = MLPClassifier(activation = 'logistic', solver = 'sgd', hidden\_layer\_sizes=(100), random\_state=1, max\_iter=3000)  
NN\_PCA.fit(PCA\_train, Y\_train)

## MLPClassifier(activation='logistic', hidden\_layer\_sizes=100, max\_iter=3000,  
## random\_state=1, solver='sgd')

gnb = GaussianNB()  
gnb.fit(X\_train\_scaled, Y\_train)

## GaussianNB()

gnb\_PCA = GaussianNB()  
gnb\_PCA.fit(PCA\_train, Y\_train)

## GaussianNB()

Models = [SVM\_model, rf\_Model, NN, gnb]  
PCA\_Models= [SVM\_PCA\_model, rf\_PCA\_Model, NN\_PCA, gnb\_PCA]  
Model\_name = ['Support Vector Classifier','Random Forest Classifier', 'Neural network', 'Gaussian Naive Bayes']  
AS = []  
AS\_PCA = []  
AS\_val = []  
  
  
for Model, PCA\_Model in zip(Models, PCA\_Models):  
 Predicted\_labels= Model.predict(X\_test\_scaled)  
 PCA\_Predicted\_labels= PCA\_Model.predict(PCA\_test)  
 AS.append(accuracy\_score(Predicted\_labels, Y\_test))   
 AS\_PCA.append(accuracy\_score(PCA\_Predicted\_labels, Y\_test))  
  
Accurracy\_Store = pd.DataFrame(data=[AS, AS\_PCA], columns=Model\_name, index=['Standard', 'PCA'])  
  
  
#for Model in Models  
# kfold\_validation=KFold(10)  
# results=cross\_val\_score(Model,X\_train\_scaled,Y\_train,cv=kfold\_validation)  
# AS\_val.append(np.mean(results))  
  
#Accurracy\_Store = pd.DataFrame(data=[AS, AS\_PCA, AS\_val], columns=Model\_name, index=['Standard', 'PCA', 'Kfold'])

dat <- data.frame(y=scal\_PCA2Df$status, x1=scal\_PCA2Df$PC1, x2=scal\_PCA2Df$PC2)  
SVM\_X <- scal\_PCA2Df %>% dplyr::select(-status)  
SVM\_Y <- scal\_PCA2Df$status  
SVM\_Y <- as.factor(SVM\_Y)  
s <- seq(from=-3,to=5,length=400)  
  
# for standard SVM usage, do not set this `C` parameter so high  
# this will be discussed later when we talk about "soft margin" SVM  
tg <- data.frame(C=100)  
fit <- train(SVM\_X, SVM\_Y, method="svmLinear", tuneGrid=tg)  
alpha <- fit$finalModel@alpha[[1]]  
sv <- as.data.frame(SVM\_X[fit$finalModel@SVindex,]) # the "support vectors"  
sv.y <- 2 \* (as.numeric(SVM\_Y[fit$finalModel@SVindex]) - 1.5)  
w <- colSums(alpha \* sv.y \* as.matrix(sv))  
b <- fit$finalModel@b  
grid <- expand.grid(x1=s,x2=s)  
grid$y.cont <- (as.matrix(grid[,1:2]) %\*% w - b)[,1]  
ggplot(dat, aes(dat$x1,dat$x2,col=dat$y)) + geom\_point() +   
 geom\_contour(data=grid, aes(x1,x2,z=y.cont), breaks=c(-1,0,1), col="black")

## Warning: Use of `dat$x1` is discouraged. Use `x1` instead.

## Warning: Use of `dat$x2` is discouraged. Use `x2` instead.

## Warning: Use of `dat$y` is discouraged. Use `y` instead.

