Chase Moreno

@2078503

MATH 4323

Homework 4

1. Give a real-life data example for each of the following three cases:

(a) False negatives are less tolerable than false positives.

Pregnancy testing. If a woman receives a false negative result, then she might put off going to the doctor, missing out on necessary prenatal treatment.

(b) False positives are less tolerable than false negatives.

Fraud detection. If a person detected a fraud, then the person is innocent after the investigation is done. So, it wasted time and cause frustration to that person.

(c) False positives and false negatives are of equivalent importance.

Covid-19 Testing. If a person receives a false positive result, then it requires them to isolate even if they don't have covid-19. Causing them to miss work, school, or else. If a person receives a false negative result, then they don't have covid-19 and don't have to isolate. However, if that person really do have covid-19 then it is spreading the virus and infecting others.

2. In Section 10.2.3, a formula for calculating PVE was given in Equation 10.8. We also saw that the PVE can be obtained using the sdev output of the prcomp() function. On the USArrests data, calculate the PVE in two ways:

(a) Using the sdev output of the prcomp() function, as was done in Section 10.2.3.

My R code:

```
> data("USArrests")
> pr.out = prcomp(USArrests, scale = T)
> pr.var=pr.out$sdev^2
> pr.var
[1] 2.4802416 0.9897652 0.3565632 0.1734301
> pve = pr.var/sum(pr.var)
> pve
[1] 0.62006039 0.24744129 0.08914080 0.04335752
```

(b) By applying Equation 10.8 directly. That is, use the prcomp() function to compute the principal component loadings. Then, use those loadings in Equation 10.8 to obtain the PVE.

My R code:

```
> pr.out$rotation
                PC1        PC2        PC3         PC4
Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
> USArrests.scale = scale(USArrests)
> sumValue = colSums((USArrests.scale %*% pr.out$rotation)^2)
> sumValue
       PC1        PC2        PC3        PC4
121.531837  48.498492  17.471596   8.498074
> sum(sumValue)
[1] 196
> sumValue/sum(sumValue)
       PC1        PC2        PC3        PC4
0.62006039 0.24744129 0.08914080 0.04335752
```

3. Generate a simulated two-class data set with 200 observations and two features in which there is a visible but non-linear separation between the two classes. For example, you could do

```
set.seed(1)

x1 <- rnorm(200)

x2 <- 4 * x1^2 + 1 + rnorm(200)

y <- as.factor(c(rep(1,100), rep(-1,100)))

x2[y==1] <- x2[y==1] + 3

x2[y==-1] <- x2[y==-1] – 3

plot(x1[y==1], x2[y==1], col = "red", xlab = "X", ylab = "Y", ylim = c(-6, 30))

points(x1[y==-1], x2[y==-1], col = "blue")

dat <- data.frame(x1,x2,y)
```

(a) Subdivide the data 80%/20% into training and test subsets. Use set.seed(1) when generating this random split.

My R code:

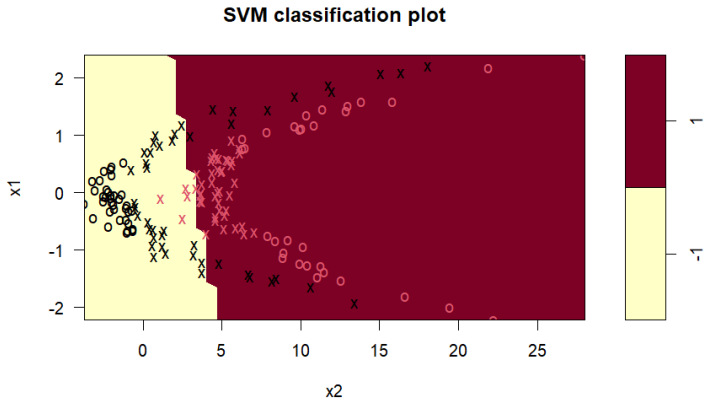
```
> set.seed(1)
> n <-nrow(dat)
> train <-sample(1:n, 0.8 * n)
> test <- sample(1:n, 0.2 * n)
```

(b) (Please make sure to use set.seed(1) once again prior to each tune() operation.) On training subset, use tune() function - in similar fashion to what we did in Lab #5 - to select optimal:

i. support vector classifier model with respect to cost value,

My R code:

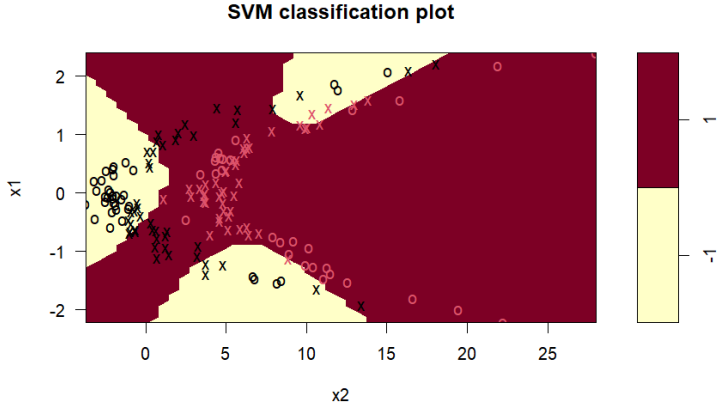
```
> library(e1071)
> set.seed(1)
> tune.out <- tune(svm,
+                  y~., data=dat[train,],
+                  kernel="linear",
+                  ranges=list(cost=c(0.1,1,10,100,1000)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.15

- Detailed performance results:
   cost  error dispersion
1 1e-01 0.1875 0.08838835
2 1e+00 0.1500 0.08436857
3 1e+01 0.1625 0.06718548
4 1e+02 0.1625 0.06718548
5 1e+03 0.1625 0.06718548

> plot(tune.out$best.model, data = dat[train,])
```



ii. polynomial SVM model with respect to cost and degree values,

```
> plot(tune.out$best.model, data = dat[train,])
> set.seed(1)
> tune.out.poly <- tune(svm,
+                       y~., data=dat[train,],
+                       kernel="poly",
+                       ranges=list(cost=c(0.1,1,10,100,1000),
```

```
+                     degree=c(2,3)))
> summary(tune.out.poly)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
 1000      3

- best performance: 0.225

- Detailed performance results:
    cost degree   error dispersion
1  1e-01      2 0.55000  0.1243036
2  1e+00      2 0.52500  0.1564582
3  1e+01      2 0.53750  0.1748015
4  1e+02      2 0.53750  0.1748015
5  1e+03      2 0.53750  0.1748015
6  1e-01      3 0.43125  0.1299372
7  1e+00      3 0.28125  0.1187683
8  1e+01      3 0.25000  0.1141089
9  1e+02      3 0.23125  0.1104363
10 1e+03      3 0.22500  0.1185854

> plot(tune.out.poly$best.model, data = dat[train,])
```

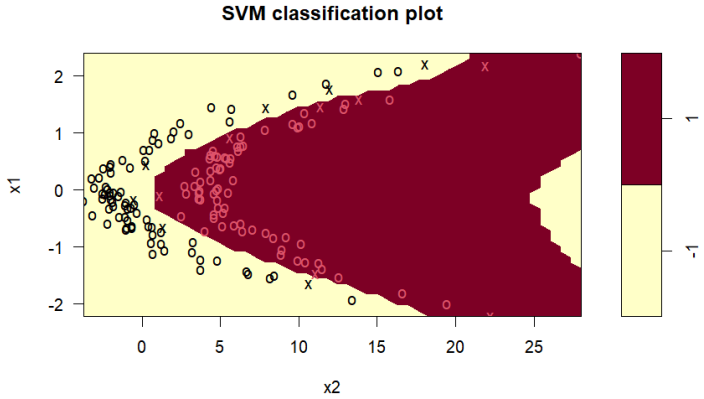


iii. radial SVM model with respect to cost and gamma values.

```
> plot(tune.out.poly$best.model, data = dat[train,])
> set.seed(1)
> tune.out.rad <- tune(svm,
+                      y~., data=dat[train,],
+                      kernel="radial",
+                      ranges=list(cost=c(0.1,1,10,100,1000),
+                      gamma=c(0.5,1,2,3,4)))
> summary(tune.out.rad)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  100   0.5

- best performance: 0
```

```
- Detailed performance results:
    cost gamma   error dispersion
1  1e-01   0.5 0.12500 0.07216878
2  1e+00   0.5 0.04375 0.05929271
3  1e+01   0.5 0.01250 0.03952847
4  1e+02   0.5 0.00000 0.00000000
5  1e+03   0.5 0.00000 0.00000000
6  1e-01   1.0 0.11250 0.08228507
7  1e+00   1.0 0.04375 0.07822910
8  1e+01   1.0 0.00625 0.01976424
9  1e+02   1.0 0.00625 0.01976424
10 1e+03   1.0 0.00625 0.01976424
11 1e-01   2.0 0.10000 0.06718548
12 1e+00   2.0 0.03125 0.04419417
13 1e+01   2.0 0.00625 0.01976424
14 1e+02   2.0 0.00625 0.01976424
15 1e+03   2.0 0.00625 0.01976424
16 1e-01   3.0 0.09375 0.07933097
17 1e+00   3.0 0.01875 0.04218428
18 1e+01   3.0 0.00625 0.01976424
19 1e+02   3.0 0.00625 0.01976424
20 1e+03   3.0 0.00625 0.01976424
21 1e-01   4.0 0.09375 0.07365696
22 1e+00   4.0 0.01875 0.04218428
23 1e+01   4.0 0.01875 0.03019037
24 1e+02   4.0 0.01875 0.03019037
25 1e+03   4.0 0.01875 0.03019037
```

```
> plot(tune.out.rad$best.model, data = dat[train,])
```



For each model: provide code, report the selected optimal tuning parameter values, plot the fitted boundary.

(c) For each optimal model from part (b), calculate

i. Training error. Which model is best with respect to training error?

```
> mean(predict(tune.out$best.model) != y[train])
[1] 0.14375

> mean(predict(tune.out.poly$best.model) != y[train])
[1] 0.20625

> mean(predict(tune.out.rad$best.model) != y[train])
[1] 0
```

The radial SVM model has the best training error which is 0.

ii. Test error. Which model is best with respect to test error?

```
> mean(predict(tune.out$best.model, newdata = dat[-train,]) != y[
-train])
[1] 0.175

> mean(predict(tune.out.poly$best.model, newdata = dat[-train,])
!= y[-train])
[1] 0.15

> mean(predict(tune.out.rad$best.model, newdata = dat[-train,]) !
= y[-train])
[1] 0
```

The radial SVM model has the best test error which is 0.

4. In this problem, you will use support vector machines in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

My R code:

```
> library(ISLR)
> data("Auto")
> Auto$mpg01 <- as.factor(ifelse(Auto$mpg > median(Auto$mpg), 1, 0))
> Auto$mpg <- NULL
```

(b) Fit a support vector classifier to the data for a grid of cost values, c(0.01, 0.1, 1, 5, 10, 100) (via tune() function), in order to predict whether a car gets high or low gas mileage. Report the optimal cost value, the optimal model's corresponding training error and cross-validation error.

```
> set.seed(1)
> tune.out <- tune(svm,
+                  mpg01~., data = Auto,
+                  kernel="linear",
+                  ranges=list(cost=c(0.01,0.1,1,5,10,100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.09179487

- Detailed performance results:
   cost      error dispersion
1 1e-02 0.09179487 0.05812971
2 1e-01 0.09692308 0.06369443
3 1e+00 0.09179487 0.04543280
4 5e+00 0.10198718 0.04338864
5 1e+01 0.11480769 0.05828005
6 1e+02 0.11730769 0.06521821
```

The optimal cost value is 1 which I did from doing this:

```
> print(tune.out$best.parameters)
  cost
3    1
```

The optimal training error is 0.02806122 which I did from doing this:

```
> mean(predict(tune.out$best.model) != Auto$mpg01)
[1] 0.02806122
```

The optimal cross-validation error is 0.09179487 which I did from doing this:

```
> print(tune.out$best.performance)
[1] 0.09179487
```

(c) Now repeat (b), this time using SVM with polynomial kernels for the same grid of cost values, but now also for two degree values - degree = 2, 3 and 4 (similar to how we picked gamma for radial kernel in Lab #5). Report the optimal parameter values, the optimal model's corresponding training error and cross-validation error.

```
> set.seed(1)
> tune.out.poly <- tune(svm,
+                   mpg01~., data = Auto,
+                   kernel="poly",
+                   ranges=list(cost=c(0.01,0.1,1,5,10,100),
+                   degree = c(2,3,4)))
> summary(tune.out.poly)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
  100      2

- best performance: 0.3115385

- Detailed performance results:
    cost degree     error dispersion
1  1e-02      2 0.5611538 0.04344202
2  1e-01      2 0.5611538 0.04344202
3  1e+00      2 0.5611538 0.04344202
4  5e+00      2 0.5611538 0.04344202
5  1e+01      2 0.5558333 0.04951408
6  1e+02      2 0.3115385 0.08212248
7  1e-02      3 0.5611538 0.04344202
8  1e-01      3 0.5611538 0.04344202
9  1e+00      3 0.5611538 0.04344202
10 5e+00      3 0.5611538 0.04344202
11 1e+01      3 0.5611538 0.04344202
12 1e+02      3 0.3858974 0.13255994
13 1e-02      4 0.5611538 0.04344202
14 1e-01      4 0.5611538 0.04344202
15 1e+00      4 0.5611538 0.04344202
16 5e+00      4 0.5611538 0.04344202
17 1e+01      4 0.5611538 0.04344202
18 1e+02      4 0.5611538 0.04344202
```

The optimal parameters value for cost is 100 and degree is 2. I get this from doing this:

```
> print(tune.out.poly$best.parameters)
  cost degree
6  100      2
```

The optimal training error is 0.2933673 which I did from doing this:

```
> mean(predict(tune.out.poly$best.model) != Auto$mpg01)
[1] 0.2933673
```

The optimal cross-validation error is 0.3115385 which I did from doing this:

```
> print(tune.out.poly$best.performance)
[1] 0.3115385
```

(d) Now repeat (b), this time using SVM with radial kernels for the same grid of cost values, but now also for a grid of gamma values in (0.01, 0.1, 1, 5). Report the optimal parameter values, the optimal model's corresponding training error and cross-validation error.

```
> set.seed(1)
> tune.out.rad <- tune(svm,
+                      mpg01~., data = Auto,
+                      kernel="radial",
+                      ranges=list(cost=c(0.01,0.1,1,5,10,100),
+                      gamma = c(0.01,0.1,1,5)))
> summary(tune.out.rad)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.1

- best performance: 0.07641026

- Detailed performance results:
    cost gamma      error dispersion
1  1e-02  0.01 0.56115385 0.04344202
2  1e-01  0.01 0.11237179 0.05585231
3  1e+00  0.01 0.08673077 0.05819036
4  5e+00  0.01 0.09179487 0.06178488
5  1e+01  0.01 0.08410256 0.05905312
6  1e+02  0.01 0.08929487 0.05571152
7  1e-02  0.10 0.25025641 0.09544717
8  1e-01  0.10 0.09179487 0.05812971
9  1e+00  0.10 0.08416667 0.05540253
10 5e+00  0.10 0.07897436 0.05705152
11 1e+01  0.10 0.07641026 0.05513715
12 1e+02  0.10 0.10461538 0.05724406
13 1e-02  1.00 0.56115385 0.04344202
14 1e-01  1.00 0.56115385 0.04344202
15 1e+00  1.00 0.07647436 0.05657355
16 5e+00  1.00 0.08160256 0.06250579
17 1e+01  1.00 0.08416667 0.06164376
18 1e+02  1.00 0.08416667 0.06164376
19 1e-02  5.00 0.56115385 0.04344202
```

```
20 1e-01  5.00 0.56115385 0.04344202
21 1e+00  5.00 0.48461538 0.04704835
22 5e+00  5.00 0.47185897 0.05585590
23 1e+01  5.00 0.47185897 0.05585590
24 1e+02  5.00 0.47185897 0.05585590
```

The optimal parameters value for cost is 10 and gamma is 0.1 and I get this from doing this:

```
> print(tune.out.rad$best.parameters)
   cost gamma
11   10   0.1
```

The optimal training error is 0.01020408 which I did from doing this:

```
> mean(predict(tune.out.rad$best.model) != Auto$mpg01)
[1] 0.01020408
```

The optimal cross-validation error is 0.07641026 which I did from doing this:

```
> print(tune.out.rad$best.performance)
[1] 0.07641026
```

(e) From parts (b) – (d), which model yielded best training error? Best cross-validation error?

The radial SVM model yielded the best training error which is 0.01020408. Also, the radial SVM model yielded the best cross-validation error which is 0.07641026.

5. This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. Make sure to use set.seed(1).

```
> library(ISLR)
> set.seed(1)
> dataOJ <- sample(1:nrow(OJ), 800)
> trainOJ <- OJ[dataOJ,]
> testOJ <- OJ[-dataOJ,]
```

(b) (Please make sure to use set.seed(1) once again prior to each tune() operation, for parts (c) – (d) as well.) Use tune() function to find the optimal support vector classifier with respect to cost value, using Purchase as the response and the other variables as predictors. Consider the following grid of values for cost: 0.01, 0.05, 0.1, 0.5, 1, 10. Provide code and report both the training error and test error of the optimal selected model.

```
> set.seed(1)
> tune.out <- tune(svm,
+                  Purchase~., data = trainOJ,
+                  kernel="linear",
+                  ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10)))
> summary(tune.out)
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.05

- best performance: 0.17125

- Detailed performance results:
   cost   error dispersion
1  0.01 0.17500 0.03996526
2  0.05 0.17125 0.03682259
3  0.10 0.17875 0.03821086
4  0.50 0.17625 0.03747684
5  1.00 0.17750 0.03717451
6 10.00 0.18000 0.04005205
```

The training error of the optimal selected model is 0.16125. I got this from doing:

```
> mean(predict(tune.out$best.model) != trainOJ$Purchase)
[1] 0.16125
```

The test error of the optimal selected model is 0.1888889. I got this from doing:

```
> test.pred <- predict(tune.out$best.model,testOJ)
> mean(testOJ$Purchase != test.pred)
[1] 0.1888889
```

(c) Repeat part (b) using an SVM with a polynomial kernel with degree = 3.

```
> set.seed(1)
> tune.out.poly <- tune(svm,
+                   Purchase~., data = trainOJ,
+                   kernel="poly",
+                   ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10),
+                   degree = 3))
> summary(tune.out.poly)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      3

- best performance: 0.18125

- Detailed performance results:
   cost degree   error dispersion
1  0.01      3 0.36750 0.05041494
2  0.05      3 0.32750 0.06118052
3  0.10      3 0.30250 0.07402139
4  0.50      3 0.19750 0.05458174
5  1.00      3 0.18750 0.05368374
6 10.00      3 0.18125 0.04686342
```

The training error of the optimal selected model is 0.12875. I got this from doing:

```
> mean(predict(tune.out.poly$best.model) != trainOJ$Purchase)
[1] 0.12875
```

The test error of the optimal selected model is 0.1925926. I got this from doing:

```
> test.pred <- predict(tune.out.poly$best.model,testOJ)
> mean(testOJ$Purchase != test.pred)
[1] 0.1925926
```

(d) Repeat part (b) using an SVM with a radial kernel with default gamma value (gamma = 2).

```
> set.seed(1)
> tune.out.rad <- tune(svm,
+                      Purchase~., data = trainOJ,
+                      kernel="radial",
+                      ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10),
+                      gamma = 2))
> summary(tune.out.rad)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1     2

- best performance: 0.21375

- Detailed performance results:
   cost gamma   error dispersion
1  0.01     2 0.38250 0.05596378
2  0.05     2 0.38250 0.05596378
3  0.10     2 0.36250 0.06373774
4  0.50     2 0.22250 0.05062114
5  1.00     2 0.21375 0.04910660
6 10.00     2 0.23500 0.04158325
```

The training error of the optimal selected model is 0.10125. I got this from doing:

```
> mean(predict(tune.out.rad$best.model) != trainOJ$Purchase)
[1] 0.10125
```

The test error of the optimal selected model is 0.2222222. I got this from doing:

```
> test.pred <- predict(tune.out.rad$best.model,testOJ)
> mean(testOJ$Purchase != test.pred)
[1] 0.2222222
```

(e) Which approach gave best results in terms of training error? Test error?

The radial SVM model gave the best result for training error which is 0.10125. The linear SVM model gave the best result for test error which is 0.1888889.

6. For iris data set (that is available in base R, simply type in "iris"), which includes measurements on three species of iris flower, proceed to:

(a) Subdivide the data 80%/20% into training and test subsets. Use set.seed(1) when generating this random split.

```
> set.seed(1)
> data.iris <- sample(nrow(iris), 0.8 * nrow(iris))
> train.iris <- iris[data.iris,]
> test.iris <- iris[-data.iris,]
```

(b) (Please make sure to use set.seed(1) once again prior to each tune() operation.) On training subset, use tune() function - in similar fashion to what we did in Lab #5 - to select optimal:

i. support vector classifier model with respect to cost value,

```
> set.seed(1)
> tune.out <- tune(svm,
+                  Species~., data = train.iris,
+                  kernel="linear",
+                  ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10)))
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.03333333

- Detailed performance results:
   cost      error dispersion
1  0.01 0.11666667 0.10540926
2  0.05 0.08333333 0.09622504
3  0.10 0.04166667 0.05892557
4  0.50 0.04166667 0.05892557
5  1.00 0.03333333 0.05826716
6 10.00 0.05000000 0.05826716
```

The selected optimal tuning parameter value for cost is 1. Which I got it from doing:

```
> print(tune.out$best.parameters)
  cost
5    1
```

ii. polynomial SVM model with respect to cost and degree values,

```
> set.seed(1)
> tune.out.poly <- tune(svm,
+                  Species~., data = train.iris,
+                  kernel="poly",
+                  ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10),
+                  degree=c(2,3)))
> summary(tune.out.poly)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation
```

```
- best parameters:
 cost degree
   10      3

- best performance: 0.04166667

- Detailed performance results:
    cost degree      error dispersion
1   0.01      2 0.71666667 0.07027284
2   0.05      2 0.36666667 0.08958064
3   0.10      2 0.25000000 0.11785113
4   0.50      2 0.18333333 0.12909944
5   1.00      2 0.16666667 0.12422600
6  10.00      2 0.12500000 0.09001029
7   0.01      3 0.44166667 0.17145888
8   0.05      3 0.21666667 0.13146844
9   0.10      3 0.14166667 0.13058510
10  0.50      3 0.10833333 0.11145779
11  1.00      3 0.10833333 0.09662515
12 10.00      3 0.04166667 0.07081972
```

The selected optimal tuning parameter value for cost is 10 and degree is 3. I got it from doing:

```
> print(tune.out.poly$best.parameters)
   cost degree
12   10      3
```

iii. radial SVM model with respect to cost and gamma values. For each model: provide code, report the selected optimal tuning parameter values. NO NEED to plot the boundary.

```
> set.seed(1)
> tune.out.rad <- tune(svm,
+                      Species~., data = train.iris,
+                      kernel="radial",
+                      ranges=list(cost=c(0.01,0.05,0.1,0.5,1,10
),
+                      gamma=c(0.25,0.5,1,2,3,4)))
> summary(tune.out.rad)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1  0.25

- best performance: 0.025

- Detailed performance results:
    cost gamma      error dispersion
1   0.01  0.25 0.71666667 0.07027284
2   0.05  0.25 0.21666667 0.15811388
3   0.10  0.25 0.13333333 0.08958064
4   0.50  0.25 0.05833333 0.08827916
5   1.00  0.25 0.02500000 0.05624571
6  10.00  0.25 0.04166667 0.05892557
7   0.01  0.50 0.71666667 0.07027284
8   0.05  0.50 0.24166667 0.13861083
```

```
9   0.10  0.50 0.11666667 0.09781565
10  0.50  0.50 0.05000000 0.07027284
11  1.00  0.50 0.03333333 0.05826716
12 10.00  0.50 0.05000000 0.05826716
13  0.01  1.00 0.71666667 0.07027284
14  0.05  1.00 0.37500000 0.16315750
15  0.10  1.00 0.10833333 0.11145779
16  0.50  1.00 0.04166667 0.05892557
17  1.00  1.00 0.05000000 0.05826716
18 10.00  1.00 0.05833333 0.05624571
19  0.01  2.00 0.71666667 0.07027284
20  0.05  2.00 0.54166667 0.11947674
21  0.10  2.00 0.23333333 0.14593251
22  0.50  2.00 0.05000000 0.05826716
23  1.00  2.00 0.05000000 0.05826716
24 10.00  2.00 0.05833333 0.05624571
25  0.01  3.00 0.71666667 0.07027284
26  0.05  3.00 0.71666667 0.07027284
27  0.10  3.00 0.36666667 0.13146844
28  0.50  3.00 0.05833333 0.06860605
29  1.00  3.00 0.05000000 0.07027284
30 10.00  3.00 0.05833333 0.06860605
31  0.01  4.00 0.71666667 0.07027284
32  0.05  4.00 0.71666667 0.07027284
33  0.10  4.00 0.47500000 0.12453618
34  0.50  4.00 0.08333333 0.10393493
35  1.00  4.00 0.05833333 0.06860605
36 10.00  4.00 0.05000000 0.07027284
```

The selected optimal tuning parameter value for cost is 1 and gamma is 0.25. I got it from doing:

```
> print(tune.out.rad$best.parameters)
  cost gamma
5    1  0.25
```

(c) For each optimal model from part (b), calculate

i. Training error. Which model (or models) is best with respect to training error?

```
> mean(predict(tune.out$best.model) != train.iris$Species)
[1] 0.03333333

> mean(predict(tune.out.poly$best.model) != train.iris$Species)
[1] 0.025

> mean(predict(tune.out.rad$best.model) != train.iris$Species)
[1] 0.01666667
```

The radial SVM model is best with respect to training error which is 0.01666667.

ii. Test error. Which model (or models) is best with respect to test error?

```
> test.pred <- predict(tune.out$best.model,test.iris)
> mean(test.iris$Species != test.pred)
[1] 0.03333333

> test.pred <- predict(tune.out.poly$best.model,test.iris)
> mean(test.iris$Species != test.pred)
[1] 0.06666667

> test.pred <- predict(tune.out.rad$best.model,test.iris)
```

```
> mean(test.iris$Species != test.pred)
[1] 0.06666667
```

The Linear model is best with respect to test error which is 0.03333333.

(d) Given that it was a three-class problem, K = 3, what two types of SVMs can one use for classification? Explain the main ideas behind each type (see slides). Which of those two types does the svm() function implement?

There are two types of SVMs can use for classification which is One-vs-One and One-vs-All.

For one-vs-one: construct (K choose 2) SVMs for all possible pairs of classes. It uses those models to get (K choose 2) class predictions for a test observation. Then assign the test observations the most frequent class among the (K choose 2) predictions.

For one-vs-all: for class k, k = 1, 2, 3, …, K. Fit SVM model like it compares the observations from class k like y = +1, with those are not in class such as y = -1. Then denote hyperplane parameters given that yi = +1 for arbitrary observation x in class k. Given test observation x and assign to the class k which get

$$k = \max_{k,k=1,\ldots,K} \beta_{0k} + \beta_{1k}x_1^* + \beta_{2k}x_2^* + \cdots + \beta_{pk}x_p^*$$

A high level of confidence that the test observation belongs to the k^{th} class rather than other classes.

Use One-vs-One for svm() function implement because if K is not too large, then One-vs-All approach will not be good to use.

7. In this problem, you will practice PCA on the BOSTON dataset from the library MASS.

(a) First, scan through the dataset and remove the categorical variable(s) from the dataset, because PCA only works for numerical variables.

My R Code:

```
> library(MASS)
> data(Boston)
> str(Boston)
'data.frame':     506 obs. of  14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.8
7 ...
 $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524
0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9
...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
```

```
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.
2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9
...
> Boston <- Boston[,-4]
> str(Boston)
'data.frame':     506 obs. of  13 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.8
7 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524
0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9
...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.
2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9
...
```

(b) Remove the variable medv so you can perform PCA on the rest of the variables.

```
> Boston <- Boston[,-13]
> str(Boston)
'data.frame':     506 obs. of  12 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.8
7 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524
0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9
...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.
2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
```

(c) Use prcomp() function in R to perform PCA. Make sure that you set scale = TRUE, so R will do the data scaling for you automatically. Report the standard deviation of each principle component.

```
> pr <- prcomp(Boston, scale = T)
> summary(pr)
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9
Standard deviation     2.4752 1.1587 1.08618 0.91382 0.81527 0.73308 0.62962 0.52637 0.46932
Proportion of Variance 0.5106 0.1119 0.09832 0.06959 0.05539 0.04478 0.03303 0.02309 0.01836
Cumulative Proportion  0.5106 0.6224 0.72075 0.79034 0.84573 0.89051 0.92355 0.94663 0.96499
                          PC10    PC11    PC12
Standard deviation     0.43146 0.41148 0.25426
Proportion of Variance 0.01551 0.01411 0.00539
Cumulative Proportion  0.98050 0.99461 1.00000
```

(d) Calculate the proportion of variance explained by each principle component. How many principle components are needed if you would like to explain at least 80% of the variation of the BOSTON dataset?

```
> pr.var <- pr$sdev^2
> pr.var / sum(pr.var)
          PC1           PC2           PC3           PC4           PC5
 [1] 0.510559900 0.111873274 0.098315403 0.069588829 0.055389284
          PC6           PC7
0.044783915 0.033034787
          PC8           PC9          PC10          PC11          PC12
 [8] 0.023088961 0.018355454 0.015513457 0.014109600 0.005387137

> cpr <- cumsum(pr.var / sum(pr.var))
> pc <- which(cpr > 0.8)[1]
> cat(pc,"\n")
5
```

Only need 5 principal components if you would like to explain at least 80% of the variation of the Boston dataset.