MATH 4323, Fall 2023, Homework # 4.

Name: Hyundoo Jeong
PSID:2212332

.

Conceptual.

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

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

   Answer: In medical testing for a deadly disease, missing a genuine case (false negative) can lead to a delayed diagnosis and potentially untreatable conditions, making it less tolerable than a false positive.

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

   Answer: In airport security screening, false positives (flagging innocent passengers or items) can lead to inconvenience, but the priority is safety, making them less tolerable compared to missing a genuine threat (false negative).

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

   Answer: In spam email filtering, both false positives and false negatives are equally important to strike a balance between not missing important messages and filtering out unwanted spam.

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.

   Answer:

```
data(USArrests)

USArrests <- scale(USArrests)

comp.arrests <- prcomp(USArrests)
```

```
comp.var <- comp.arrests$sdev^2

comp.var/sum(comp.var)
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.

```
Answer: USArrests %*%
    comp.arrests$rotation %>%
    (function(x) x^2) %>%
    as.tibble %>%
    rowwise() %>%
    map_df(sum) %>%
    ungroup() %>%
    (function(x) x/sum(USArrests^2))
    0.62006039 0.24744129 0.08914080 0.04335752
```

Applied.

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)
```

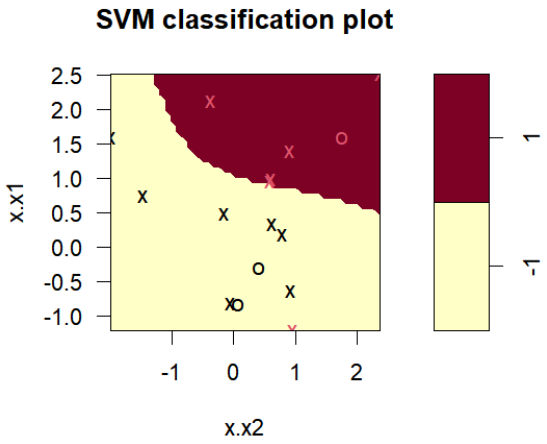
(a) Subdivide the data 80%/20% into training and test subsets. Use *set.seed*(1) when generating this random split.

Answer:
```
> set.seed(1)
> rand_ind <- sample(c(rep(1, 0.8 * nrow(dat)), rep(0, 0.2 * nrow(dat))))
> train_df = dat[rand_ind == 1,]
> test_df = dat[rand_ind == 0,]
```

(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,

Answer:
```
> set.seed(1)
> tuned_model <- tune(svm, y ~ ., data = train_df,
+                     ranges = list(cost = c(0.1,1,10,100,1000)))
> best_cost <- tuned_model$best.parameters$cost
> cat("Optimal cost parameter:", best_cost, "\n")
Optimal cost parameter: 1
```

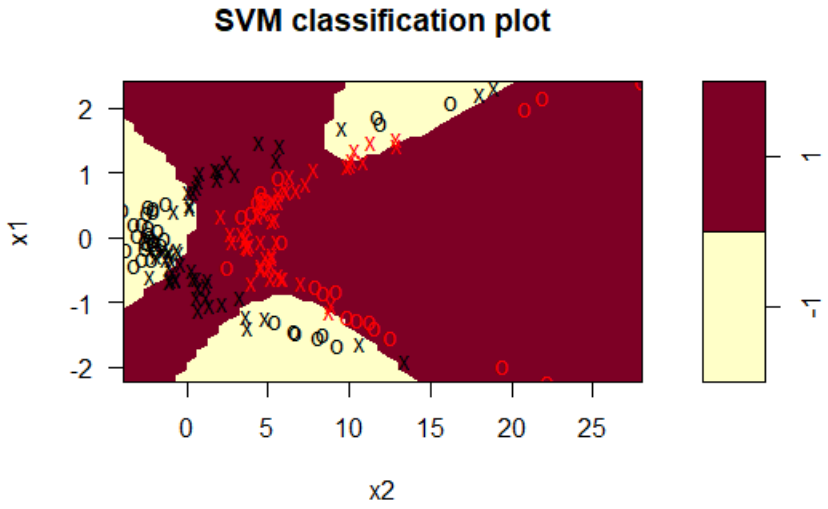


Plot:

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

Answer:
```
> set.seed(1)
> tuned_model <- tune(svm, y ~ ., data = train_df,
+                     kernel = "polynomial",
+                     ranges = list(cost = c(0.1, 1, 5, 10),
+                                   degree = c(2, 3, 4))
+ )
>
> best_cost <- tuned_model$best.parameters$cost
> cat("Optimal cost parameter:", best_cost, "\n")
Optimal cost parameter: 0.1
> best_degree<-tuned_model$best.parameters$degree
> cat("Optimal Degree parameter:", best_degree, "\n")
```

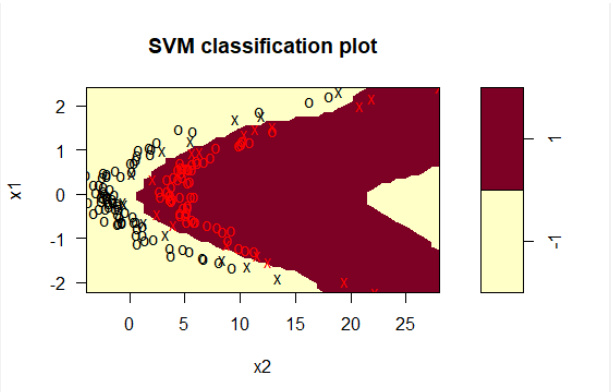
```
Optimal Degree parameter: 3
```



Plot:

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

Answer:
```
set.seed(1)
> tune.out=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)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
cost gamma
100 0.5
- best performance: 0.1
```



Plot:

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?

Answer: It is SVM-Radial because SVM-respect to cost value is also same training error, but it is not controlled by any parameter.

Training error is 0.0625.

*SVM-Polynomial's training error is 0.375

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

Answer: It is SVM-Radial because SVM-respect to cost value is also same test error ,but it is not controlled by any parameter.

Test error is 0.25.

*SVM-polynomial's test error is 0.5

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.

```
Answer: library(ISLR)
gas.med = median(Auto$mpg)
new.var = ifelse(Auto$mpg > gas.med, 1, 0)
Auto$mpglevel = as.factor(new.var)
```

(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.

```
Answer: > set.seed(1)
        > tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "lin
ear", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)))
        > summary(tune.out)
```

Cost value is 1. It is lowest error rate.
Training error: (1 + 18) / (I + 18 + 141 + 149) = 0.06
Cross-validation error:0.084

(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.

```
Answer: > set.seed(1)
> tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", range
s = list(cost = c(0.01, 0.1, 1, 5, 10, 100), degree = c(2, 3, 4)))
> summary(tune.out)
```

Cost value is 100 because it has the lowest error rate.
Training error: (93+ 0)/(66+93+154) = 0.29
Cross-validation error:0.316

(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.

```
Answer: > set.seed(1)
> tune.out = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges =
list(cost = c(0.01, 0.1, 1, 5, 10, 100), gamma = c(0.1, 1, 5, 10)))
> summary(tune.out)
```

Optimal cost value is 1.

Training error is 0(0/(159 + 154))

Cross-validation error is 0.078

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

Answer: It is SVM with radical because its training error rate is 0 which is the lowest among the models.

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).

```
Answer: > set.seed(1)
        > obs <- sample(1:nrow(OJ), 800)
        > train8 <- OJ[obs,]
        > test8 <- OJ[-obs,]
```

(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.

```
Answer: > set.seed(1)
> tuned_model <- tune(svm, Purchase ~ ., data = OJ,
+                     type = "C-classification",
+                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 10)))
```

Optimal Cost Value is 0.5
Training error: 0.1475
Test error: 0.1666667

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

Answer:
```
> set.seed(1)
> tuned_model <- tune(svm, Purchase ~ ., data = OJ,
+                     type = "C-classification",
+                     kernel = "polynomial",
+                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 10),
+                                   degree = 3
+                     )
+ )
```

Optical Cost value is 10.
The training error is 0.12875.
The test error is 0.1925926.

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

Answer:
```
> set.seed(1)
> tuned_model <- tune(svm, Purchase ~ ., data = OJ,
+                                      type = "C-classification",
+                                       kernel = "radial",
+                                       ranges = list(cost = c(0.01, 0.05,
0.1, 0.5, 1, 10),
+
              gamma = (2)
+                                                                          )
+                     )
> summary(tuned_model)
```

Optimal Cost value is 1.
The training error is 0.10125.
The Test error is 0.222222.

(e) Which approach gave best results in terms of training error? Test error?
Answer:
In terms of Training error, best model is SVM-radical because it is the lowest training error.
In terms of Test error, best model is SVM-respect to cost value because it is the lowest test error.

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.

Answer:
```
> set.seed(1)
> rand_ind <- sample(c(rep(1, 0.8 * nrow(iris)), rep(0, 0.2 * n
row(iris))))
> train_df = iris[rand_ind == 1,]
> test_df = iris[rand_ind == 0,]
```

(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

```
Answer: > set.seed(1)
> tuned_model <- tune(svm, Species ~ ., data = iris,
+                     type = "C-classification",
+                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 10)
+                     )
+ )
> summary(tuned_model)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1
```

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

```
Answer: > set.seed(1)
> tuned_model <- tune(svm, Species ~ ., data = iris,
+                     type = "C-classification",
+                     kernel = "polynomial",
+                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 10),
+                                   degree = 3
+                     )
+ )
> summary(tuned_model)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
   10      3
```

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

```
Answer: > set.seed(1)
> tuned_model <- tune(svm, Species ~ ., data = iris,
+                     type = "C-classification",
+                     kernel = "radial",
+                     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 10),
+                                   gamma=c(0.5,1,2,3,4)
+                     )
+ )
> summary(tuned_model)

Parameter tuning of ‘svm’:
```

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

- best parameters:
 cost gamma
    1   0.5
```

For each model: provide code, report the selected optimal tuning parameter values. NO NEED to plot the boundary.
(c) For each optimal model from part (b), calculate
i. Training error. Which model (or models) is best with respect to training error?
Answer: SVM-Polynomial and SVM-Radical are best with respect to value is 0.02.

SVM(cost):0.03, SVM(polynomial):0.02, SVM-radial:0.02.

ii. Test error. Which model (or models) is best with respect to test error?
Answer: SVM-radial is the best(value is 0.02).
SVM(cost):0.06, SVM(Poly):0.08, SVM-radial(0.02)

(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?

Answer:
In a three-class classification problem (K = 3), two common SVM-based approaches are One-Versus-One (OvO) and One-Versus-All (OvA). OvO creates binary classifiers for every pair of classes, while OvA creates K binary classifiers, one for each class, distinguishing it from the rest. The **svm()** function in R's **e1071** package typically implements the OvA approach, where it trains separate binary classifiers for each class and selects the class with the highest decision value during predictions.

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.

Answer: There is no categorical variable and everything are numerical variables.

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

```
Answer: Boston_num <- Boston[, -which(names(Boston_num) == "medv")]
```

(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.

```
Answer: pca <- prcomp(Boston_num1, scale = TRUE)
> summary(pca)$sdev
 [1] 2.4752472 1.1971947 1.1147272 0.9260535 0.9136826 0.8108065 0.7316803 0.6
293626 0.5262541 0.4692950
[11] 0.4312938 0.4114644 0.2520104
```

(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?

```
Answer: > stdev<-pca$sdev
> prop_var_explained <- stdev^2 / sum(stdev^2)
> cumulative_prop_var <- cumsum(prop_var_explained)
> cat("Proportion of Variance Explained by Each Principal Component:\n")
Proportion of Variance Explained by Each Principal Component:
> print(prop_var_explained)
 [1] 0.471296064 0.110251932 0.095585898 0.065967316 0.064216611 0.050569783 0.
041181237 0.030469024
 [9] 0.021303333 0.016941371 0.014308797 0.013023306 0.004885328
> cat("Cumulative Proportion of Variance Explained:\n")
Cumulative Proportion of Variance Explained:
> print(cumulative_prop_var)
 [1] 0.4712961 0.5815480 0.6771339 0.7431012 0.8073178 0.8578876 0.8990688 0.9
295379 0.9508412 0.9677826
[11] 0.9820914 0.9951147 1.0000000
```

(e) (For interested students, will not be graded) If you project the 506 observations onto the direction represented by the first principle component, on average, how far each projected point is away from their mean value?

```
Answer: num_components_needed <- sum(cumulative_prop_var < 0.8)
```

It is 4.