ST563 Project

Henry van Eijk

2025-03-30

Read in the Data:

My data is based off 900 images of two different types of raisins. The Kecimen and the Besni. There are 900 observations in the dataset with exactly 450 for each type. The target variable is the type of raisin, either Kecimen or Besni. The goal of the project is to classify whether an observation is a Kecimen or Besni raisin conditioned on the given seven features: Area, Perimeter, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, and Extent. This task is important in biological applications.

```
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.3.3
```

```
data <- read.xlsx("/Users/henryvaneijk/Desktop/Raisin_Dataset.xlsx")
head(data)
```

```
##    Area MajorAxisLength MinorAxisLength Eccentricity ConvexArea    Extent
## 1 87524        442.2460        253.2912    0.8197384      90546 0.7586506
## 2 75166        406.6907        243.0324    0.8018052      78789 0.6841296
## 3 90856        442.2670        266.3283    0.7983536      93717 0.6376128
## 4 45928        286.5406        208.7600    0.6849892      47336 0.6995994
## 5 79408        352.1908        290.8275    0.5640113      81463 0.7927719
## 6 49242        318.1254        200.1221    0.7773513      51368 0.6584564
##   Perimeter   Class
## 1  1184.040 Kecimen
## 2  1121.786 Kecimen
## 3  1208.575 Kecimen
## 4   844.162 Kecimen
## 5  1073.251 Kecimen
## 6   881.836 Kecimen
```

Data Cleaning & Transformations:

```
str(data)
```

```
## 'data.frame':    900 obs. of  8 variables:
##  $ Area           : num  87524 75166 90856 45928 79408 ...
##  $ MajorAxisLength: num  442 407 442 287 352 ...
##  $ MinorAxisLength: num  253 243 266 209 291 ...
##  $ Eccentricity   : num  0.82 0.802 0.798 0.685 0.564 ...
##  $ ConvexArea     : num  90546 78789 93717 47336 81463 ...
##  $ Extent         : num  0.759 0.684 0.638 0.7 0.793 ...
##  $ Perimeter      : num  1184 1122 1209 844 1073 ...
##  $ Class          : chr  "Kecimen" "Kecimen" "Kecimen" "Kecimen" ...
```

```
summary(data)
```

```
##       Area        MajorAxisLength MinorAxisLength  Eccentricity
##  Min.   : 25387   Min.   :225.6   Min.   :143.7   Min.   :0.3487
##  1st Qu.: 59348   1st Qu.:345.4   1st Qu.:219.1   1st Qu.:0.7418
##  Median : 78902   Median :407.8   Median :247.8   Median :0.7988
##  Mean   : 87804   Mean   :430.9   Mean   :254.5   Mean   :0.7815
##  3rd Qu.:105028   3rd Qu.:494.2   3rd Qu.:279.9   3rd Qu.:0.8426
##  Max.   :235047   Max.   :997.3   Max.   :492.3   Max.   :0.9621
##    ConvexArea         Extent          Perimeter         Class
##  Min.   : 26139   Min.   :0.3799   Min.   : 619.1   Length:900
##  1st Qu.: 61513   1st Qu.:0.6709   1st Qu.: 966.4   Class :character
##  Median : 81651   Median :0.7074   Median :1119.5   Mode  :character
##  Mean   : 91186   Mean   :0.6995   Mean   :1165.9
##  3rd Qu.:108376   3rd Qu.:0.7350   3rd Qu.:1308.4
##  Max.   :278217   Max.   :0.8355   Max.   :2697.8
```

```
missing_summary <- colSums(is.na(data))
print(missing_summary)
```

```
##            Area MajorAxisLength MinorAxisLength    Eccentricity      ConvexArea
##               0               0               0               0               0
##          Extent       Perimeter           Class
##               0               0               0
```

```
data$Class <- ifelse(data$Class == "Kecimen", 1, 0)
```

There are no issues with the data, no missing values, etc. I did one transformation where I created a binary indicator for the target variable so we can use it correctly for modeling.

Split the Data into a Train and Test Set:

```
set.seed(123)
n <- nrow(data)
# Create the training indicies then define train and test set
train_idx <- sample(seq_len(n), size = round(0.8 * n))
train_data <- data[train_idx, ]
test_data <- data[-train_idx, ]
# Need to treat as a factor so R does not do regression
train_data$Class <- factor(train_data$Class, levels = c(0, 1),
                           labels = c("No", "Yes"))
test_data$Class  <- factor(test_data$Class,  levels = c(0, 1),
                           labels = c("No", "Yes"))
# Create y_train, y_test, x_train, and x_test
x_train <- subset(train_data, select = -Class)
y_train <- train_data$Class
x_test <- subset(test_data, select = -Class)
y_test <- test_data$Class
# I will go ahead and scale the predictors once
x_train <- scale(x_train)
x_test  <- scale(x_test,
                 center = attr(x_train, "scaled:center"),
                 scale  = attr(x_train, "scaled:scale"))
# I use the package 'caret' thus I define the control once for CV
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.2
```

```
ctrl <- trainControl(
  method = "cv",
  number = 5
)
```

Next, I will go ahead and use Caret for all the following models. It automatically does the hyperparameter tuning since I specified the trainControl above. I am going to save all of these models, then once done training, I will show the testing results. This training step is basically identical for each model. I set class as the target variable, use the training data, specify which model, set accuracy as the metric, set the ctrl to what I defined above (i.e, 5 fold CV), and set the number of hyperparameters to try if needed. I also describe the model below using the 5 bullet points from the project description.

For CV, instead of specifying a grid of hyperparameters to use, I use caret's build in tuneLength parameter. It uses some built-in heuristics to pick values of the hyperparameter, essentially creating tuneGrid for you.

kNN:

```
set.seed(123)
knn_fit <- train(Class ~ .,
                 data       = train_data,
                 method     = "knn",
                 metric     = "Accuracy",
                 trControl  = ctrl,
                 tuneLength = 10)
```

1. Non-parametric 2) Yes k is the tuning parameter which represents the how many neighbors to consider 3) No, it's difficult since its non-parametric 4) No 5) Yes because this algorithm measures distances so it could be smart to do so

Penalized Logistic Regression:

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
set.seed(123)
glm_fit <- train(
  Class ~ .,
  data      = train_data,
  method    = "glmnet",
  trControl = ctrl,
  metric    = "Accuracy",
  tuneLength = 10
)
```

1. Parametric 2) Yes alpha controls the Lasso vs. ridge (e.g., alpha=1 indicates LASSO) while lambda is the strength of the penalty 3) Yes, you can look at p-values etc 4) If LASSO, then yes 5) Penalized regression does require standarization

GAMs:

```
set.seed(123)
gam_fit <- train(
  Class ~ .,
  data      = train_data,
  method    = "gam",
  trControl = ctrl,
  metric    = "Accuracy"
)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
## Warning: package 'nlme' was built under R version 4.3.3
```

```
## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
```

1. Parametric 2) Yes the degree of each basis function 3) Yes, you can look at p-values etc 4) No 5) Not required

Single tree:

```
set.seed(123)
tree_fit <- train(
  Class ~ .,
  data      = train_data,
  method    = "rpart",
  trControl = ctrl,
  metric    = "Accuracy",
  tuneLength = 10
)
```

1. Non-parametric 2) Yes the tree depth which controls how many nodes in the TREE 3) They are interpretable but no ways to get p-values like linear regression 4) Technically yes since the tree picks what feature to split on and features could be left out 5) Not required

Random forest:

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(123)
rf_fit <- train(
  Class ~ .,
  data      = train_data,
  method    = "rf",
  trControl = ctrl,
  metric    = "Accuracy",
  tuneLength = 5
)
```

1. Non-parametric 2) Yes a tuning parameter that controls how many predictors to consider when splitting 3) No more black-box due to many trees 4) Similar to the tree case 5) Not required

SVM:

```
library("kernlab")
```

```
## Warning: package 'kernlab' was built under R version 4.3.3
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
set.seed(123)
svm_fit <- train(
  Class ~ .,
  data      = train_data,
  method    = "svmRadial",
  trControl = ctrl,
  metric    = "Accuracy",
  tuneLength = 5
)
```

1. Non-parametric 2) Yes the margin violation hyperparameter 3) No 4) No 5) Yes

Below, I produce all the confusion matrices. Then I display the test set accuracy across all models. As you can see, the GAM model produced the highest out-of-sample accuracy of 87.7%. I chose accuracy for simplicity. In certain settings such as medicine, preventing false negatives are crucial. Here, we are just classifying which type of raisin so no need to overcomplicate things.

Testing:

```
# kNN predictions on test
knn_preds  <- predict(knn_fit,  newdata = test_data)
cm_knn     <- confusionMatrix(knn_preds, test_data$Class)

# Logistic predictions on test
glm_preds  <- predict(glm_fit,  newdata = test_data)
cm_glm     <- confusionMatrix(glm_preds, test_data$Class)

# GAM predictions on test
gam_preds  <- predict(gam_fit,  newdata = test_data)
cm_gam     <- confusionMatrix(gam_preds, test_data$Class)

# Tree predictions on test
tree_preds <- predict(tree_fit, newdata = test_data)
cm_tree    <- confusionMatrix(tree_preds, test_data$Class)

# RF predictions on test
rf_preds   <- predict(rf_fit,   newdata = test_data)
cm_rf      <- confusionMatrix(rf_preds, test_data$Class)

# SVM predictions on test
svm_preds  <- predict(svm_fit,  newdata = test_data)
cm_svm     <- confusionMatrix(svm_preds, test_data$Class)

# Compare final accuracy
acc_knn  <- cm_knn$overall["Accuracy"]
acc_glm  <- cm_glm$overall["Accuracy"]
acc_gam  <- cm_gam$overall["Accuracy"]
acc_tree <- cm_tree$overall["Accuracy"]
acc_rf   <- cm_rf$overall["Accuracy"]
acc_svm  <- cm_svm$overall["Accuracy"]

# Show the results in a single table
results <- data.frame(
  Model    = c("kNN","Logistic","GAM","Tree","RF","SVM"),
  Accuracy = c(acc_knn, acc_glm, acc_gam, acc_tree, acc_rf, acc_svm)
)
print(results)
```

```
##      Model  Accuracy
## 1      kNN 0.8611111
## 2 Logistic 0.8666667
## 3      GAM 0.8777778
## 4     Tree 0.8388889
## 5       RF 0.8500000
## 6      SVM 0.8500000
```

Fit entire model to dataset:

```
# Combine test and train sets so I can fit across all data
combined_x <- rbind(x_train, x_test)
combined_y <- c(y_train, y_test)
all_data <- data.frame(Class = combined_y, combined_x)
all_data$Class <- factor(all_data$Class, levels = c("No", "Yes"))
# Refit across all data
final_gam <- train(
  Class ~ .,
  data      = all_data,
  method    = "gam",
  trControl = trainControl(method = "none")
)
# Summarize for inference
summary(final_gam$finalModel)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## .outcome ~ s(ConvexArea) + s(Area) + s(MajorAxisLength) + s(MinorAxisLength) +
##     s(Eccentricity) + s(Extent) + s(Perimeter)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.978      1.001  -1.977   0.0481 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                          edf Ref.df Chi.sq  p-value
## s(ConvexArea)      1.000e+00      9  9.583 0.000208 ***
## s(Area)            3.537e+00      9 11.202 0.003476 **
## s(MajorAxisLength) 2.325e-04      9  0.000 0.695388
## s(MinorAxisLength) 6.722e+00      9  9.632 0.109653
## s(Eccentricity)    2.426e+00      9  8.174 0.010411 *
## s(Extent)          3.688e-05      9  0.000 0.955450
## s(Perimeter)       6.784e+00      9 40.119  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.644   Deviance explained = 59.6%
## UBRE = -0.39169  Scale est. = 1         n = 900
```

In terms of inference, it looks like the p-values for s(MajorAxisLength), s(MinorAxisLength), and s(Extent) are > alpha=0.05. Thus these features are not significant to the model.