Kernel Classfication - 10/23

Rice Seed Classification

On this R markdown file, we'll be investigating a Rice seed classification dataset found on Kaggle.

(Link for pdf viewers: https://www.kaggle.com/datasets/mssmartypants/rice-type-classification)

The dataset has two classes to categorize. A "0" indicates a Gonen rice seed (commonly found and produced in Northwest Turkey) and "1" for jasmine rice. This is tied to the class attribute of the dataset.

Data Exploration & Train/Test

```
# Data Exploration - Taking a look at min and max areas as well.
df <- read.csv("riceClassification.csv", header = TRUE)
str(df)
```

```
## 'data.frame':    18185 obs. of  12 variables:
##  $ id             : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Area           : int  4537 2872 3048 3073 3693 2990 3556 3788 2629 5719 ...
##  $ MajorAxisLength: num  92.2 74.7 76.3 77 85.1 ...
##  $ MinorAxisLength: num  64 51.4 52 51.9 56.4 ...
##  $ Eccentricity   : num  0.72 0.726 0.731 0.739 0.749 ...
##  $ ConvexArea     : int  4677 3015 3132 3157 3802 3080 3636 3866 2790 5819 ...
##  $ EquivDiameter  : num  76 60.5 62.3 62.6 68.6 ...
##  $ Extent         : num  0.658 0.713 0.759 0.784 0.769 ...
##  $ Perimeter      : num  273 208 210 211 230 ...
##  $ Roundness      : num  0.765 0.832 0.868 0.87 0.875 ...
##  $ AspectRation   : num  1.44 1.45 1.47 1.48 1.51 ...
##  $ Class          : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
head(df)
```

```
##   id Area MajorAxisLength MinorAxisLength Eccentricity ConvexArea EquivDiameter
## 1  1 4537        92.22932        64.01277    0.7199162       4677      76.00452
## 2  2 2872        74.69188        51.40045    0.7255527       3015      60.47102
## 3  3 3048        76.29316        52.04349    0.7312109       3132      62.29634
## 4  4 3073        77.03363        51.92849    0.7386387       3157      62.55130
## 5  5 3693        85.12478        56.37402    0.7492816       3802      68.57167
## 6  6 2990        77.41707        50.95434    0.7528609       3080      61.70078
##      Extent Perimeter Roundness AspectRation Class
## 1 0.6575362   273.085 0.7645096     1.440796     1
## 2 0.7130089   208.317 0.8316582     1.453137     1
## 3 0.7591532   210.012 0.8684336     1.465950     1
## 4 0.7835288   210.657 0.8702031     1.483456     1
## 5 0.7693750   230.332 0.8747433     1.510000     1
## 6 0.5848983   216.930 0.7984391     1.519342     1
```

```
tail(df)
```

```
##          id Area MajorAxisLength MinorAxisLength Eccentricity ConvexArea
```

```
## 18180 18180 5757        146.5050        50.65003    0.9383369       5859
## 18181 18181 5853        148.6246        51.02928    0.9392099       6008
## 18182 18182 7585        169.5940        58.14166    0.9393980       7806
## 18183 18183 6365        154.7771        52.90808    0.9397603       6531
## 18184 18184 5960        151.3979        51.47460    0.9404271       6189
## 18185 18185 6134        153.0820        51.59061    0.9415000       6283
##       EquivDiameter    Extent Perimeter Roundness AspectRation Class
## 18180      85.61565 0.4757851   326.511 0.6785947     2.892496     0
## 18181      86.32654 0.4985944   332.960 0.6634440     2.912535     0
## 18182      98.27269 0.6474605   385.506 0.6413619     2.916910     0
## 18183      90.02316 0.5612875   342.253 0.6828318     2.925396     0
## 18184      87.11204 0.4923992   343.371 0.6352269     2.941216     0
## 18185      88.37450 0.4899752   338.613 0.6722741     2.967245     0
```

```
class_occur <- data.frame(table(df$Class))
print(class_occur)
```

```
##   Var1 Freq
## 1    0 8200
## 2    1 9985
```

```
print(subset(df, Area == max(Area)))
```

```
##          id  Area MajorAxisLength MinorAxisLength Eccentricity ConvexArea
## 11502 11502 10210        165.9325        78.81152    0.8800067      10428
## 17651 17651 10210        177.2386        74.40660    0.9076118      10544
##       EquivDiameter    Extent Perimeter Roundness AspectRation Class
## 11502      114.0166 0.5949883   398.452 0.8081344     2.105435     0
## 17651      114.0166 0.6102445   417.438 0.7362946     2.382028     0
```

```
print(subset(df, Area == min(Area)))
```

```
##    id Area MajorAxisLength MinorAxisLength Eccentricity ConvexArea
## 49 49 2522        77.09079        42.87188    0.8311007       2579
##    EquivDiameter    Extent Perimeter Roundness AspectRation Class
## 49      56.66666 0.5987654   197.015 0.8165003     1.798167     1
```

```
## Split into train/test/validate
set.seed(1234)
groups <- c(train=0.6, test=0.2, validate=0.2)
i <- sample(cut(1:nrow(df), nrow(df)*cumsum(c(0,groups)), labels = names(groups)))
train <- df[i=="train",]
test <- df[i=="test",]
vald <- df[i=="validate",]
str(train)
```

```
## 'data.frame':    10911 obs. of  12 variables:
##  $ id             : int  1 2 3 4 5 6 8 9 11 15 ...
##  $ Area           : int  4537 2872 3048 3073 3693 2990 3788 2629 2665 4301 ...
##  $ MajorAxisLength: num  92.2 74.7 76.3 77 85.1 ...
##  $ MinorAxisLength: num  64 51.4 52 51.9 56.4 ...
##  $ Eccentricity   : num  0.72 0.726 0.731 0.739 0.749 ...
##  $ ConvexArea     : int  4677 3015 3132 3157 3802 3080 3866 2790 2777 4427 ...
##  $ EquivDiameter  : num  76 60.5 62.3 62.6 68.6 ...
##  $ Extent         : num  0.658 0.713 0.759 0.784 0.769 ...
##  $ Perimeter      : num  273 208 210 211 230 ...
##  $ Roundness      : num  0.765 0.832 0.868 0.87 0.875 ...
```

```
##  $ AspectRation   : num  1.44 1.45 1.47 1.48 1.51 ...
##  $ Class          : int  1 1 1 1 1 1 1 1 1 1 ...
```

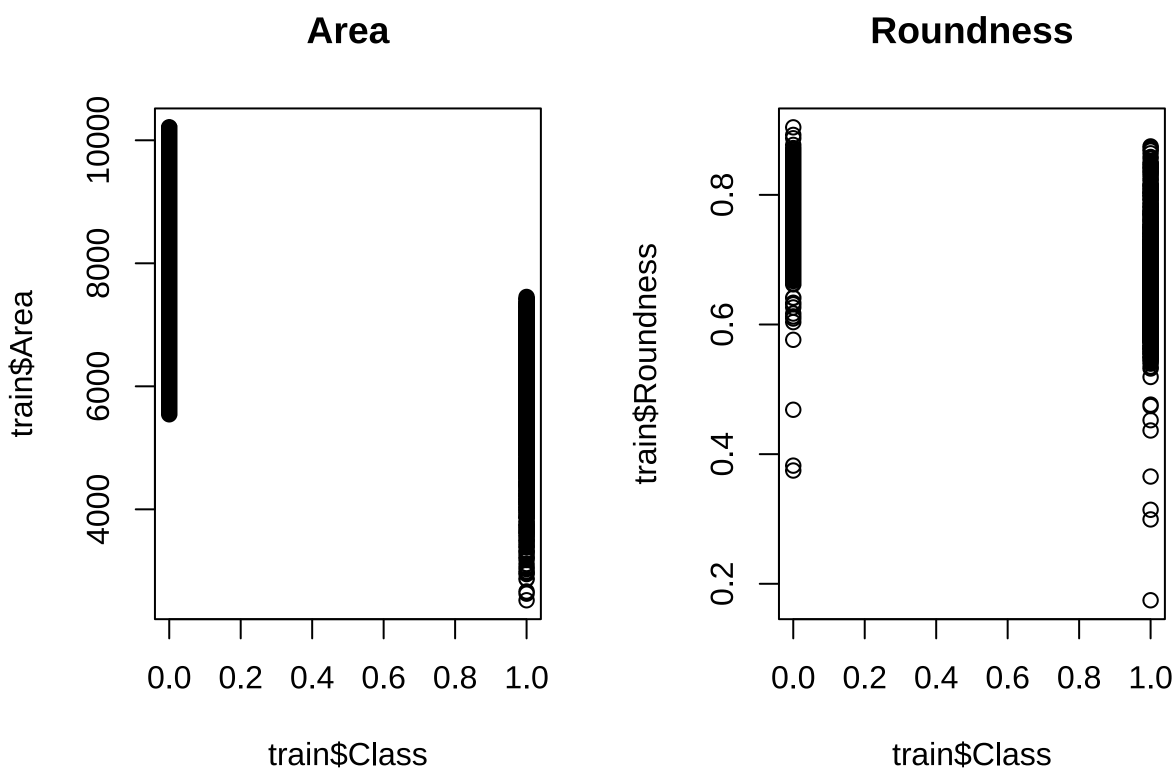
It looks like we have a much more equal split of gonen/jasmine rice, which is a pleasure to see after fiddling around with wine and spotify song datasets that seems so scattered. However, I still don't have any idea of how gonen and jasmine rice differ! Luckily we have some kernel methods to sift through.

```r
par(mfrow = c(1,2))
plot(train$Class, train$Area, data=train, main = "Area", varwidth=TRUE)
```

```
## Warning in plot.window(...): "data" is not a graphical parameter

## Warning in plot.window(...): "varwidth" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "data" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "varwidth" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "varwidth" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "varwidth" is not a
## graphical parameter

## Warning in box(...): "data" is not a graphical parameter

## Warning in box(...): "varwidth" is not a graphical parameter

## Warning in title(...): "data" is not a graphical parameter

## Warning in title(...): "varwidth" is not a graphical parameter
```

```r
plot(train$Class, train$Roundness, data=train, main = "Roundness", varwidth=TRUE)
```

```
## Warning in plot.window(...): "data" is not a graphical parameter

## Warning in plot.window(...): "varwidth" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "data" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "varwidth" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "varwidth" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "varwidth" is not a
## graphical parameter

## Warning in box(...): "data" is not a graphical parameter

## Warning in box(...): "varwidth" is not a graphical parameter

## Warning in title(...): "data" is not a graphical parameter

## Warning in title(...): "varwidth" is not a graphical parameter
```



SVM Regression

```
library(e1071)

svm1 <- svm(Class~., data=train, kernel="linear" , cost=10, scale = TRUE)
summary(svm1)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "linear", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.09090909
##     epsilon:  0.1
##
##
## Number of Support Vectors:  1461
```

Let's try some tuning

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

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

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.002145993
##
## - Detailed performance results:
##    cost       error   dispersion
## 1 1e-03 0.006390473 0.0013651699
## 2 1e-02 0.004971036 0.0012844447
## 3 1e-01 0.002570922 0.0004044403
## 4 1e+00 0.002182949 0.0003087127
## 5 5e+00 0.002152849 0.0003117028
## 6 1e+01 0.002150326 0.0003117880
## 7 1e+02 0.002145993 0.0003045929
```

Let's make a prediction based on our best model for svm1.

```
pred <- predict(tune_svm1$best.model, newdata=test)
options(max.print = 100) #limit output of table pred function
table(pred, test$Class)
```

```
##
## pred                    0 1
##   -0.170332163953867    1 0
##   -0.104211616016358    1 0
##   -0.0715125124784483   1 0
##   -0.0617165999468288   1 0
##   -0.0603068901526024   1 0
##   -0.0598986008502543   1 0
##   -0.0586396991001316   1 0
##   -0.0585206993511832   1 0
##   -0.0584196639215182   1 0
##   -0.0582180053943596   1 0
##   -0.0581328361005117   1 0
##   -0.0579371746199019   1 0
##   -0.0577535625982942   1 0
##   -0.0576863080381741   1 0
##   -0.0575527101665834   1 0
##   -0.0574538685898055   1 0
##   -0.0571269029203965   1 0
##   -0.0570607123071054   1 0
##   -0.056722241420452    1 0
##   -0.0566776679505081   1 0
##   -0.0564596861882738   1 0
##   -0.0563920895086157   1 0
##   -0.0562024776195346   1 0
##   -0.0561333290629018   1 0
##   -0.0560539602037037   1 0
##   -0.0558086680625426   1 0
##   -0.0557505182212761   1 0
##   -0.055672437654289    1 0
```

```
##   -0.0554594381563792   1 0
##   -0.0554562717267619   1 0
##   -0.0554553304174883   1 0
##   -0.0553801583656114   1 0
##   -0.055355020207288    1 0
##   -0.0552948667263502   1 0
##   -0.0550208808439542   1 0
##   -0.0550176909187372   1 0
##   -0.0548688705639058   1 0
##   -0.054848949307170 2  1 0
##   -0.0547780891316677   1 0
##   -0.0544378673494785   1 0
##   -0.0542183883903161   1 0
##   -0.0541589004629199   1 0
##   -0.0540601014510481   1 0
##   -0.0540182210918505   1 0
##   -0.0536229079578648   1 0
##   -0.0535348857747054   1 0
##   -0.05343988028945     1 0
##   -0.0533393750855944   1 0
##   -0.0532625512901258   1 0
##   -0.0532105867477445   1 0
##  [ reached getOption("max.print") -- omitted 3587 rows ]
```

Let's try meddling with our kernel types.

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

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "polynomial", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.09090909
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  537
```

```
#prediction
pred <- predict(svm2, newdata=test)
```

And radial kernels...

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

```
##
## Call:
```

```
## svm(formula = Class ~ ., 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:  785
```

```r
pred <- predict(svm3, newdata=test)
```

Let's tune our hyper parameters

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.001823944
##
## - Detailed performance results:
##     cost gamma       error   dispersion
## 1  1e-01   0.5 0.004559848 0.0010069682
## 2  1e+00   0.5 0.001825410 0.0004234979
## 3  1e+01   0.5 0.001823944 0.0004229951
## 4  1e+02   0.5 0.001823944 0.0004229951
## 5  1e+03   0.5 0.001823944 0.0004229951
## 6  1e-01   1.0 0.009865972 0.0018394328
## 7  1e+00   1.0 0.003918596 0.0008966973
## 8  1e+01   1.0 0.003915440 0.0008945071
## 9  1e+02   1.0 0.003915440 0.0008945071
## 10 1e+03   1.0 0.003915440 0.0008945071
## 11 1e-01   2.0 0.021458804 0.0030380542
## 12 1e+00   2.0 0.008826466 0.0016268660
## 13 1e+01   2.0 0.008820503 0.0016197051
## 14 1e+02   2.0 0.008820503 0.0016197051
## 15 1e+03   2.0 0.008820503 0.0016197051
## 16 1e-01   3.0 0.034612964 0.0044193665
## 17 1e+00   3.0 0.013976537 0.0021852877
## 18 1e+01   3.0 0.013968265 0.0021759613
## 19 1e+02   3.0 0.013968265 0.0021759613
## 20 1e+03   3.0 0.013968265 0.0021759613
```

```
## 21 1e-01   4.0 0.049146887 0.0057651937
## 22 1e+00   4.0 0.019225956 0.0026208510
## 23 1e+01   4.0 0.019218107 0.0026125641
## 24 1e+02   4.0 0.019218107 0.0026125641
## 25 1e+03   4.0 0.019218107 0.0026125641
```

Finally, from the tuned data output, we can see that we get best results with a cost = 10 and a gamma = 0.5.

```
svm_final <- svm(Class~., data = train, kernel="radial", cost = 10, gamma = 0.5, scale = TRUE)
summary(svm_final)
```

```
##
## Call:
## svm(formula = Class ~ ., data = train, kernel = "radial", cost = 10,
##     gamma = 0.5, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##     epsilon:  0.1
##
##
## Number of Support Vectors:  391
```

```
pred <- predict(svm_final, newdata = test)
```

Reflections

Let's take a look at how these kernels work.

Linear Kernels: Linear kernels are most advantageous when the data is linearly classifiable, so drawing a straight line across the plot would adequately separate the dataset. Commonly used when there are many features in a dataset.

Polynomial Kernels: Data is rarely ever linearly separable, so polynomial kernels add another dimension to allow so that we can find another way to fit the data to become linearly separable.

Radial Kernels: Radial kernels introduce an additional hyperparameter, the gamma parameter. Gamma manipulates the boundary of the hyperplane where lower values give sharper peaks and higher gamma values give more rounded peaks. Rounded peaks are more susceptible to points closer to the boundary, so we have a bias-variance tradeoff with the gamma value where we have a lower bias - higher variance with smaller gamma values (as we see with this rice classification) and a higher bias - lower variance with larger gamma values.

This was a fun dataset to mess around with. I think one of the most confusing factors is the gonen rice class itself. Google searches on gonen rice don't yield any real information other than a wikipedia article on the city of Gönen and a passing remark about it's rice production. So quite honesty Perhaps I'll have to interview some Turks over a nice plate of Turkish food to gather some more information.