BIOSTAT 285 Spring 2020 Project 3

Use SVM approaches to predict whether a given car gets high or low gas mileage

Nan Liu

SupportVector Approaches

First let's import the data:

```
library(ISLR)
data(Auto)
#mpgadd <- Auto$mpg
```

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.

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## Warning: package 'ggplot2' was built under R version 3.6.2

## Warning: package 'tibble' was built under R version 3.6.2

## Warning: package 'tidyr' was built under R version 3.6.2

## Warning: package 'purrr' was built under R version 3.6.2

## Warning: package 'dplyr' was built under R version 3.6.2

## -- Conflicts -------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
Auto <- Auto %>%
  mutate (mpg01 = as.factor(ifelse(mpg > median(mpg), 1, 0)))
```

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.

We tune the parameter `cost` and find the optimal value of `cost`:

```r
#install.packages("e1071")
set.seed(1)
library(e1071)
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07653846 0.03617137
## 2 1e-01 0.04596154 0.03378238
## 3 1e+00 0.01025641 0.01792836
## 4 5e+00 0.02051282 0.02648194
## 5 1e+01 0.02051282 0.02648194
## 6 1e+02 0.03076923 0.03151981
```

The optimal value of cost is 1 and the cross-validation error is 0.01025641 at this time. The cross-validation error is relatively small, so we conclude the model can predict whether a car gets high or low gas mileage pretty well.

SVM with radial and polynomial basis kernels

For **radial** kernal, we tune the parameter of `cost` and `gamma`:

```r
#radial kernal
set.seed(1)
tuneradial <- tune(svm, mpg01 ~ .,
```

```r
                   data = Auto, kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100),
                               gamma = 1/nrow(Auto) * c(1,10,100)))
summary(tuneradial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost      gamma
##   100 0.00255102
##
## - best performance: 0.01282051
##
## - Detailed performance results:
##     cost      gamma      error dispersion
## 1    0.1 0.00255102 0.11493590 0.05448680
## 2    1.0 0.00255102 0.08160256 0.04297800
## 3   10.0 0.00255102 0.05365385 0.02830539
## 4  100.0 0.00255102 0.01282051 0.01813094
## 5    0.1 0.02551020 0.08160256 0.04297800
## 6    1.0 0.02551020 0.06128205 0.03252808
## 7   10.0 0.02551020 0.01538462 0.02477158
## 8  100.0 0.02551020 0.02307692 0.02549818
## 9    0.1 0.25510204 0.08160256 0.04297800
## 10   1.0 0.25510204 0.04865385 0.03304230
## 11  10.0 0.25510204 0.04871795 0.04264949
## 12 100.0 0.25510204 0.04871795 0.04264949
```

When using the SVM with radial basis kernel, the optiaml value of `gamma` is 0.00255102 and the optimal value of `cost` is 100. The cross-validation error is 0.01282051 at this time.

For **polynomial** kernal, we tune the parameter of `cost` and `gamma`:

```r
#polynomial
set.seed(1)
tunepoly <- tune(svm, mpg01 ~ .,
                 data = Auto, kernel = "polynomial",
                 ranges = list(cost = c(0.1, 1, 10, 100),
                               gamma = 1/nrow(Auto) * c(1,10,100),
                               degree = c(2, 3, 4)))
summary(tunepoly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost    gamma degree
##     1 0.255102      3
```

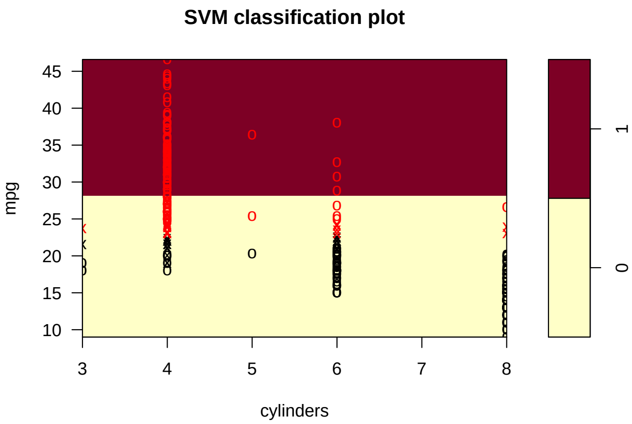
```
##
## - best performance: 0.03839744
##
## - Detailed performance results:
##     cost      gamma degree      error dispersion
## 1    0.1 0.00255102      2 0.55115385 0.04366593
## 2    1.0 0.00255102      2 0.55115385 0.04366593
## 3   10.0 0.00255102      2 0.55115385 0.04366593
## 4  100.0 0.00255102      2 0.32698718 0.09393958
## 5    0.1 0.02551020      2 0.55115385 0.04366593
## 6    1.0 0.02551020      2 0.32698718 0.09393958
## 7   10.0 0.02551020      2 0.30160256 0.10208544
## 8  100.0 0.02551020      2 0.14801282 0.04647959
## 9    0.1 0.25510204      2 0.30160256 0.10208544
## 10   1.0 0.25510204      2 0.14801282 0.04647959
## 11  10.0 0.25510204      2 0.16070513 0.01706742
## 12 100.0 0.25510204      2 0.18621795 0.02958495
## 13   0.1 0.00255102      3 0.55115385 0.04366593
## 14   1.0 0.00255102      3 0.55115385 0.04366593
## 15  10.0 0.00255102      3 0.55115385 0.04366593
## 16 100.0 0.00255102      3 0.42115385 0.12702484
## 17   0.1 0.02551020      3 0.42115385 0.12702484
## 18   1.0 0.02551020      3 0.25538462 0.09577427
## 19  10.0 0.02551020      3 0.16871795 0.11308600
## 20 100.0 0.02551020      3 0.06391026 0.03669252
## 21   0.1 0.25510204      3 0.06391026 0.03669252
## 22   1.0 0.25510204      3 0.03839744 0.03872235
## 23  10.0 0.25510204      3 0.04615385 0.03783922
## 24 100.0 0.25510204      3 0.04358974 0.03636247
## 25   0.1 0.00255102      4 0.55115385 0.04366593
## 26   1.0 0.00255102      4 0.55115385 0.04366593
## 27  10.0 0.00255102      4 0.55115385 0.04366593
## 28 100.0 0.00255102      4 0.55115385 0.04366593
## 29   0.1 0.02551020      4 0.55115385 0.04366593
## 30   1.0 0.02551020      4 0.48000000 0.08809526
## 31  10.0 0.02551020      4 0.36512821 0.08400844
## 32 100.0 0.02551020      4 0.25794872 0.09840025
## 33   0.1 0.25510204      4 0.18628205 0.08711672
## 34   1.0 0.25510204      4 0.16833333 0.08429979
## 35  10.0 0.25510204      4 0.18621795 0.07710760
## 36 100.0 0.25510204      4 0.19397436 0.07775911
```

When using the SVM with polynomial basis kernel, the optiaml value of `degree` is 3, the optimal value of `gamma` is 0.255102 and the optimal value of `cost` is 1.The cross-validation error is 0.55115385 at this time.

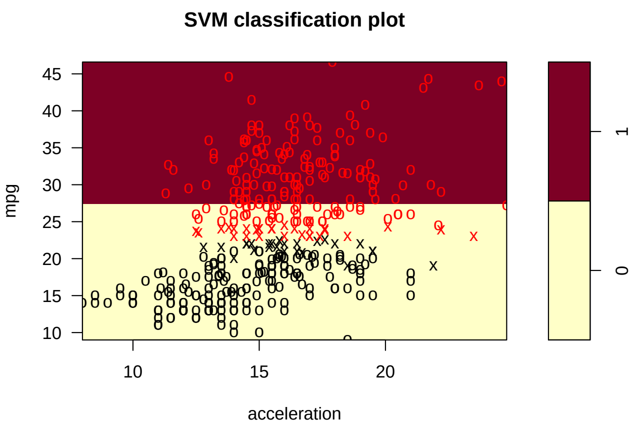
Visualize

```
svm_lin <- svm(mpg01 ~ ., data=Auto, kernel="linear", cost = 1)
svm_rad <- svm(mpg01 ~ ., data=Auto, kernel="radial", cost = 100, gamma=0.00255)
svm_poly <- svm(mpg01 ~ ., data=Auto, kernel="polynomial", cost = 1, gamma=0.255, degree =3)
```

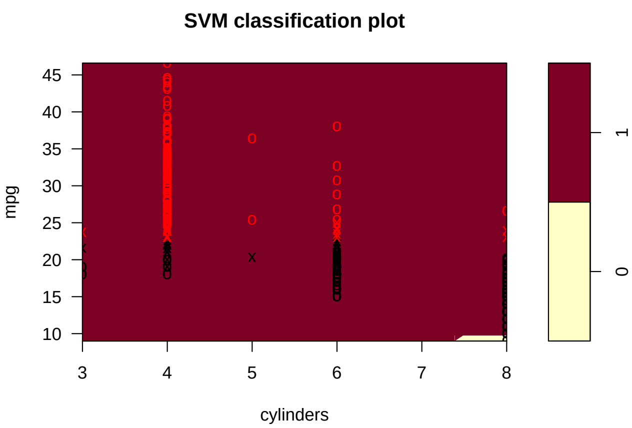
```r
plot(svm_lin, Auto, mpg~cylinders)
```



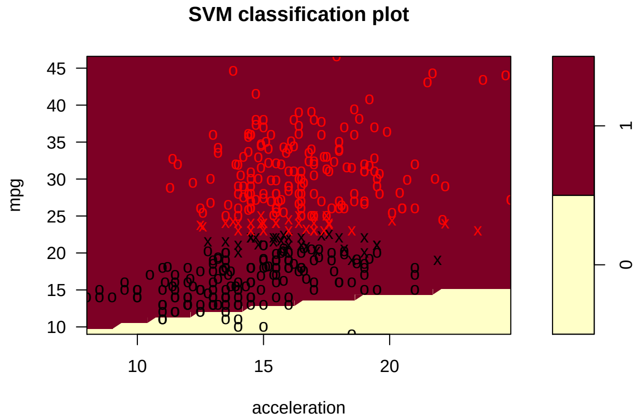

```r
plot(svm_lin, Auto, mpg~acceleration)
```



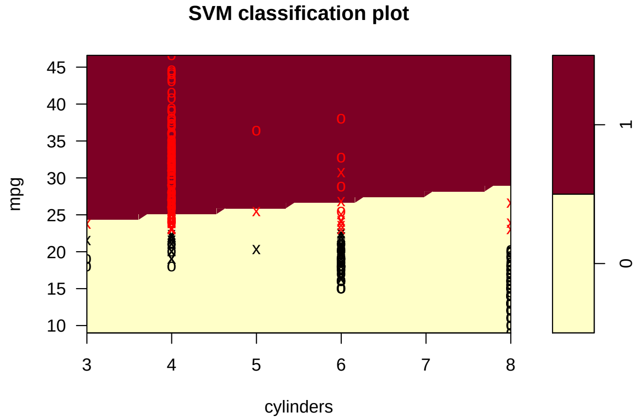

```r
plot(svm_rad, Auto, mpg~cylinders)
```



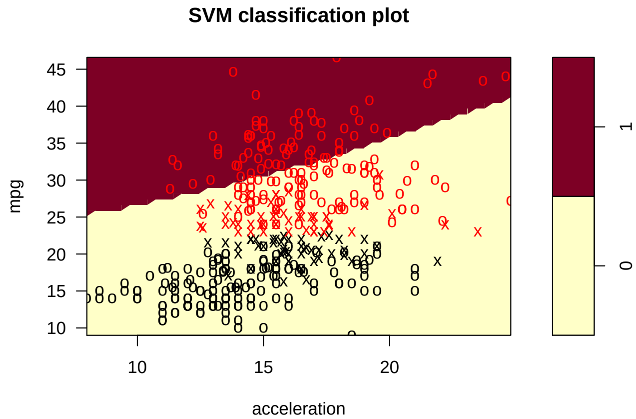

```r
plot(svm_rad, Auto, mpg~acceleration)
```




```r
plot(svm_poly, Auto, mpg~cylinders)
```




```r
plot(svm_poly, Auto, mpg~acceleration)
```



Compare the cross validation errors and the plots of the three method, we conclude that SVM with linear kernal fits the data best. there is evidence that linear kernel seems to fit the data the best. From the plots above, SVM using radial and polynomial kernel do not separate the two classes well.