Kernel and Ensemble Methods

SVM Classification

CS 4375 - Intro to Machine Learning

Dr. Karen Mazidi

Author:

- Leo Nguyen - ldn190002
- Cory Pekkala - cdp190005

Skin Segmentation Data Set

Data Set Information

- The skin dataset is collected by randomly sampling B,G,R values from face images of various age groups (young, middle, and old), race groups (white, black, and asian), and genders obtained from FERET database and PAL database. Total learning sample size is 245057; out of which 50859 is the skin samples and 194198 is non-skin samples. Color FERET Image Database: [Web Link], PAL Face Database from Productive Aging Laboratory, The University of Texas at Dallas: [Web Link].

Citation:

- Rajen Bhatt, Abhinav Dhall, 'Skin Segmentation Dataset', UCI Machine Learning Repository
- https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation (https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation)

Attribute Information:

- This dataset is of the dimension 245057 * 4 where first three columns are B,G,R (x1,x2, and x3 features) values and fourth column is of the class labels (decision variable y).

Load the data

```
df_origin <- read.csv("data/Skin Segmentation.csv", header=TRUE)
str(df_origin)
```

```
## 'data.frame':    245057 obs. of  4 variables:
##  $ B   : int  74 73 72 70 70 69 70 70 76 76 ...
##  $ G   : int  85 84 83 81 81 80 81 81 87 87 ...
##  $ R   : int  123 122 121 119 119 118 119 119 125 125 ...
##  $ Skin: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
dim(df_origin)
```

```
## [1] 245057      4
```

Data Cleaning

Create a subsample by produce a subset randomly from a data frame

- Just randomly pick up 10000 observations to create a subset

```
set.seed(1234)
df <- df_origin[sample(1:nrow(df_origin), 10000, replace = FALSE),]
str(df)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ B   : int  183 12 102 37 170 89 59 50 63 20 ...
##  $ G   : int  180 13 149 0 185 91 58 54 57 17 ...
##  $ R   : int  135 3 223 146 231 39 20 19 14 2 ...
##  $ Skin: int  2 2 1 2 1 2 2 2 2 2 ...
```

```
dim(df)
```

```
## [1] 10000     4
```

- Convert column Skin to factor, and label them as Skin and NonSkin

```
df$Skin <- factor(df$Skin)
levels(df$Skin) <- c("Skin", "NonSkin")
str(df)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ B   : int  183 12 102 37 170 89 59 50 63 20 ...
##  $ G   : int  180 13 149 0 185 91 58 54 57 17 ...
##  $ R   : int  135 3 223 146 231 39 20 19 14 2 ...
##  $ Skin: Factor w/ 2 levels "Skin","NonSkin": 2 2 1 2 1 2 2 2 2 2 ...
```

Divide into train, test, validate

```
set.seed(1234)
groups <- c(train=.6, test=.2, validate=.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",]
```

Data Exploration

1. Look at the first 10 rows in training data

```
head(train, n=10)
```

```
##          B   G   R    Skin
## 126055  59  58  20 NonSkin
## 120410  50  54  19 NonSkin
## 83023   63  57  14 NonSkin
## 80756   20  17   2 NonSkin
## 85374   22  51  35 NonSkin
## 199230 148 231 232 NonSkin
## 59944  173 173 127 NonSkin
## 199608 199 197 162 NonSkin
## 148452 154 153  95 NonSkin
## 164319 174 169 124 NonSkin
```

2. View the summary of entire training data set

```
summary(train)
```

```
##        B                G               R            Skin
##  Min.   :  0.00   Min.   :  0.0   Min.   :  0   Skin   :1263
##  1st Qu.: 69.75   1st Qu.: 87.0   1st Qu.: 71   NonSkin:4737
##  Median :139.00   Median :153.0   Median :127
##  Mean   :125.59   Mean   :132.2   Mean   :123
##  3rd Qu.:176.00   3rd Qu.:177.0   3rd Qu.:165
##  Max.   :255.00   Max.   :255.0   Max.   :255
```

3. Check the level, encoding matrix of Skin sample in training data

```
levels(train$Skin)
```

```
## [1] "Skin"    "NonSkin"
```

```
contrasts(train$Skin)
```

```
##         NonSkin
## Skin          0
## NonSkin       1
```

4. Summary statistics by group.

```
aggregate(.~train$Skin, train, mean)
```

```
##   train$Skin        B        G        R Skin
## 1       Skin 112.1718 145.4014 203.1449    1
## 2    NonSkin 129.1737 128.6795 101.6597    2
```

5. Number of sample is Skin and NonSkin

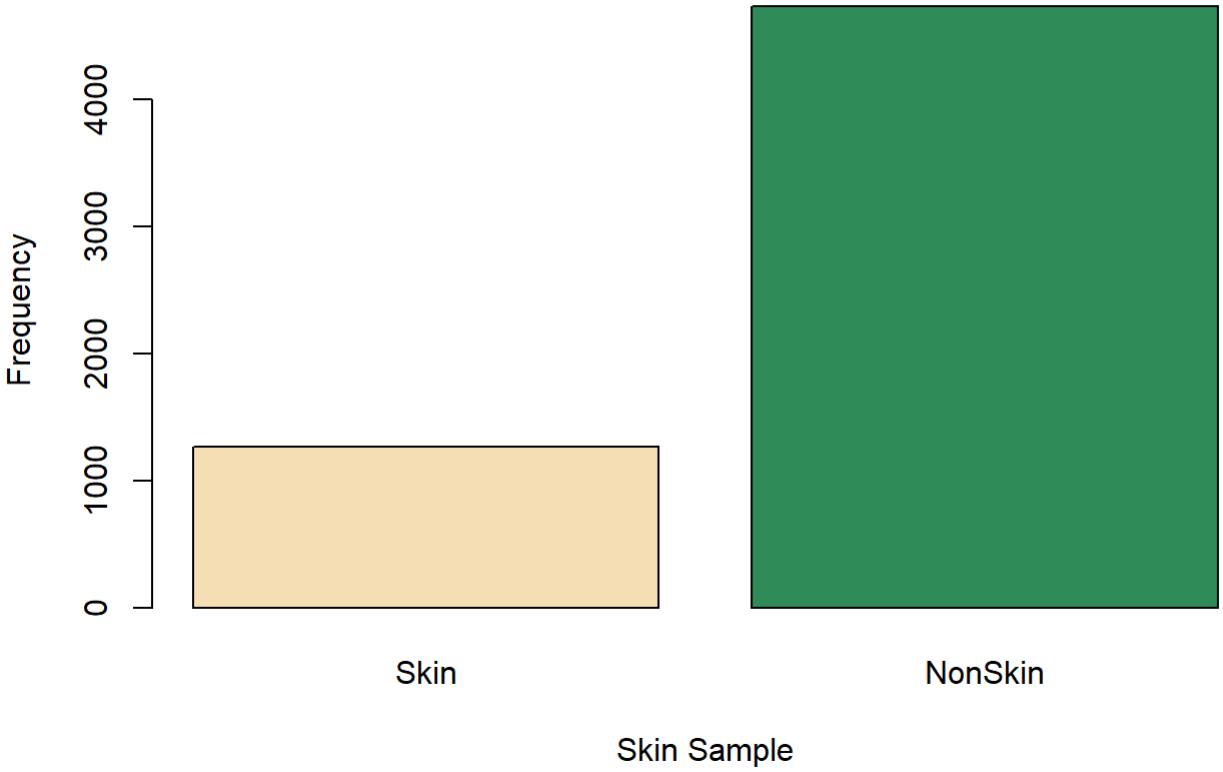
```
table(train$Skin)
```

```
##
##    Skin NonSkin
##    1263    4737
```

Data Visualization using informative graph

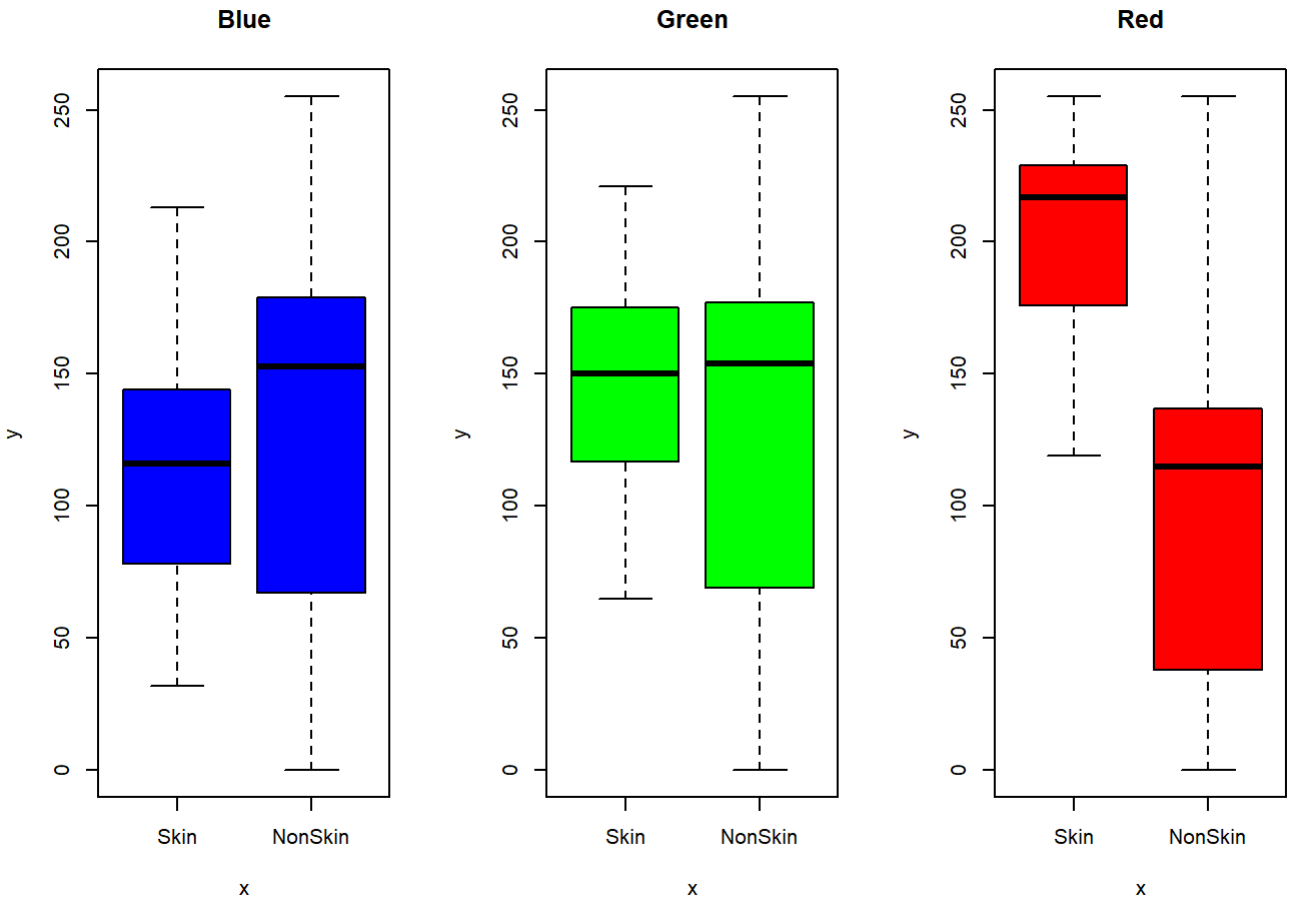
1. Using barplots to visualize the 2 factor level of Skin Sample

```
counts <- table(train$Skin)
barplot(counts, xlab="Skin Sample", ylab="Frequency", col=c("wheat", "seagreen"))
```



2. Visualize the relationship between Skin and B, G, R

```
par(mfrow=c(1,3))
plot(train$Skin, train$B, data = train, main = "Blue", col="blue")
plot(train$Skin, train$G, data = train, main = "Green", col="green")
plot(train$Skin, train$R, data = train, main = "Red", col="red")
```



SVM Classitification with Linear

Building linear kernel with random cost=10

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

```
##
## Call:
## svm(formula = Skin ~ ., data = train, kernel = "linear", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  1171
##
##  ( 586 585 )
##
##
## Number of Classes:  2
##
## Levels:
##  Skin NonSkin
```

Evaluate linear kernel with cost=10

```
pred1 <- predict(svm1, newdata = test)
table(pred1, test$Skin)
```

```
##
## pred1     Skin NonSkin
##   Skin     386     120
##   NonSkin   34    1460
```

```
acc_ln <- mean(pred1==test$Skin)

print(paste('Accuracy:', acc_ln))
```

```
## [1] "Accuracy: 0.923"
```

- Even with a random cost=10 we have a very good accuracy 92.3% for SVM linear kernel. It is because SVM works well when there is a clear cut between classes. As we can see on the box plot above, there is a clear cut between Skin and NonSkin mean value of Red and Blue,

Tuning cost hyperparameter

- After tuning, the best cost value is 0.01

```
set.seed(1234)
tune_svm1 <- tune(svm, Skin~., 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
##  0.01
##
## - best performance: 0.0635
##
## - Detailed performance results:
##    cost  error dispersion
## 1 1e-03 0.2175 0.04244277
## 2 1e-02 0.0635 0.01415195
## 3 1e-01 0.0690 0.01505545
## 4 1e+00 0.0725 0.01687371
## 5 5e+00 0.0710 0.01612452
## 6 1e+01 0.0710 0.01612452
## 7 1e+02 0.0710 0.01612452
```

Evaluate linear kernel with cost=0.01

```
pred1_tune <- predict(tune_svm1$best.model, newdata = test)
table(pred1_tune, test$Skin)
```

```
##
## pred1_tune Skin NonSkin
##    Skin     402     102
##    NonSkin   18    1478
```

```
acc_ln_tune <- mean(pred1_tune==test$Skin)

print(paste('Accuracy:', acc_ln_tune))
```

```
## [1] "Accuracy: 0.94"
```

- The accuracy improve from 92.3% to 94% because after tuning we have a better value for cost which is 0.01. This happen because the new cost hyper parameter was significantly reduce from 10 to 0.01. When we have smaller C hyper-parameter, it means we allow less slack variables on the model. This definitely improve the accuracy, but the model tend to be more overfiting.

SVM Classitification with polynomial

Building polynomial kernel with random cost=10

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

```
##
## Call:
## svm(formula = Skin ~ ., data = train, kernel = "polynomial", cost = 10,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  1041
##
##  ( 524 517 )
##
##
## Number of Classes:  2
##
## Levels:
##  Skin NonSkin
```

Evaluate polynomial kernel with cost=10

```
pred2 <- predict(svm2, newdata = test)
table(pred2, test$Skin)
```

```
##
## pred2     Skin NonSkin
##   Skin     365      43
##   NonSkin   55    1537
```

```
acc_poly <- mean(pred2==test$Skin)

print(paste('Accuracy:', acc_poly))
```

```
## [1] "Accuracy: 0.951"
```

- When we use the same c=10 without tuning on SVM polynomial kernel the accuracy improve quite a a bit compare to linear kernel. It is because the polynomial provide a higher and more complex shape on hyperplanes. Now, we can map the data to a higher dimension which have better chance to separate the classes correctly.

Tuning cost hyperparameter

- After tuning, the best cost value is 100

```
set.seed(1234)
tune_svm2 <- tune(svm, Skin~., data=vald, kernel="polynomial", ranges=list(cost=c(0.001, 0.01,
0.1, 1, 5, 10, 100)))
summary(tune_svm2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.0605
##
## - Detailed performance results:
##    cost  error dispersion
## 1 1e-03 0.2175 0.04244277
## 2 1e-02 0.1995 0.03312015
## 3 1e-01 0.1150 0.02198484
## 4 1e+00 0.0695 0.02006240
## 5 5e+00 0.0640 0.02118700
## 6 1e+01 0.0630 0.02002776
## 7 1e+02 0.0605 0.02100926
```

Evaluate polynomial kernel with cost=100

```
pred2_tune <- predict(tune_svm2$best.model, newdata = test)
table(pred2_tune, test$Skin)
```

```
##
## pred2_tune Skin NonSkin
##    Skin     358      51
##    NonSkin   62    1529
```

```
acc_poly_tune <- mean(pred2_tune==test$Skin)

print(paste('Accuracy:', acc_poly_tune))
```

```
## [1] "Accuracy: 0.9435"
```

- The accuracy reduce from 95% to 94.35% on SVM polynomial kernel because after tunning, the new cost increase from 10 to 100. When the cost increase, more slack variable allow which mean the accuracy will decrease. This is happen because the main objective of SVM to find the best separating hyperplanes with

the largest margin value. So in the case, best model is the compromise between reduce overfitting and increase the margin on support vector.

SVM Classitification with radial

Building radial kernel with random cost=10, gamma=1

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

```
##
## Call:
## svm(formula = Skin ~ ., data = train, kernel = "radial", cost = 10,
##     gamma = 1, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  77
##
##  ( 38 39 )
##
##
## Number of Classes:  2
##
## Levels:
##  Skin NonSkin
```

Evaluate radial kernel with cost=10, gamma=1

```
pred3 <- predict(svm3, newdata = test)
table(pred3, test$Skin)
```

```
##
## pred3     Skin NonSkin
##   Skin     420       2
##   NonSkin    0    1578
```

```
acc_rad <- mean(pred3==test$Skin)

print(paste('Accuracy:', acc_rad))
```

```
## [1] "Accuracy: 0.999"
```

- The SVM radial kernel has the best accuracy so far 99.9%. It is because the radial have the highest and most complex dimension compare to linear and polynomial. However, because we randomly choose the

small cost = 10 and large gamma=1, the model tend to be overfiting which explain a very high accuracy.

Tuning cost and gamma hyperparameter

- After tuning, the best cost value is 100, and gamma is 0.5

```
set.seed(1234)
tune_svm3 <- tune(svm, Skin~., data=vald, kernel="radial", ranges=list(cost=c(0.001, 0.01, 0.1,
1, 5, 10, 100),
                                                                       gamma=c(0.5,1,2,3,4)))
summary(tune_svm3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100   0.5
##
## - best performance: 0.001
##
## - Detailed performance results:
##     cost gamma  error  dispersion
## 1  1e-03   0.5 0.2175 0.042442772
## 2  1e-02   0.5 0.0435 0.015643600
## 3  1e-01   0.5 0.0180 0.009189366
## 4  1e+00   0.5 0.0070 0.004830459
## 5  5e+00   0.5 0.0040 0.004594683
## 6  1e+01   0.5 0.0035 0.004116363
## 7  1e+02   0.5 0.0010 0.002108185
## 8  1e-03   1.0 0.2175 0.042442772
## 9  1e-02   1.0 0.0330 0.013984118
## 10 1e-01   1.0 0.0130 0.007527727
## 11 1e+00   1.0 0.0040 0.004594683
## 12 5e+00   1.0 0.0040 0.004594683
## 13 1e+01   1.0 0.0035 0.004116363
## 14 1e+02   1.0 0.0015 0.002415229
## 15 1e-03   2.0 0.2175 0.042442772
## 16 1e-02   2.0 0.0525 0.021634592
## 17 1e-01   2.0 0.0090 0.006146363
## 18 1e+00   2.0 0.0040 0.004594683
## 19 5e+00   2.0 0.0025 0.003535534
## 20 1e+01   2.0 0.0020 0.002581989
## 21 1e+02   2.0 0.0020 0.002581989
## 22 1e-03   3.0 0.2175 0.042442772
## 23 1e-02   3.0 0.0945 0.030409246
## 24 1e-01   3.0 0.0070 0.007527727
## 25 1e+00   3.0 0.0035 0.004116363
## 26 5e+00   3.0 0.0020 0.002581989
## 27 1e+01   3.0 0.0020 0.002581989
## 28 1e+02   3.0 0.0020 0.002581989
## 29 1e-03   4.0 0.2175 0.042442772
## 30 1e-02   4.0 0.1270 0.039524957
## 31 1e-01   4.0 0.0065 0.007090682
## 32 1e+00   4.0 0.0020 0.003496029
## 33 5e+00   4.0 0.0010 0.002108185
## 34 1e+01   4.0 0.0015 0.002415229
## 35 1e+02   4.0 0.0015 0.002415229
```

Evaluate radial kernel with cost=100, gamma=0.5

```
pred3_tune <- predict(tune_svm3$best.model, newdata = test)
table(pred3_tune, test$Skin)
```

```
##
## pred3_tune Skin NonSkin
##    Skin     420       5
##    NonSkin    0    1575
```

```
acc_rad_tune <- mean(pred3_tune==test$Skin)

print(paste('Accuracy:', acc_rad_tune))
```

```
## [1] "Accuracy: 0.9975"
```

- The accuracy reduce from 99.9% to 99.75% because after tuning we have larger cost=100 and smaller gamma=0.5. These new value reduce the overfitting on the model. This happen with same reason on polynomial kernel, the SVM main objective is find the best separating hyperplane with the biggest margin. SO after tuning, and considering all these factor the algorithm pick up these value.

Conclusion

- The SVM model will have the better accuracy when we increase the higher dimension and more complex shape on separating hyperplane from linear to polynomial and finally to radial. In general after tuning, we will gave the better accuracy with new tuned hyper-parameter. But it is not always the case, because, we want a model that can generalize the data than the model tend to be overfitting. This is also the main objective of SVM model, it need to find the best separating hyperplan with largest margin. That is why sometime, we have the accuracy reduce after tuned value.