DS-6030 Homework Module 9

Matt Scheffel

DS 6030 | Spring 2022 | University of Virginia

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

```
library(ISLR)
data(Auto)
library(caret)
library(e1071)
library(tidyverse)
```

(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.

```
data(Auto)
Auto <- as_tibble(Auto)

# binary variable for high or low gas mileage
Auto <- Auto %>%
    mutate(high_mpg = as.integer(mpg > median(mpg))) %>%
    mutate(high_mpg = factor(high_mpg),
           cylinders = factor(cylinders))

head(Auto)
```

```
#> # A tibble: 6 x 10
#>     mpg cylinders displacement horse~1 weight accel~2  year origin name  high_~3
#>   <dbl> <fct>            <dbl>   <dbl>  <dbl>   <dbl> <dbl>  <dbl> <fct> <fct>
#> 1    18 8                  307     130   3504    12      70      1 chev~ 0
#> 2    15 8                  350     165   3693    11.5    70      1 buic~ 0
#> 3    18 8                  318     150   3436    11      70      1 plym~ 0
#> 4    16 8                  304     150   3433    12      70      1 amc ~ 0
#> 5    17 8                  302     140   3449    10.5    70      1 ford~ 0
#> 6    15 8                  429     198   4341    10      70      1 ford~ 0
#> # ... with abbreviated variable names 1: horsepower, 2: acceleration,
#> #   3: high_mpg
```

(b) Fit a support vector classifier to the data with various values of `cost`, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

```
Auto <- Auto %>%
    select(-mpg, -name)
```

```
dummy <- dummyVars(high_mpg ~ ., data = Auto)
Auto_Dummy <- predict(dummy, Auto)
```

```
Linear_SVM <- train(x = Auto_Dummy, y = Auto$high_mpg,
                    method = 'svmLinear2',
                    trControl = trainControl(method = 'cv', number = 10, allowParallel = TRUE),
                    preProcess = c('center', 'scale'),
                    tuneGrid = expand.grid(cost = seq(1, 20, by = 1)))
```

```
Linear_SVM$finalModel
```

```
#>
#> Call:
#> svm.default(x = as.matrix(x), y = y, kernel = "linear", cost = param$cost,
#>     probability = classProbs)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  2
#>
#> Number of Support Vectors:  75
```

The best-performing linear SVM model had a cost parameter of 11 and used a linear kernel. The output also shows that the number of support vectors in the final model is 74.

(c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of `gamma` and `degree` and `cost`. Comment on your results.

```
Polynomial_SVM <- train(x = Auto_Dummy, y = Auto$high_mpg,
                    method = 'svmPoly',
                    trControl = trainControl(method = 'cv', number = 10, allowParallel = TRUE),
                    preProcess = c('center', 'scale'),
                    tuneGrid = expand.grid(degree = seq(1, 8, by = 1),
                                           C = seq(1, 5, by = 1),
                                           scale = TRUE))
```

```
Polynomial_SVM$finalModel
```

```
#> Support Vector Machine object of class "ksvm"
#>
#> SV type: C-svc  (classification)
#>  parameter : cost C = 3
#>
#> Polynomial kernel function.
#>  Hyperparameters : degree =  1  scale =  TRUE  offset =  1
#>
#> Number of Support Vectors : 73
#>
#> Objective Function Value : -200.2077
#> Training error : 0.071429
```

The best-performing model had a cost parameter of 4, a degree parameter of 1, and used a polynomial kernel with hyper parameters of scale = TRUE and offset = 1. The number of support vectors in the final model was 73. Overall, the polynomial kernel SVM achieved a relatively high level of accuracy in cross-validation,

with a training error of 0.068878.

```
Radial_SVM <- train(x = Auto_Dummy, y = Auto$high_mpg,
                    method = 'svmRadial',
                    trControl = trainControl(method = 'cv', number = 10, allowParallel = TRUE),
                    preProcess = c('center', 'scale'),
                    tuneGrid = expand.grid(C = seq(0.001, 3, length.out = 10),
                                           sigma = seq(0.2, 2, length.out = 5)))
```

```
Radial_SVM$finalModel
```

```
#> Support Vector Machine object of class "ksvm"
#>
#> SV type: C-svc  (classification)
#>  parameter : cost C = 2.00033333333333
#>
#> Gaussian Radial Basis kernel function.
#>  Hyperparameter : sigma =  1.55
#>
#> Number of Support Vectors : 227
#>
#> Objective Function Value : -93.8982
#> Training error : 0.017857
```

The best-performing radial SVM model had a cost parameter of 0.334 and a sigma parameter of 1.1. The number of support vectors in the final model was 226.The training error was 0.038265, which suggests that the model achieved a relatively high level of accuracy in cross-validation. Overall, the radial kernel SVM achieved a high level of accuracy in cross-validation and may be a good choice for modeling the Auto dataset.

(d) Make some plots to back up your assertions in (b) and (c). > Hint: In the lab, we used the plot() function for svm objects only in cases with $p = 2$. When $p > 2$, you can use the `plot()` function to create plots displaying pairs of variables at a time. Essentially, instead of typing

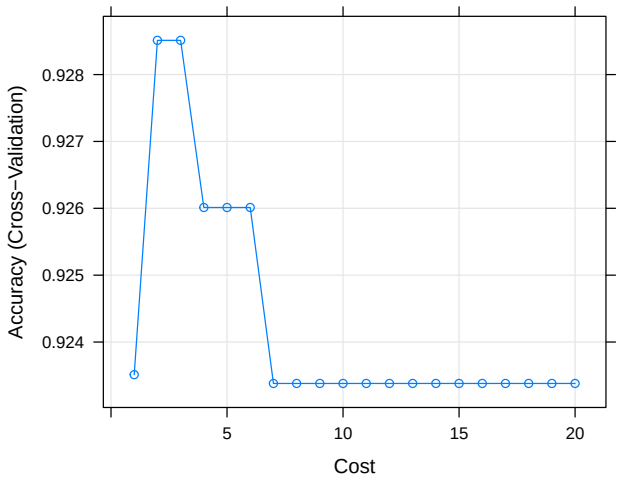
```
plot(svmfit, dat)
```

where svmfit contains your fitted model and dat is a data frame containing your data, you can type

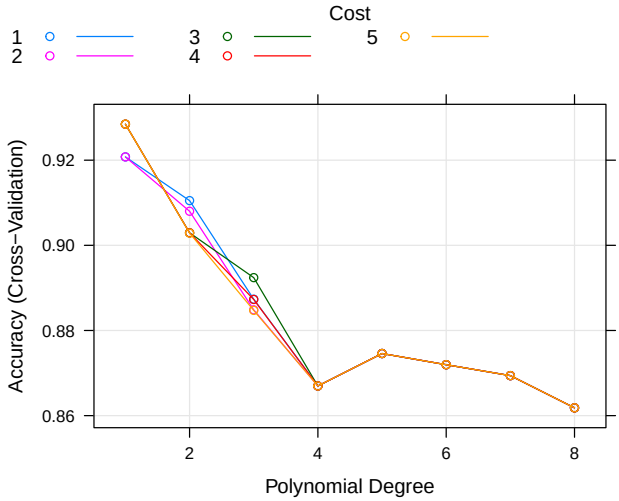
```
plot(svmfit, dat, x1 ~ x4)
```

in order to plot just the first and fourth variables. However, you must replace `x1` and `x4` with the correct variable names. To find out more, type `?plot.svm`.

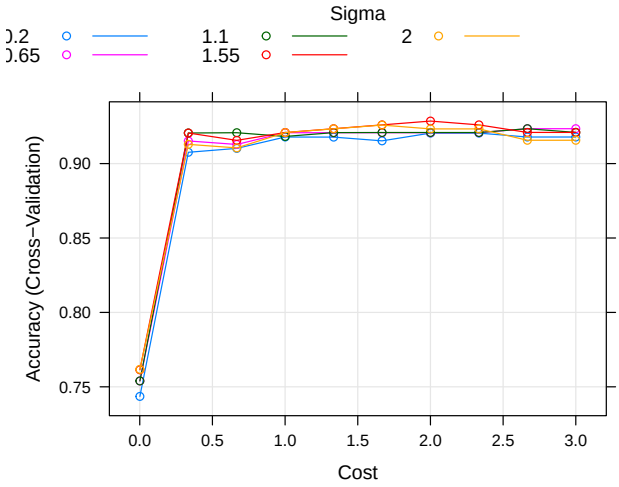
```
plot(Linear_SVM)
```



```
plot(Polynomial_SVM)
```




```
plot(Radial_SVM)
```



8. This problem involves the OJ data set which is part of the ISLR2 package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
library(ISLR2)
data(OJ)

set.seed(123)

# random sample of 800 observations for the training set
train <- sample(1:nrow(OJ), 800, replace = FALSE)

# remaining observations for the test set
test <- setdiff(1:nrow(OJ), train)
```

```
OJ_train <- OJ[train, ]
OJ_test <- OJ[test, ]
```

(b) Fit a support vector classifier to the training data using `cost=0.01`, with `Purchase` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics, and describe the results obtained.

```
# fit an SVM with cost = 0.01 to the training data
svm_fit <- svm(Purchase ~ ., data = OJ_train, cost = 0.01)

summary(svm_fit)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ_train, cost = 0.01)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  radial
#>        cost:  0.01
#>
#> Number of Support Vectors:  629
#>
#>  ( 313 316 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

The number of support vectors in the final model was 629, with 313 support vectors from the CH class and 316 support vectors from the MM class. The output also indicates that there are two classes in the data (CH and MM).

(c) What are the training and test error rates?

```
# predictions on the training and test datasets
train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)

# calculate the training and test error rates
train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

# results
cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.3912
```

```
cat(paste0("Test error rate: ", round(test_error, 4)))
```

```
#> Test error rate: 0.3852
```

(d) Use the `tune()` function to select an optimal `cost`. Consider values in the range 0.01 to 10.

```r
# tuning grid with values of cost from 0.01 to 10
tune_grid <- expand.grid(cost = seq(0.01, 10, length.out = 20))

# tune function to find the optimal value of cost
tune_result <- tune(svm, Purchase ~ ., data = OJ_train, ranges = tune_grid)

# results
print(tune_result)
```

```
#>
#> Parameter tuning of 'svm':
#>
#> - sampling method: 10-fold cross validation
#>
#> - best parameters:
#>       cost
#>   1.587368
#>
#> - best performance: 0.16125
```

The output gives us an optimal cost of \$1.58.

(e) Compute the training and test error rates using this new value for `cost`.

```r
# SVM with the optimal cost value to the training data
svm_fit <- svm(Purchase ~ ., data = OJ_train, cost = 1.587368)

# predictions on the training and test datasets
train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)

# training and test error rates
train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

# results
cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.1363
```

```r
cat(paste0("Test error rate: ", round(test_error, 4)))
```

```
#> Test error rate: 0.1963
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for `gamma`.

```r
# (b)
# SVM with radial kernel to the training data with cost = 0.01
svm_fit <- svm(Purchase ~ ., data = OJ_train, kernel = "radial", cost = 0.01)

# model summary
summary(svm_fit)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ_train, kernel = "radial", cost = 0.01)
```

```
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  radial
#>        cost:  0.01
#>
#> Number of Support Vectors:  629
#>
#>  ( 313 316 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

```
# (c)
# training and test error rates
train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)

train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.3912
```

```
cat(paste0("Test error rate: ", round(test_error, 4), "\n"))
```

```
#> Test error rate: 0.3852
```

```
# (d)
# tune() function to select an optimal cost
tune_grid <- expand.grid(cost = seq(0.01, 10, length.out = 20))
tune_result <- tune(svm, Purchase ~ ., data = OJ_train, kernel = "radial", ranges = tune_grid)
print(tune_result)
```

```
#>
#> Parameter tuning of 'svm':
#>
#> - sampling method: 10-fold cross validation
#>
#> - best parameters:
#>      cost
#>  1.061579
#>
#> - best performance: 0.1625
```

```
# (e)
# Compute the training and test error rates using the optimal cost
svm_fit <- svm(Purchase ~ ., data = OJ_train, kernel = "radial", cost = tune_result$best.parameters$cos

train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)
```

```r
train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

# results
cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.14
```

```r
cat(paste0("Test error rate: ", round(test_error, 4), "\n"))
```

```
#> Test error rate: 0.1852
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set `degree=2`.

```r
# (b)
# SVM with polynomial kernel to the training data with cost = 0.01 and degree = 2
svm_fit <- svm(Purchase ~ ., data = OJ_train, kernel = "polynomial", cost = 0.01, degree = 2)

# model summary
summary(svm_fit)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ_train, kernel = "polynomial",
#>     cost = 0.01, degree = 2)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  polynomial
#>        cost:  0.01
#>      degree:  2
#>      coef.0:  0
#>
#> Number of Support Vectors:  631
#>
#>  ( 313 318 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

```r
# (c)
# training and test error rates
train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)

train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.3725
```

```r
cat(paste0("Test error rate: ", round(test_error, 4), "\n"))
```

```
#> Test error rate: 0.3741
```

```r
# (d)
# tune() function to select an optimal cost
tune_grid <- expand.grid(cost = seq(0.01, 10, length.out = 20))
tune_result <- tune(svm, Purchase ~ ., data = OJ_train, kernel = "polynomial", degree = 2, ranges = tune
print(tune_result)
```

```
#>
#> Parameter tuning of 'svm':
#>
#> - sampling method: 10-fold cross validation
#>
#> - best parameters:
#>       cost
#>  6.319474
#>
#> - best performance: 0.165
```

```r
# (e)
# training and test error rates using the optimal cost
svm_fit <- svm(Purchase ~ ., data = OJ_train, kernel = "polynomial", cost = tune_result$best.parameters$

train_pred <- predict(svm_fit, newdata = OJ_train)
test_pred <- predict(svm_fit, newdata = OJ_test)

train_error <- mean(train_pred != OJ_train$Purchase)
test_error <- mean(test_pred != OJ_test$Purchase)

cat(paste0("Training error rate: ", round(train_error, 4), "\n"))
```

```
#> Training error rate: 0.1425
```

```r
cat(paste0("Test error rate: ", round(test_error, 4), "\n"))
```

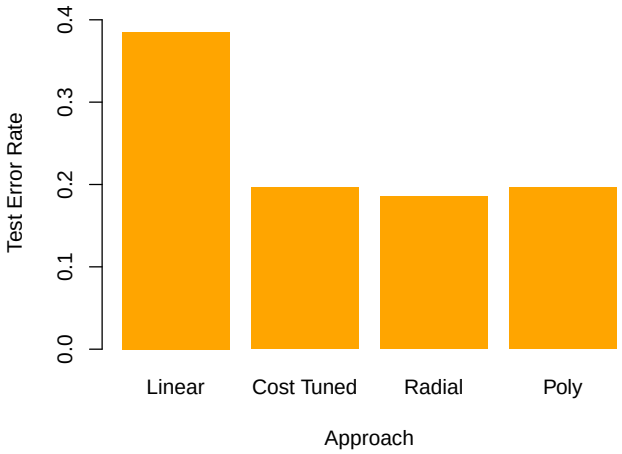
```
#> Test error rate: 0.1963
```

(h) Overall, which approach seems to give the best results on this data?

```r
# test error rates for each approach
test_errors <- c(0.3852, 0.1963, 0.1852, 0.1963)

# vector of approach names
approaches <- c("Linear", "Cost Tuned", "Radial", "Poly")

# bar plot of the test error rates
barplot(test_errors, names.arg = approaches, xlab = "Approach", ylab = "Test Error Rate",
        ylim = c(0, max(test_errors) + 0.02), col = "orange", border = NA)
```



Overall, based on test error rates we see that the linear method performs significantly worse and the radial method slightly outperforms the other 2 methods. The radial method also gives the lowest cost, which may be attractive depending on who is viewing the data (consumer vs. producer).