# P8106: Data Science II, Homework #5

Zachary Katz (UNI: zak2132)

5/5/2022

## Contents

## Question 1

### Set-Up and Data Preprocessing

```r
set.seed(2132)

# Load data, clean column names, eliminate rows containing NA entries
cars_data = read_csv("./Data/auto.csv") %>%
  janitor::clean_names() %>%
  na.omit() %>%
  distinct() %>%
  mutate(
    cylinders = as.factor(cylinders),
    origin = case_when(origin == "1" ~ "American",
                       origin == "2" ~ "European",
                       origin == "3" ~ "Japanese"),
    origin = as.factor(origin),
    mpg_cat = as.factor(mpg_cat),
    mpg_cat = fct_relevel(mpg_cat, "low", "high")
  ) %>%
  as.data.frame()
```

```r
# Partition data into training/test sets (70% split)
indexTrain = createDataPartition(y = cars_data$mpg_cat,
                                 p = 0.7,
                                 list = FALSE)
```

## Part (a): Support Vector Classifier (Linear Kernel)

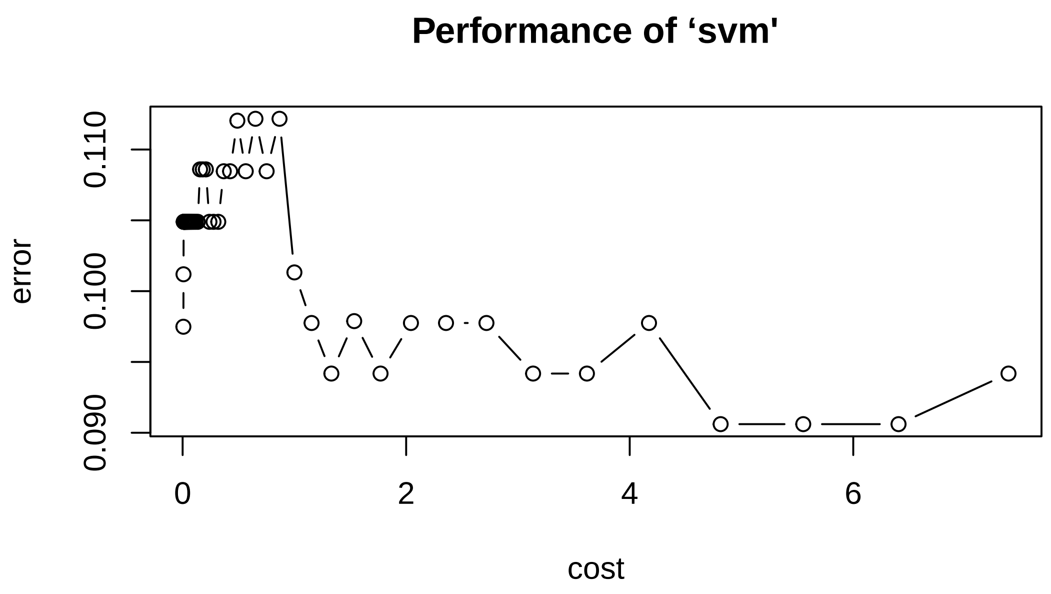
After loading the data, we fit a support vector classifier (linear kernel) to the training data.

```r
# Set seed
set.seed(2132)

# Fit model with tune.svm (no `caret`)
linear_svc = tune.svm(mpg_cat ~ .,
                      data = cars_data[indexTrain, ],
                      kernel = "linear",
                      cost = exp(seq(-5, 2, len = 50)),
                      scale = TRUE)
```

With cross-validation, we find the optimal tuning parameter is when cost is 4.81352, which minimizes the error.

```r
# Plot cost against error with cross-validation
plot(linear_svc)
```



```r
# Extract optimal tuning parameter with minimum cross-validation error
linear_svc$best.parameters
```

```
##       cost
## 47 4.81352
```

```
# Extract final model and summarize
best_linear_svc = linear_svc$best.model
summary(best_linear_svc)
```

```
##
## Call:
## best.svm(x = mpg_cat ~ ., data = cars_data[indexTrain, ], cost = exp(seq(-5,
##     2, len = 50)), kernel = "linear", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  4.81352
##
## Number of Support Vectors:  52
##
##  ( 25 27 )
##
##
## Number of Classes:  2
##
## Levels:
##  low high
```

The optimal support vector classifier with linear kernel incorrectly classifies 20 out of 276 training observations, giving a 7.2% error rate. When applied to the test data, it incorrectly classifies 7 out of 116 observations, leading to a 6.0% error rate.

```
# Training error

# Check confusion matrix; 93% accurate (7% training error rate)
confusionMatrix(data = linear_svc$best.model$fitted,
                reference = cars_data$mpg_cat[indexTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low  125    7
##       high  13  131
##
##                Accuracy : 0.9275
##                  95% CI : (0.8903, 0.9552)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8551
##
##  Mcnemar's Test P-Value : 0.2636
##
##             Sensitivity : 0.9058
##             Specificity : 0.9493
##          Pos Pred Value : 0.9470
##          Neg Pred Value : 0.9097
##              Prevalence : 0.5000
##          Detection Rate : 0.4529
##    Detection Prevalence : 0.4783
##       Balanced Accuracy : 0.9275
##
##        'Positive' Class : low
##
```

```
# Test error

# Make predictions
linear_test_preds = predict(best_linear_svc, newdata = cars_data[-indexTrain, ])

# Check confusion matrix; 94% accurate (6% test error rate)
confusionMatrix(data = linear_test_preds,
                reference = cars_data$mpg_cat[-indexTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low   56    5
##       high   2   53
##
##                Accuracy : 0.9397
##                  95% CI : (0.8796, 0.9754)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8793
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.9655
##             Specificity : 0.9138
##          Pos Pred Value : 0.9180
##          Neg Pred Value : 0.9636
##              Prevalence : 0.5000
##          Detection Rate : 0.4828
##    Detection Prevalence : 0.5259
##       Balanced Accuracy : 0.9397
##
##        'Positive' Class : low
##
```

## Part (b): Support Vector Machine (Radial Kernel)

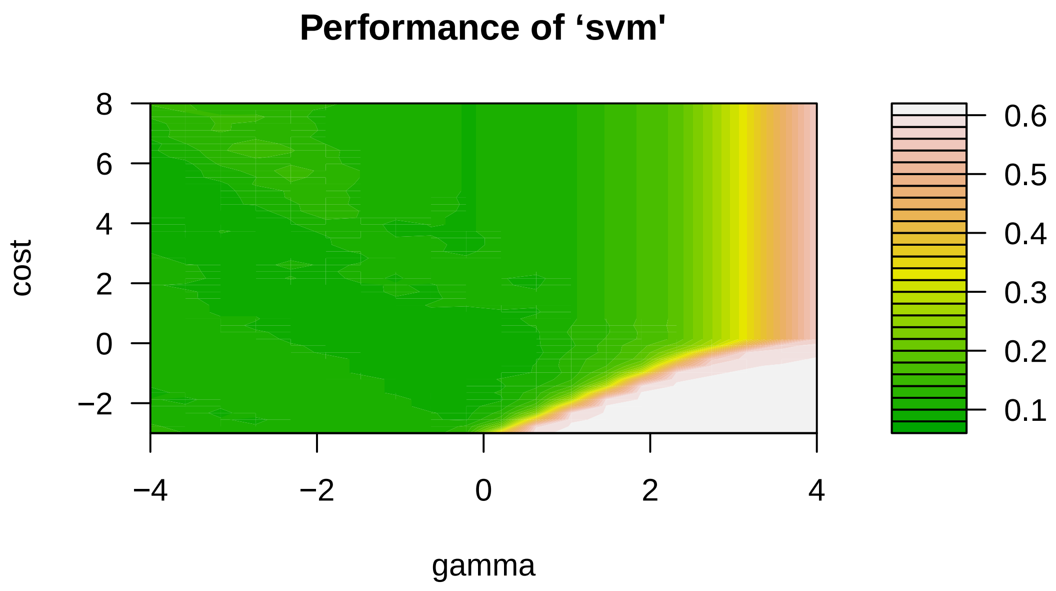
Now, we want to try a support vector machine with radial kernel, which gives a nonlinear decision boundary.

```r
# Set seed
set.seed(2132)

# Fit model with tune.svm (no `caret`)
radial_svm = tune.svm(mpg_cat ~ .,
                      data = cars_data[indexTrain, ],
                      kernel = "radial",
                      cost = exp(seq(-3, 8, len = 50)),
                      gamma = exp(seq(-4, 4, len = 20)),
                      scale = TRUE)
```

With cross-validation, we find the optimal tuning parameters of cost = 41.9 and gamma = 0.03, then use the best model to determine our training and testing error rates.

```r
# Performance plotted across tuning parameters
plot(radial_svm, transform.y = log, transform.x = log, color.palette = terrain.colors)
```



```r
# Optimal tuning parameters
best_radial = radial_svm$best.parameters

best_radial
```

```
##          gamma    cost
## 602 0.02790506 41.8752
```

```r
# Extract final model and summarize
best_radial_svm = radial_svm$best.model
summary(best_radial_svm)
```

```
##
## Call:
## best.svm(x = mpg_cat ~ ., data = cars_data[indexTrain, ], gamma = exp(seq(-4,
##     4, len = 20)), cost = exp(seq(-3, 8, len = 50)), kernel = "radial",
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  41.8752
##
## Number of Support Vectors:  54
##
##  ( 26 28 )
##
##
## Number of Classes:  2
##
## Levels:
##  low high
```

The best radial kernel SVM performs a bit better than the best linear kernel SVC when obtained using cross-validation on the training data; our radial SVM incorrectly classifies 15 out of 276, i.e has a training error rate of 94.6%. However, the radial kernel SVM actually performs the same as the linear SVC when applied to the unseen test data, incorrectly classifying 7 out of 116, or ~6.0%, of our testing set observations.

```r
# Training error

# Check confusion matrix; 95% accurate (5% training error rate)
confusionMatrix(data = radial_svm$best.model$fitted,
                reference = cars_data$mpg_cat[indexTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low  126    3
##       high  12  135
##
##                Accuracy : 0.9457
##                  95% CI : (0.9119, 0.9693)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8913
##
##  Mcnemar's Test P-Value : 0.03887
##
##             Sensitivity : 0.9130
##             Specificity : 0.9783
##          Pos Pred Value : 0.9767
##          Neg Pred Value : 0.9184
##               Prevalence : 0.5000
##           Detection Rate : 0.4565
##     Detection Prevalence : 0.4674
##        Balanced Accuracy : 0.9457
##
##         'Positive' Class : low
##
```

```
# Test error

# Make predictions
radial_test_preds = predict(best_radial_svm, newdata = cars_data[-indexTrain, ])

# Check confusion matrix; 94% accurate (6% test error rate)
confusionMatrix(data = radial_test_preds,
                reference = cars_data$mpg_cat[-indexTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction low high
##       low   53    3
##       high   5   55
##
##                Accuracy : 0.931
##                  95% CI : (0.8686, 0.9698)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8621
##
##  Mcnemar's Test P-Value : 0.7237
##
##             Sensitivity : 0.9138
##             Specificity : 0.9483
##          Pos Pred Value : 0.9464
##          Neg Pred Value : 0.9167
##              Prevalence : 0.5000
##          Detection Rate : 0.4569
##    Detection Prevalence : 0.4828
##       Balanced Accuracy : 0.9310
##
##        'Positive' Class : low
##
```

## Question 2

### Part (a): Hierarchical Clustering (Without Scaling)

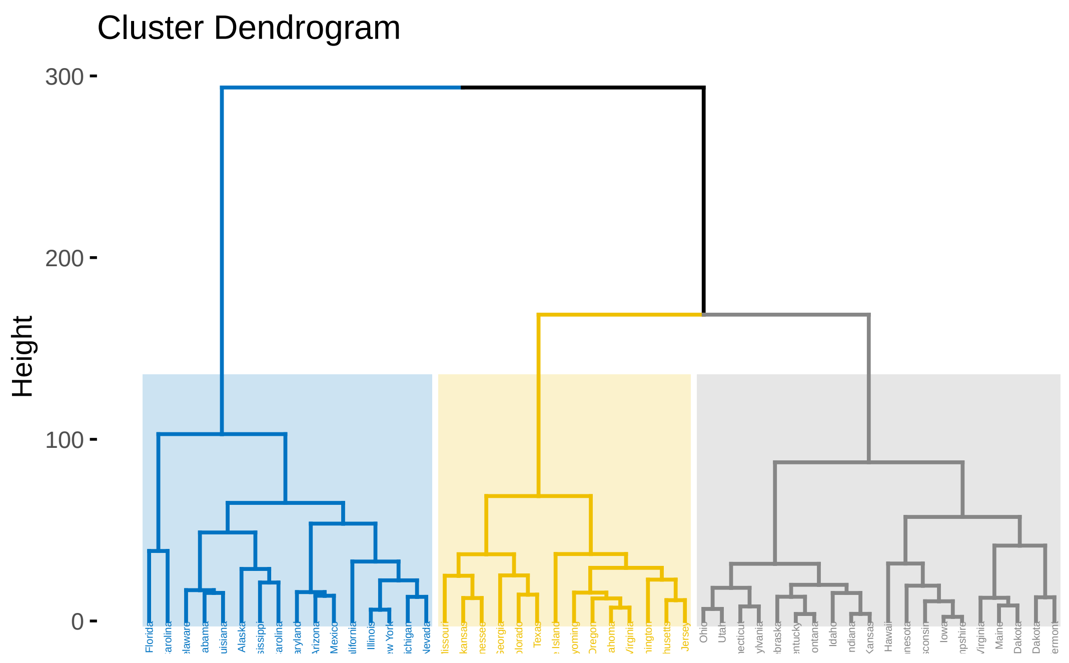
```
# Load data
data(USArrests)
```

```r
arrests_data = USArrests %>% as.data.frame()
```

```r
# Hierarchical clustering with complete linkage and Euclidean distance, no scaling
cluster_no_scale = hclust(dist(arrests_data), method = "complete")
```

Without scaling our variables, we perform hierarchical clustering with complete linkage and Euclidean distance, then build a dendrogram that has three distinct clusters, as below.

```r
# Cut the dendrogram at a height that results in three distinct clusters
fviz_dend(cluster_no_scale, k = 3,
          cex = 0.3,
          palette = "jco",
          color_labels_by_k = TRUE,
          rect = TRUE, rect_fill = TRUE, rect_border = "jco",
          labels_track_height = 2.5)
```



Each cluster has a number of states included in it. For example, the first cluster has a number of quite populous states, including Florida, North Carolina, New York, Michigan, among others, as well as a few less populous states like Mississippi. The second cluster includes some Southern states, such as Missouri, Arkansas, Tennessee, but also some others, like Wyoming and Oregon. Finally, the third cluster contains some less populous states, like North Dakota and Vermont, as well as Ohio, Utah, and others. There doesn't seem to be any clearly discernible pattern when we cut at three clusters, at least based on geography, though there is somewhat noticeable grouping tendencies based on population.

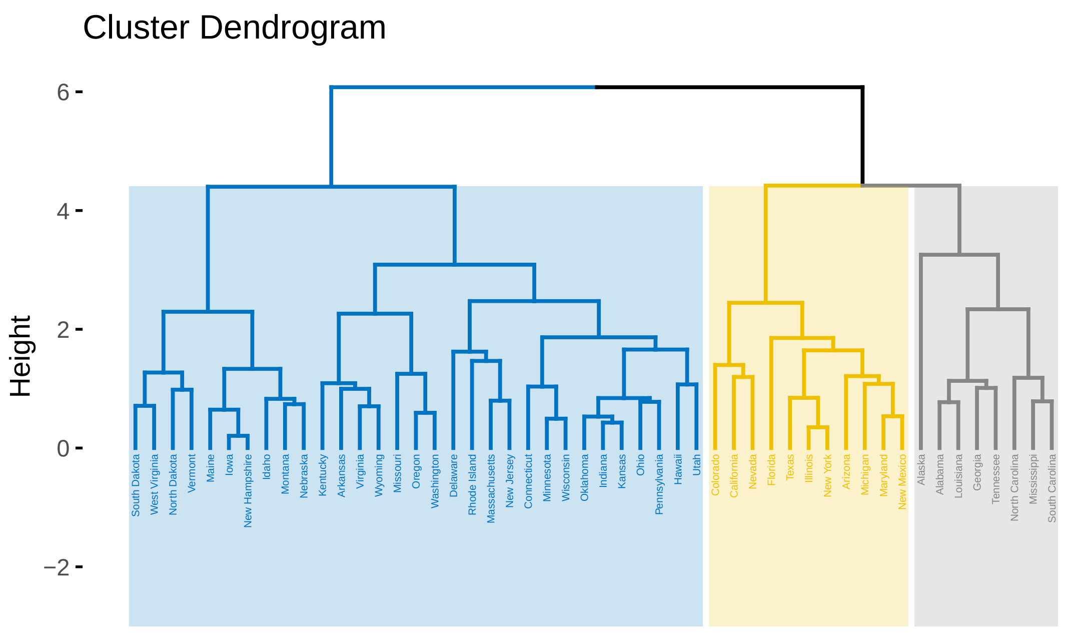
## Part (b): Hierarchical Clustering (With Scaling)

```r
# Scale and center data
arrests_data_scaled = scale(arrests_data, center = TRUE, scale = TRUE)
```

```r
# Hierarchical clustering with complete linkage and Euclidean distance, with scaling
cluster_scale = hclust(dist(arrests_data_scaled), method = "complete")
```

In this example, we perform the same kind of Hierarchical clustering with Euclidean distance and complete linkage, but first make sure that our variables are scaled to have standard deviation equal to 1, as well as centered. The dendrogram cut at a height that results in three distinct clusters is shown below.

```r
# Cut the dendrogram at a height that results in three distinct clusters
fviz_dend(cluster_scale, k = 3,
          cex = 0.3,
          palette = "jco",
          color_labels_by_k = TRUE,
          rect = TRUE, rect_fill = TRUE, rect_border = "jco",
          labels_track_height = 2.5)
```



In this example, we have one cluster with many more states than each of the other two. See part (c) for further elaboration on the clustering outcome after scaling.

## Part (c): Considerations

Once we've scaled our data, we observe that the clusters do change compared to what we observe without scaling. With scaling, we have one cluster that contains South Dakota, West Virginia, any many other of the less populous states; another cluster with Colorado, California, Nevada, Texas, New York, and primarily more populous states with major urban metro areas; and a third cluster with Alaska, Alabama, Louisiana, Georgia, and a number of other mostly Southern U.S. states. This is different from what we find in part (a), without scaling.

Scaling the variables does indeed change the clustering results. Because our clustering algorithms require some definition of distance (here, Euclidean), failing to scale our numeric variables means that we may unfairly attribute more importance to those variables that have greater magnitudes. In our case, that means that without scaling, we're more likely to cluster based on `assault` or `urbanpop`, since these tend to have greater values per unit population than `murder` or `rape`. As we see below, the unscaled mean for `assault` is 171 with unscaled SD of 83.3, whereas for `murder`, the unscaled mean is much lower, at 7.79, with SD 4.36.

```
skimr::skim_without_charts(arrests_data)
```

Table 1: Data summary

| | |
|---|---|
| Name | arrests_data |
| Number of rows | 50 |
| Number of columns | 4 |
| Column type frequency: | |
| numeric | 4 |
| Group variables | None |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| Murder | 0 | 1 | 7.79 | 4.36 | 0.8 | 4.08 | 7.25 | 11.25 | 17.4 |
| Assault | 0 | 1 | 170.76 | 83.34 | 45.0 | 109.00 | 159.00 | 249.00 | 337.0 |
| UrbanPop | 0 | 1 | 65.54 | 14.47 | 32.0 | 54.50 | 66.00 | 77.75 | 91.0 |
| Rape | 0 | 1 | 21.23 | 9.37 | 7.3 | 15.08 | 20.10 | 26.17 | 46.0 |

In my opinion, for the reasons stated above, the variables **should be scaled before the inter-observation dissimilarities are computed** in order to ensure our variables are of comparable units. In the cases of `murder`, `assault`, and `rape`, for example, it is not enough that they are all expressed as the number of arrests per 100K residents; we also prefer each one to have SD = 1 (and mean = 0, ideally).