Support vector machines (SVM), cleaning noisy data using k-nearest neighbors (kNN)

Introduction to Topological Data Analysis with R - Lab 3

Peter Bubenik (September 2020), with modifications by Johnathan Bush and Iryna Hartsock (July 2022)

Functions

Load functions that we defined in the previous labs

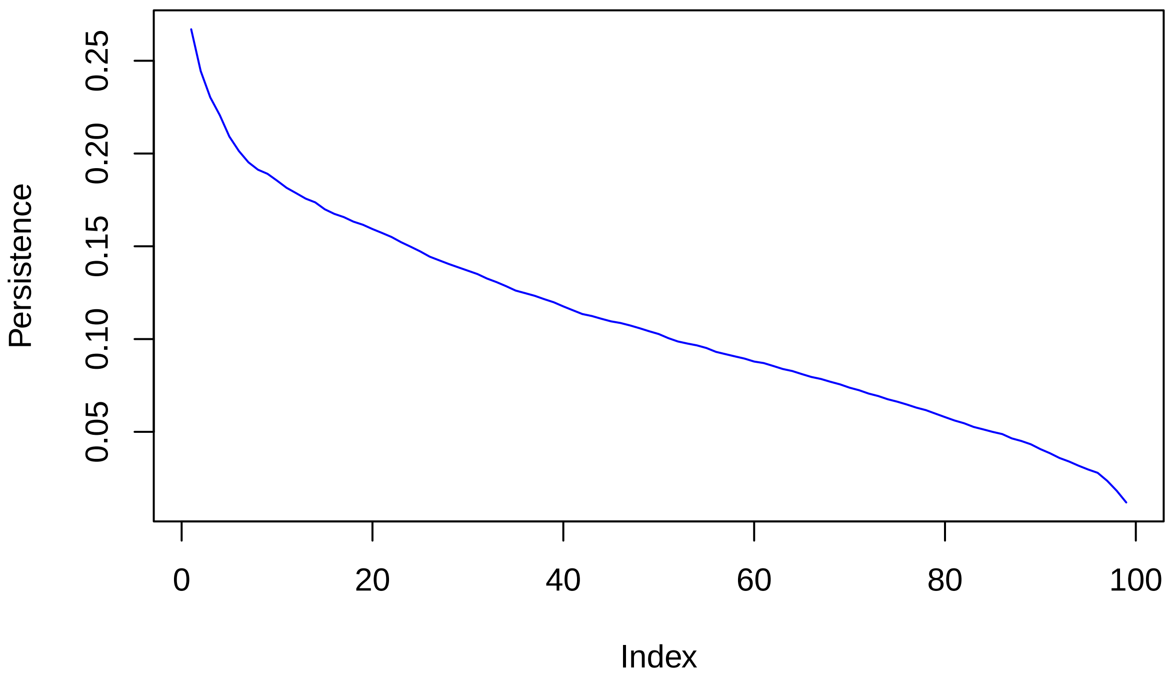
```r
euclidean.distance <- function(u, v) sqrt(sum((u - v) ^ 2))

# Sample points from an annulus
sample.annulus <- function(num.pts, inner.radius, outer.radius){
  theta <- runif(num.pts) * 2 * pi
  radius <- sqrt(runif(num.pts, inner.radius^2, outer.radius^2))
  x <- radius * cos(theta)
  y <- radius * sin(theta)
  cbind(x,y)
}

# Take difference of vectors of potentially different lengths
difference.vectors <-function(vector.1,vector.2){
  length.1 <- length(vector.1)
  length.2 <- length(vector.2)
  difference.vector = numeric(max(length.1,length.2))
  difference.vector = as(difference.vector, "sparseVector")
  difference.vector[1:length.1] = difference.vector[1:length.1] + vector.1
  difference.vector[1:length.2] = difference.vector[1:length.2] - vector.2
}

# Compute death vector from persistence diagram
death.vector <- function(PD){
  zeroes <- which(PD[, 1] == 0)
  PD.0 <- PD[zeroes,2:3]
  dv <- vector()
  if ((min(PD.0[,"Birth"]) == 0) && (max(PD.0[,"Birth"]) == 0))
    dv <- sort(PD.0[,2], decreasing=TRUE)
  return(dv)
}

# Matrix of death vectors from a list of persistence diagrams
death.vector.matrix <- function(PD.list){
  num.pts <- length(which(PD.list[[1]][,1] == 0))
  DVM <- matrix(0L, nrow = length(PD.list), ncol = num.pts - 1)
  for (c in 1 : length(PD.list))
    DVM[c,] <- death.vector(PD.list[[c]])[-1]
  return(DVM)
```

```
}

# Matrix of persistence landscape row vectors from list of persistence landscapes
landscape.matrix.from.list <- function(PL.list){
  n <- length(PL.list)
  m <- ncol(PL.list[[1]])
  max.depth <- integer(n)
  for (i in 1:n)
    max.depth[i] <- nrow(PL.list[[i]])
  K <- max(max.depth)
  PL.matrix <- Matrix(0, nrow = n, ncol = m*K, sparse = TRUE)
  for (i in 1:n)
    for (j in 1:max.depth[i])
      PL.matrix[i,(1+(j-1)*m):(j*m)] <- PL.list[[i]][j,]
  return(PL.matrix)
}

# Convert a vector to a persistence landscape
landscape.from.vector <- function(PL.vector, t.vals){
  m <- length(t.vals)
  K <- length(PL.vector)/m
  PL <- Matrix(0, nrow = K, ncol=m, sparse = TRUE)
  for (i in 1:K){
    PL[i,1:m] <- PL.vector[(1+(i-1)*m):(i*m)]
  }
  return(PL)
}

# Plot Persistence Landscape
plot.landscape <- function(PL,t.vals){
  plot(t.vals,PL[1,],type="l",ylab="Persistence",xlab="Parameter values",col=1,ylim=c(min(PL),max(PL)))
  for(i in 2:dim(PL)[1])
    lines(t.vals,PL[i,],type="l",col=i)
}

# Remove dimension zero points from a persistence diagram
remove.dimzero.from.pd <- function(PD){
  zeroes <- which(PD[, 1] == 0)
  newPD <- PD[-zeroes,]
  return(newPD)
}

# Discrete Persistence Landscape
discrete.PL <- function(PD,t.vals){
  get.tent <- function(interval,t.vals){
    y <- numeric(length(t.vals))
    # Compute slope +1 part of tent
    going_up <- which(t.vals > interval[1] & t.vals <= mean(interval))
    y[going_up] <- (t.vals[going_up] - interval[1])
    # Compute slope -1 part of tent
    going_down <- which(t.vals > mean(interval) & t.vals < interval[2])
    y[going_down] <- (interval[2] - t.vals[going_down])
    return(y)
  }

# discrete landscape using sparse matrix
PL <- Matrix(0, nrow = dim(PD)[1], ncol = length(t.vals), sparse = TRUE)
  for(i in 1:dim(PD)[1])
    PL[i, ] <- get.tent(PD[i, ],t.vals)
  for(j in 1:length(t.vals)){
    PL[ ,j] <- sort(PL[ ,j], decreasing = TRUE)
  }
  actual.depth <- sum( rowSums(PL) != 0)
  PL <- PL[1:actual.depth,]
  return(PL)
}

# Plot persistence landscape
plot.persistence.landscape <- function(PL.single.degree,t.vals){
  plot(t.vals,PL.single.degree[1,],type="l",ylab="Persistence",
       xlab="Parameter values",col=1)
  for(i in 2:dim(PL.single.degree)[1])
    lines(t.vals,PL.single.degree[i,],type="l",col=i)
}

# Permutation test for two matrices consisting of row vectors
permutation.test <- function(M1 ,M2, num.repeats = 10000){
  # append zeros if necessary so that the matrices have the same number of columns
  num.columns <- max(ncol(M1),ncol(M2))
  M1 <- cbind(M1, Matrix(0,nrow=nrow(M1),ncol=num.columns-ncol(M1)))
  M2 <- cbind(M2, Matrix(0,nrow=nrow(M2),ncol=num.columns-ncol(M2)))
  t.obs <- euclidean.distance(colMeans(M1),colMeans(M2))
  k <- dim(M1)[1]
  M <- rbind(M1,M2)
  n <- dim(M)[1]
  count <- 0
  for (i in 1:num.repeats){
    permutation <- sample(1:n)
    t <- euclidean.distance(colMeans(M[permutation[1:k],]),colMeans(M[permutation[(k+1):n],]))
    if (t >= t.obs)
      count <- count + 1
  }
  return(count/num.repeats)
}
```

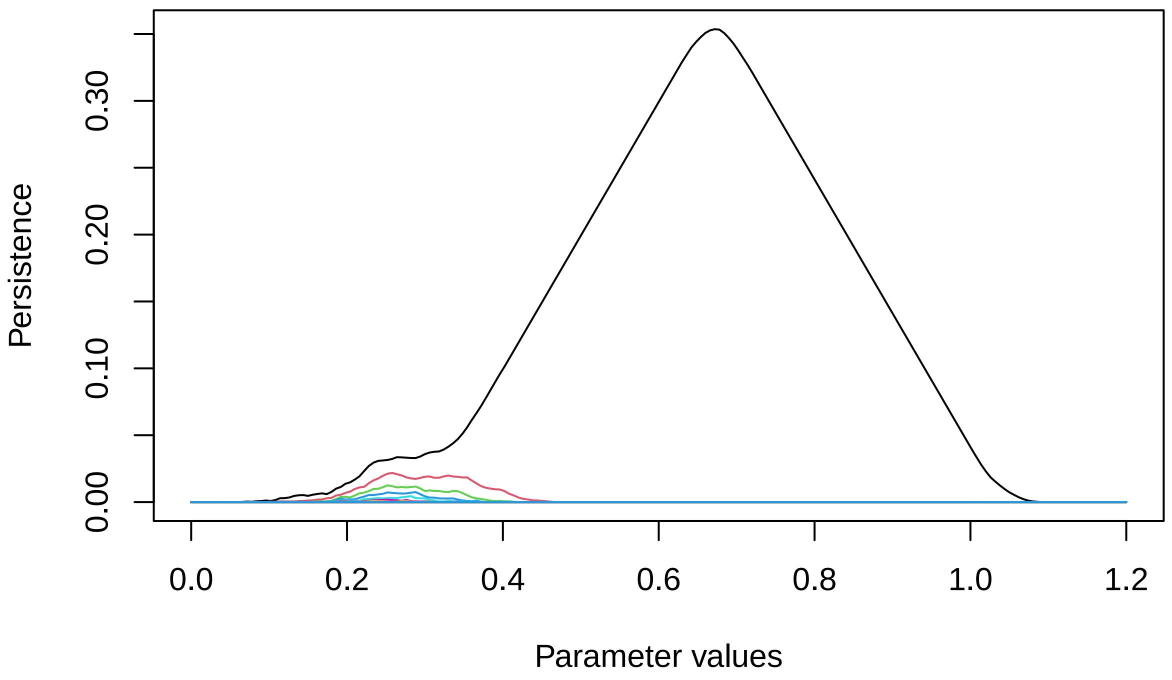
Sample from two classes

```
num.pts <- 100
inner.radius <- 1
outer.radius <- 2
min.t <- 0
max.t <- 1.2
t.steps <- 200
t.vals <- seq(min.t,max.t,(max.t-min.t)/t.steps)
num.repeats <- 20
```

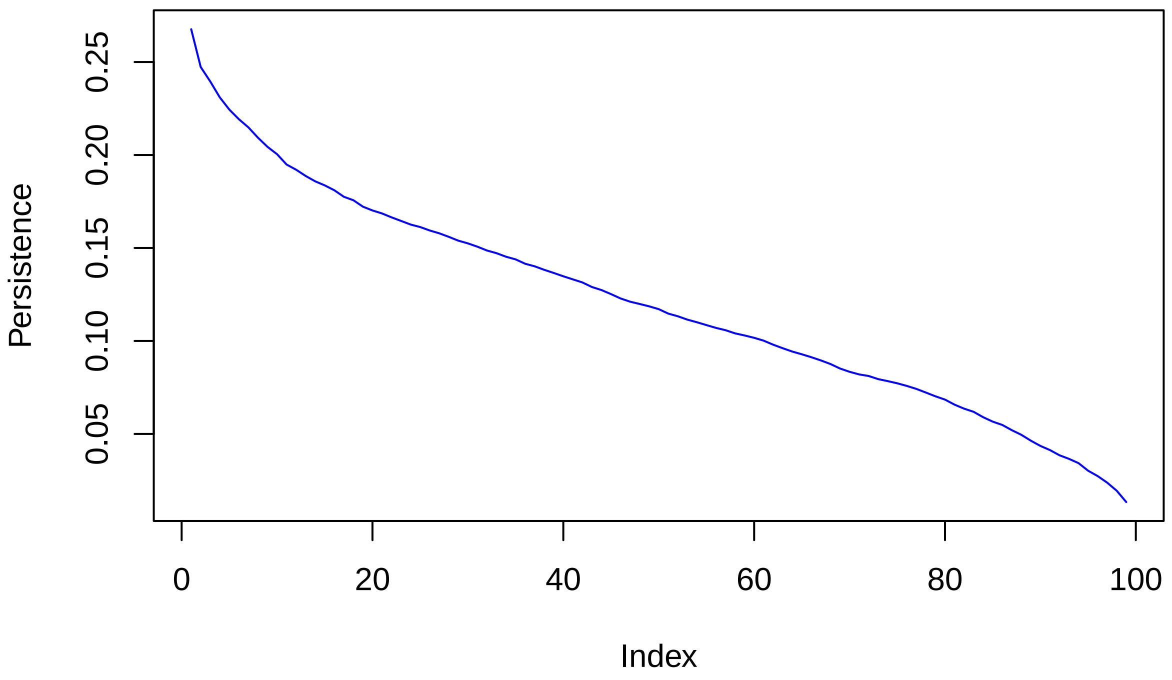
```
#Compute the death vectors and persistence landscapes for the samples and average them

PD.list <- vector("list",num.repeats)
for (c in 1 : num.repeats){
  X <- sample.annulus(num.pts,inner.radius,outer.radius)
  PH.output <- alphaComplexDiag(X)
  PD.list[[c]] <- PH.output[["diagram"]]
  PD.list[[c]][,2:3] <- sqrt(PD.list[[c]][,2:3])
}

DV.matrix <- death.vector.matrix(PD.list)
average.DV <- colMeans(DV.matrix)
plot(average.DV, type="l", col="blue", ylab = "Persistence")
```



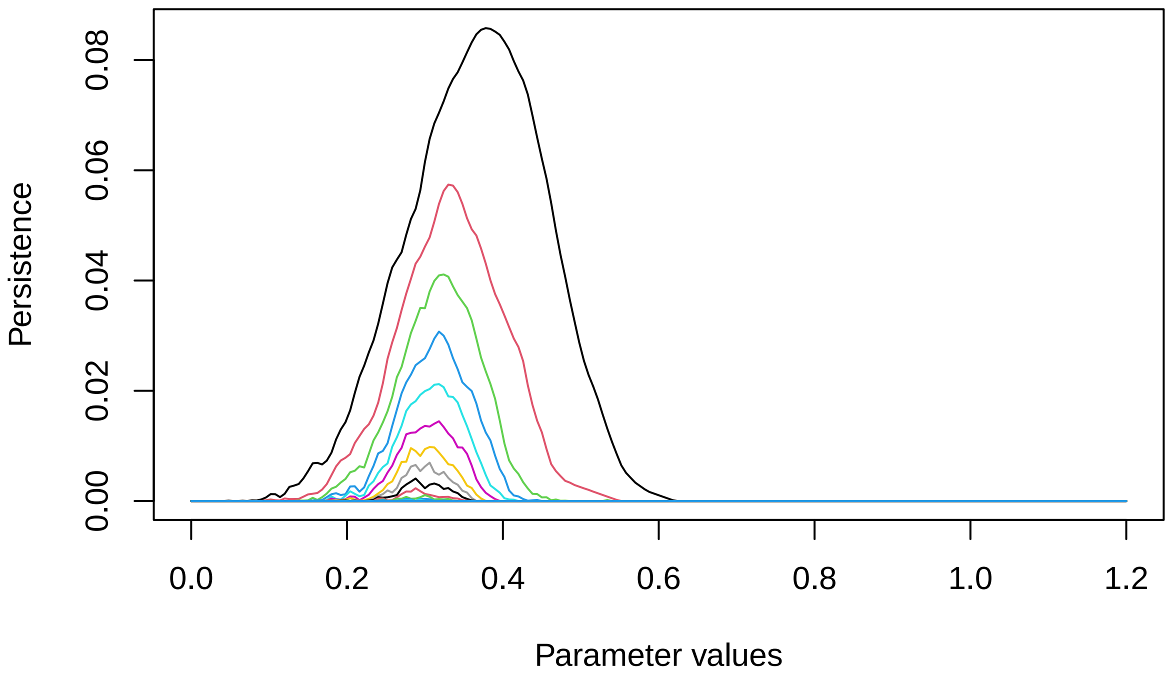
```
PL.list <- vector("list",num.repeats)
for (c in 1 : num.repeats)
  PL.list[[c]] <- t(landscape(PD.list[[c]],dimension=1,KK=1:100,tseq=t.vals))
PL.matrix <- landscape.matrix.from.list(PL.list)
average.PL.vector <- colMeans(PL.matrix, sparseResult = TRUE)
average.PL <- landscape.from.vector(average.PL.vector,t.vals)
plot.landscape(average.PL,t.vals)
```



```r
#Repeat the computations above with inner.radius = 0.25.

inner.radius <- 0.25

PD.list.2 <- vector("list",num.repeats)
for (c in 1 : num.repeats){
  X <- sample.annulus(num.pts,inner.radius,outer.radius)
  PH.output <- alphaComplexDiag(X)
  PD.list.2[[c]] <- PH.output[["diagram"]]
  PD.list.2[[c]][,2:3] <- sqrt(PD.list.2[[c]][,2:3])
}

DV.matrix.2 <- death.vector.matrix(PD.list.2)
average.DV.2 <- colMeans(DV.matrix.2)
plot(average.DV.2, type="l", col="blue", ylab = "Persistence")
```



```r
PL.list.2 <- vector("list",num.repeats)
for (c in 1 : num.repeats)
  PL.list.2[[c]] <- t(landscape(PD.list.2[[c]],dimension=1,KK=1:100,tseq=t.vals))
PL.matrix.2 <- landscape.matrix.from.list(PL.list.2)
average.PL.vector.2 <- colMeans(PL.matrix.2, sparseResult = TRUE)
average.PL.2 <- landscape.from.vector(average.PL.vector.2,t.vals)
plot.landscape(average.PL.2,t.vals)
```

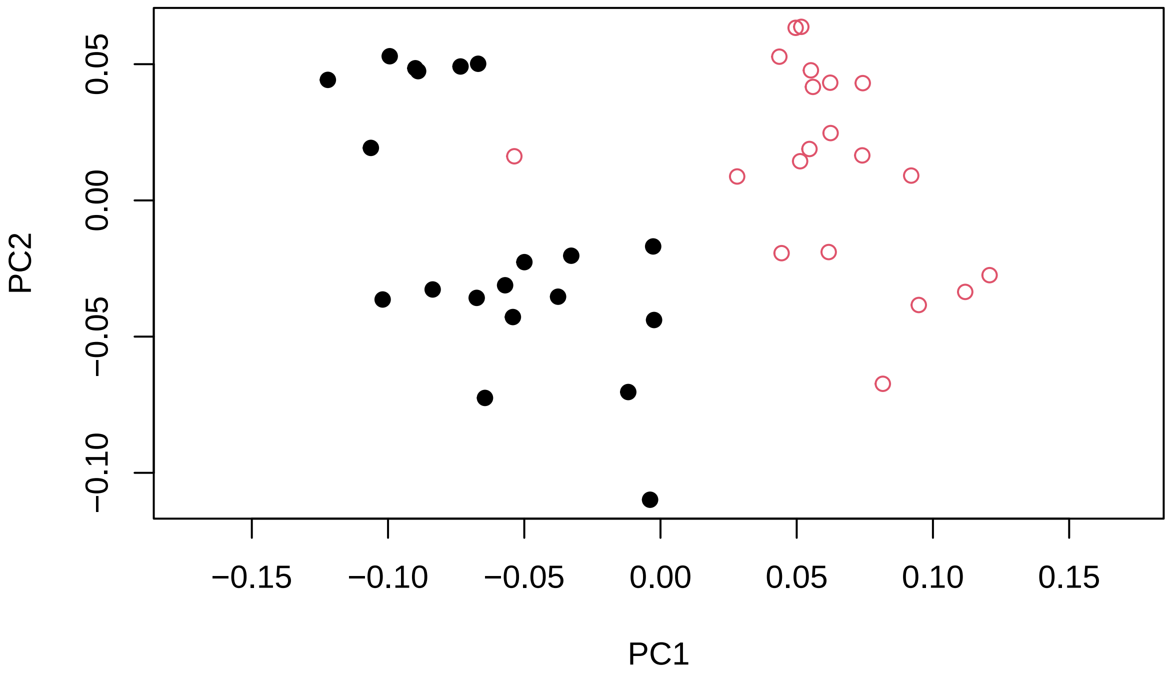


##

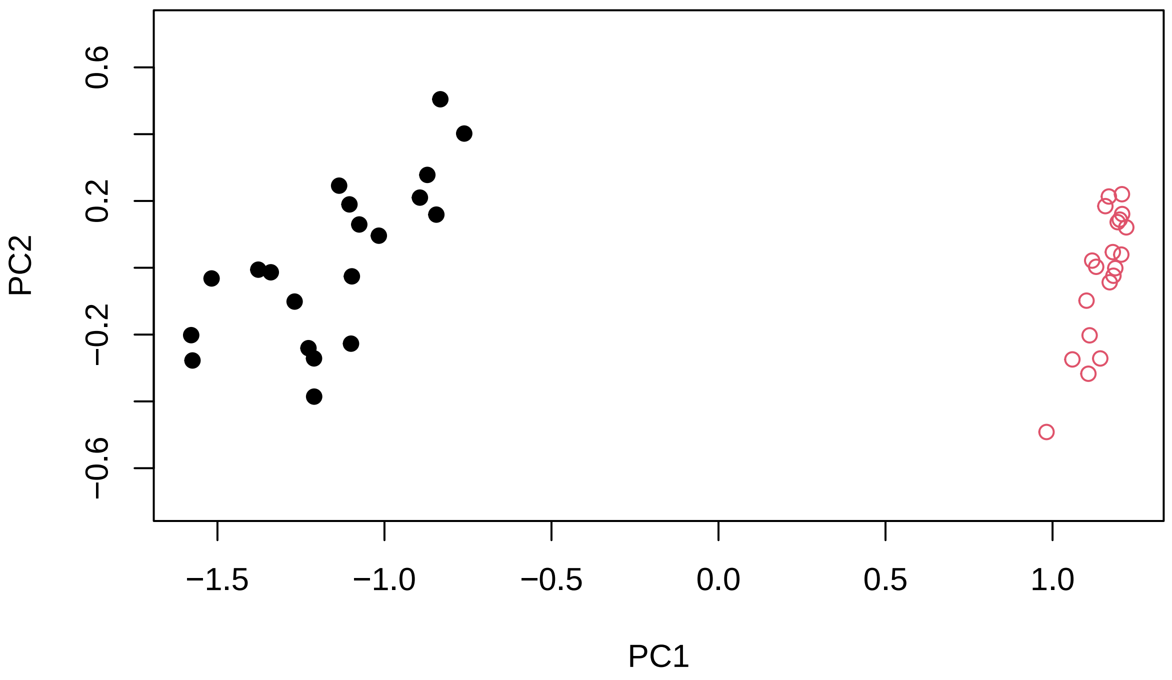
Compute the PCA projections

```r
# Principal Components Analysis (from Lab 2)
data.labels <- c(rep(1,nrow(PL.matrix)), rep(2,nrow(PL.matrix.2)))
```

```r
DV.vectors <- rbind(DV.matrix,DV.matrix.2)
pca.0 <- prcomp(DV.vectors,rank=10)
#plot(pca.0,type="l")
plot(pca.0$x[,1:2], col=data.labels, pch=17+(2*data.labels), asp=1)
```



```r
#loading.vectors <- t(pca.0$rotation)
#plot(loading.vectors[1,],type="l")
#plot(loading.vectors[2,],type="l")
PL.vectors <- rbind(PL.matrix,PL.matrix.2)
pca.1 <- prcomp(PL.vectors,rank=10)
#plot(pca.1,type="l")
plot(pca.1$x[,1:2], col=data.labels, pch=17+(2*data.labels), asp=1)
```



```
#loading.vectors <- t(pca.1$rotation)
#plot.landscape(landscape.from.vector(loading.vectors[1,],t.vals),t.vals)
#plot.landscape(landscape.from.vector(loading.vectors[2,],t.vals),t.vals)
```

Support Vector Machine (SVM) classification of 2d PCA projections

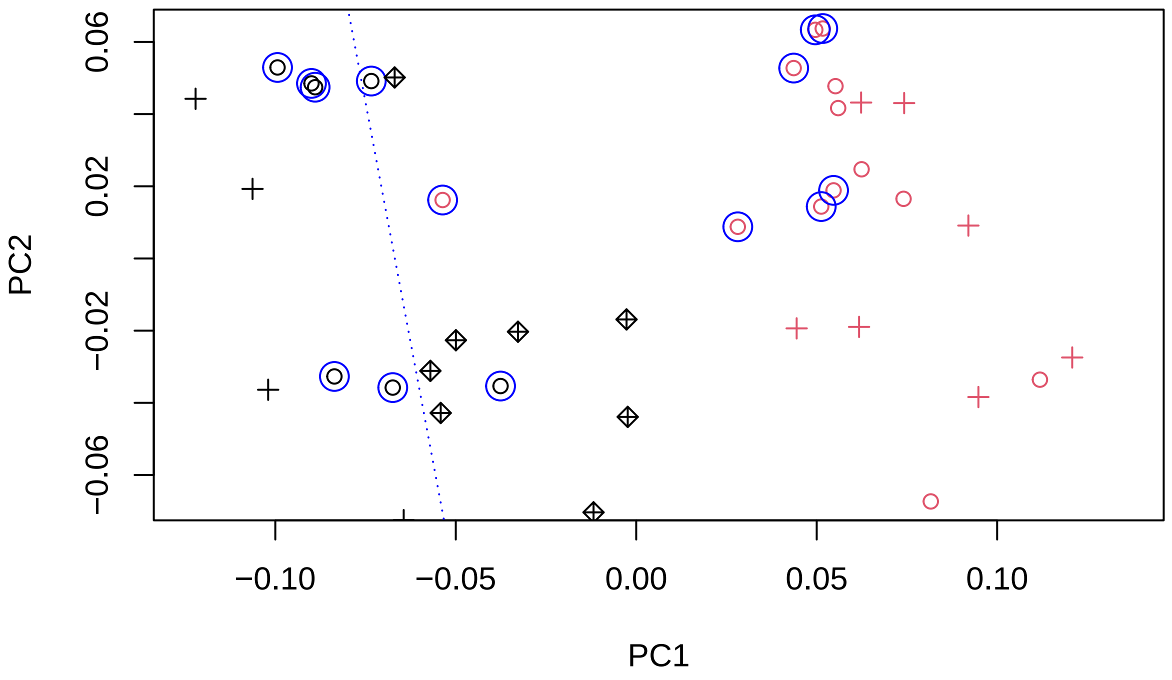
We apply SVM to the 2D PCA projections of the death vectors and the persistence landscapes. This is the wrong way to do it, but is easiest to understand and visualize. The better way to do it is to the use the high-dimensional death vectors and persistence landscapes directly.

Black and red are used from the two classes, respectively. Training data is marked with circles. The separating line is indicated with a dashed line. The training data used to compute the separating line are circled with blue circles (these are called the support vectors). Correctly labeled test data is marked with a plus symbol. Incorrectly labeled test data is marked with a plus inside a diamond.

```
# SVM for 2d pca projections of death vectors from persistence diagram in degree 0.

# Split data into training data and testing data
num.data <- length(data.labels)
test.index <- sample(num.data, trunc(num.data/2))
test.x   <- pca.0$x[test.index,1:2]
test.y   <- data.labels[test.index]
train.x   <- pca.0$x[-test.index,1:2]
train.y   <- data.labels[-test.index]

# Build the Support Vector Machine Classifier using the training data
svm.model.0 <- svm(x=train.x,y=train.y,scale=FALSE,type="C-classification",kernel="linear",cost=10)

# Use the SVM Classifier to predict the classes of the test data
svm.predict.0 <- predict(svm.model.0,test.x)

# Compute SVM confusion matrix
table(pred = svm.predict.0, true = test.y)
```

```
##      true
## pred 1 2
##    1 4 0
##    2 9 7
```

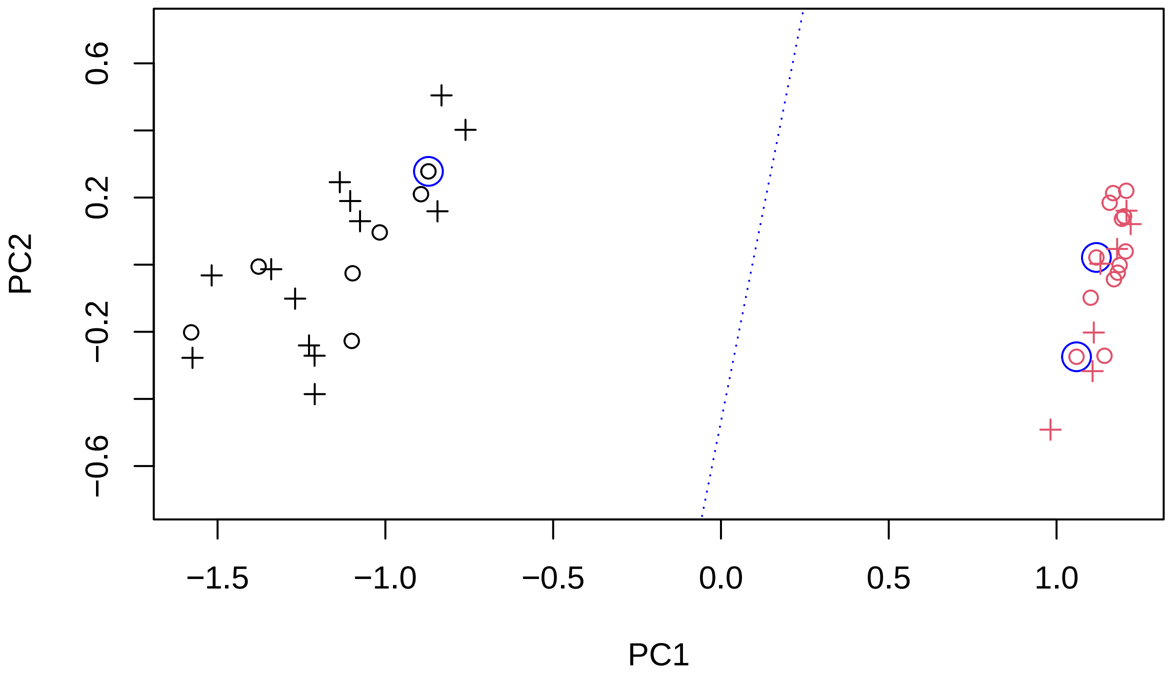
```
# Visualization:
# plot the 2D pca projection of the training data
plot(pca.0$x[-test.index,1:2], col=data.labels[-test.index], pch=1, asp=1)

# mark the support vectors
points(svm.model.0$SV, col="blue",cex=2)

# Obtain the equation of the Separating Hyperplane
w <- t(svm.model.0$coefs) %*% svm.model.0$SV
b <- -svm.model.0$rho

# plot the intersection of the separating hyperplane with the 2D pca plane
abline(a=-b/w[1,2], b=-w[1,1]/w[1,2], col="blue", lty=3)

# which testing points were correctly predicted
correct.points.0 <- svm.predict.0 == data.labels[test.index]

# plot the 2D pca projection of the testing data, colored by true label, marking incorrect predictions
points(pca.0$x[test.index,1:2], col=data.labels[test.index], pch=9-6*correct.points.0, asp=1)
```



```
# SVM for 2d pca projections of persistence landscapes from persistence diagram in degree 1.

test.x    <- pca.1$x[test.index,1:2]
train.x    <- pca.1$x[-test.index,1:2]
svm.model.1 <- svm(x=train.x,y=train.y,scale=FALSE,type="C-classification",kernel="linear",cost=1000)
svm.predict.1 <- predict(svm.model.1,test.x)
table(pred = svm.predict.1, true = test.y)
```

```
##      true
## pred  1  2
##    1 13  0
##    2  0  7
```

```
plot(pca.1$x[-test.index,1:2], col=data.labels[-test.index], pch=1, asp=1)
points(svm.model.1$SV, col="blue",cex=2)
w <- t(svm.model.1$coefs) %*% svm.model.1$SV
b <- -svm.model.1$rho
abline(a=-b/w[1,2], b=-w[1,1]/w[1,2], col="blue", lty=3)
correct.points.1 <- svm.predict.1 == data.labels[test.index]
points(pca.1$x[test.index,1:2], col=data.labels[test.index], pch=9-6*correct.points.1, asp=1)
```



Exercises (Part I)

In collaboration with some of the other participants, do the following.

1. In the SVM Classification explain each of the following:

(a) how the data is split into training data and testing data
(b) each of the parameters for the command that builds and SVM classifier
(c) how the classifier predicts the classes for the test data

2. Compare and contrast the confusion matrices for persistent homology in degrees 0 and 1
3. Compare and contrast the visualizations of SVM prediction for persistent homology in degrees 0 and 1

SVM classification of high dimensional data

Now redo without projecting to two dimensions and using “5-folds”: split the data in 5 groups, for each group, use the other 4 groups as training data and that group as testing data. Each data point gets tested exactly once.

```
cost <- 10
num.folds <- 5
svm_model <- svm(DV.vectors,data.labels,scale=FALSE,type="C-classification",kernel="linear",cost=cost,c
summary(svm_model)
```

```
##
## Call:
## svm.default(x = DV.vectors, y = data.labels, scale = FALSE, type = "C-classification",
##     kernel = "linear", cost = cost, cross = num.folds)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  30
##
##  ( 15 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 95
## Single Accuracies:
##  87.5 100 100 100 87.5
```

```
svm_model <- svm(PL.vectors,data.labels,scale=FALSE,type="C-classification",kernel="linear",cost=cost,c
summary(svm_model)
```

```
##
## Call:
## svm.default(x = PL.vectors, y = data.labels, scale = FALSE, type = "C-classification",
##     kernel = "linear", cost = cost, cross = num.folds)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  5
##
##  ( 1 4 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
##
## 5-fold cross-validation on training data:
##
## Total Accuracy: 100
## Single Accuracies:
```

```
##  100 100 100 100 100
```

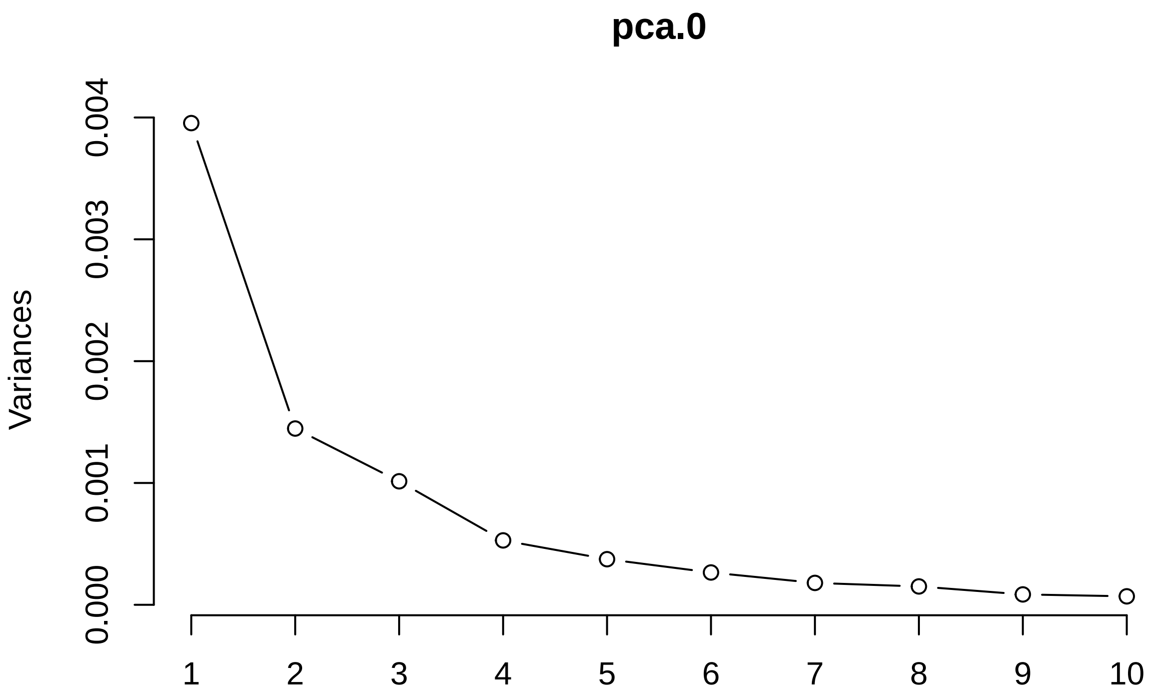
SVM regression

Above we had binary data, coming from one of two classes, and our learning task was to predict the class, called "classification". Now, we consider real-valued data, and our learning task is to predict the real-value, called "regression".

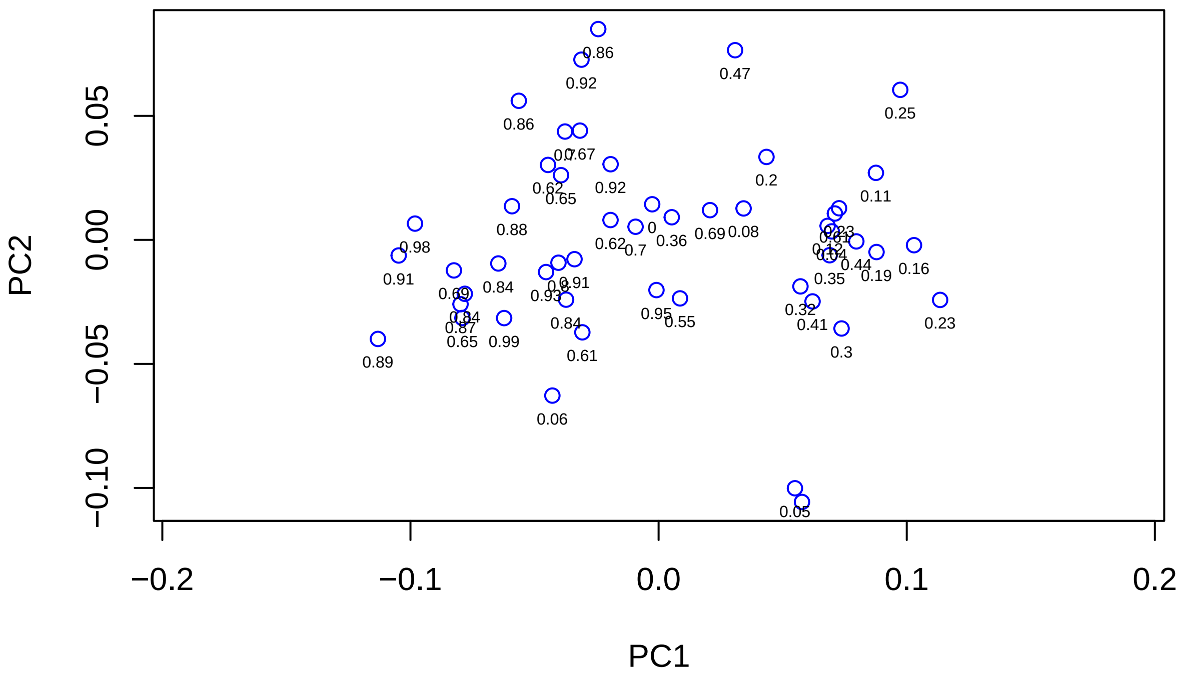
```
# Annuli of varying inner radii
num.repeats <- 50
max.t <- 1
t.vals <- seq(min.t,max.t,(max.t-min.t)/t.steps)

PD.list.3 <- vector("list",num.repeats)
inner.radius <- runif(num.repeats)
for (c in 1 : num.repeats){
  X <- sample.annulus(num.pts,inner.radius[c],outer.radius)
  PH.output <- alphaComplexDiag(X)
  PD.list.3[[c]] <- PH.output[["diagram"]]
  PD.list.3[[c]][,2:3] <- sqrt(PD.list.3[[c]][,2:3])
}
DV.matrix.3 <- death.vector.matrix(PD.list.3)
PL.list.3 <- vector("list",num.repeats)
for (c in 1 : num.repeats)
  PL.list.3[[c]] <- t(landscape(PD.list.3[[c]],dimension=1,KK=1:100,tseq=t.vals))
PL.matrix.3 <- landscape.matrix.from.list(PL.list.3)
```

The following is only to help us understand our data. We use PCA to give a 2d projection of the high-dimensional data together with their corresponding real values.

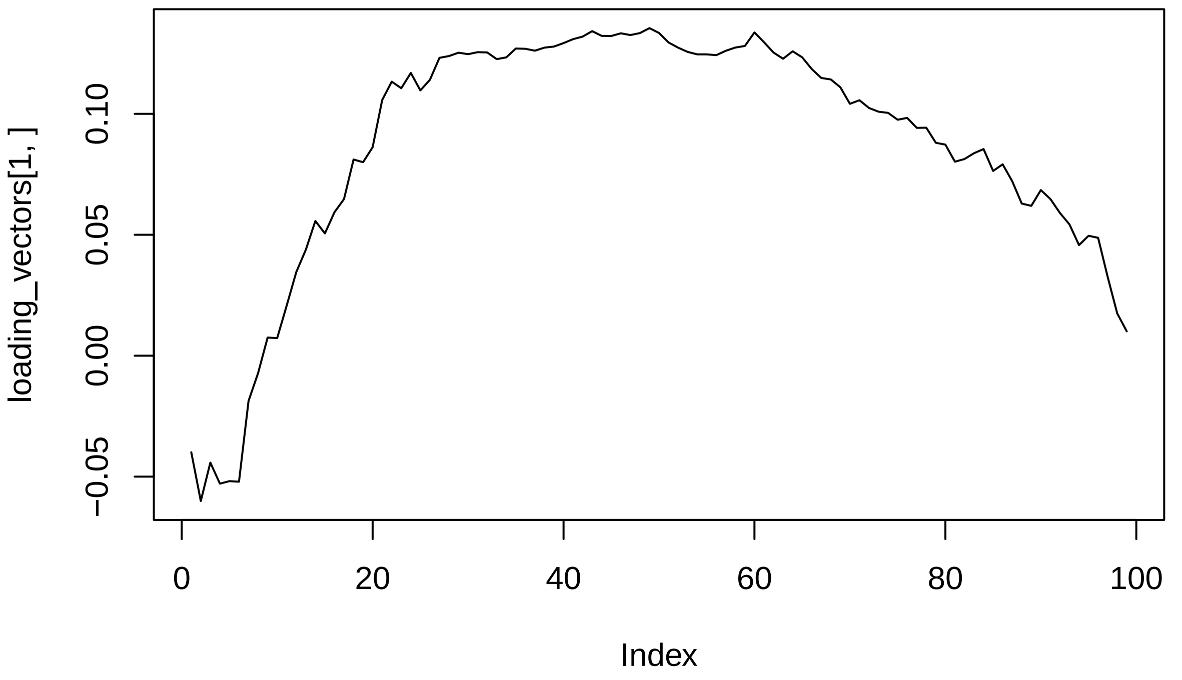
```
# Principal Component Analysis
pca.0 <- prcomp(DV.matrix.3,rank=10)
plot(pca.0,type="l")
```



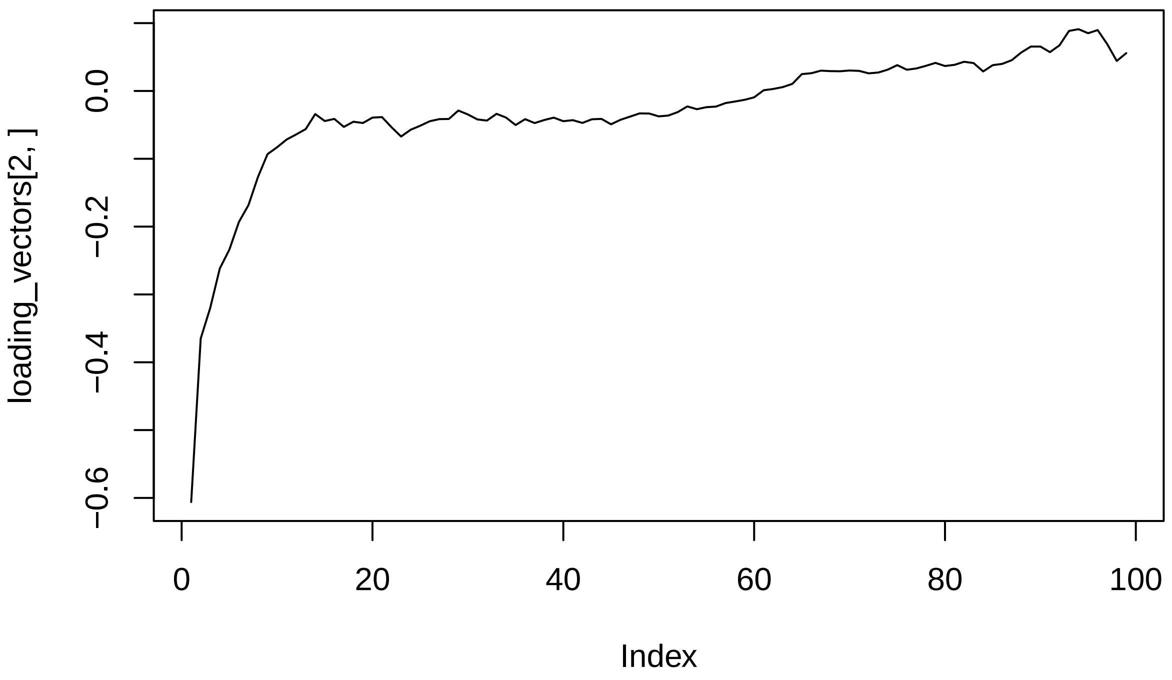
```
plot(pca.0$x[,1:2], col="blue", asp=1)
text(pca.0$x[,1:2], labels=trunc(inner.radius*100)/100, cex=.5, pos=1)
```



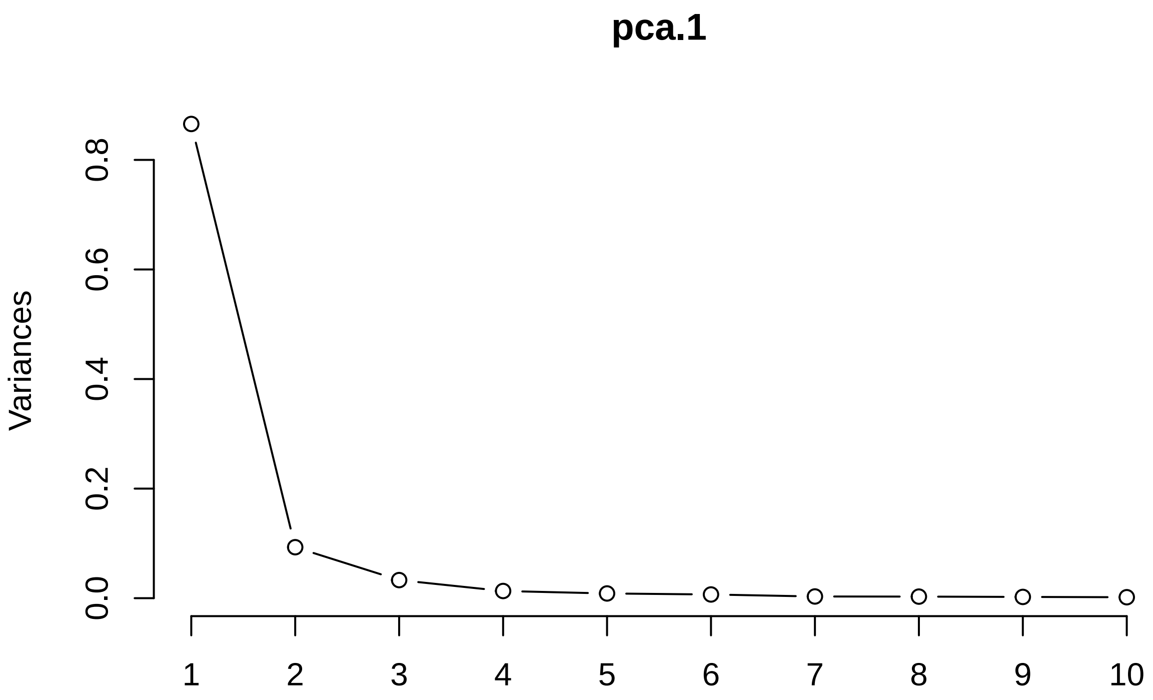
```
loading_vectors <- t(pca.0$rotation)
plot(loading_vectors[1,],type="l")
```



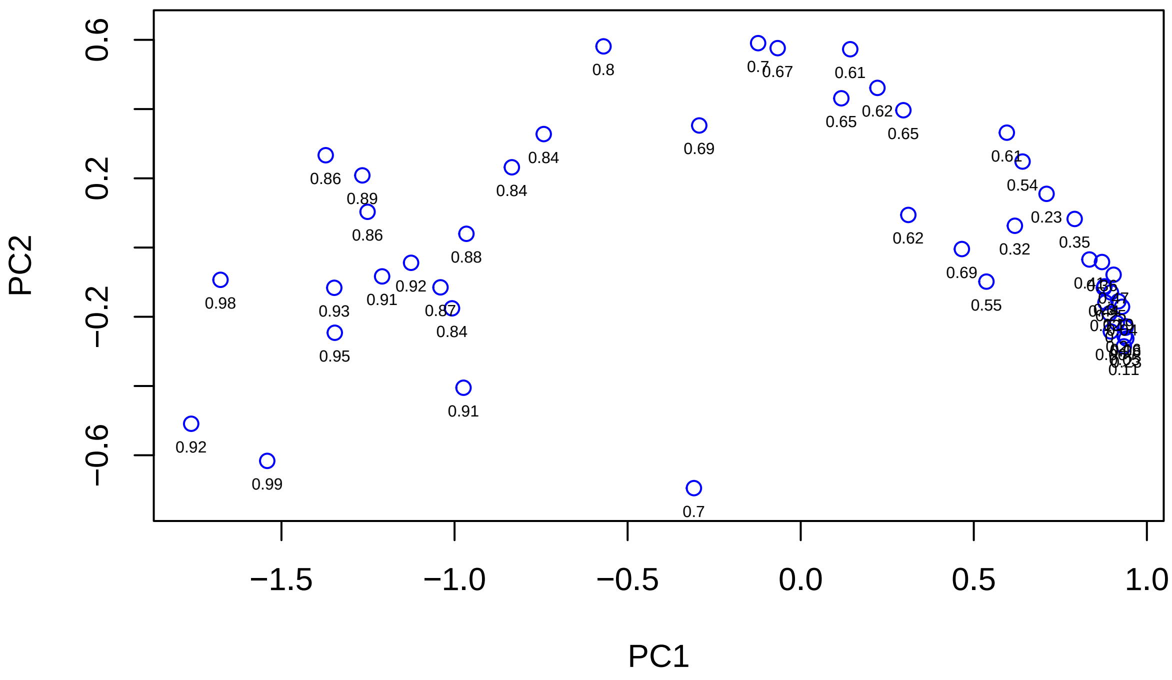
```
plot(loading_vectors[2,],type="l")
```



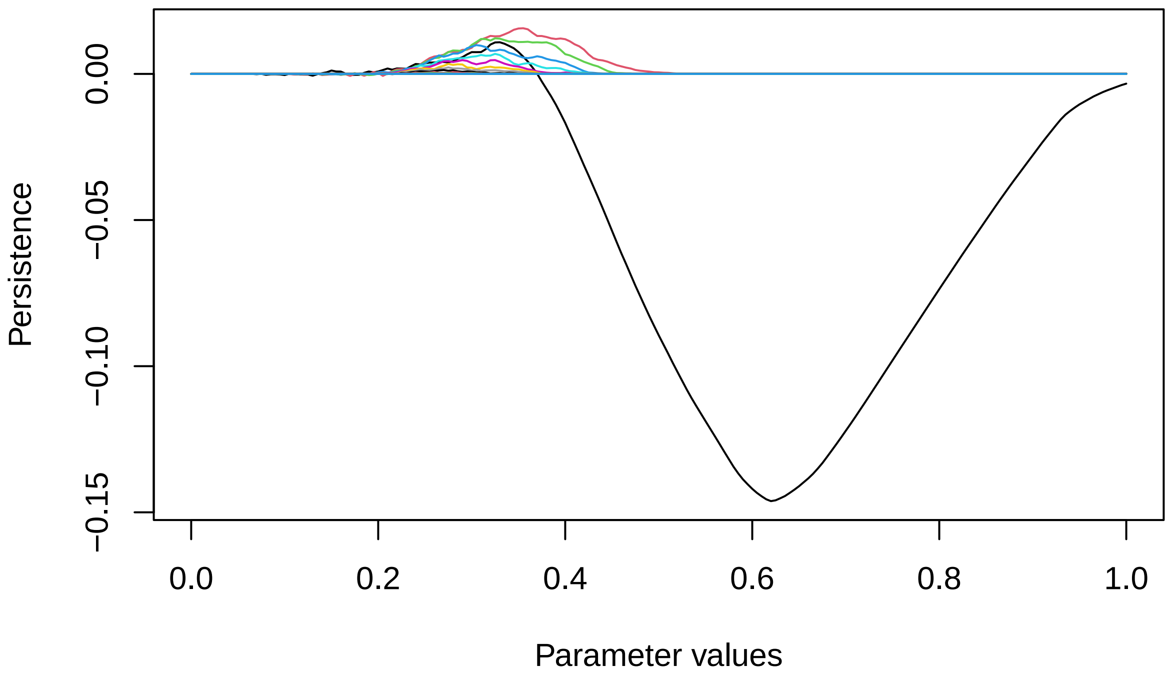
```
pca.1 <- prcomp(PL.matrix.3,rank=10)
plot(pca.1,type="l")
```



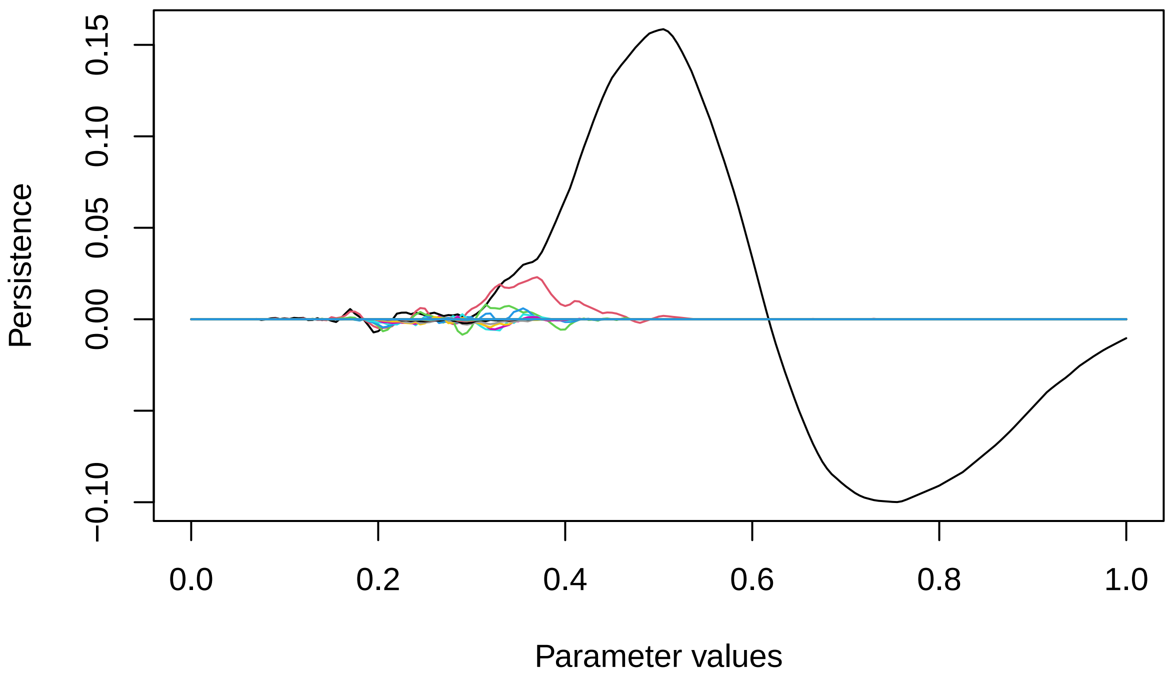
```
plot(pca.1$x[,1:2], col="blue", asp=1)
text(pca.1$x[,1:2], labels=trunc(inner.radius*100)/100, cex=.5, pos=1)
```



```
loading_vectors <- t(pca.1$rotation)
plot.landscape(landscape.from.vector(loading_vectors[1,],t.vals),t.vals)
```

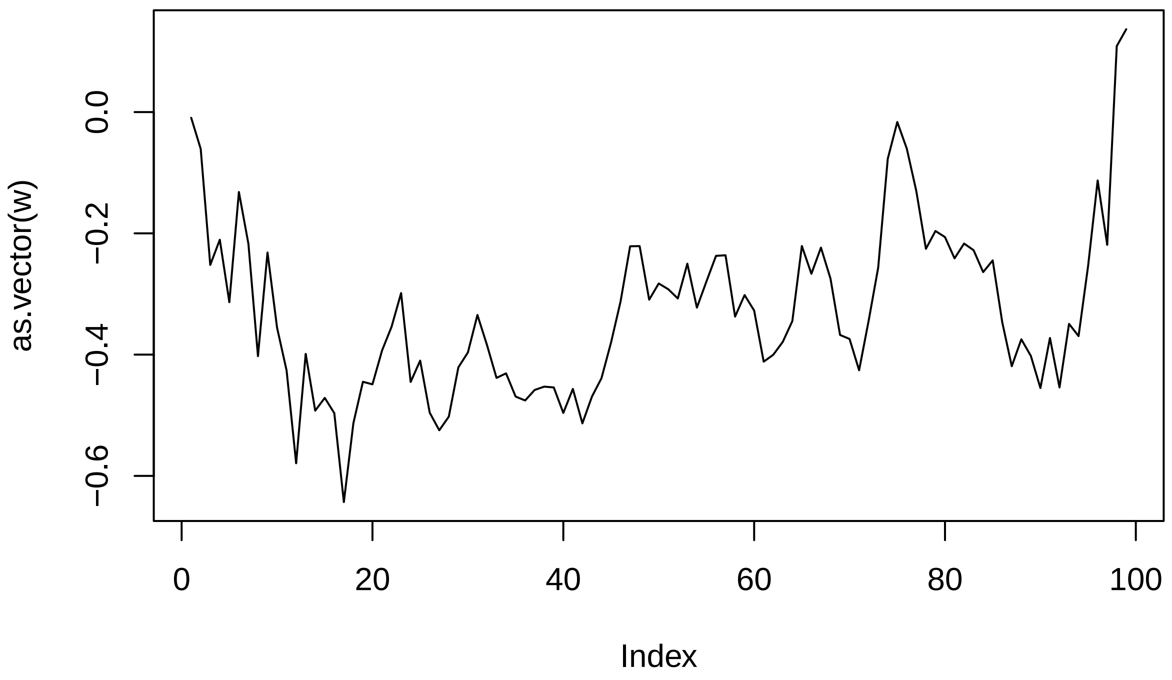


```
plot.landscape(landscape.from.vector(loading_vectors[2,],t.vals),t.vals)
```

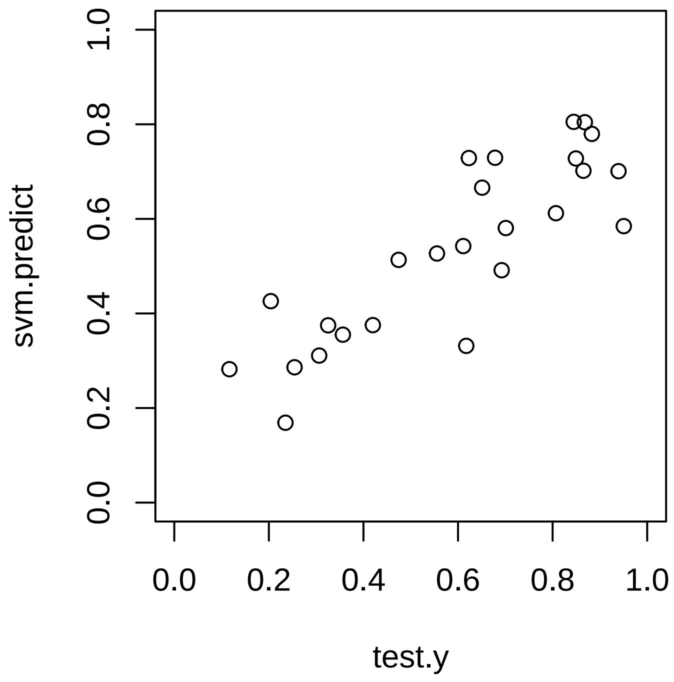


Now we apply SVM regression to our high-dimensional data.

```
# Support vector regression for persistent homology in degree 0 using death vectors

# Split data into training data and testing data
num.data <- length(inner.radius)
test.index <- sample(num.data, trunc(num.data/2))
test.x   <- DV.matrix.3[test.index,]
test.y   <- inner.radius[test.index]
train.x   <- DV.matrix.3[-test.index,]
train.y   <- inner.radius[-test.index]
cost <- 10
num.folds <- 5

# Build the Support Vector Machine Regression using the training data
svm.model <- svm(x=train.x,y=train.y,scale=FALSE,type="eps-regression",kernel="linear",cost=cost)

# Obtain the equation of the Separating Hyperplane
w <- t(svm.model$coefs) %*% svm.model$SV
b <- -svm.model$rho
plot(as.vector(w),type="l")
```

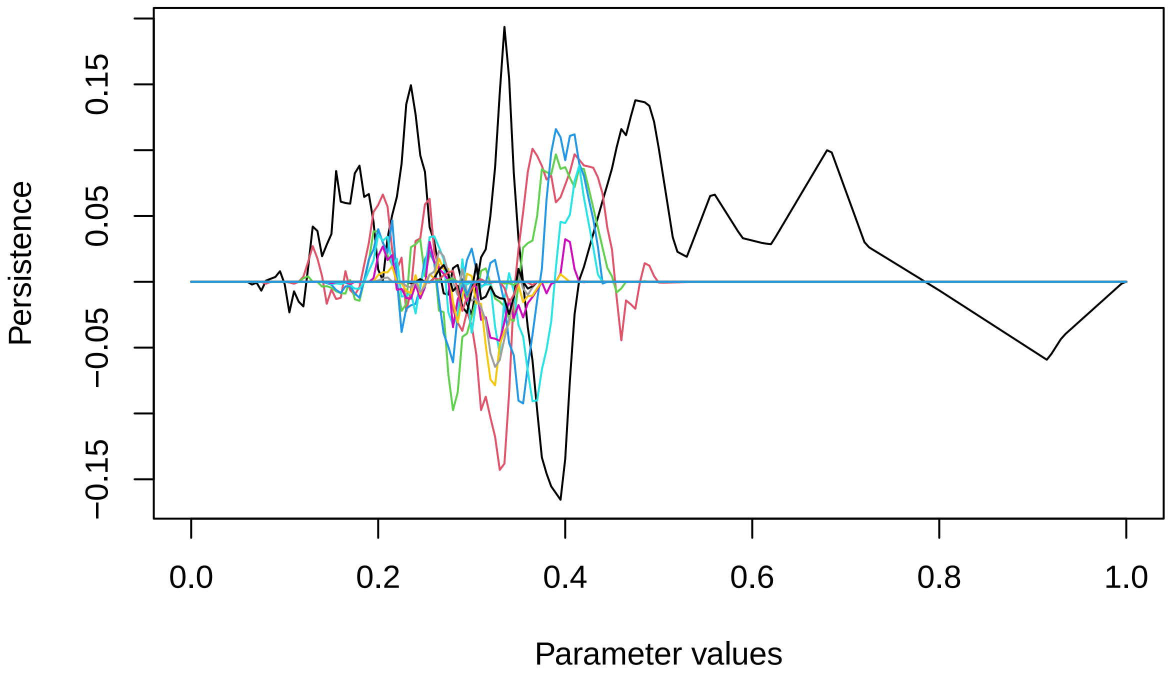


```
# Use the SVM Regression to predict the classes of the test data
svm.predict <- predict(svm.model,test.x)

## Plot actual value vs SVM Regression predicted value
par(pty="s") # force the plotting region to be square
plot(test.y,svm.predict,xlim=c(0,1),ylim=c(0,1),asp=1)
```

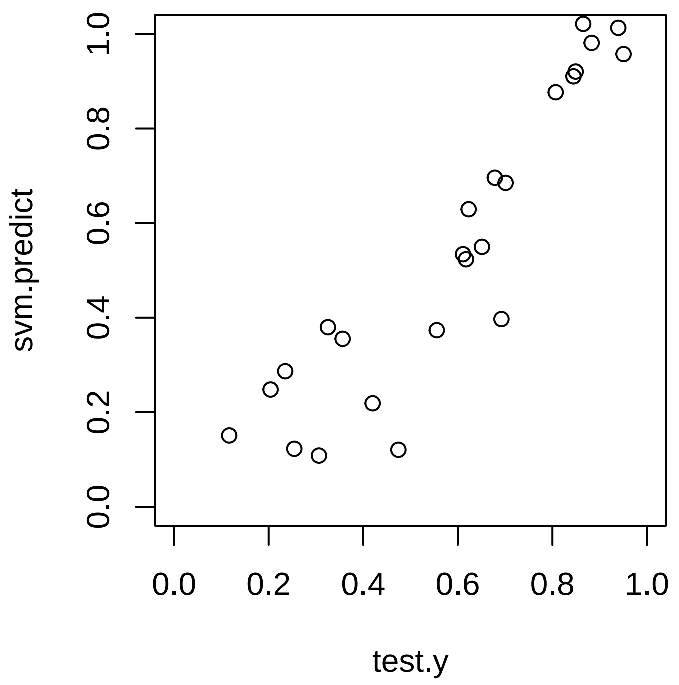


```
# Support vector regression for persistent homology in degree 1 using persistence landscapes

test.x   <- PL.matrix.3[test.index,]
train.x   <- PL.matrix.3[-test.index,]
```

```
svm.model <- svm(x=train.x,y=train.y,scale=FALSE,type="eps-regression",kernel="linear",cost=cost)

# Obtain the equation of the Separating Hyperplane
w <- t(svm.model$coefs) %*% svm.model$SV
b <- -svm.model$rho

# plot the normal vector to the hyperplane
plot.landscape(landscape.from.vector(as.matrix(w),t.vals),t.vals)
```



```
svm.predict <- predict(svm.model,test.x)
par(pty="s") # force the plotting region to be square
plot(test.y,svm.predict,xlim=c(0,1),ylim=c(0,1),asp=1)
```



```
# Get Mean Squared Error using Cross Validation
svm.model <- svm(DV.matrix.3,inner.radius,scale=FALSE,type="eps-regression",kernel="linear",cost=cost,c
summary(svm.model)
```

```
##
## Call:
## svm.default(x = DV.matrix.3, y = inner.radius, scale = FALSE, type = "eps-regression",
##     kernel = "linear", cost = cost, cross = num.folds)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.01010101
##     epsilon:  0.1
##
##
## Number of Support Vectors:  27
##
##
##
## 5-fold cross-validation on training data:
##
## Total Mean Squared Error: 0.04322504
## Squared Correlation Coefficient: 0.5600073
## Mean Squared Errors:
##  0.02657531 0.08468378 0.03705669 0.01441354 0.05339587
```

```
svm.model <- svm(PL.matrix.3,inner.radius,scale=FALSE,type="eps-regression",kernel="linear",cost=cost,c
summary(svm.model)
```

```
##
```

```
## Call:
## svm.default(x = PL.matrix.3, y = inner.radius, scale = FALSE, type = "eps-regression",
##     kernel = "linear", cost = cost, cross = num.folds)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  4.975124e-05
##     epsilon:  0.1
##
##
## Number of Support Vectors:  19
##
##
##
## 5-fold cross-validation on training data:
##
## Total Mean Squared Error: 0.02183437
## Squared Correlation Coefficient: 0.7775447
## Mean Squared Errors:
##  0.02456173 0.01971345 0.01089599 0.02525705 0.02874362
```

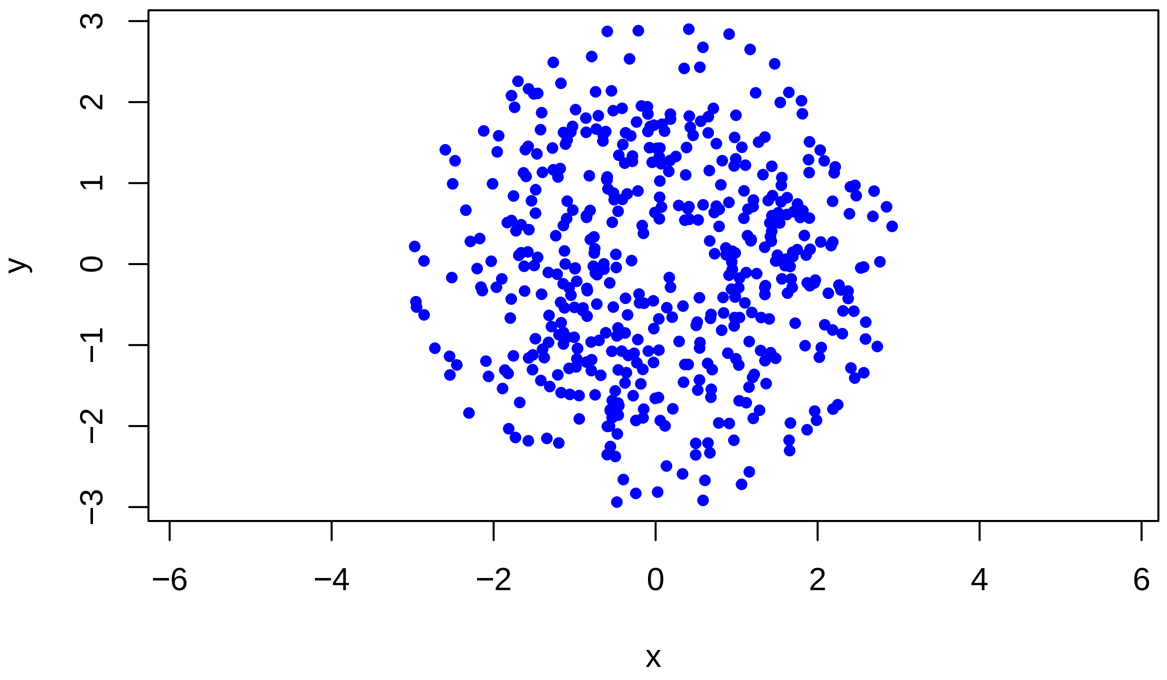
Exercises (Part II)

4. Discuss 5-fold cross validation for SVM-Classification accuracy
5. Explain the section of code labeled "Annuli of varying inner radii"
6. Discuss the 2d PCA plots for the death vector and persistence landscape for the annuli of varying inner radii
7. Discuss the first loading vector for PCA of the Persistence Landscape for the annuli of varying inner radii
8. Support Vector Machine Regression:

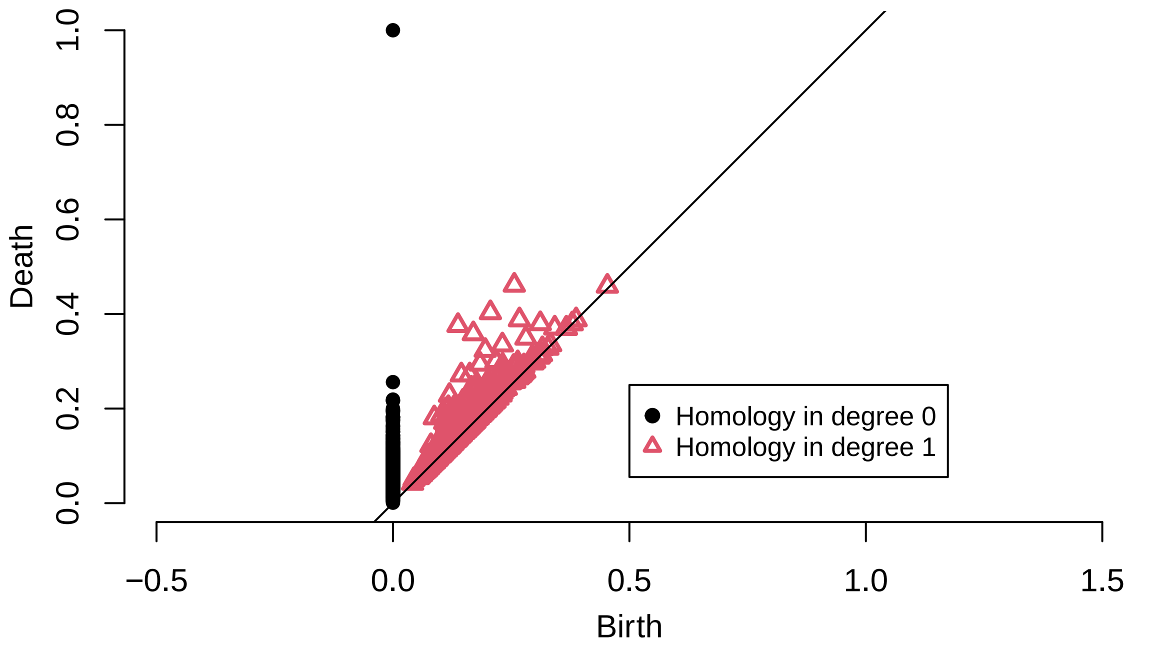
(a) Discuss splitting the data into training data and testing data
(b) Discuss the creation of the SVM regression model
(c) Discuss the variables w and b
(d) Discuss the SVM regression model prediction
(e) Discuss the mean squared errors

Cleaning noisy data using kNN

```r
# Sample Points with Noise
inner.radius <- 0.5
num.pts <- 250
num.pts.noisy <- 250
X <- sample.annulus(num.pts,inner.radius,outer.radius)
Y <- sample.annulus(num.pts.noisy,inner.radius=0,outer.radius=1.5*outer.radius)
X <- rbind(X,Y)
plot(X, pch=20, col="blue", asp=1)
```

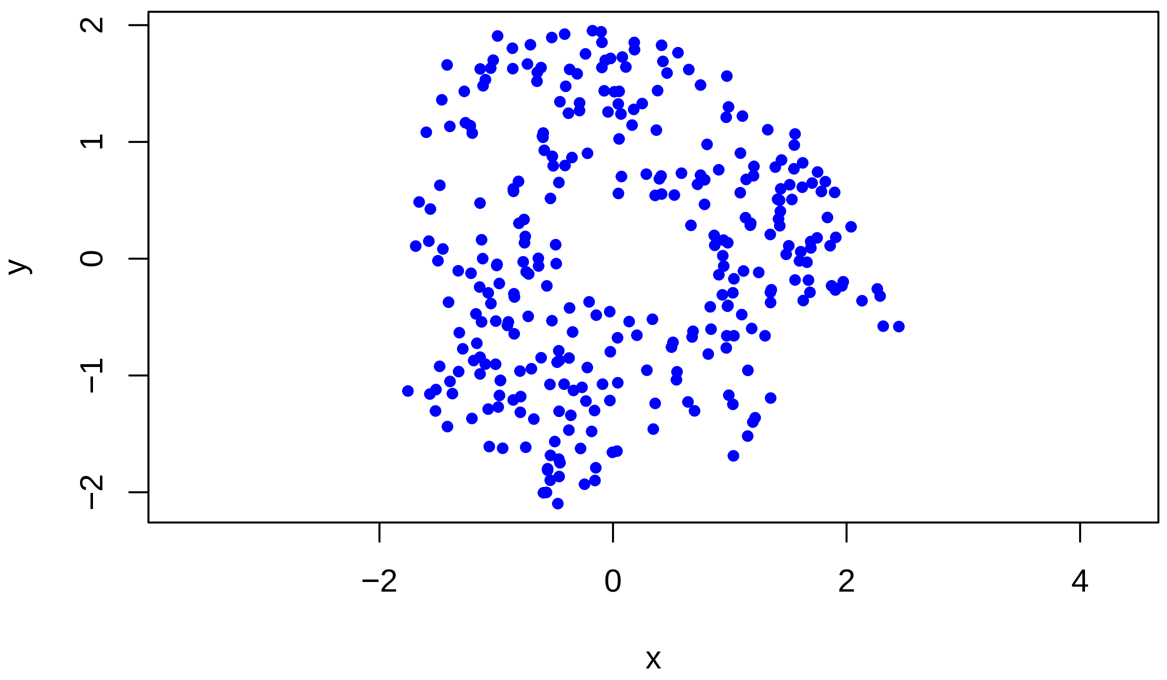


```r
PH.output <- alphaComplexDiag(X)
PD <- PH.output[["diagram"]]
PD[,2:3] <- sqrt(PD[,2:3])
plot(PD, asp=1, diagLim = c(0,max.t))
legend(0.5*max.t, 0.25*max.t, c("Homology in degree 0","Homology in degree 1"),
       col = c(1,2), pch = c(19,2), cex = .8, pt.lwd = 2)
```

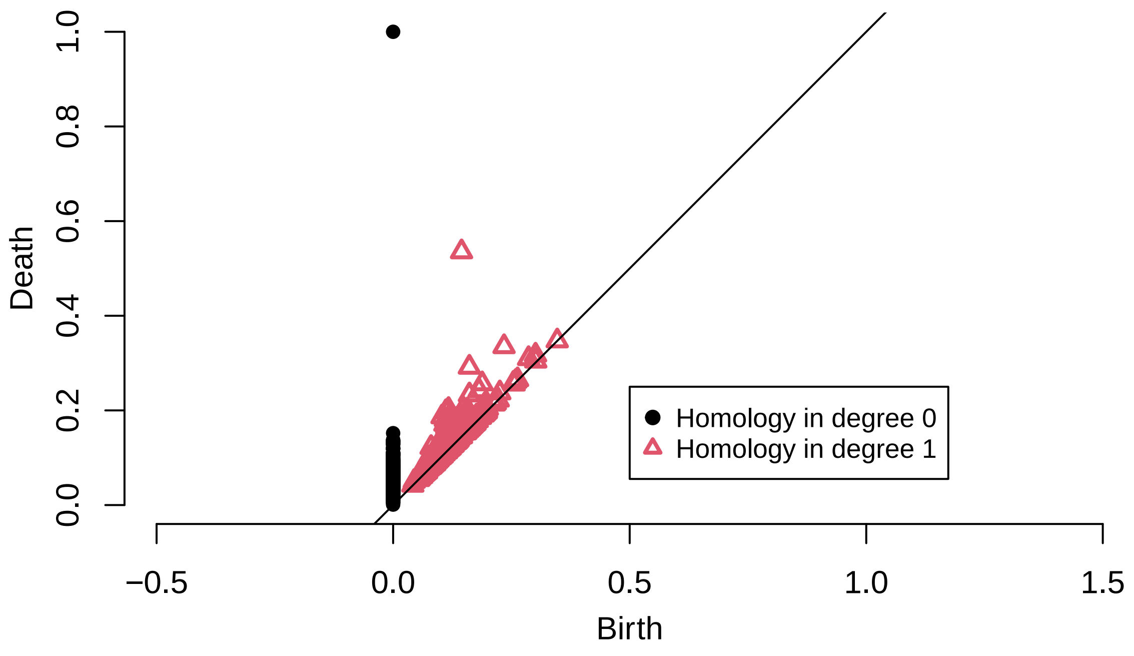


```r
# Clean noise using kNN
k <- 10
Threshold <- 0.4
dist.to.knn <- knn.dist(X)[,k]
X.cleaned <- X[dist.to.knn < Threshold,]
```

```
plot(X.cleaned, pch=20, col="blue", asp=1)
```



```
PH.output <- alphaComplexDiag(X.cleaned)
PD <- PH.output[["diagram"]]
PD[,2:3] <- sqrt(PD[,2:3])
plot(PD, asp=1, diagLim = c(0,max.t))
legend(0.5*max.t, 0.25*max.t, c("Homology in degree 0","Homology in degree 1"),
       col = c(1,2), pch = c(19,2), cex = .8, pt.lwd = 2)
```



Exercises (Part III)

9. Noise and cleaning:

(a) Discuss the code for adding noise to the sample
(b) Discuss the code for cleaning the noise
(c) Compare the persistence diagrams for the noisy sample and cleaned sample

10. Can you do better job cleaning this data?