Homework 3

Lorenzo Bocchi

2022-05-29

Introduction

This analysis will focus on gene expression data of 79 patients with leukemia belonging to two groups:

- patients with a chromosomal translocation (marked as 1 in the response)
- patients cytogenetically normal (marked as -1 in the response)

Some more information about translocation:

"Translocations generate novel chromosomes. In a translocation, a segment from one chromosome is transferred to a nonhomologous chromosome or to a new site on the same chromosome. [...] Translocations are more often associated with negative consequences such as aneuploidy, infertility, or cancer." [1]

We start by loading the data from the 'gene_expr.tsv' file, which contains data of 2000 genes and the response for 79 individuals:

```r
gene_data <- read.csv("./gene_expr.tsv", sep = "")
```

Let's also make sure that there are no missing samples:

```r
any(is.na(gene_data))
```

```
## [1] FALSE
```

First part: full dataset

Nothing is missing, so we can proceed to divide the data in training and test sets. We will assign 80% of the dataset to the training data and the remaining to the test data:

```r
set.seed(1)
sample <- createDataPartition(gene_data$y, p = 0.8)
train_data <- gene_data[sample$Resample1, ]
test_data <- gene_data[-sample$Resample1, ]
x_train <- model.matrix(y ~ ., data = train_data)[, -1]
x_test <- model.matrix(y ~ ., data = test_data)[, -1]
y_train <- train_data$y
y_test <- test_data$y
```

For this analysis, we will classify each individual using Support Vector Machines with three different kernels:

- linear

[1] O'Connor, C. (2008) Human chromosome translocations and cancer. Nature Education 1(1):56

- radial
- polynomial

For each of these kernels, we want to find the best tuning parameters. The cost is shared by all three kernels, the radial adds the gamma and the polynomial the degree.

Given this premise, the first step is to find the optimal cost. To do that we do a k-fold cross validation on the training set and, since the dataset contains only 79 observations, we choose k = 5. In each fold we test the three SVMs with different parameters and save the average MSE for each cost to be able to compare them later. For both the gamma and the cost, we choose the values suggested by Chih-Wei Hsu et al. in their paper “A Practical Guide to Support Vector Classification” [2], which are between 2^-5 and 2^15 for the cost and between 2^-15 and 2^3 for the gamma. For the degree of the polynomial, we choose to test between 2 and 5 to avoid overfitting:

```r
set.seed(1)
dfTrain <- data.frame(x = x_train, y = as.factor(y_train))
dfTest <- data.frame(x = x_test, y = as.factor(y_test))
tuneGrid <- expand.grid(cost = 2^(-5:15), gamma_vals = 2^(-15:3))
costArr <- tuneGrid$cost
costArr <- costArr[!duplicated(costArr)]
gammaArr <- tuneGrid$gamma_vals
gammaArr <- gammaArr[!duplicated(gammaArr)]
degArr <- c(2:5)

mat_res_lin <- matrix(0, nrow = length(costArr), ncol = 4)
mat_res_rad <- matrix(0, nrow = length(costArr), ncol = 4)
mat_res_pol <- matrix(0, nrow = length(costArr), ncol = 4)
folds <- createFolds(dfTrain$y, k = 5)
i <- 1
for (c in costArr) {
    res_lin_internal <- matrix(0, nrow = length(folds), ncol = 3)
    res_rad_internal <- matrix(0, nrow = length(folds), ncol = 3)
    res_pol_internal <- matrix(0, nrow = length(folds), ncol = 3)
    j <- 1
    for (f in folds) {
        out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "linear", cost = c, type = "C-classification")
        res_lin_internal[j, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
            ]$y)
        res_lin_internal[j, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]),
            dfTrain[f, ]$y, positive = levels(dfTrain[f, ]$y)[1])
        res_lin_internal[j, 3] <- specificity(predict(out, newdata = dfTrain[f, ]),
            dfTrain[f, ]$y, positive = levels(dfTrain[f, ]$y)[1])

        radAcc <- matrix(0, nrow = length(gammaArr), ncol = 3)
        z <- 1
        for (g in gammaArr) {
            out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "radial", cost = c,
                gamma = g, type = "C-classification")
            radAcc[z, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
                ]$y)
            radAcc[z, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            radAcc[z, 3] <- specificity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            z <- z + 1
        }
        res_rad_internal[j, 1] <- mean(radAcc[, 1])
        res_rad_internal[j, 2] <- mean(radAcc[, 2])
```

[2] https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

```
        res_rad_internal[j, 3] <- mean(radAcc[, 3])

        polAcc <- matrix(0, nrow = length(degArr), ncol = 3)
        z <- 1
        for (d in degArr) {
            out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "polynomial", cost = c,
                degree = d, type = "C-classification")
            polAcc[z, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
                ]$y)
            polAcc[z, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            polAcc[z, 3] <- specificity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            z <- z + 1
        }
        res_pol_internal[j, 1] <- mean(polAcc[, 1])
        res_pol_internal[j, 2] <- mean(polAcc[, 2])
        res_pol_internal[j, 3] <- mean(polAcc[, 3])
        j <- j + 1
    }
    mat_res_lin[i, 1] <- c
    mat_res_lin[i, 2] <- mean(res_lin_internal[, 1])
    mat_res_lin[i, 3] <- mean(res_lin_internal[, 2])
    mat_res_lin[i, 4] <- mean(res_lin_internal[, 3])

    mat_res_rad[i, 1] <- c
    mat_res_rad[i, 2] <- mean(res_rad_internal[, 1])
    mat_res_rad[i, 3] <- mean(res_rad_internal[, 2])
    mat_res_rad[i, 4] <- mean(res_rad_internal[, 3])

    mat_res_pol[i, 1] <- c
    mat_res_pol[i, 2] <- mean(res_pol_internal[, 1])
    mat_res_pol[i, 3] <- mean(res_pol_internal[, 2])
    mat_res_pol[i, 4] <- mean(res_pol_internal[, 3])
    i <- i + 1
}
colnames(mat_res_lin) <- c("cost", "accuracy", "sensitivity", "specificity")
colnames(mat_res_rad) <- c("cost", "accuracy", "sensitivity", "specificity")
colnames(mat_res_pol) <- c("cost", "accuracy", "sensitivity", "specificity")
```

Linear kernel

After a pretty lengthy computation, we can now check and compare the results. The first we will check is the SVM with the linear kernel:

```
mat_res_lin
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.8282051   0.8285714   0.8266667
##  [2,] 6.2500e-02 0.8282051   0.8285714   0.8266667
##  [3,] 1.2500e-01 0.8282051   0.8285714   0.8266667
##  [4,] 2.5000e-01 0.8282051   0.8285714   0.8266667
##  [5,] 5.0000e-01 0.8282051   0.8285714   0.8266667
##  [6,] 1.0000e+00 0.8282051   0.8285714   0.8266667
##  [7,] 2.0000e+00 0.8282051   0.8285714   0.8266667
##  [8,] 4.0000e+00 0.8282051   0.8285714   0.8266667
##  [9,] 8.0000e+00 0.8282051   0.8285714   0.8266667
## [10,] 1.6000e+01 0.8282051   0.8285714   0.8266667
## [11,] 3.2000e+01 0.8282051   0.8285714   0.8266667
## [12,] 6.4000e+01 0.8282051   0.8285714   0.8266667
## [13,] 1.2800e+02 0.8282051   0.8285714   0.8266667
## [14,] 2.5600e+02 0.8282051   0.8285714   0.8266667
## [15,] 5.1200e+02 0.8282051   0.8285714   0.8266667
## [16,] 1.0240e+03 0.8282051   0.8285714   0.8266667
## [17,] 2.0480e+03 0.8282051   0.8285714   0.8266667
## [18,] 4.0960e+03 0.8282051   0.8285714   0.8266667
## [19,] 8.1920e+03 0.8282051   0.8285714   0.8266667
## [20,] 1.6384e+04 0.8282051   0.8285714   0.8266667
## [21,] 3.2768e+04 0.8282051   0.8285714   0.8266667
```

```
print(paste("Best cost for: accuracy =", mat_res_lin[which.max(mat_res_lin[, 2]),
    1], "sensitivity =", mat_res_lin[which.max(mat_res_lin[, 3]), 1], "specificity =",
    mat_res_lin[which.max(mat_res_lin[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 0.03125 sensitivity = 0.03125 specificity = 0.03125"
```

Apparently, there is very little difference with the choice of the cost. In particular, the smallest and second smallest are the only ones that change. The best accuracy and sensitivity are obtained with a cost of 0.03125 while the best specificity is obtained with 0.0625. The difference in accuracy is pretty much nonexistent, while the sensitivity ans specificity change both around 2% - 3%. Since our patients have leukemia, we are interested to know if they really suffer from a chromosomal translocation, so we feel more confident in giving the priority to sensitivity in order to avoid missing possible cases:

```
set.seed(1)
out <- svm(y ~ ., data = dfTrain, kernel = "linear", cost = mat_res_lin[which.max(mat_res_lin[,
    3]), 1], type = "C-classification")
pred <- predict(out, newdata = dfTest)
prob_lin_full <- attributes(predict(out, dfTest, decision.values = T))$decision.values
accLin <- mean(pred == dfTest$y)
sensLin <- sensitivity(pred, dfTest$y, positive = levels(dfTest$y)[1])
specLin <- specificity(pred, dfTest$y, positive = levels(dfTest$y)[1])
table(pred, dfTest$y)
```

```
##
## pred -1 1
##   -1  6 3
##   1   1 5
```

```
print(paste("Accuracy:", accLin, "Sensitivity:", sensLin, "Specificity:", specLin))
```

```
## [1] "Accuracy: 0.733333333333333 Sensitivity: 0.857142857142857 Specificity: 0.625"
```

Radial kernel

We proceed to analyze the SVM with the radial kernel:

```
mat_res_rad
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.5474359   1.0000000  0.00000000
##  [2,] 6.2500e-02 0.5474359   1.0000000  0.00000000
##  [3,] 1.2500e-01 0.5474359   1.0000000  0.00000000
##  [4,] 2.5000e-01 0.5474359   1.0000000  0.00000000
##  [5,] 5.0000e-01 0.5474359   1.0000000  0.00000000
##  [6,] 1.0000e+00 0.5547233   0.9894737  0.02877193
##  [7,] 2.0000e+00 0.5844804   0.9744361  0.11298246
##  [8,] 4.0000e+00 0.6016869   0.9639098  0.16421053
##  [9,] 8.0000e+00 0.6122807   0.9563910  0.19649123
## [10,] 1.6000e+01 0.6155870   0.9563910  0.20350877
## [11,] 3.2000e+01 0.6188259   0.9593985  0.20701754
## [12,] 6.4000e+01 0.6188259   0.9593985  0.20701754
## [13,] 1.2800e+02 0.6188259   0.9593985  0.20701754
## [14,] 2.5600e+02 0.6188259   0.9593985  0.20701754
## [15,] 5.1200e+02 0.6188259   0.9593985  0.20701754
## [16,] 1.0240e+03 0.6188259   0.9593985  0.20701754
## [17,] 2.0480e+03 0.6188259   0.9593985  0.20701754
## [18,] 4.0960e+03 0.6188259   0.9593985  0.20701754
## [19,] 8.1920e+03 0.6188259   0.9593985  0.20701754
## [20,] 1.6384e+04 0.6188259   0.9593985  0.20701754
## [21,] 3.2768e+04 0.6188259   0.9593985  0.20701754
```

```r
print(paste("Best cost for: accuracy =", mat_res_rad[which.max(mat_res_rad[, 2]),
    1], "sensitivity =", mat_res_rad[which.max(mat_res_rad[, 3]), 1], "specificity =",
    mat_res_rad[which.max(mat_res_rad[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 32 sensitivity = 0.03125 specificity = 32"
```

Here the choice of the cost are not obvious. While up to a cost of 0.5 we obtain a perfect sensitivity, the specificity is 0. The accuracy ranges between 55% and 62%. The highest specificity is 21%. A good compromise between the three values could be obtained with a cost of 32, where we obtain both the highest accuracy and the highest specificity, while only sacrificing around 4% of sensitivity. This time we sacrifice the sensitivity because it is very high compared to the specificity. If they were close enough we would have chosen to stick with the sensitivity, but in this case there is more than 70% difference between the two measures and, ideally, we would prefer to maximize both. After choosing the right parameter, we proceed to fit the model with the fixed cost and test each gamma value to find the best one. For this test, we fit the model on the training data and test against the test data:

```r
set.seed(1)
radAcc <- matrix(0, nrow = length(gammaArr), ncol = 4)
z <- 1
for (g in gammaArr) {
    out <- svm(y ~ ., data = dfTrain, kernel = "radial", cost = 32, gamma = g, type = "C-classification")
    radAcc[z, 1] <- g
    radAcc[z, 2] <- mean(predict(out, newdata = dfTest) == dfTest$y)
    radAcc[z, 3] <- sensitivity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    radAcc[z, 4] <- specificity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    z <- z + 1
}
colnames(radAcc) <- c("gamma", "accuracy", "sensitivity", "specificity")
radAcc
```

```
##               gamma  accuracy sensitivity specificity
##  [1,] 3.051758e-05 0.7333333   0.8571429       0.625
##  [2,] 6.103516e-05 0.7333333   0.8571429       0.625
##  [3,] 1.220703e-04 0.7333333   0.8571429       0.625
##  [4,] 2.441406e-04 0.7333333   0.8571429       0.625
##  [5,] 4.882812e-04 0.5333333   0.7142857       0.375
##  [6,] 9.765625e-04 0.6000000   0.8571429       0.375
##  [7,] 1.953125e-03 0.5333333   1.0000000       0.125
##  [8,] 3.906250e-03 0.4666667   1.0000000       0.000
##  [9,] 7.812500e-03 0.4666667   1.0000000       0.000
## [10,] 1.562500e-02 0.4666667   1.0000000       0.000
## [11,] 3.125000e-02 0.4666667   1.0000000       0.000
## [12,] 6.250000e-02 0.4666667   1.0000000       0.000
## [13,] 1.250000e-01 0.4666667   1.0000000       0.000
## [14,] 2.500000e-01 0.4666667   1.0000000       0.000
## [15,] 5.000000e-01 0.4666667   1.0000000       0.000
## [16,] 1.000000e+00 0.4666667   1.0000000       0.000
## [17,] 2.000000e+00 0.4666667   1.0000000       0.000
## [18,] 4.000000e+00 0.4666667   1.0000000       0.000
## [19,] 8.000000e+00 0.4666667   1.0000000       0.000
```

```
print(paste("Best gamma for: accuracy =", radAcc[which.max(radAcc[, 2]), 1], "sensitivity =",
    radAcc[which.max(radAcc[, 3]), 1], "specificity =", radAcc[which.max(radAcc[,
        4]), 1]))
```

```
## [1] "Best gamma for: accuracy = 3.0517578125e-05 sensitivity = 0.001953125 specificity = 3.0517578125e-05"
```

Once again, the choice for the best parameter is not immediate. When the sensitivity is 1, the specificity is 0 or close to it and the accuracy is much lower compared to smaller values of sensitivity. A good balance can be obtained by sacrificing around 14% in sensitivity, but gaining up to 63% in specificity and around 30% in accuracy. A gamma of 2.441406e-04 provides these results. We can fit the model with this value and check the final result

```
set.seed(1)
out <- svm(y ~ ., data = dfTrain, kernel = "radial", cost = 32, gamma = 0.0002441406,
    type = "C-classification")
pred <- predict(out, newdata = dfTest)
prob_rad_full <- attributes(predict(out, dfTest, decision.values = T))$decision.values
accRad <- mean(pred == dfTest$y)
sensRad <- sensitivity(pred, dfTest$y, positive = levels(dfTest$y)[1])
specRad <- specificity(pred, dfTest$y, positive = levels(dfTest$y)[1])
table(pred, dfTest$y)
```

```
##
## pred -1 1
##   -1  6 3
##   1   1 5
```

```
print(paste("Accuracy:", accRad, "Sensitivity:", sensRad, "Specificity:", specRad))
```

```
## [1] "Accuracy: 0.733333333333333 Sensitivity: 0.857142857142857 Specificity: 0.625"
```

Curiously, we get the same results as with the linear kernel.

Polynomial kernel

Finally, we check the polynomial kernel's performance:

```
mat_res_pol
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.5474359   1.0000000  0.00000000
##  [2,] 6.2500e-02 0.5474359   1.0000000  0.00000000
##  [3,] 1.2500e-01 0.5474359   1.0000000  0.00000000
##  [4,] 2.5000e-01 0.5474359   1.0000000  0.00000000
##  [5,] 5.0000e-01 0.5314103   0.9714286  0.00000000
##  [6,] 1.0000e+00 0.5153846   0.9285714  0.01833333
##  [7,] 2.0000e+00 0.5073718   0.7928571  0.16333333
##  [8,] 4.0000e+00 0.5666667   0.7071429  0.40000000
##  [9,] 8.0000e+00 0.5474359   0.5357143  0.56666667
## [10,] 1.6000e+01 0.5166667   0.4000000  0.65833333
## [11,] 3.2000e+01 0.5243590   0.3214286  0.76666667
## [12,] 6.4000e+01 0.5083333   0.2928571  0.76666667
## [13,] 1.2800e+02 0.5083333   0.2928571  0.76666667
## [14,] 2.5600e+02 0.5083333   0.2928571  0.76666667
## [15,] 5.1200e+02 0.5083333   0.2928571  0.76666667
## [16,] 1.0240e+03 0.5083333   0.2928571  0.76666667
## [17,] 2.0480e+03 0.5083333   0.2928571  0.76666667
## [18,] 4.0960e+03 0.5083333   0.2928571  0.76666667
## [19,] 8.1920e+03 0.5083333   0.2928571  0.76666667
## [20,] 1.6384e+04 0.5083333   0.2928571  0.76666667
## [21,] 3.2768e+04 0.5083333   0.2928571  0.76666667
```

```
print(paste("Best cost for: accuracy =", mat_res_pol[which.max(mat_res_pol[, 2]),
    1], "sensitivity =", mat_res_pol[which.max(mat_res_pol[, 3]), 1], "specificity =",
    mat_res_pol[which.max(mat_res_pol[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 4 sensitivity = 0.03125 specificity = 32"
```

With the polynomial kernel, the accuracy is pretty low. In this case it is also difficult to choose a good compromise between all three values, since for the specificity to get to a decent level we must sacrifice around 50% of the sensitivity. The best we could do would be to choose a cost of 4. In this case, the accuracy is the highest, the sensitivity is around 70% and the specificity is at 40%. We fit the model with the chosen cost and check for the best degree:

```
set.seed(1)
polAcc <- matrix(0, nrow = length(degArr), ncol = 4)
z <- 1
for (d in degArr) {
    out <- svm(y ~ ., data = dfTrain, kernel = "polynomial", cost = 4, degree = d,
        type = "C-classification")
    polAcc[z, 1] <- d
    polAcc[z, 2] <- mean(predict(out, newdata = dfTest) == dfTest$y)
    polAcc[z, 3] <- sensitivity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    polAcc[z, 4] <- specificity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    z <- z + 1
}
colnames(polAcc) <- c("degree", "accuracy", "sensitivity", "specificity")
polAcc
```

```
##      degree  accuracy sensitivity specificity
## [1,]      2 0.5333333   0.4285714       0.625
## [2,]      3 0.4666667   0.7142857       0.250
## [3,]      4 0.4666667   0.8571429       0.125
## [4,]      5 0.4666667   1.0000000       0.000
```

```
print(paste("Best degree for: accuracy =", polAcc[which.max(polAcc[, 2]), 1], "sensitivity =",
    polAcc[which.max(polAcc[, 3]), 1], "specificity =", polAcc[which.max(polAcc[,
        4]), 1]))
```

```
## [1] "Best degree for: accuracy = 2 sensitivity = 5 specificity = 2"
```

The situation is again difficult. the accuracy doesn't change much, but it's low to start with. The sensitivity starts from 43% and reaches 100% but, as it gets bigger, the specificity converges to 0. To obtain a decent specificity we would have to drop the sensitivity from 71% to 43%, which is unacceptable. Therefore, we maximize the sensitivity and pick 2 as the degree (which also gives us 25% specificity compared to a degree 3 which gives us only 13%):

```
set.seed(1)
out <- svm(y ~ ., data = dfTrain, kernel = "polynomial", cost = 4, degree = 2, type = "C-classification")
pred <- predict(out, newdata = dfTest)
prob_pol_full <- attributes(predict(out, dfTest, decision.values = T))$decision.values
accPol <- mean(pred == dfTest$y)
sensPol <- sensitivity(pred, dfTest$y, positive = levels(dfTest$y)[1])
specPol <- specificity(pred, dfTest$y, positive = levels(dfTest$y)[1])
table(pred, dfTest$y)
```

```
##
## pred -1 1
##   -1  3 3
##   1   4 5
```

```
print(paste("Accuracy:", accPol, "Sensitivity:", sensPol, "Specificity:", specPol))
```

```
## [1] "Accuracy: 0.533333333333333 Sensitivity: 0.428571428571429 Specificity: 0.625"
```

The results are pretty bad compared to both the other kernels.

Second part: reducing the variables

The first part of the analysis is done. At this point, we can reduce the number of variables considered and compare the results once again. A popular approach in gene expression analysis is to keep only the most variable genes for downstream analysis. Since most of the 2K genes have low expression or do not vary much across the experiments, this step usually minimizes the contribution of noise. We will select only genes whose standard deviation is among the top 5% and repeat the analyses performed previously.

We start the second part of the analysis by creating the new subset according to the just mentioned rule:

```
# Extract the names of the genes (without sampleID and y)
geneNames <- names(gene_data)[-1]
geneNames <- geneNames[-length(geneNames)]

# Calculate standard deviation for all genes
stDevs <- c()
for (gene in geneNames) {
    stDevs <- c(stDevs, sd(gene_data[, gene]))
}

# Only extract the first 5% ordered by decreasing deviation
dataf <- data.frame(gene = geneNames, std = stDevs)
dataf <- head(dataf[order(dataf$std, decreasing = T), ], round(length(stDevs) * 0.05))
```

We create the new train and test sets:

```
set.seed(1)
gene_2 <- subset(gene_data, select = c(dataf[, 1], "y"))
sample <- caret::createDataPartition(gene_2$y, p = 0.8)
train_data <- gene_2[sample$Resample1, ]
test_data <- gene_2[-sample$Resample1, ]
x_train <- model.matrix(y ~ ., data = train_data)[, -1]
x_test <- model.matrix(y ~ ., data = test_data)[, -1]
y_train <- train_data$y
y_test <- test_data$y
```

Now we can re-analyse the models for the best cost. The analysis is the same as before:

```r
set.seed(1)
dfTrain <- data.frame(x = x_train, y = as.factor(y_train))
dfTest <- data.frame(x = x_test, y = as.factor(y_test))
tuneGrid <- expand.grid(cost = 2^(-5:15), gamma_vals = 2^(-15:3))
costArr <- tuneGrid$cost
costArr <- costArr[!duplicated(costArr)]
gammaArr <- tuneGrid$gamma_vals
gammaArr <- gammaArr[!duplicated(gammaArr)]
degArr <- c(2:5)

mat_res_lin <- matrix(0, nrow = length(costArr), ncol = 4)
mat_res_rad <- matrix(0, nrow = length(costArr), ncol = 4)
mat_res_pol <- matrix(0, nrow = length(costArr), ncol = 4)
folds <- createFolds(dfTrain$y, k = 5)
i <- 1
for (c in costArr) {
    res_lin_internal <- matrix(0, nrow = length(folds), ncol = 3)
    res_rad_internal <- matrix(0, nrow = length(folds), ncol = 3)
    res_pol_internal <- matrix(0, nrow = length(folds), ncol = 3)
    j <- 1
    for (f in folds) {
        out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "linear", cost = c, type = "C-classification")
        res_lin_internal[j, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
            ]$y)
        res_lin_internal[j, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]),
            dfTrain[f, ]$y, positive = levels(dfTrain[f, ]$y)[1])
        res_lin_internal[j, 3] <- specificity(predict(out, newdata = dfTrain[f, ]),
            dfTrain[f, ]$y, positive = levels(dfTrain[f, ]$y)[1])

        radAcc <- matrix(0, nrow = length(gammaArr), ncol = 3)
        z <- 1
        for (g in gammaArr) {
            out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "radial", cost = c,
                gamma = g, type = "C-classification")
            radAcc[z, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
                ]$y)
            radAcc[z, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            radAcc[z, 3] <- specificity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            z <- z + 1
        }
        res_rad_internal[j, 1] <- mean(radAcc[, 1])
        res_rad_internal[j, 2] <- mean(radAcc[, 2])
        res_rad_internal[j, 3] <- mean(radAcc[, 3])

        polAcc <- matrix(0, nrow = length(degArr), ncol = 3)
        z <- 1
        for (d in degArr) {
            out <- svm(y ~ ., data = dfTrain[-f, ], kernel = "polynomial", cost = c,
                degree = d, type = "C-classification")
            polAcc[z, 1] <- mean(predict(out, newdata = dfTrain[f, ]) == dfTrain[f,
                ]$y)
            polAcc[z, 2] <- sensitivity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            polAcc[z, 3] <- specificity(predict(out, newdata = dfTrain[f, ]), dfTrain[f,
                ]$y, positive = levels(dfTrain[f, ]$y)[1])
            z <- z + 1
        }
        res_pol_internal[j, 1] <- mean(polAcc[, 1])
        res_pol_internal[j, 2] <- mean(polAcc[, 2])
        res_pol_internal[j, 3] <- mean(polAcc[, 3])
        j <- j + 1
    }
    mat_res_lin[i, 1] <- c
    mat_res_lin[i, 2] <- mean(res_lin_internal[, 1])
    mat_res_lin[i, 3] <- mean(res_lin_internal[, 2])
    mat_res_lin[i, 4] <- mean(res_lin_internal[, 3])

    mat_res_rad[i, 1] <- c
    mat_res_rad[i, 2] <- mean(res_rad_internal[, 1])
    mat_res_rad[i, 3] <- mean(res_rad_internal[, 2])
    mat_res_rad[i, 4] <- mean(res_rad_internal[, 3])

    mat_res_pol[i, 1] <- c
    mat_res_pol[i, 2] <- mean(res_pol_internal[, 1])
    mat_res_pol[i, 3] <- mean(res_pol_internal[, 2])
    mat_res_pol[i, 4] <- mean(res_pol_internal[, 3])
    i <- i + 1
}
colnames(mat_res_lin) <- c("cost", "accuracy", "sensitivity", "specificity")
colnames(mat_res_rad) <- c("cost", "accuracy", "sensitivity", "specificity")
colnames(mat_res_pol) <- c("cost", "accuracy", "sensitivity", "specificity")
```

Linear kernel

Again, we start by analyzing the results for the linear kernel:

```
mat_res_lin
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.8141026   0.8285714   0.7933333
##  [2,] 6.2500e-02 0.8128205   0.8000000   0.8266667
##  [3,] 1.2500e-01 0.8128205   0.8000000   0.8266667
##  [4,] 2.5000e-01 0.8128205   0.8000000   0.8266667
##  [5,] 5.0000e-01 0.8128205   0.8000000   0.8266667
##  [6,] 1.0000e+00 0.8128205   0.8000000   0.8266667
##  [7,] 2.0000e+00 0.8128205   0.8000000   0.8266667
##  [8,] 4.0000e+00 0.8128205   0.8000000   0.8266667
##  [9,] 8.0000e+00 0.8128205   0.8000000   0.8266667
## [10,] 1.6000e+01 0.8128205   0.8000000   0.8266667
## [11,] 3.2000e+01 0.8128205   0.8000000   0.8266667
## [12,] 6.4000e+01 0.8128205   0.8000000   0.8266667
## [13,] 1.2800e+02 0.8128205   0.8000000   0.8266667
## [14,] 2.5600e+02 0.8128205   0.8000000   0.8266667
## [15,] 5.1200e+02 0.8128205   0.8000000   0.8266667
## [16,] 1.0240e+03 0.8128205   0.8000000   0.8266667
## [17,] 2.0480e+03 0.8128205   0.8000000   0.8266667
## [18,] 4.0960e+03 0.8128205   0.8000000   0.8266667
## [19,] 8.1920e+03 0.8128205   0.8000000   0.8266667
## [20,] 1.6384e+04 0.8128205   0.8000000   0.8266667
## [21,] 3.2768e+04 0.8128205   0.8000000   0.8266667
```

```
print(paste("Best cost for: accuracy =", mat_res_lin[which.max(mat_res_lin[, 2]),
    1], "sensitivity =", mat_res_lin[which.max(mat_res_lin[, 3]), 1], "specificity =",
    mat_res_lin[which.max(mat_res_lin[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 0.03125 sensitivity = 0.03125 specificity = 0.0625"
```

In this case we get the same results as with 2000 variables. Therefore, the same reasoning as before still stands and we apply the first cost:

```
set.seed(1)
out_lin <- svm(y ~ ., data = dfTrain, kernel = "linear", cost = mat_res_lin[which.max(mat_res_lin[,
    3]), 1], type = "C-classification")
pred_lin <- predict(out_lin, newdata = dfTest)
prob_lin <- attributes(predict(out_lin, dfTest, decision.values = T))$decision.values
accLin_purged <- mean(pred_lin == dfTest$y)
sensLin_purged <- sensitivity(pred_lin, dfTest$y, positive = levels(dfTest$y)[1])
specLin_purged <- specificity(pred_lin, dfTest$y, positive = levels(dfTest$y)[1])
table(pred_lin, dfTest$y)
```

```
##
## pred_lin -1 1
##       -1  6 2
##       1   1 6
```

```
print(paste("Accuracy:", accLin_purged, "Sensitivity:", sensLin_purged, "Specificity:",
    specLin_purged))
```

```
## [1] "Accuracy: 0.8 Sensitivity: 0.857142857142857 Specificity: 0.75"
```

Both the accuracy and the specificity have improved.

Radial kernel

Let’s check if the radial kernel also improved:

```
mat_res_rad
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.5474359   1.0000000  0.00000000
##  [2,] 6.2500e-02 0.5474359   1.0000000  0.00000000
##  [3,] 1.2500e-01 0.5474359   1.0000000  0.00000000
##  [4,] 2.5000e-01 0.5474359   1.0000000  0.00000000
##  [5,] 5.0000e-01 0.5715924   0.9939850  0.06140351
##  [6,] 1.0000e+00 0.6058704   0.9578947  0.18105263
##  [7,] 2.0000e+00 0.6328610   0.9488722  0.25087719
##  [8,] 4.0000e+00 0.6435223   0.9338346  0.29263158
##  [9,] 8.0000e+00 0.6566802   0.9187970  0.33964912
## [10,] 1.6000e+01 0.6723347   0.9097744  0.38491228
## [11,] 3.2000e+01 0.6838731   0.8977444  0.42491228
## [12,] 6.4000e+01 0.6896761   0.8917293  0.44456140
## [13,] 1.2800e+02 0.6913630   0.8902256  0.44982456
## [14,] 2.5600e+02 0.6922402   0.8917293  0.44982456
## [15,] 5.1200e+02 0.6922402   0.8917293  0.44982456
## [16,] 1.0240e+03 0.6921727   0.8902256  0.45157895
## [17,] 2.0480e+03 0.6921727   0.8902256  0.45157895
## [18,] 4.0960e+03 0.6921727   0.8902256  0.45157895
## [19,] 8.1920e+03 0.6921727   0.8902256  0.45157895
## [20,] 1.6384e+04 0.6921727   0.8902256  0.45157895
## [21,] 3.2768e+04 0.6921727   0.8902256  0.45157895
```

```
print(paste("Best cost for: accuracy =", mat_res_rad[which.max(mat_res_rad[, 2]),
    1], "sensitivity =", mat_res_rad[which.max(mat_res_rad[, 3]), 1], "specificity =",
    mat_res_rad[which.max(mat_res_rad[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 256 sensitivity = 0.03125 specificity = 1024"
```

Same as before, it's not easy to choose the best parameter. Previously, 32 was the best cost overall. Here, the differences in sensitivity after a cost of 16 are all within 1%. In this case, it makes the most sense to maximize the accuracy and specificity since the sensitivity after that point doesn't really change much and locks at 89% after a cost of 1024, so we will choose 256 as the cost. We now test the gamma values:

```
set.seed(1)
radAcc <- matrix(0, nrow = length(gammaArr), ncol = 4)
z <- 1
for (g in gammaArr) {
    out <- svm(y ~ ., data = dfTrain, kernel = "radial", cost = mat_res_rad[which.max(mat_res_rad[,
        2]), 1], gamma = g, type = "C-classification")
    radAcc[z, 1] <- g
    radAcc[z, 2] <- mean(predict(out, newdata = dfTest) == dfTest$y)
    radAcc[z, 3] <- sensitivity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    radAcc[z, 4] <- specificity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    z <- z + 1
}
colnames(radAcc) <- c("gamma", "accuracy", "sensitivity", "specificity")
radAcc
```

```
##              gamma  accuracy sensitivity specificity
##  [1,] 3.051758e-05 0.8000000   1.0000000       0.625
##  [2,] 6.103516e-05 0.8000000   0.8571429       0.750
##  [3,] 1.220703e-04 0.8000000   0.8571429       0.750
##  [4,] 2.441406e-04 0.8000000   0.8571429       0.750
##  [5,] 4.882812e-04 0.8000000   0.8571429       0.750
##  [6,] 9.765625e-04 0.8000000   0.8571429       0.750
##  [7,] 1.953125e-03 0.7333333   0.8571429       0.625
##  [8,] 3.906250e-03 0.7333333   0.8571429       0.625
##  [9,] 7.812500e-03 0.8000000   1.0000000       0.625
## [10,] 1.562500e-02 0.7333333   1.0000000       0.500
## [11,] 3.125000e-02 0.6000000   1.0000000       0.250
## [12,] 6.250000e-02 0.5333333   1.0000000       0.125
## [13,] 1.250000e-01 0.4666667   1.0000000       0.000
## [14,] 2.500000e-01 0.4666667   1.0000000       0.000
## [15,] 5.000000e-01 0.4666667   1.0000000       0.000
## [16,] 1.000000e+00 0.4666667   1.0000000       0.000
## [17,] 2.000000e+00 0.4666667   1.0000000       0.000
## [18,] 4.000000e+00 0.4666667   1.0000000       0.000
## [19,] 8.000000e+00 0.4666667   1.0000000       0.000
```

```
print(paste("Best gamma for: accuracy =", radAcc[which.max(radAcc[, 2]), 1], "sensitivity =",
    radAcc[which.max(radAcc[, 3]), 1], "specificity =", radAcc[which.max(radAcc[,
        4]), 1]))
```

```
## [1] "Best gamma for: accuracy = 3.0517578125e-05 sensitivity = 3.0517578125e-05 specificity = 6.103515625e
```

Here it’s easier to make a decision. The very first cost gives a high accuracy, perfect sensitivity and good specificity. The second best cost would reduce the sensitivity by 14% and increase the specificity by 16%. We consider this to be a good tradeoff, since the sensitivity would still be a respectable 86% and the specificity would increase to 75%:

```r
set.seed(1)
out_rad <- svm(y ~ ., data = dfTrain, kernel = "radial", cost = mat_res_rad[which.max(mat_res_rad[,
    2]), 1], gamma = radAcc[which.max(radAcc[, 4]), 1], type = "C-classification")
pred_rad <- predict(out_rad, newdata = dfTest)
prob_rad <- attributes(predict(out_rad, dfTest, decision.values = T))$decision.values
accRad_purged <- mean(pred_rad == dfTest$y)
sensRad_purged <- sensitivity(pred_rad, dfTest$y, positive = levels(dfTest$y)[1])
specRad_purged <- specificity(pred_rad, dfTest$y, positive = levels(dfTest$y)[1])
table(pred_rad, dfTest$y)
```

```
##
## pred_rad -1 1
##       -1  6 2
##       1   1 6
```

```r
print(paste("Accuracy:", accRad_purged, "Sensitivity:", sensRad_purged, "Specificity:",
    specRad_purged))
```

```
## [1] "Accuracy: 0.8 Sensitivity: 0.857142857142857 Specificity: 0.75"
```

Both the accuracy and the specificity increased over the previous model with 2000 variables.

Polynomial kernel

At last, we analyse the SVM with the polynomial kernel:

```r
mat_res_pol
```

```
##             cost  accuracy sensitivity specificity
##  [1,] 3.1250e-02 0.5474359   1.0000000  0.00000000
##  [2,] 6.2500e-02 0.5474359   1.0000000  0.00000000
##  [3,] 1.2500e-01 0.5474359   1.0000000  0.00000000
##  [4,] 2.5000e-01 0.5474359   1.0000000  0.00000000
##  [5,] 5.0000e-01 0.5592949   1.0000000  0.02666667
##  [6,] 1.0000e+00 0.6429487   0.7714286  0.49666667
##  [7,] 2.0000e+00 0.6612179   0.4785714  0.88166667
##  [8,] 4.0000e+00 0.6496795   0.4214286  0.92500000
##  [9,] 8.0000e+00 0.6413462   0.4071429  0.92500000
## [10,] 1.6000e+01 0.6493590   0.4214286  0.92500000
## [11,] 3.2000e+01 0.6493590   0.4214286  0.92500000
## [12,] 6.4000e+01 0.6493590   0.4214286  0.92500000
## [13,] 1.2800e+02 0.6493590   0.4214286  0.92500000
## [14,] 2.5600e+02 0.6493590   0.4214286  0.92500000
## [15,] 5.1200e+02 0.6493590   0.4214286  0.92500000
## [16,] 1.0240e+03 0.6493590   0.4214286  0.92500000
## [17,] 2.0480e+03 0.6493590   0.4214286  0.92500000
## [18,] 4.0960e+03 0.6493590   0.4214286  0.92500000
## [19,] 8.1920e+03 0.6493590   0.4214286  0.92500000
## [20,] 1.6384e+04 0.6493590   0.4214286  0.92500000
## [21,] 3.2768e+04 0.6493590   0.4214286  0.92500000
```

```
print(paste("Best cost for: accuracy =", mat_res_pol[which.max(mat_res_pol[, 2]),
    1], "sensitivity =", mat_res_pol[which.max(mat_res_pol[, 3]), 1], "specificity =",
    mat_res_pol[which.max(mat_res_pol[, 4]), 1]))
```

```
## [1] "Best cost for: accuracy = 2 sensitivity = 0.03125 specificity = 4"
```

Even with less variables, choosing the right cost parameter is not easy. Once again we have a situation where the sensitivity decreases dramatically and the specificity goes in the opposite way (and just as dramatically), while the accuracy ranges between 55% and 65%. Since the difference is so big, we choose to prioritize the sensitivity and choose a cost of 1:

```
set.seed(1)
polAcc <- matrix(0, nrow = length(degArr), ncol = 4)
z <- 1
for (d in degArr) {
    out <- svm(y ~ ., data = dfTrain, kernel = "polynomial", cost = 1, degree = d,
        type = "C-classification")
    polAcc[z, 1] <- d
    polAcc[z, 2] <- mean(predict(out, newdata = dfTest) == dfTest$y)
    polAcc[z, 3] <- sensitivity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    polAcc[z, 4] <- specificity(predict(out, newdata = dfTest), dfTest$y, positive = levels(dfTest$y)[1])
    z <- z + 1
}
colnames(polAcc) <- c("degree", "accuracy", "sensitivity", "specificity")
polAcc
```

```
##      degree  accuracy sensitivity specificity
## [1,]      2 0.6666667           1       0.375
## [2,]      3 0.7333333           1       0.500
## [3,]      4 0.5333333           1       0.125
## [4,]      5 0.6666667           1       0.375
```

```
print(paste("Best degree for: accuracy =", polAcc[which.max(polAcc[, 2]), 1], "sensitivity =",
    polAcc[which.max(polAcc[, 3]), 1], "specificity =", polAcc[which.max(polAcc[,
        4]), 1]))
```

```
## [1] "Best degree for: accuracy = 3 sensitivity = 2 specificity = 3"
```

The decision for the degree is immediate. The sensitivity is perfect with all the degrees, while both the accuracy and specificity peak with a degree of 3:

```
set.seed(1)
out_pol <- svm(y ~ ., data = dfTrain, kernel = "polynomial", cost = 1, degree = 3,
    type = "C-classification")
pred_pol <- predict(out_pol, newdata = dfTest)
prob_pol <- attributes(predict(out_pol, dfTest, decision.values = T))$decision.values
accPol_purged <- mean(pred_pol == dfTest$y)
sensPol_purged <- sensitivity(pred_pol, dfTest$y, positive = levels(dfTest$y)[1])
specPol_purged <- specificity(pred_pol, dfTest$y, positive = levels(dfTest$y)[1])
table(pred_pol, dfTest$y)
```

```
##
## pred_pol -1 1
##       -1  7 4
##       1   0 4
```

```
print(paste("Accuracy:", accPol_purged, "Sensitivity:", sensPol_purged, "Specificity:",
    specPol_purged))
```

```
## [1] "Accuracy: 0.733333333333333 Sensitivity: 1 Specificity: 0.5"
```

Both the accuracy and the sensitivity increased notably, while the specificity decreased.

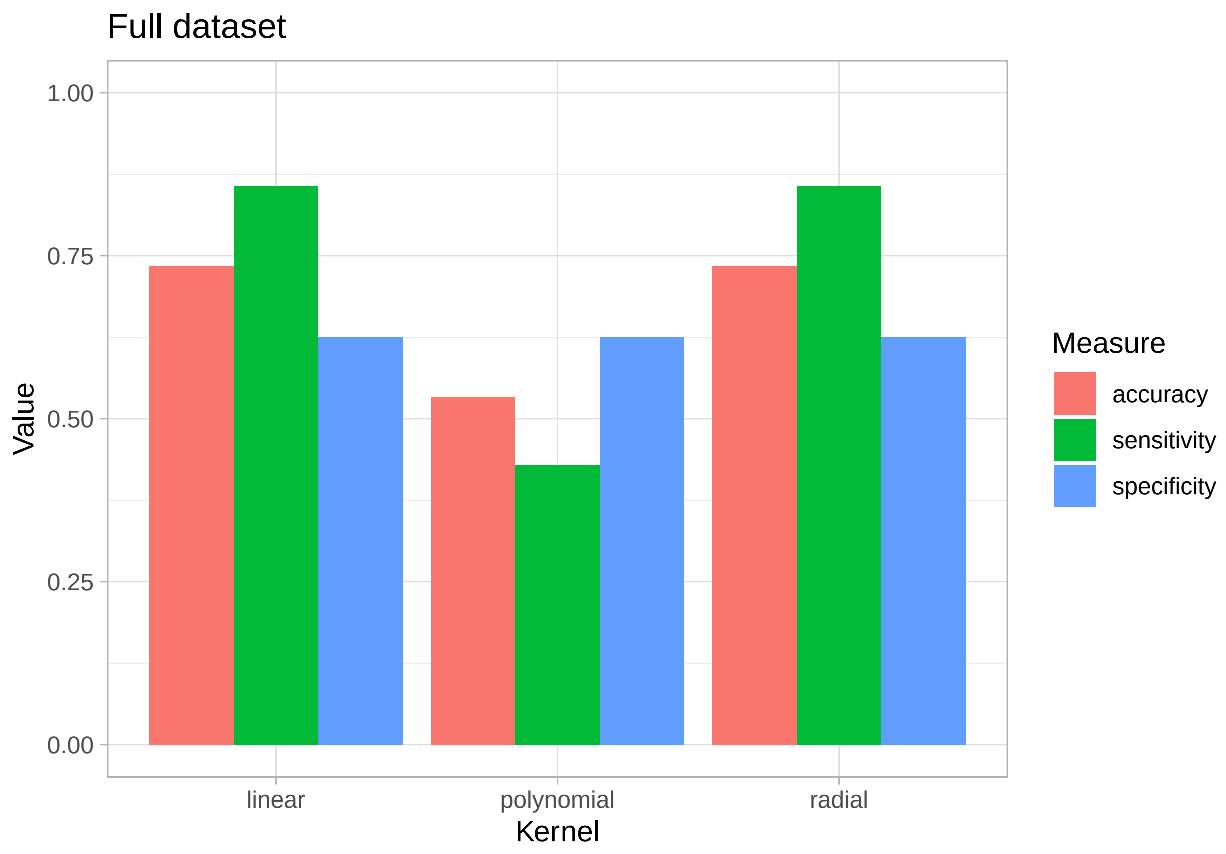
Visually comparing the models

Our final analysis will be visual. We plot the values for the various models to have a better representation of the differences:

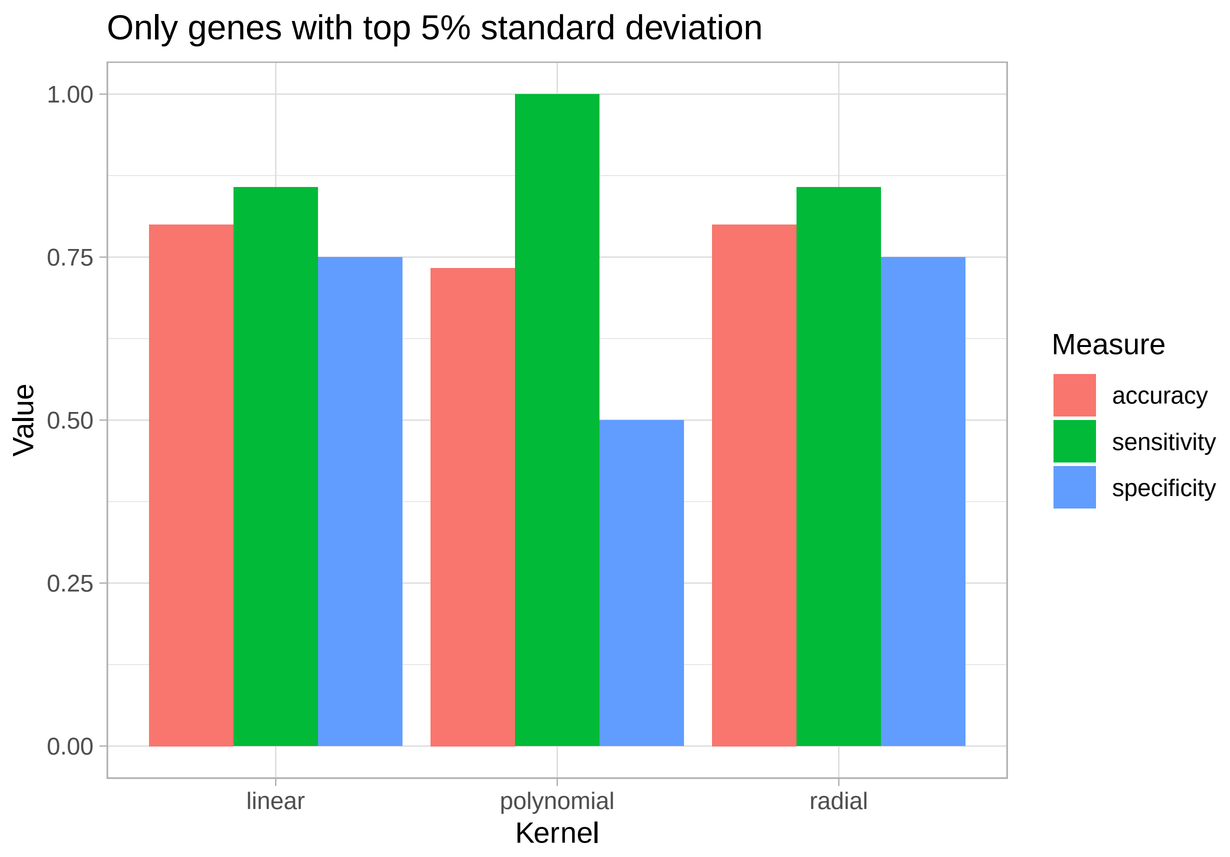
```
kernel_full <- c(rep("linear", 3), rep("radial", 3), rep("polynomial", 3))
measure_full <- rep(c("accuracy", "sensitivity", "specificity"), 3)
value_full <- c(accLin, sensLin, specLin, accRad, sensRad, specRad, accPol, sensPol,
    specPol)
toPlot1 <- data.frame(kernel_full, measure_full, value_full)

kernel_top5 <- c(rep("linear", 3), rep("radial", 3), rep("polynomial", 3))
measure_top5 <- rep(c("accuracy", "sensitivity", "specificity"), 3)
value_top5 <- c(accLin_purged, sensLin_purged, specLin_purged, accRad_purged, sensRad_purged,
    specRad_purged, accPol_purged, sensPol_purged, specPol_purged)
toPlot2 <- data.frame(kernel_top5, measure_top5, value_top5)

ggplot(toPlot1, aes(fill = measure_full, y = value_full, x = kernel_full)) + theme_light() +
    geom_bar(position = "dodge", stat = "identity") + coord_cartesian(ylim = c(0,
    1)) + labs(title = "Full dataset", y = "Value", x = "Kernel") + guides(fill = guide_legend(title = "Measu
```



```r
ggplot(toPlot2, aes(fill = measure_top5, y = value_top5, x = kernel_top5)) + theme_light() +
    geom_bar(position = "dodge", stat = "identity") + coord_cartesian(ylim = c(0,
    1)) + labs(title = "Only genes with top 5% standard deviation", y = "Value",
    x = "Kernel") + guides(fill = guide_legend(title = "Measure"))
```

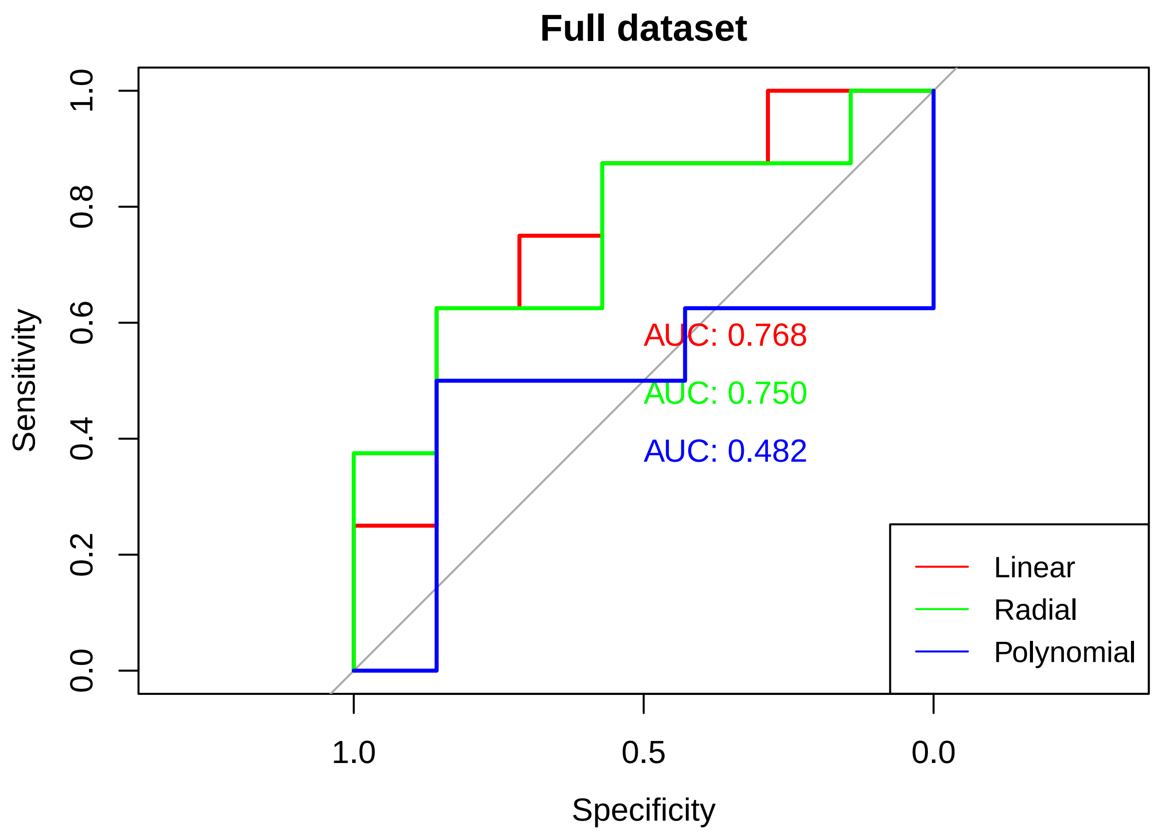


```r
set.seed(1)
roc_lin <- roc(dfTest$y ~ prob_lin_full)
plot.roc(roc_lin, print.auc = TRUE, col = "red", print.auc.y = 0.6, main = "Full dataset")

roc_rad <- roc(dfTest$y ~ prob_rad_full)
plot.roc(roc_rad, add = TRUE, print.auc = TRUE, col = "green")

roc_pol <- roc(dfTest$y ~ prob_pol_full)
plot.roc(roc_pol, add = TRUE, print.auc = TRUE, col = "blue", print.auc.y = 0.4)

par(cex = 0.9)
legend("bottomright", legend = c("Linear", "Radial", "Polynomial"), col = c("red",
    "green", "blue"), lty = c(1, 1, 1))
```

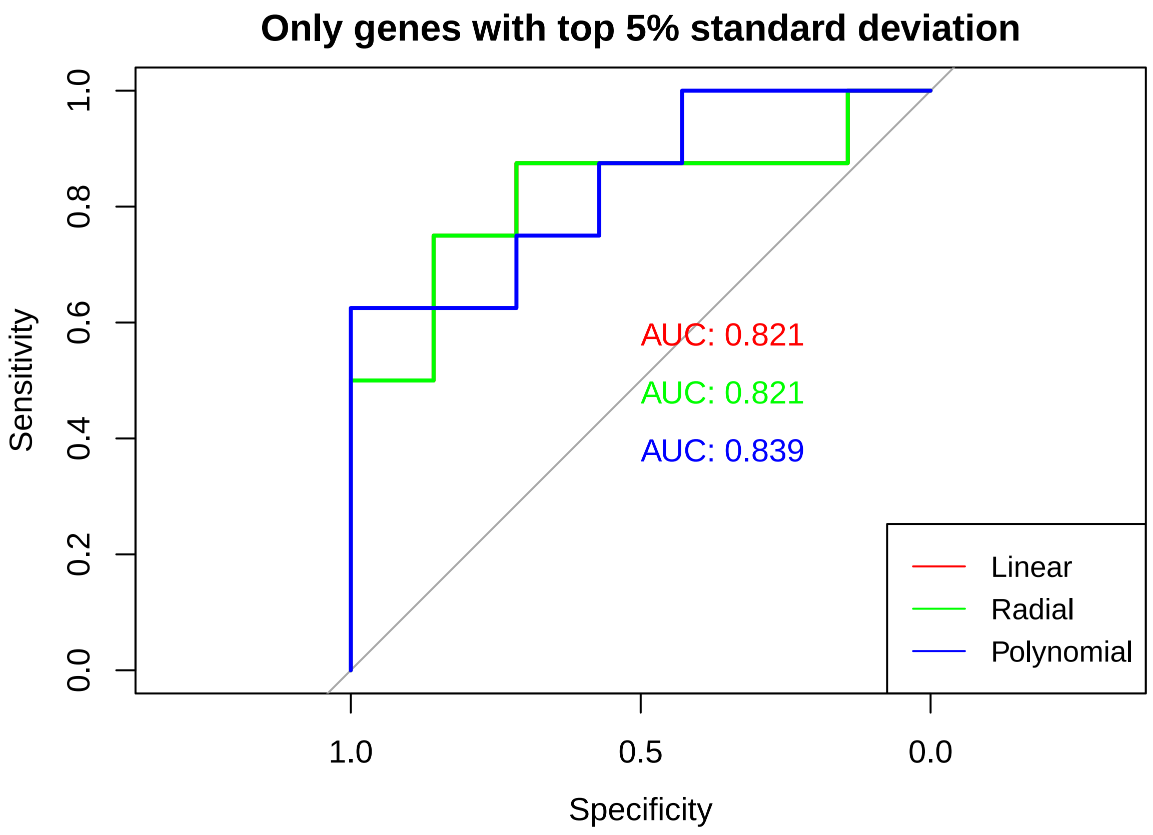


```
set.seed(1)
roc_lin <- roc(dfTest$y ~ prob_lin)
plot.roc(roc_lin, print.auc = TRUE, col = "red", print.auc.y = 0.6, main = "Only genes with top 5% standard d

roc_rad <- roc(dfTest$y ~ prob_rad)
plot.roc(roc_rad, add = TRUE, print.auc = TRUE, col = "green")

roc_pol <- roc(dfTest$y ~ prob_pol)
plot.roc(roc_pol, add = TRUE, print.auc = TRUE, col = "blue", print.auc.y = 0.4)

par(cex = 0.9)
legend("bottomright", legend = c("Linear", "Radial", "Polynomial"), col = c("red",
    "green", "blue"), lty = c(1, 1, 1))
```



Both in the bar plot and the ROC curves, we can see that the linear kernel and radial kernel produce the same output. The polynomial kernel has a perfect sensitivity but a pretty bad specificity. In the ROC plot, the AUC value of the polynomial is slightly higher than the other two.

Conclusion

In conclusion, only choosing the genes with the top 5% deviation helped. This is especially true for the SVM with the polynomial kernel, which has achieved perfect sensitivity, but has a slightly lower accuracy than the other two and a much lower specificity. Overall, it seems that either the linear kernel or the radial kernel are best suited for classification with this dataset. This is true for both the full dataset and the "purged" one. Another very important fact is that by removing noisy variables the computation becomes much quicker. For instance, the first k-fold cross validation takes more than one hour to complete, while the second takes a few minutes.