**CSC 791, Fall 2014 - Solution to Project 2 (Kernelization, Kernel Tricks)**

# install.packages('ggplot2') install.packages('plyr') # install.packages('kernlab') install.packages('caret') library(ggplot2) library(plyr) library(kernlab) library(caret)

## Loading required package: cluster ## Loading required package: foreach ## Loading required package: lattice ## Loading required package: reshape2

**Exercise 2: Generating the data sets.**

Write a script (in R, Matlab, or SAS) that generates three data sets in a 2-dimensional space, defined as follows (see examples in <http://cran.r-project.org/web/packages/kernlab/vignettes/kernlab.pdf):>

1. BAD\_kmeans: The data set for which the kmeans clustering algorithm will not perform well.
2. BAD\_pca: The data set for which the Principal Component Analysis (PCA) dimension reduction method upon projection of the original points into 1-dimensional space (i.e., the first eigenvector) will not perform well.
3. BAD\_svm: The data set for which the linear Support Vector Machine (SVM) supervised classification method using two classes of points (positive and negative) will not perform well.
4. Plot each data set in a 2-dimensional space.

**Part 1**

set.seed(42) N = 200 bad\_kmeans <- rbind(data.frame(X = rnorm(N/4, 1, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(1, times = N/4))), data.frame(X = rnorm(N/4, 0, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(1, times = N/4))), data.frame(X = rnorm(N/4, 0.5, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(2, times = N/4))), data.frame(X = rnorm(N/4, 1.5, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(2, times = N/4)))) ggplot(bad\_kmeans) + aes(x = X, y = Y, color = class) + geom\_point() + ggtitle("Bad k-means")

**Part 2**

bad\_pca <- rbind(data.frame(X = (1:(N/2))/(N/2), Y = (1:(N/2)/(N/2) - 0.5 + rnorm(N/2, sd = 0.05))^2, class = factor(rep(1, times = N/2))), data.frame(X = rnorm(N/2, 0.5, 0.1), Y = rnorm(N/2, 0.2, 0.1), class = factor(rep(2, times = N/2)))) ggplot(bad\_pca) + aes(x = X, y = Y, color = class) + geom\_point() + ggtitle("Bad PCA")

**Part 3**

bad\_svm <- rbind(data.frame(X = rnorm(N/4, 1, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(1, times = N/4))), data.frame(X = rnorm(N/4, 0, 0.1), Y = rnorm(N/4, 0, 0.1), class = factor(rep(1, times = N/4))), data.frame(X = rnorm(N/2, 0.5, 0.1), Y = rnorm(N/2, 0, 0.1), class = factor(rep(2, times = N/2)))) ggplot(bad\_svm) + aes(x = X, y = Y, color = class) + geom\_point() + ggtitle("Bad SVM")

**Exercise 3: Evaluating the “badness” of the data mining methods.**

Write a script that uses the BAD data set in Exercise 2, runs the corresponding data mining method, produces the output from the method, and evaluates how bad the performance of this method is. You may use various performance metrics to assess each method (e.g., the variance, precision, recall, F1 measure). Not all the metrics could equally apply to each of the technique. Reading the Performance Metrics chapter by Kanchana and John from the Practical Graph Mining with R book is strongly encouraged for performing this exercise. Also, the book web-site provides the R scripts to play with these metrics, if interested. Report the summary of the performance metrics used and the performance results obtained.

If you are an R user, here is an R script that could give you an idea of how to perform the projection of the data using principal components:

data (iris); iris; X = iris [ , 1:4]; pca = princomp (X, center=TRUE); pca; plot (pca); # screeplot loadings(pca); # matrix of eigenvectors summary (pca); # check proportion of variance P=pca$scores; # projection of X onto eigenvectors plot (P[ ,1], P[ ,2]); points (P [1:50, 1], P[1:50,2], col="red"); points (P [51:100, 1], P[51:100,2], col="blue");

# function from 'Performance Metrics' Chapter twoCrossConfusionMatrixMetrics <- function(mat) { if ((nrow(mat) != 2) & (ncol(mat) != 2)) { print("Not a 2x2 Matrix") } else { f11 = mat[1, 1] f10 = mat[1, 2] f01 = mat[2, 1] f00 = mat[2, 2] T = sum(mat) AandB = colSums(mat) CandD = rowSums(mat) # List of Performance Metrics a = (f11 + f00)/T ER = (f10 + f01)/T TPR = f11/CandD[1] TNR = f00/CandD[2] FPR = f01/CandD[2] FNR = f10/CandD[1] PrecisionPositive = f11/AandB[1] PrecisionNegative = f00/AandB[2] FMeasurePositive = (2 \* PrecisionPositive \* TPR)/(PrecisionPositive + TPR) FMeasureNegative = (2 \* PrecisionNegative \* TNR)/(PrecisionNegative + TNR) GMean = sqrt(TPR \* TNR) return(c(Accuray = a, `Error Rate` = ER, `True Positive Rate` = TPR, `True Negative Rate` = TNR, `False Positive Rate` = FPR, `False Negative Rate` = FNR, `Precision+` = PrecisionPositive, `Precision-` = PrecisionNegative, `F-measure+` = FMeasurePositive, `F-measure-` = FMeasureNegative, `G-Mean` = GMean)) } }

# Note: Calculation for communities, not just classification!! ClusteringConfusionMatrix = function(mat1, mat2) { n = length(mat1) cm = matrix(0, nrow = 2, ncol = 2) for (i in 1:n) { for (j in 1:n) { if (mat1[i] == mat1[j] && mat2[i] == mat2[j]) cm[1, 1] = cm[1, 1] + 1 else if (mat1[i] != mat1[j] && mat2[i] == mat2[j]) cm[1, 2] = cm[1, 2] + 1 else if (mat1[i] == mat1[j] && mat2[i] != mat2[j]) cm[2, 1] = cm[2, 1] + 1 else if (mat1[i] != mat1[j] && mat2[i] != mat2[j]) cm[2, 2] = cm[2, 2] + 1 } } return(cm) # confusion matrix }

**k-Means**

cluster <- kmeans(x = as.matrix(bad\_kmeans[, c("X", "Y")]), centers = 2) bad\_kmeans$class\_kmeans <- factor(cluster$cluster) ggplot(bad\_kmeans) + aes(x = X, y = Y, color = class\_kmeans, shape = class) + geom\_point(size = 3) + ggtitle("Vanilla k-Means")

kmeans\_results <- data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) kmeans\_results["Vanilla k-means"] <- twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans$class, bad\_kmeans$class\_kmeans)) kmeans\_results

## Label Vanilla k-means ## 1 Accuray 0.5000 ## 2 Error Rate 0.5000 ## 3 True Positive Rate 0.5000 ## 4 True Negative Rate 0.5001 ## 5 False Positive Rate 0.4999 ## 6 False Negative Rate 0.5000 ## 7 Precision+ 0.5001 ## 8 Precision- 0.5000 ## 9 F-measure+ 0.5001 ## 10 F-measure- 0.5000 ## 11 G-Mean 0.5001

**SVM**

# Version 1: EXAMPLE OF HOW \*NOT\* TO PERFORM SVM - no use of # crossvalidation, uses all data for training and all data for testing # instead of splitting the data! mat <- as.matrix(bad\_svm[, c("X", "Y")]) s <- ksvm(mat, bad\_svm$class, type = "C-svc", kernel = "vanilladot") # train the model with all the data

## Setting default kernel parameters

bad\_svm$class\_svm = predict(s, mat) # test the model with all the data ggplot(bad\_svm) + aes(x = X, y = Y, color = class\_svm, shape = class) + geom\_point(size = 3) + ggtitle("Linear SVM (incorrectly done)")

cm <- table(bad\_svm$class, bad\_svm$class\_svm) # confusion matrix svm\_results\_incorrect = data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) svm\_results\_incorrect["Linear SVM (incorrectly done)"] = twoCrossConfusionMatrixMetrics(cm) svm\_results\_incorrect

## Label Linear SVM (incorrectly done) ## 1 Accuray 0.6500 ## 2 Error Rate 0.3500 ## 3 True Positive Rate 0.4800 ## 4 True Negative Rate 0.8200 ## 5 False Positive Rate 0.1800 ## 6 False Negative Rate 0.5200 ## 7 Precision+ 0.7273 ## 8 Precision- 0.6119 ## 9 F-measure+ 0.5783 ## 10 F-measure- 0.7009 ## 11 G-Mean 0.6274

# Version 2: CORRECT 10-fold crossvalidation for SVM nfolds <- 10 sizeN <- nrow(bad\_svm) # N x <- sample(1:sizeN, size = sizeN, replace = FALSE) folds <- createFolds(x, k = nfolds) # creates random folds of the data kksvm\_results\_pred\_all <- matrix(0, nrow = sizeN, ncol = 1) # will contain the predictions for all (test) data sv\_per\_fold <- data.frame(Label = c("No. support vectors in %")) # number of support vectors in percent svm\_results\_per\_fold <- data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) cm\_per\_fold <- data.frame(Label = c("True Positive", "False Positive", "False Negative", "True Negative")) for (i in 1:nfolds) { train <- bad\_svm[-folds[[i]], ] # training data test <- bad\_svm[folds[[i]], ] # test data kksvm\_results <- ksvm(train$class ~ ., data = train, type = "C-svc", kernel = "vanilladot") # train the model with the training data only kksvm\_results\_pred <- predict(kksvm\_results, test) # get the model prediction for this test data only kksvm\_results\_pred\_all[folds[[i]]] <- kksvm\_results\_pred # save the model prediction for each test data cm <- table(test$class, kksvm\_results\_pred) # confusion matrix for this fold cm\_per\_fold[paste("fold", i)] <- as.vector(cm) # confusion matrix for each fold, as vector svm\_results\_per\_fold[paste("fold", i)] <- twoCrossConfusionMatrixMetrics(cm) sv\_per\_fold[paste("fold", i)] <- 100 \* nSV(kksvm\_results)/nrow(train) }

## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters ## Setting default kernel parameters

ggplot(test) + aes(x = X, y = Y, color = kksvm\_results\_pred, shape = test$class) + geom\_point(size = 3) + ggtitle("Linear SVM (test set of just one fold)")

rbind(svm\_results\_per\_fold, sv\_per\_fold) # linear SVM results per fold (for the test data only)

## Label fold 1 fold 2 fold 3 fold 4 fold 5 ## 1 Accuray 0.90 0.8000 0.9500 0.90000 0.8000 ## 2 Error Rate 0.10 0.2000 0.0500 0.10000 0.2000 ## 3 True Positive Rate 0.90 1.0000 1.0000 0.91667 0.8889 ## 4 True Negative Rate 0.90 0.6923 0.8571 0.87500 0.7273 ## 5 False Positive Rate 0.10 0.3077 0.1429 0.12500 0.2727 ## 6 False Negative Rate 0.10 0.0000 0.0000 0.08333 0.1111 ## 7 Precision+ 0.90 0.6364 0.9286 0.91667 0.7273 ## 8 Precision- 0.90 1.0000 1.0000 0.87500 0.8889 ## 9 F-measure+ 0.90 0.7778 0.9630 0.91667 0.8000 ## 10 F-measure- 0.90 0.8182 0.9231 0.87500 0.8000 ## 11 G-Mean 0.90 0.8321 0.9258 0.89559 0.8040 ## 12 No. support vectors in % 64.44 58.8889 65.5556 62.77778 58.8889 ## fold 6 fold 7 fold 8 fold 9 fold 10 ## 1 0.8500 0.9000 0.9500 0.9000 0.95000 ## 2 0.1500 0.1000 0.0500 0.1000 0.05000 ## 3 1.0000 1.0000 1.0000 1.0000 1.00000 ## 4 0.7000 0.7778 0.9000 0.7500 0.92857 ## 5 0.3000 0.2222 0.1000 0.2500 0.07143 ## 6 0.0000 0.0000 0.0000 0.0000 0.00000 ## 7 0.7692 0.8462 0.9091 0.8571 0.85714 ## 8 1.0000 1.0000 1.0000 1.0000 1.00000 ## 9 0.8696 0.9167 0.9524 0.9231 0.92308 ## 10 0.8235 0.8750 0.9474 0.8571 0.96296 ## 11 0.8367 0.8819 0.9487 0.8660 0.96362 ## 12 61.6667 62.7778 65.0000 62.7778 65.55556

sv\_avg = data.frame(Label = c("Avg. % suppport vectors"), Linear\_SVM = rowMeans(sv\_per\_fold[c(2:(1 + nfolds))])) svm\_avg\_results = data.frame(Label = c("Average Accuray", "Avg. Error Rate", "Avg. True Positive Rate", "Avg. True Negative Rate", "Avg. False Positive Rate", "Avg. False Negative Rate", "Avg. Precision+", "Avg. Precision-", "Avg. F-measure+", "Avg. F-measure-", "Avg. G-Mean")) svm\_avg\_results["Linear\_SVM"] = twoCrossConfusionMatrixMetrics(table(bad\_svm$class, kksvm\_results\_pred\_all)) svm\_avg\_results\_incl\_sv <- rbind(svm\_avg\_results, sv\_avg) svm\_avg\_results\_incl\_sv

## Label Linear\_SVM ## 1 Average Accuray 0.8900 ## 2 Avg. Error Rate 0.1100 ## 3 Avg. True Positive Rate 0.9700 ## 4 Avg. True Negative Rate 0.8100 ## 5 Avg. False Positive Rate 0.1900 ## 6 Avg. False Negative Rate 0.0300 ## 7 Avg. Precision+ 0.8362 ## 8 Avg. Precision- 0.9643 ## 9 Avg. F-measure+ 0.8981 ## 10 Avg. F-measure- 0.8804 ## 11 Avg. G-Mean 0.8864 ## 12 Avg. % suppport vectors 62.8333

The number of support vectors is over 50 percent, we might be overfitting the data.

**PCA**

pca = princomp(bad\_pca[, c("X", "Y")], center = TRUE) plot(pca)

loadings(pca) # matrix of eigenvectors

## ## Loadings: ## Comp.1 Comp.2 ## X -0.999 ## Y -0.999 ## ## Comp.1 Comp.2 ## SS loadings 1.0 1.0 ## Proportion Var 0.5 0.5 ## Cumulative Var 0.5 1.0

summary(pca) # check proportion of variance

## Importance of components: ## Comp.1 Comp.2 ## Standard deviation 0.2175 0.1032 ## Proportion of Variance 0.8163 0.1837 ## Cumulative Proportion 0.8163 1.0000

P = pca$scores # projection of bad\_pca onto eigenvectors bad\_pca$first\_component = P[, 1] ggplot(bad\_pca) + aes(x = X, y = Y, color = first\_component, shape = class) + geom\_point(size = 6) + ggtitle("Linear PCA")

**Exercise 4: Kernelizing the methods.**

Write a script that uses the kernalized version of each of the data mining method in Exercise 3 (e.g., you may consider using kernlab and kkmeans packages in R for kernel SVM+PCA and kmeans, resp.).

1. Choose at least two kernels for each of the methods.
2. Use the same performance metrics as in Ex. 3, and compare the performance obtained by the methods after applying the kernel trick versus the original un-kernelized versions of the techniques.
3. Do you observe the difference in performance when you use different kernels?
4. What are the best performance results do you get by playing with different kernels and kernel parameters? Also, make sure to report the number of support vectors for the SVM (the good rule of thumb is to strive for no more than 35%-50% support vectors to avoid model overfitting.

**k-Means**

mat = as.matrix(bad\_kmeans[, c("X", "Y")])

**Gaussian Kernel**

s <- kkmeans(mat, centers = 2, kernel = "rbfdot")

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

bad\_kmeans$class\_kmeans\_rbf = as.vector(s) ggplot(bad\_kmeans) + aes(x = X, y = Y, color = class\_kmeans\_rbf, shape = class) + geom\_point(size = 3) + ggtitle("k-Means with Gaussian Kernel")

Vanilla and Gaussian stats:

kmeans\_results["Gaussian k-means"] = twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans$class, bad\_kmeans$class\_kmeans\_rbf)) kmeans\_results

## Label Vanilla k-means Gaussian k-means ## 1 Accuray 0.5000 0.5004 ## 2 Error Rate 0.5000 0.4995 ## 3 True Positive Rate 0.5000 0.5004 ## 4 True Negative Rate 0.5001 0.5005 ## 5 False Positive Rate 0.4999 0.4995 ## 6 False Negative Rate 0.5000 0.4996 ## 7 Precision+ 0.5001 0.5009 ## 8 Precision- 0.5000 0.5000 ## 9 F-measure+ 0.5001 0.5007 ## 10 F-measure- 0.5000 0.5002 ## 11 G-Mean 0.5001 0.5005

Gaussian kernel performs around the same as k-means without a kernel (results are also in the area of random guessing).

**Polynomial Kernel**

s <- kkmeans(mat, centers = 2, kernel = "polydot", kpar = list(degree = 3)) bad\_kmeans$class\_kmeans\_poly = as.vector(s) ggplot(bad\_kmeans) + aes(x = X, y = Y, color = class\_kmeans\_poly, shape = class) + geom\_point(size = 3) + ggtitle("k-Means with Polynomial Kernel")

Vanilla and Polynomial stats:

kmeans\_results["Polynomial k-means"] = twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans$class, bad\_kmeans$class\_kmeans\_poly)) kmeans\_results

## Label Vanilla k-means Gaussian k-means Polynomial k-means ## 1 Accuray 0.5000 0.5004 0.6105 ## 2 Error Rate 0.5000 0.4995 0.3896 ## 3 True Positive Rate 0.5000 0.5004 0.5905 ## 4 True Negative Rate 0.5001 0.5005 0.6418 ## 5 False Positive Rate 0.4999 0.4995 0.3582 ## 6 False Negative Rate 0.5000 0.4996 0.4095 ## 7 Precision+ 0.5001 0.5009 0.7209 ## 8 Precision- 0.5000 0.5000 0.5000 ## 9 F-measure+ 0.5001 0.5007 0.6492 ## 10 F-measure- 0.5000 0.5002 0.5621 ## 11 G-Mean 0.5001 0.5005 0.6156

The polynomial kernel gives slightly better results than the two other k-means algorithms.

**SVM**

mat = as.matrix(bad\_svm[, c("X", "Y")])

**Gaussian Kernel**

nfolds <- 10 # 10-fold crossvalidation sizeN <- nrow(bad\_svm) # N x <- sample(1:sizeN, size = sizeN, replace = FALSE) folds <- createFolds(x, k = nfolds) # creates random folds of the data kksvm\_results\_pred\_all <- matrix(0, nrow = sizeN, ncol = 1) # will contain the predictions for all (test) data sv\_per\_fold <- data.frame(Label = c("No. support vectors in %")) # number of support vectors in percent svm\_results\_per\_fold <- data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) cm\_per\_fold <- data.frame(Label = c("True Positive", "False Positive", "False Negative", "True Negative")) for (i in 1:nfolds) { train <- bad\_svm[-folds[[i]], ] # training data test <- bad\_svm[folds[[i]], ] # test data kksvm\_results <- ksvm(train$class ~ ., data = train, type = "C-svc", kernel = "rbfdot", kpar = list(sigma = 1)) # train the model with the training data only kksvm\_results\_pred <- predict(kksvm\_results, test) # get the model prediction for this test data only kksvm\_results\_pred\_all[folds[[i]]] <- kksvm\_results\_pred # save the model prediction for each test data cm <- table(test$class, kksvm\_results\_pred) # confusion matrix for this fold cm\_per\_fold[paste("fold", i)] <- as.vector(cm) # confusion matrix for each fold, as vector svm\_results\_per\_fold[paste("fold", i)] <- twoCrossConfusionMatrixMetrics(cm) sv\_per\_fold[paste("fold", i)] <- 100 \* nSV(kksvm\_results)/nrow(train) } ggplot(test) + aes(x = X, y = Y, color = kksvm\_results\_pred, shape = test$class) + geom\_point(size = 3) + ggtitle("SVM with Gaussian Kernel (test set of just one fold)")

rbind(svm\_results\_per\_fold, sv\_per\_fold) # Gaussian SVM results per fold (for the test data only)

## Label fold 1 fold 2 fold 3 fold 4 fold 5 fold 6 ## 1 Accuray 0.90000 1 1.00 1.00 0.95000 0.9500 ## 2 Error Rate 0.10000 0 0.00 0.00 0.05000 0.0500 ## 3 True Positive Rate 0.90909 1 1.00 1.00 0.92308 0.9000 ## 4 True Negative Rate 0.88889 1 1.00 1.00 1.00000 1.0000 ## 5 False Positive Rate 0.11111 0 0.00 0.00 0.00000 0.0000 ## 6 False Negative Rate 0.09091 0 0.00 0.00 0.07692 0.1000 ## 7 Precision+ 0.90909 1 1.00 1.00 1.00000 1.0000 ## 8 Precision- 0.88889 1 1.00 1.00 0.87500 0.9091 ## 9 F-measure+ 0.90909 1 1.00 1.00 0.96000 0.9474 ## 10 F-measure- 0.88889 1 1.00 1.00 0.93333 0.9524 ## 11 G-Mean 0.89893 1 1.00 1.00 0.96077 0.9487 ## 12 No. support vectors in % 23.88889 25 23.89 23.89 24.44444 23.8889 ## fold 7 fold 8 fold 9 fold 10 ## 1 1.00 1.00 1.00 0.9500 ## 2 0.00 0.00 0.00 0.0500 ## 3 1.00 1.00 1.00 0.9000 ## 4 1.00 1.00 1.00 1.0000 ## 5 0.00 0.00 0.00 0.0000 ## 6 0.00 0.00 0.00 0.1000 ## 7 1.00 1.00 1.00 1.0000 ## 8 1.00 1.00 1.00 0.9091 ## 9 1.00 1.00 1.00 0.9474 ## 10 1.00 1.00 1.00 0.9524 ## 11 1.00 1.00 1.00 0.9487 ## 12 23.33 22.78 23.89 23.8889

sv\_avg["Gaussian\_SVM"] = rowMeans(sv\_per\_fold[c(2:(1 + nfolds))]) svm\_avg\_results["Gaussian\_SVM"] = twoCrossConfusionMatrixMetrics(table(bad\_svm$class, kksvm\_results\_pred\_all)) svm\_avg\_results\_incl\_sv <- rbind(svm\_avg\_results, sv\_avg)

Linear and Gaussian stats:

svm\_avg\_results\_incl\_sv # Gaussian SVM results per fold (for the test data only)

## Label Linear\_SVM Gaussian\_SVM ## 1 Average Accuray 0.8900 0.9750 ## 2 Avg. Error Rate 0.1100 0.0250 ## 3 Avg. True Positive Rate 0.9700 0.9600 ## 4 Avg. True Negative Rate 0.8100 0.9900 ## 5 Avg. False Positive Rate 0.1900 0.0100 ## 6 Avg. False Negative Rate 0.0300 0.0400 ## 7 Avg. Precision+ 0.8362 0.9897 ## 8 Avg. Precision- 0.9643 0.9612 ## 9 Avg. F-measure+ 0.8981 0.9746 ## 10 Avg. F-measure- 0.8804 0.9754 ## 11 Avg. G-Mean 0.8864 0.9749 ## 12 Avg. % suppport vectors 62.8333 23.8889

We see an enormous improvement when using the Gaussian kernel. The number of support vectors drops to less than half of what it was before and the SVM seems to not overfit the data.

**Hyperbolic Tangent Kernel**

nfolds <- 10 # 10-fold crossvalidation sizeN <- nrow(bad\_svm) # N x <- sample(1:sizeN, size = sizeN, replace = FALSE) folds <- createFolds(x, k = nfolds) # creates random folds of the data kksvm\_results\_pred\_all <- matrix(0, nrow = sizeN, ncol = 1) # will contain the predictions for all (test) data sv\_per\_fold <- data.frame(Label = c("No. support vectors in %")) # number of support vectors in percent svm\_results\_per\_fold <- data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) cm\_per\_fold <- data.frame(Label = c("True Positive", "False Positive", "False Negative", "True Negative")) for (i in 1:nfolds) { train <- bad\_svm[-folds[[i]], ] # training data test <- bad\_svm[folds[[i]], ] # test data kksvm\_results <- ksvm(train$class ~ ., data = train, type = "C-svc", kernel = "tanhdot", kpar = list(scale = 1, offset = 1)) # train the model with the training data only kksvm\_results\_pred <- predict(kksvm\_results, test) # get the model prediction for this test data only kksvm\_results\_pred\_all[folds[[i]]] <- kksvm\_results\_pred # save the model prediction for each test data cm <- table(test$class, kksvm\_results\_pred) # confusion matrix for this fold cm\_per\_fold[paste("fold", i)] <- as.vector(cm) # confusion matrix for each fold, as vector svm\_results\_per\_fold[paste("fold", i)] <- twoCrossConfusionMatrixMetrics(cm) sv\_per\_fold[paste("fold", i)] <- 100 \* nSV(kksvm\_results)/nrow(train) } ggplot(test) + aes(x = X, y = Y, color = kksvm\_results\_pred, shape = test$class) + geom\_point(size = 3) + ggtitle("SVM with hyperbolic tangent kernel (test set of just one fold)")

rbind(svm\_results\_per\_fold, sv\_per\_fold) # Hyperbolic Tangent SVM results per fold (for the test data only)

## Label fold 1 fold 2 fold 3 fold 4 fold 5 ## 1 Accuray 0.00 0.20000 0.10000 0.00 0.05000 ## 2 Error Rate 1.00 0.80000 0.90000 1.00 0.95000 ## 3 True Positive Rate 0.00 0.33333 0.07692 0.00 0.07692 ## 4 True Negative Rate 0.00 0.09091 0.14286 0.00 0.00000 ## 5 False Positive Rate 1.00 0.90909 0.85714 1.00 1.00000 ## 6 False Negative Rate 1.00 0.66667 0.92308 1.00 0.92308 ## 7 Precision+ 0.00 0.23077 0.14286 0.00 0.12500 ## 8 Precision- 0.00 0.14286 0.07692 0.00 0.00000 ## 9 F-measure+ NaN 0.27273 0.10000 NaN 0.09524 ## 10 F-measure- NaN 0.11111 0.10000 NaN NaN ## 11 G-Mean 0.00 0.17408 0.10483 0.00 0.00000 ## 12 No. support vectors in % 91.11 93.33333 93.33333 91.11 87.77778 ## fold 6 fold 7 fold 8 fold 9 fold 10 ## 1 0.10000 0.05000 0.2000 0.1500 0.1000 ## 2 0.90000 0.95000 0.8000 0.8500 0.9000 ## 3 0.16667 0.00000 0.2727 0.1111 0.0000 ## 4 0.07143 0.11111 0.1111 0.1818 0.2222 ## 5 0.92857 0.88889 0.8889 0.8182 0.7778 ## 6 0.83333 1.00000 0.7273 0.8889 1.0000 ## 7 0.07143 0.00000 0.2727 0.1000 0.0000 ## 8 0.16667 0.08333 0.1111 0.2000 0.1538 ## 9 0.10000 NaN 0.2727 0.1053 NaN ## 10 0.10000 0.09524 0.1111 0.1905 0.1818 ## 11 0.10911 0.00000 0.1741 0.1421 0.0000 ## 12 92.22222 87.77778 92.2222 91.1111 92.2222

sv\_avg["Hyperbolic\_tangent\_SVM"] = rowMeans(sv\_per\_fold[c(2:(1 + nfolds))]) svm\_avg\_results["Hyperbolic\_tangent\_SVM"] = twoCrossConfusionMatrixMetrics(table(bad\_svm$class, kksvm\_results\_pred\_all)) svm\_avg\_results\_incl\_sv <- rbind(svm\_avg\_results, sv\_avg)

Linear and Hyperbolic Tangent stats:

svm\_avg\_results\_incl\_sv # Hyperbolic Tangent SVM results per fold (for the test data only)

## Label Linear\_SVM Gaussian\_SVM Hyperbolic\_tangent\_SVM ## 1 Average Accuray 0.8900 0.9750 0.09500 ## 2 Avg. Error Rate 0.1100 0.0250 0.90500 ## 3 Avg. True Positive Rate 0.9700 0.9600 0.10000 ## 4 Avg. True Negative Rate 0.8100 0.9900 0.09000 ## 5 Avg. False Positive Rate 0.1900 0.0100 0.91000 ## 6 Avg. False Negative Rate 0.0300 0.0400 0.90000 ## 7 Avg. Precision+ 0.8362 0.9897 0.09901 ## 8 Avg. Precision- 0.9643 0.9612 0.09091 ## 9 Avg. F-measure+ 0.8981 0.9746 0.09950 ## 10 Avg. F-measure- 0.8804 0.9754 0.09045 ## 11 Avg. G-Mean 0.8864 0.9749 0.09487 ## 12 Avg. % suppport vectors 62.8333 23.8889 91.22222

We do not see an improvement at all. The Hyperbolic Tangent SVM performs poorly and the number of support vectors is remarkably high, strongly indicating that we are overfitting the training data.

**PCA**

mat = as.matrix(bad\_pca[, c("X", "Y")])

**Polynomial Kernel**

p <- kpca(mat, kernel = "polydot", kpar = list(scale = 1, offset = 0, degree = 3), features = 1) bad\_pca$first\_component\_poly <- pcv(p) cumsum(eig(p)/sum(eig(p)))

## Comp.1 ## 1

ggplot(bad\_pca) + aes(x = X, y = Y, color = first\_component\_poly, shape = class) + geom\_point(size = 6) + ggtitle("Third Degree Polynomial PCA")

**Gaussian Kernel**

p <- kpca(mat, kernel = "rbfdot", kpar = list(sigma = 0.01), features = 1) bad\_pca$first\_component\_rbf <- pcv(p) cumsum(eig(p)/sum(eig(p)))

## Comp.1 ## 1

ggplot(bad\_pca) + aes(x = X, y = Y, color = first\_component\_rbf, shape = class) + geom\_point(size = 6) + ggtitle("Gaussian PCA")

In both cases, the Kernalized PCA's first component isn't addequate for linear discrimination.

**Exercise 5: Pipelining.**

Dimension reduction is often used as the key data preprocessing step to other data mining techniques downstream of end-to-end data analysis. In this exercise we will use unsupervised kernel PCA as a preprocessing step to clustering. Later in the course, we will use supervised dimension reduction methods as a preprocessor to the supervised classification methods.

1. Generalize your BAD\_kmeans data set to very high-dimensional space (d>>2).
2. Show that the kmeans clustering method does not perform well on that data.
3. Apply the kernel PCA method to this high dimensional data and identify the number (m<<d) of principal components (i.e., eigenvectors) that provide a reasonably good low-dimensional approximation to your data (i.e., based on eigenvalue distribution). How much total variability of the data will be preserved upon using this low-dimensional representation?
4. Project your original data onto the top m eigenvectors corresponding the largest eigenvalues.
5. Run the kmeans clustering algorithm on the projected low dimensional data.
6. Compare the performance of the kmeans on d-dimensional original data vs. the m-dimensional projected data. Has the performance improved?
7. If you run the kernel kmeans clustering method on the original data, will get better/worse performance? Can you discuss the pros and cons of using kernel kmeans on the original data directly versus applying the kernel pca as the pre-processing step and then running the kmeans on the low-dimensional data.

**Part 1**

# function mat.gen mat.gen <- function(X) { as.matrix(rbind(data.frame(X = rnorm(N/2, 1, 0.1), Y = rnorm(N/2, 0, 0.1)), data.frame(X = rnorm(N/2, 0, 0.1), Y = rnorm(N/2, 0, 0.1)), data.frame(X = rnorm(N/2, 0.5, 0.1), Y = rnorm(N/2, 0, 0.1)), data.frame(X = rnorm(N/2, 1.5, 0.1), Y = rnorm(N/2, 0, 0.1)))) } bad\_kmeans\_hd <- do.call("cbind", lapply(1:50, mat.gen)) bad\_kmeans\_hd\_classes <- c(rep(1, N), rep(2, N))

**Part 2**

k <- kmeans(bad\_kmeans\_hd, centers = 2) # predicted = k$cluster, actual truth = bad\_kmeans\_hd\_classes kmeans\_results = data.frame(Label = c("Accuray", "Error Rate", "True Positive Rate", "True Negative Rate", "False Positive Rate", "False Negative Rate", "Precision+", "Precision-", "F-measure+", "F-measure-", "G-Mean")) kmeans\_results["Vanilla k-means in high dim"] = twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans\_hd\_classes, k$cluster)) kmeans\_results

## Label Vanilla k-means in high dim ## 1 Accuray 0.5 ## 2 Error Rate 0.5 ## 3 True Positive Rate 0.5 ## 4 True Negative Rate 0.5 ## 5 False Positive Rate 0.5 ## 6 False Negative Rate 0.5 ## 7 Precision+ 0.5 ## 8 Precision- 0.5 ## 9 F-measure+ 0.5 ## 10 F-measure- 0.5 ## 11 G-Mean 0.5

**Part 3**

p <- kpca(bad\_kmeans\_hd, kernel = "rbfdot", kpar = list(sigma = 1), features = 10) plot(eig(p))

From this plot, it looks like 3 components will be good.

**Part 4**

mat <- pcv(kpca(bad\_kmeans\_hd, kernel = "rbfdot", kpar = list(sigma = 1), features = 3)) df\_pca <- data.frame(mat) df\_pca$classes <- bad\_kmeans\_hd\_classes

**Part 5**

k <- kmeans(mat, centers = 2) # predicted = k$cluster, actual truth = bad\_kmeans\_hd\_classes

**Part 6**

kmeans\_results["PCA with Vanilla k-means in low dim"] = twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans\_hd\_classes, k$cluster)) kmeans\_results

## Label Vanilla k-means in high dim ## 1 Accuray 0.5 ## 2 Error Rate 0.5 ## 3 True Positive Rate 0.5 ## 4 True Negative Rate 0.5 ## 5 False Positive Rate 0.5 ## 6 False Negative Rate 0.5 ## 7 Precision+ 0.5 ## 8 Precision- 0.5 ## 9 F-measure+ 0.5 ## 10 F-measure- 0.5 ## 11 G-Mean 0.5 ## PCA with Vanilla k-means in low dim ## 1 0.6250 ## 2 0.3750 ## 3 0.6000 ## 4 0.6667 ## 5 0.3333 ## 6 0.4000 ## 7 0.7500 ## 8 0.5000 ## 9 0.6667 ## 10 0.5714 ## 11 0.6325

k-Means performance has improved (it went from 50% accuracy (random guessing) to ~60%).

**Part 7**

s <- kkmeans(bad\_kmeans\_hd, centers = bad\_kmeans\_hd[c(1, 2), ], kernel = "rbfdot")

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

# predicted = as.vector(s), actual truth = bad\_kmeans\_hd\_classes kmeans\_results["RBF kernel k-means in high dim"] = twoCrossConfusionMatrixMetrics(ClusteringConfusionMatrix(bad\_kmeans\_hd\_classes, as.vector(s))) kmeans\_results

## Label Vanilla k-means in high dim ## 1 Accuray 0.5 ## 2 Error Rate 0.5 ## 3 True Positive Rate 0.5 ## 4 True Negative Rate 0.5 ## 5 False Positive Rate 0.5 ## 6 False Negative Rate 0.5 ## 7 Precision+ 0.5 ## 8 Precision- 0.5 ## 9 F-measure+ 0.5 ## 10 F-measure- 0.5 ## 11 G-Mean 0.5 ## PCA with Vanilla k-means in low dim RBF kernel k-means in high dim ## 1 0.6250 0.5 ## 2 0.3750 0.5 ## 3 0.6000 0.5 ## 4 0.6667 0.5 ## 5 0.3333 0.5 ## 6 0.4000 0.5 ## 7 0.7500 0.5 ## 8 0.5000 0.5 ## 9 0.6667 0.5 ## 10 0.5714 0.5 ## 11 0.6325 0.5

RBF k-Means on the original gives 50% accuracy while k-Means on the low dimensional data performs better with ~60% accuracy.

When applying the PCA first to reduce the dimensions, one has to determine how many components to use, based on the eigenvalues. It might be tricky to decide on a “good” number of components if there is no clear cut. The benefit is that some noise seems to be removed from the data, increasing the performance of the clustering. Using a kernel k-means on the original high dimensional data, one has to choose which kernel is apropriate.