CS 229 - Machine Learning
Xiangliang Zhang
Computer Science (CS)/Statistics (STAT) Program
Computer, Electrical and Mathematical Sciences & Engineering (CEMSE) Division
King Abdullah University of Science and Technology (KAUST)

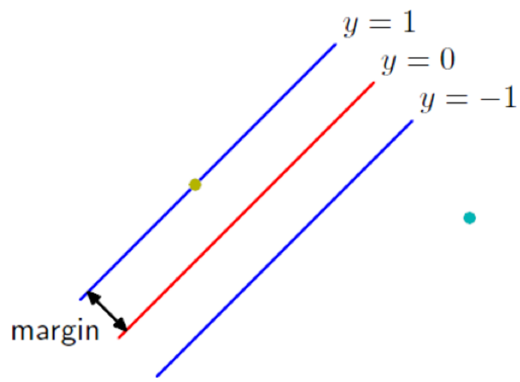
HOMEWORK
VII

Henrique Aparecido Laureano

Spring Semester 2018

Contents

[30 points] Question 1:
Margin for the maximum-margin hyper-plane (Exercise 7.4 of Bishop's book)



Show that the value ρ of the margin for the maximum-margin hyperplane is given by

$$\frac{1}{\rho^2} = \sum_{n=1}^{N} a_n$$

where $\{a_n\}$ are given by maximizing

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.10}$$

subject to the constrains

$$a_n \geq 0, \quad n = 1, \ldots, N, \tag{7.11}$$

and

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{7.12}$$

Solution:

From the theory of discriminant functions we have that

$$\rho = \frac{1}{\|\mathbf{w}\|},$$

with $\mathbf{w}$ being a weight vector that determines the orientation of the decision surface. Then we have

$$\rho = \frac{1}{\|\mathbf{w}\|} \quad \Rightarrow \quad \rho^2 = \frac{1}{\|\mathbf{w}\|^2} \quad \Rightarrow \quad \|\mathbf{w}\|^2 = \frac{1}{\rho^2}.$$

In (7.10) we have the dual representation of the maximum margin problem, that is obtained by eliminating $\mathbf{w}$ and b from the Lagrangian function $L(\mathbf{w}, b, \mathbf{a})$

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \{t_n(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n) + b) - 1\}. \tag{7.7}$$

Setting the derivatives of $L(\mathbf{w}, b, \mathbf{a})$ w.r.t. $\mathbf{w}$ and b equal to zero, we obtain the following two conditions

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{7.9}$$

This constrained optimization problem satisfies (have to satisfie) three KKT (Karush-Kuhn-Tucker) conditions, that are

$$a_n \geq 0 \tag{7.14}$$

$$t_n y(\mathbf{x}_n)) - 1 \geq 0 \tag{7.15}$$

$$a_n\{t_n y(\mathbf{x}_n)) - 1\} = 0, \tag{7.16}$$

with

$$y(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b. \tag{7.1}$$

And with either $a_n = 0$ or $t_n y(\mathbf{x}_n)) = 1$.

From the condition (7.16) we see that the summation term disappear in (7.7) and then we have

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2.$$

Combining this with the condition (7.8), and remembering that the kernel function is defined by $k(\mathbf{x}_n, \mathbf{x}_m) = \boldsymbol{\phi}(\mathbf{x}_n, \mathbf{x}_m)$, the dual representation of the maximum margin problem (7.10) can be defined as

$$\frac{1}{2}\|\mathbf{w}\|^2 = \sum_{n=1}^{N} a_n - \frac{1}{2}\|\mathbf{w}\|^2 \quad \Rightarrow \quad \|\mathbf{w}\|^2 = \sum_{n=1}^{N} a_n.$$

Therefore,

$$\boxed{\frac{1}{\rho^2} = \sum_{n=1}^{N} a_n.}$$

□

[70 points] Question 2: Classification by SVM

Code: You could use the LIBSVM Support Vector Machine library for the classification; or the built-in Matlab SVM functions.

Here I'm using the R language and the package `e1071` [1]

```
# <r code> =================================================================== #
library(e1071)                                               # loading the library
        # if you don't have the package installed, run: install.packages("e1071")
# </r code> ================================================================== #
```

Data: You can download the data from `http://archive.ics.uci.edu/ml/datasets/Wine+Quality`. Take either the red-wine or the white-wine data set.

Choosing the red-wine!

```
# <r code> =================================================================== #
path <- "~/Dropbox/KAUST/machine_learning/hw7/"                       # files path
                                                   # df: dataframe. reading the dataset
df <- read.csv(paste0(path, "winequality-red.csv"), header = TRUE, sep = ";")
# </r code> ================================================================== #
```

Take "quality" as class label, e.g., 1-5 as *negative*, while 6-10 as *positive*.

```
# <r code> =================================================================== #
                                  # defining class label: 1-5: negative, 6-10: positive
df$quality <- as.factor(ifelse(df$quality <= 5, "negative", "positive"))
table(df$quality)                                                # frequency table
# </r code> ================================================================== #
```

```
negative positive
     744      855
```

Evaluating and Testing: Divide the whole data set into training data and testing data, e.g., 60% for training and 40% for testing. Use 5-fold Cross Validation for setting parameters, e.g., C and kernel parameters.

We have 3 different types of models for learning classifiers (SVM with 3 different types of kernel):

- **SVM with linear kernel;**
- **SVM with polynomial kernel;**
- **SVM with the radial basis function kernel.**

[1]**e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien**. Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier, ...

[15 points] (1)

For each learning model (each type of kernel), use 60% of data for training SVM model (with the default parameters), and use the remaining 40% for testing.

OBS. By default the method `svm` uses a classification machine algorithm, C-classification, and scale the variables to zero mean and unit variance.

```
# <r code> =================================================================== #
          # selecting random row numbers for training data (60% of the whole data)
id <- sample(1:nrow(df), round(nrow(df)*.6, 0))
df.train <- df[id, ]                                      # setting 60% for training
df.test <- df[-id, ]                            # setting the remaining 40% for test
# </r code> ================================================================== #
```

Linear kernel: $k(x_i, x_j) = \gamma \|x_i, x_j\|$

C : default cost of constraints violation in `e1071::svm` is 1;

γ : default value in `e1071::svm` is 1/dataset size (number of variables).

```
# <r code> =================================================================== #
 # fitting SVM with linear kernel (training data) using all 11 available variables
linear.train <- svm(quality ~ ., df.train, kernel = "linear")
linear.test <- predict(linear.train, df.test[ , -12]) # prediction in testing data
# </r code> ================================================================== #
```

Polynomial kernel: $k(x_i, x_j) = (c_0 + \gamma \|x_i, x_j\|)^d$

C : default cost of constraints violation in `e1071::svm` is 1;

c_0 : default value in `e1071::svm` is 0;

γ : default value in `e1071::svm` is 1/dataset size (number of variables);

d : polynominal degree, default value in `e1071::svm` is 3.

```
# <r code> =================================================================== #
   # fitting SVM with poly kernel (training data) using all 11 available variables
poly.train <- svm(quality ~ ., df.train, kernel = "polynomial")
poly.test <- predict(poly.train, df.test[ , -12])      # prediction in testing data
# </r code> ================================================================== #
```

Radial kernel: $k(x_i, x_j) = \exp(-\gamma \|x_i, x_j\|)^2$

C : default cost of constraints violation in `e1071::svm` is 1;

γ : default value in `e1071::svm` is 1/dataset size (number of variables).

```r
# <r code> ================================================================== #
 # fitting SVM with radial kernel (training data) using all 11 available variables
radial.train <- svm(quality ~ ., df.train, kernel = "radial")
radial.test <- predict(radial.train, df.test[ , -12]) # prediction in testing data
# </r code> ================================================================= #
```

a)

Report the number of support vectors.

Linear kernel : 588 support vectors (the dataset have 959 samples).
294 of the class label **negative** and 294 of the class label **positive**.

```r
# <r code> ================================================================== #
summary(linear.train)
# </r code> ================================================================= #
```

```


Call:
svm(formula = quality ~ ., data = df.train, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.09090909

Number of Support Vectors:  588

 ( 294 294 )


Number of Classes:  2

Levels:
 negative positive
```

Polynomial kernel : 659 support vectors (the dataset have 959 samples).
327 of the class label **negative** and 332 of the class label **positive**.

```r
# <r code> ================================================================== #
summary(poly.train)
# </r code> ================================================================= #
```

```
Call:
svm(formula = quality ~ ., data = df.train, kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  0.09090909
     coef.0:  0

Number of Support Vectors:  659

 ( 327 332 )


Number of Classes:  2

Levels:
 negative positive
```

Radial kernel : 605 support vectors (the dataset have 959 samples).
307 of the class label **negative** and 298 of the class label **positive**.

```
# <r code> ==================================================================== #
summary(radial.train)
# </r code> =================================================================== #
```

```
Call:
svm(formula = quality ~ ., data = df.train, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.09090909

Number of Support Vectors:  605

 ( 307 298 )


Number of Classes:  2

Levels:
 negative positive
```

□

b)

Plot the ROC curve of testing results by ranking the decision values (3 curves in one figure).

```
# <r code> =================================================================== #
library(pROC)                                                  # loading library
                                  # computing roc curves for the testing datasets
roc.linear <- roc(df.test$quality, as.numeric(linear.test))
roc.poly   <- roc(df.test$quality, as.numeric(poly.test))
roc.radial <- roc(df.test$quality, as.numeric(radial.test))
                                                            # plotting roc curves
plot(roc.linear)
plot(roc.poly, col = "#0080ff", add = TRUE)
plot(roc.radial, col = 2, add = TRUE)
                                                                 # adding legend
legend(x = .55, y = .35, legend = c("Linear", "Polynomial", "Radial")
       , bty = "n", col = c(1, "#0080ff", 2), lwd = 2)
# </r code> ================================================================== #
```

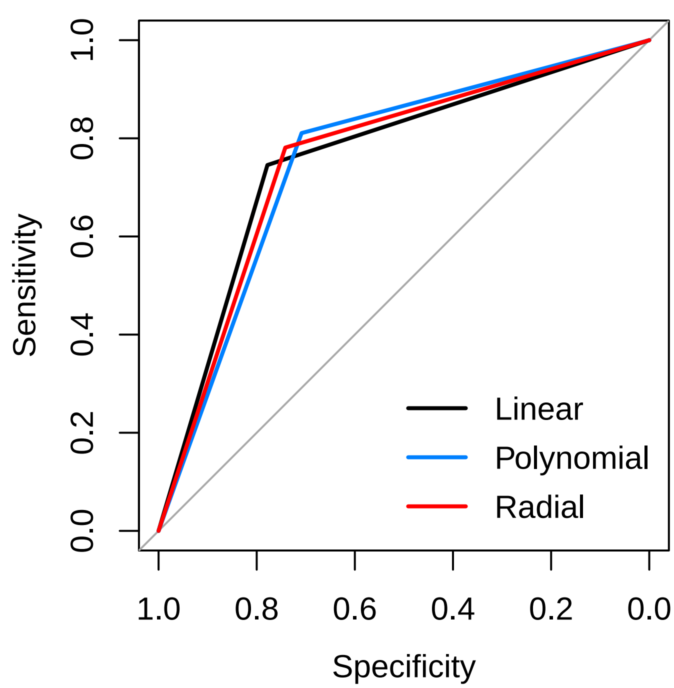


Figure 1: ROC curves of testing results by ranking the decision values.

□

c)

Compute the AUC (Area Under Curve), which kernel is the best?

As we can see by the Figure 1, the AUC's (and the ROC curves) are very similar. However, by the AUC measure we see that the linear kernel is, very slightly, the best (present the bigger, slightly, AUC).

```
# <r code> =================================================================== #
auc(roc.linear)
auc(roc.poly)
auc(roc.radial)
# </r code> ================================================================== #
```

```
Area under the curve: 0.7619
Area under the curve: 0.7596
Area under the curve: 0.7614
```

□

[45 points] (2)

For each learning model (each type of kernel), Use 5-fold cross validation for setting the parameters of training process. Please note different kernels may have different parameters to set. After cross validation, choose the best parameter setting, train the model by 60% of data again (the same data used in (1)), test the model by the remaining 40% of data.

OBS. By default the method `svm` uses a classification machine algorithm, C-classification, and scale the variables to zero mean and unit variance.

Linear kernel: $k(x_i, x_j) = \gamma \|x_i, x_j\|$

C : default cost of constraints violation in `e1071::svm` is 1;

γ : default value in `e1071::svm` is 1/dataset size (number of variables).

```
# <r code> =================================================================== #
   # choosing best cost of constraints violation C by 5-fold Cross Validation (CV)
tune.linear <- tune(svm, quality ~ .        # using all the 11 available variables
                    , data = df.train
                    , kernel = "linear"
                    , tunecontrol = tune.control(cross = 5)           # 5-fold CV
                    , ranges = list(
                        # passing a grid of constraints violation C's, default: 1
                      cost = c(1e-3, 1e-2, .1, .5, 1, 2, 2.5, 3, 5, 7.5, 10),
                              # passing a grid of \gamma's, default: 1/11 = .0909
```

```
                     gamma = c(1/1000, 1/100, 1/33, 1/11, 1/3, 1/2, 1)))
tune.linear                                     # best C: .5 and \gamma: 1/1000
# </r code> ==================================================================== #
```

```
Parameter tuning of 'svm':

- sampling method: 5-fold cross validation

- best parameters:
 cost gamma
  0.5 0.001

- best performance: 0.2711551
```

```
# <r code> ===================================================================== #
                               # fitting SVM with linear kernel (training dataset)
linear_tune.train <- svm(quality ~ ., df.train  # using all 11 available variables
                         , kernel = "linear"
                         , cross = 5                                    # 5-fold CV
                         , cost = .5        # chosen cost of constraints violation
                         , gamma = 1/1000)                          # chosen \gamma
                                                     # prediction in testing data
linear_tune.test <- predict(linear_tune.train, df.test[ , -12])
# </r code> ==================================================================== #
```

Polynomial kernel: $k(x_i, x_j) = (c_0 + \gamma \|x_i, x_j\|)^d$

C : default cost of constraints violation in `e1071::svm` is 1;

c_0 : default value in `e1071::svm` is 0;

γ : default value in `e1071::svm` is 1/dataset size (number of variables);

d : polynominal degree, default value in `e1071::svm` is 3.

```
# <r code> ===================================================================== #
                    # choosing best cost of constraints violation C, c_{0}, \gamma
                    # and kernel polynomial degree by 5-fold Cross Validation (CV)
tune.poly <- tune(svm, quality ~ .          # using all the 11 available variables
                  , data = df.train
                  , kernel = "polynomial"
                  , tunecontrol = tune.control(cross = 5)               # 5-fold CV
                  , ranges = list(
                         # passing a grid of constraints violation C's, default: 1
                    cost = c(1e-3, 1e-2, .1, .5, 1, 2, 2.5, 3, 5),
                                         # passing a grid of c_{0}'s, default: 0
                    coef0 = c(-1, -.5, 0, .5, 1),
                               # passing a grid of \gamma's, default: 1/11 = .0909
```

```r
                        gamma = c(1/1000, 1/100, 1/33, 1/11, 1/3, 1/2, 1),
                                  # passing a grid of polynomial degrees, default: 3
                      degree = c(2, 3, 4, 5)))
tune.poly          # best C: .5; c_{0}: 1; \gamma: 1/11; and kernel poly degree: 3
# </r code> ==================================================================== #
```

```

Parameter tuning of 'svm':

- sampling method: 5-fold cross validation

- best parameters:
 cost coef0      gamma degree
  0.5     1 0.09090909      3

- best performance: 0.2356348
```

```r
# <r code> ===================================================================== #
                           # fitting SVM with polynomial kernel (training dataset)
poly_tune.train <- svm(quality ~ ., df.train    # using all 11 available variables
                       , kernel = "polynomial"
                       , cross = 5                                    # 5-fold CV
                       , cost = .5          # chosen cost of constraints violation
                       , coef0 = 1                                   # chosen c_{0}
                       , gamma = 1/11                               # chosen \gamma
                       , degree = 3)             # chosen kernel polynomial degree
                                                      # prediction in testing data
poly_tune.test <- predict(poly_tune.train, df.test[ , -12])
# </r code> ==================================================================== #
```

Radial kernel: $k(x_i, x_j) = \exp(-\gamma \|x_i, x_j\|)^2$

C : default cost of constraints violation in `e1071::svm` is 1;

γ : default value in `e1071::svm` is 1/dataset size (number of variables).

```r
# <r code> ===================================================================== #
                        # choosing best cost of constraints violation C and \gamma
                        #                          by 5-fold Cross Validation (CV)
tune.radial <- tune(svm, quality ~ .        # using all the 11 available variables
                    , data = df.train
                    , kernel = "radial"
                    , tunecontrol = tune.control(cross = 5)           # 5-fold CV
                    , ranges = list(
                         # passing a grid of constraints violation C's, default: 1
                      cost = c(1e-3, 1e-2, .1, .5, 1, 2, 2.5, 3, 5, 7.5, 10),
                               # passing a grid of \gamma's, default: 1/11 = .0909
```

```
                         gamma = c(1/1000, 1/100, 1/33, 1/11, 1/3, 1/2, 1)))
tune.radial                                          # best C: 5 and \gamma: 1/11
# </r code> ================================================================== #
```

```
Parameter tuning of 'svm':

- sampling method: 5-fold cross validation

- best parameters:
 cost      gamma
    5 0.09090909

- best performance: 0.2387707
```

```
# <r code> =================================================================== #
                               # fitting SVM with radial kernel (training dataset)
radial_tune.train <- svm(quality ~ ., df.train  # using all 11 available variables
                 , kernel = "radial"
                 , cross = 5                                           # 5-fold CV
                 , cost = 5                # chosen cost of constraints violation
                 , gamma = 1/11)                                   # chosen \gamma
                                                     # prediction in testing data
radial_tune.test <- predict(radial_tune.train, df.test[ , -12])
# </r code> ================================================================== #
```

OBS. Using 5-fold CV for setting the parameters of training process we saw that for each kernel we obtain a different cost of constraints violation, C, varying from 0.5 to 5. For the polynomial kernel the best polyminal degree wasn't the default, 3, and for this kernel and for the radial kernel the best γ was the default value in the `e1071::svm` implementation.

a)

Report the setting of parameters.

OBS. The step-by-step of the setting of parameters is/was shown above.

```
# <r code> =================================================================== #
                            # SVM with linear kernel: reporting setting parameters
linear_tune.train$call
# </r code> ================================================================== #
```

```
svm(formula = quality ~ ., data = df.train, kernel = "linear",
    cross = 5, cost = 0.5, gamma = 1/1000)
```

$k(x_i, x_j) = \gamma \| x_i, x_j \|$

C : chosen cost of constraints violation: 0.5. (default vaule in `e1071::svm`: 1);

γ : chosen: 0.001. (default value in `e1071::svm`: 1/number of variables = 0.0909).

```
# <r code> =================================================================== #
                          # SVM with polynomial kernel: reporting setting parameters
poly_tune.train$call
# </r code> ================================================================== #
```

```
svm(formula = quality ~ ., data = df.train, kernel = "polynomial",
    cross = 5, cost = 0.5, coef0 = 1, gamma = 1/11, degree = 3)
```

$k(x_i, x_j) = (c_0 + \gamma \|x_i, x_j\|)^d$

C : chosen cost of constraints violation: 0.5. (default vaule in `e1071::svm`: 1);

c_0 : chosen: 1. (default value in `e1071::svm`: 0);

γ : chosen the default value in `e1071::svm`: 1/number of variables = 0.0909;

d : chosen the default polynominal degree in `e1071::svm`: 3).

```
# <r code> =================================================================== #
                              # SVM with radial kernel: reporting setting parameters
radial_tune.train$call
# </r code> ================================================================== #
```

```
svm(formula = quality ~ ., data = df.train, kernel = "radial",
    cross = 5, cost = 5, gamma = 1/11)
```

$k(x_i, x_j) = \exp(-\gamma \|x_i, x_j\|)^2$

C : chosen cost of constraints violation: 5. (default vaule in `e1071::svm`: 1);

γ : chosen the default value in `e1071::svm`: 1/number of variables = 0.0909.

□

b)

Report the number of support vectors.

Linear kernel : 589 support vectors (the dataset have 959 samples).
295 of the class label **negative** and 294 of the class label **positive**.
(with the default parameters the number of support vectors was 588.)

```
# <r code> =================================================================== #
summary(linear_tune.train)
# </r code> ================================================================== #
```

```
Call:
svm(formula = quality ~ ., data = df.train, kernel = "linear",
    cross = 5, cost = 0.5, gamma = 1/1000)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.5
      gamma:  0.001

Number of Support Vectors:  589

 ( 295 294 )


Number of Classes:  2

Levels:
 negative positive

5-fold cross-validation on training data:

Total Accuracy: 72.05422
Single Accuracies:
 75.39267 68.22917 70.83333 71.35417 74.47917
```

Polynomial kernel : 558 support vectors (the dataset have 959 samples). 279 of the class label **negative** and 279 of the class label **positive**. (with the default parameters the number of support vectors was 659.)

```
# <r code> =================================================================== #
summary(poly_tune.train)
# </r code> ================================================================== #
```

```
Call:
svm(formula = quality ~ ., data = df.train, kernel = "polynomial",
    cross = 5, cost = 0.5, coef0 = 1, gamma = 1/11, degree = 3)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.5
     degree:  3
      gamma:  0.09090909
     coef.0:  1
```

```
Number of Support Vectors:  558

 ( 279 279 )


Number of Classes:  2

Levels:
 negative positive

5-fold cross-validation on training data:

Total Accuracy: 74.76538
Single Accuracies:
 75.91623 69.27083 74.47917 77.60417 76.5625
```

Radial kernel : 564 support vectors (the dataset have 959 samples).
288 of the class label **negative** and 276 of the class label **positive**.
(with the default parameters the number of support vectors was 605.)

```r
# <r code> ================================================================== #
summary(radial_tune.train)
# </r code> ================================================================= #
```

```
Call:
svm(formula = quality ~ ., data = df.train, kernel = "radial",
    cross = 5, cost = 5, gamma = 1/11)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  5
      gamma:  0.09090909

Number of Support Vectors:  564

 ( 288 276 )


Number of Classes:  2

Levels:
 negative positive

5-fold cross-validation on training data:
```

```
Total Accuracy: 76.74661
Single Accuracies:
 72.77487 75.52083 78.125 79.6875 77.60417
```

□

c)

Plot the ROC curve of testing results by ranking the decision values (3 curves in one figure).

```
# <r code> =================================================================== #
                                 # computing roc curves for the testing datasets
roc.linear_tune <- roc(df.test$quality, as.numeric(linear_tune.test))
roc.poly_tune   <- roc(df.test$quality, as.numeric(poly_tune.test))
roc.radial_tune <- roc(df.test$quality, as.numeric(radial_tune.test))
                                                          # plotting roc curves
plot(roc.linear_tune)
plot(roc.poly_tune, col = "#0080ff", add = TRUE)
plot(roc.radial_tune, col = 2, add = TRUE)
                                                                # adding legend
legend(x = .5675, y = .35, legend = c("Linear", "Polynomial", "Radial")
       , bty = "n", col = c(1, "#0080ff", 2), lwd = 2)
# </r code> ================================================================== #
```

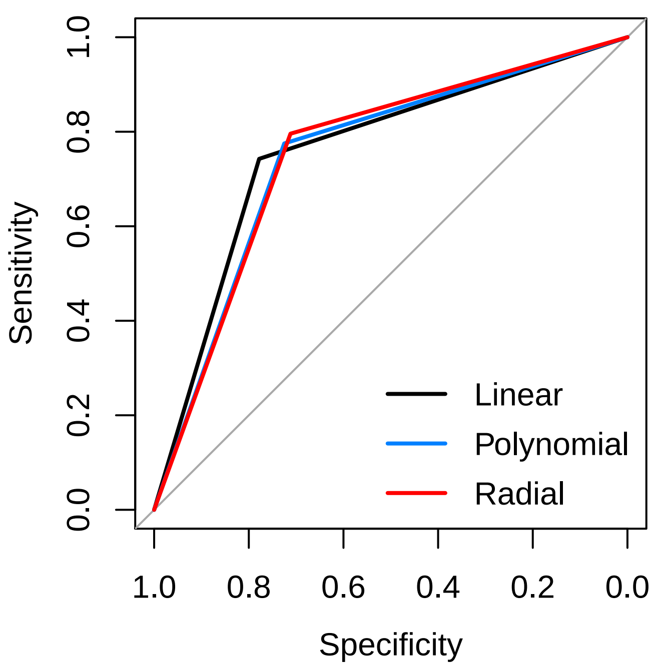


Figure 2: ROC curves of testing results by ranking the decision values.

□

d)

Compute the AUC (Area Under Curve), which kernel is the best?

As we can see by the Figure 2, the AUC's (and the ROC curves) are very similar. However, by the AUC measure we see that the linear kernel is, slightly, the best (present the bigger AUC).

```
# <r code> ================================================================== #
auc(roc.linear_tune)
auc(roc.poly_tune)
auc(roc.radial_tune)
# </r code> ================================================================= #
```

```
Area under the curve: 0.7604
Area under the curve: 0.7502
Area under the curve: 0.7539
```

□

[10 points] (3)

Make a table for comparing the AUC of different kernels with different setting of parameters (totally 6 AUC values), report the 3 best models (decided by their AUC values).

Table 1: AUC's of different kernels with the default and setting parameters. In bold is highlighted the best performance for each kernel and in red the best of all.

Linear kernel		Polynomial kernel		Radial kernel	
Default parameters	**Setting parameters**	**Default parameters**	**Setting parameters**	**Default parameters**	**Setting parameters**
0.7619	0.7604	**0.7596**	0.7502	**0.7614**	0.7539

We see in Table 1 that for all the kernels the best, slightly, results are obtained with the default parameters. The setting parameters was obtained with the training dataset and the AUC's presented in Table 1 was obtained with the testing dataset. This means that that the setting parameters generate the best results for the training data, not for the testing (in this case, with this dataset). Neverthless, the AUC's are very similar. The best AUC is obtained with the linear kernel (but the difference to the worst is of only 0.0117).

■
