SVM Regression

Team: Abed Ahmed (ASA190005) & Dylan Kapustka (DLK190000)

Date: 09/25/2022

HomeWork 5 - Kernel and Ensemble Methods

```
data <- read.csv(file="Anime-Data.csv",header=TRUE,)
data <- na.omit(data)

set.seed(2)
library(caTools)


spec <- c(train=.6, test=.2, validate=.2)
i <- sample(cut(1:nrow(data),
                nrow(data)*cumsum(c(0,spec)), labels=names(spec)))
train <- data[i=="train",]
test <- data[i=="test",]
vald <- data[i=="validate",]
summary(train)
```

```
##      MAL_ID            Name               Score          Genres
##  Min.   :    6.0   Length:739         Min.   :4.800   Length:739
##  1st Qu.:  954.5   Class :character   1st Qu.:6.885   Class :character
##  Median : 5973.0   Mode  :character   Median :7.270   Mode  :character
##  Mean   :10024.0                      Mean   :7.275
##  3rd Qu.:17735.0                      3rd Qu.:7.690
##  Max.   :32214.0                      Max.   :9.110
##  English.name       Japanese.name          Type              Episodes
##  Length:739         Length:739         Length:739         Min.   :  3.00
##  Class :character   Class :character   Class :character   1st Qu.: 12.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 13.00
##                                                           Mean   : 25.25
##                                                           3rd Qu.: 26.00
##                                                           Max.   :500.00
##     Aired            Premiered          Producers          Licensors
##  Length:739         Length:739         Length:739         Length:739
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##    Studios             Source            Duration           Rating
##  Length:739         Length:739         Length:739         Length:739
```

```
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##      Ranked        Popularity         Members          Favorites
##  Min.   :    3   Min.   :    1.0   Min.   :    763   Min.   :     0
##  1st Qu.: 1046   1st Qu.:  483.5   1st Qu.:  43113   1st Qu.:   156
##  Median : 2463   Median : 1241.0   Median : 116837   Median :   588
##  Mean   : 2872   Mean   : 1764.2   Mean   : 243222   Mean   :  4168
##  3rd Qu.: 4104   3rd Qu.: 2441.5   3rd Qu.: 291109   3rd Qu.:  2374
##  Max.   :10743   Max.   :11065.0   Max.   :2589552   Max.   :148452
##     Watching        Completed          On.Hold         Dropped
##  Min.   :    23   Min.   :    300   Min.   :    27   Min.   :    94
##  1st Qu.:  2002   1st Qu.:  21511   1st Qu.:  1752   1st Qu.:  1936
##  Median :  6068   Median :  66029   Median :  4162   Median :  5397
##  Mean   : 13055   Mean   : 162508   Mean   :  7796   Mean   :  9044
##  3rd Qu.: 13666   3rd Qu.: 181141   3rd Qu.:  8783   3rd Qu.: 11456
##  Max.   :362124   Max.   :2182587   Max.   :109707   Max.   :124253
##  Plan.to.Watch       Score.10         Score.9          Score.8
##  Min.   :   146   Min.   :     7   Min.   :     3   Min.   :     9
##  1st Qu.: 12508   1st Qu.:  1109   1st Qu.:  1637   1st Qu.:  3588
##  Median : 31152   Median :  4041   Median :  6215   Median : 11728
##  Mean   : 50819   Mean   : 18656   Mean   : 24907   Mean   : 34345
##  3rd Qu.: 68573   3rd Qu.: 15144   3rd Qu.: 23708   3rd Qu.: 37313
##  Max.   :425531   Max.   :557406   Max.   :535252   Max.   :459113
##     Score.7          Score.6          Score.5          Score.4
##  Min.   :    36   Min.   :    45   Min.   :    42   Min.   :    9.0
##  1st Qu.:  4576   1st Qu.:  2610   1st Qu.:  1350   1st Qu.:  501.5
##  Median : 12587   Median :  6720   Median :  3351   Median : 1345.0
##  Mean   : 28561   Mean   : 13257   Mean   :  6463   Mean   : 2932.7
##  3rd Qu.: 33942   3rd Qu.: 16320   3rd Qu.:  8121   3rd Qu.: 3515.5
##  Max.   :303813   Max.   :188431   Max.   :124819   Max.   :81155.0
##     Score.3          Score.2          Score.1
##  Min.   :    7   Min.   :    3.0   Min.   :    3.0
##  1st Qu.:  201   1st Qu.:   99.0   1st Qu.:   88.0
##  Median :  528   Median :  270.0   Median :  231.0
##  Mean   : 1285   Mean   :  714.1   Mean   :  623.4
##  3rd Qu.: 1476   3rd Qu.:  776.5   3rd Qu.:  667.0
##  Max.   :44204   Max.   :25371.0   Max.   :23472.0
```

```
mean(train$Score)
```

```
## [1] 7.275142
```

```
median(train$Ranked)
```

```
## [1] 2463
```

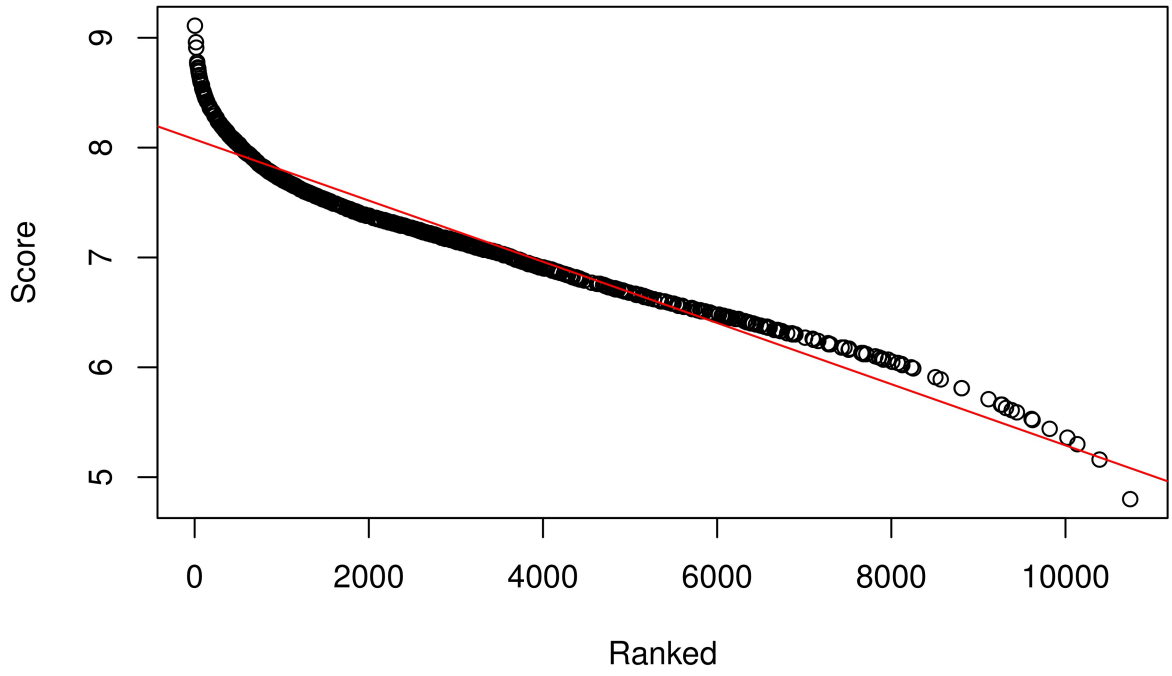
```
mean(train$Ranked)
```

```
## [1] 2872.001
```

```
median(train$Ranked)
```

```
## [1] 2463
```

```
plot(train$Score~train$Ranked,xlab="Ranked",ylab="Score")
abline(lm(train$Score~train$Ranked),col="red")
```



Lineaer Kernal

```
library(e1071)
svm1 <- svm(Score ~ Ranked, data=train, kernel="linear", cost=10, scale=TRUE)
summary(svm1)
```

```
##
## Call:
## svm(formula = Score ~ Ranked, data = train, kernel = "linear", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
```

```
##        cost:  10
##       gamma:  1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  384
```

```
pred <- predict(svm1, newdata=test)
cor_svm1 <- cor(pred, test$Score)
mse_svm1 <- mean((pred - test$Score)^2)

print(paste("Correlation of Linear Kernel before Tuning", cor_svm1))
```

```
## [1] "Correlation of Linear Kernel before Tuning 0.956859956793723"
```

```
print(paste("mse of Linear Kernel before Tuning", mse_svm1))
```

```
## [1] "mse of Linear Kernel before Tuning 0.0411521812496517"
```

```
tune_svm1 <- tune(svm, Score ~ Ranked, data=vald, kernel="linear",
                  ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune_svm1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.03728119
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.31067280 0.07820291
## 2 1e-02 0.04776888 0.02827489
## 3 1e-01 0.03830761 0.02446064
## 4 1e+00 0.03758082 0.02406671
## 5 5e+00 0.03737515 0.02399549
## 6 1e+01 0.03733120 0.02397645
## 7 1e+02 0.03728119 0.02397180
```

```
pred <- predict(tune_svm1$best.model, newdata=test)
cor_svm1_tune <- cor(pred, test$Score)
mse_svm1_tune <- mean((pred - test$Score)^2)

print(paste("Correlation of Linear Kernel after Tuning", cor_svm1_tune))
```

```
## [1] "Correlation of Linear Kernel after Tuning 0.956859956793725"
```

```
print(paste("mse of Linear Kernel after Tuning", mse_svm1_tune))
```

```
## [1] "mse of Linear Kernel after Tuning 0.040395426261333"
```

Polynomial Kernel

```
svm2 <- svm(Score ~ Ranked, data=train, kernel="polynomial", cost=10, scale=TRUE)
summary(svm2)
```

```
##
## Call:
## svm(formula = Score ~ Ranked, data = train, kernel = "polynomial",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  1
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  659
```

```
pred <- predict(svm2, newdata=test)
cor_svm2 <- cor(pred, test$Score)
mse_svm2 <- mean((pred - test$Score)^2)

print(paste("Correlation of Polynomial Kernel before tuning ", cor_svm2))
```

```
## [1] "Correlation of Polynomial Kernel before tuning  0.721831680113644"
```

```
print(paste("mse of Polynomial Kernel before tuning ", mse_svm2))
```

```
## [1] "mse of Polynomial Kernel before tuning  0.207782186150039"
```

```
set.seed(1234)
tune.out <- tune(svm, Score~Ranked, data=vald, kernel="polynomial",
                 ranges=list(cost=c(0.1,1,10,100,1000),
                             gamma=c(0.5,1)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.3631975
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.3631975  0.4507071
## 2  1e+00   0.5 0.4055354  0.5523589
## 3  1e+01   0.5 0.4117994  0.5639979
## 4  1e+02   0.5 0.4118224  0.5640146
## 5  1e+03   0.5 0.4120171  0.5641935
## 6  1e-01   1.0 0.4034985  0.5529065
## 7  1e+00   1.0 0.4117884  0.5640009
## 8  1e+01   1.0 0.4118050  0.5640149
## 9  1e+02   1.0 0.4119710  0.5641546
## 10 1e+03   1.0 0.4338111  0.5640569
```

```
svm4 <- svm(Score~Ranked, data=train, kernel="radial", cost=100, gamma=0.5, scale=TRUE)
summary(svm4)
```

```
##
## Call:
## svm(formula = Score ~ Ranked, data = train, kernel = "radial", cost = 100,
##     gamma = 0.5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  27
```

```
pred <- predict(svm4, newdata=test)
cor_svm4 <- cor(pred, test$Score)
mse_svm4 <- mean((pred - test$Score)^2)


print(paste("Correlation of Polynomial Kernel After Tuning ", cor_svm4))
```

```
## [1] "Correlation of Polynomial Kernel After Tuning  0.996611630323546"
```

```
print(paste("mse of Polynomial Kernel Aftering tuning ", mse_svm4))
```

```
## [1] "mse of Polynomial Kernel Aftering tuning  0.00288044351370964"
```

Radial Kernel

```
svm3 <- svm(Score ~ Ranked, data=train, kernel="radial", cost=10, gamma=1, scale=TRUE)
summary(svm3)
```

```
##
## Call:
## svm(formula = Score ~ Ranked, data = train, kernel = "radial", cost = 10,
##     gamma = 1, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  33
```

```
pred <- predict(svm3, newdata=test)
cor_svm3 <- cor(pred, test$Score)
mse_svm3 <- mean((pred - test$Score)^2)

print(paste("Correlation of Radial Kernel before tuning ", cor_svm3))
```

```
## [1] "Correlation of Radial Kernel before tuning  0.996307546698241"
```

```
print(paste("mse of Radial Kernel before tuning", mse_svm3))
```

```
## [1] "mse of Radial Kernel before tuning 0.00316635684200789"
```

```
set.seed(1234)
tune.out <- tune(svm, Score~Ranked, data=vald, kernel="radial",
                 ranges=list(cost=c(0.1,1,10,100,1000),
                             gamma=c(0.5,1)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##  1000   0.5
##
## - best performance: 0.009859039
##
```

```
## - Detailed performance results:
##     cost gamma       error dispersion
## 1  1e-01   0.5 0.047859762 0.05301001
## 2  1e+00   0.5 0.018851655 0.03178397
## 3  1e+01   0.5 0.011591761 0.01820545
## 4  1e+02   0.5 0.012528313 0.02491840
## 5  1e+03   0.5 0.009859039 0.01902341
## 6  1e-01   1.0 0.048168593 0.06049320
## 7  1e+00   1.0 0.020438163 0.03931643
## 8  1e+01   1.0 0.014419453 0.02905711
## 9  1e+02   1.0 0.015428973 0.03537965
## 10 1e+03   1.0 0.018003014 0.04475154
```

```
svm5 <- svm(Score~Ranked, data=train, kernel="radial", cost=100, gamma=0.5, scale=TRUE)
summary(svm5)
```

```
##
## Call:
## svm(formula = Score ~ Ranked, data = train, kernel = "radial", cost = 100,
##     gamma = 0.5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  27
```

```
pred <- predict(svm5, newdata=test)
cor_svm5 <- cor(pred, test$Score)
mse_svm5 <- mean((pred - test$Score)^2)


print(paste("Correlation of Radial Kernel After Tuning ", cor_svm5))
```

```
## [1] "Correlation of Radial Kernel After Tuning  0.996611630323546"
```

```
print(paste("mse of Polynomial Radial Aftering tuning ", mse_svm5))
```

```
## [1] "mse of Polynomial Radial Aftering tuning  0.00288044351370964"
```

Summary of results and Analysis

Radial Kernel was the best performing kernel. It had a 99% Correlation before tuning.

The Polynomial Kernel had a 72% correlation before tuning, but a 99% after tuning.

The Linear Kernel had a 95% Correlation before and after tuning.

All these results were better than using simply linear regression, most likely due to the added dimensions from the kernels. Radial Kernel was the best performing Kernel, which explains why it is so popular in a way. I suspect this was because RBF was capable of creating the most sophisticated boundaries for the data.