**R: Naive Bayes**

*Goal*: Perform Naive Bayes model on the Iris data.

*Data*: Iris data from the standard R package

**Table of Contents**

1 ---- Description of Variables and Data Set-up

1 ---- Naïve Bayes

2 ---- Model Evaluation

5 ---- Receiver Operating Characteristic (ROC) Curve

8 ---- Results Table

**Description of Variables and Data Set-Up**

The Iris Dataset includes the following variables describing flower observations:

* **Sepal.Length** - length of outer section of flower
* **Sepal.Width** - width of outer section of flower
* **Petal.Length** - length of flower's petal
* **Petal.Width** - width of flower's petal
* **Species** - type of Iris flower

As usual, we begin by importing necessary libraries, setting a seed, and performing a train-test split on the data. The **e1071** library implements the Naïve Bayes model, while **pROC** is used to plot the Receiver Operating Curve to evaluate the classification performance.

library(e1071)

library(pROC)

data(iris)

set.seed(1)

# train-test split

training.indices <- sample(1:nrow(iris), 0.7 \* nrow(iris))

x.train <- iris[training.indices, 1:4]

x.test <- iris[-training.indices, 1:4]

y.train <- iris[training.indices, 5]

y.test <- iris[-training.indices, 5]

**Naïve Bayes**

Naïve Bayes applies Bayes’ theorem to estimate the probability of a certain class given a set of data attributes. Let us apply the model to a multi-class classification problem – predicting the species in the Iris Dataset - and output confusion matrices for the test set, as well as the training set and the full dataset.

The **naiveBayes** function constructs a Naïve Bayes classifier model trained on the specified data:

* x - matrix of predictor variables
* y - class vector for target variable

Other parameters allow specifying specific predictors to use in a formula, specifying a subset of data to

use, and the type of data (probabilities or classes) to output. See **?naiveBayes** for more information.

# Construct the model

model <- naiveBayes(x.train, y.train)

# Predict for training

pred.train <- predict(model, x.train, type="class")

conf.train <- table(pred.train, y.train)

conf.train

y.train

pred.train setosa versicolor virginica

setosa 35 0 0

versicolor 0 31 3

virginica 0 2 34

# Predict for testing

# type=class tells model to output specific class predicted (setosa, versicolor, or # virginica) rather than a probability for each

pred <- predict(model, x.test, type="class")

conf <- table(pred, y.test)

conf

y.test

pred setosa versicolor virginica

setosa 15 0 0

versicolor 0 17 1

virginica 0 0 12

# Predict for all data

pred.all <- predict(model, iris[1:4], type="class")

conf.all <- table(pred.all, iris$Species)

conf.all

pred.all setosa versicolor virginica

setosa 50 0 0

versicolor 0 48 4

virginica 0 2 46

Calculating accuracy for multi-class classification amounts to summing the number of correctly predicted classes and dividing by the total number of entries in the confusion matrix.

Let us calculate accuracy and classification error for each confusion matrix we generated.

# Diagonal represents correct classifications

# Train set

acc.train <- sum(diag(conf.train)) / sum(conf.train)

acc.train

[1] 0.952381

err.train <- 1 - acc.train

err.train

[1] 0.04761905

# Test set

acc <- sum(diag(conf)) / sum(conf)

acc

[1] 0.9777778

err <- 1 - acc

err

[1] 0.02222222

# All data

acc.all <- sum(diag(conf.all)) / sum(conf.all)

acc.all

[1] 0.96

err.all <- 1 - acc.all

err.all

[1] 0.04

**Model Evaluation**

Previously, we calculated a number of metrics for classification, including sensitivity, specificity, positive predictive value (PPV), and negative predictive value (NPV). In the multi-class case, we calculate these for each individual class using a “One-Versus-All” approach.

Here, true positives are the number of cases correct for the given class, while true negatives are the correct predictions for all other classes. False positives are incorrect predictions for the given class, while false negatives are all incorrect predictions for other classes.

These metrics can be calculated using the formulas below.

### Train Set Metrics ###

macro.sens <- 0

macro.spec <- 0

# Setosa

sens.train <- conf.train['setosa','setosa'] / sum(conf.train[,'setosa'])

sens.train

[1] 1

spec.train <- sum(diag(conf.train[2:3,2:3])) / (sum(diag(conf.train[2:3,2:3])) + sum(conf.train['setosa',2:3]))

spec.train

[1] 1

ppv.train <- conf.train['setosa','setosa'] / sum(conf.train['setosa',])

ppv.train

[1] 1

npv.train <- sum(diag(conf.train[2:3,2:3])) / (sum(diag(conf.train[2:3,2:3])) + sum(conf.train[2:3,'setosa']))

npv.train

[1] 1

macro.sens <- macro.sens + sens.train

macro.spec <- macro.spec + spec.train

# versicolor

sens.train <- conf.train['versicolor','versicolor'] / sum(conf.train[,'versicolor'])

sens.train

[1] 0.9393939

spec.train <- sum(diag(conf.train[c(1,3),c(1,3)])) / (sum(diag(conf.train[c(1,3),c(1,3)])) + sum(conf.train['versicolor',c(1,3)]))

spec.train

[1] 0.9583333

ppv.train <- conf.train['versicolor','versicolor'] / sum(conf.train['versicolor',])

ppv.train

[1] 0.9117647

npv.train <- sum(diag(conf.train[c(1,3),c(1,3)])) / (sum(diag(conf.train[c(1,3),c(1,3)])) + sum(conf.train[c(1,3),'versicolor']))

npv.train

[1] 0.971831

macro.sens <- macro.sens + sens.train

macro.spec <- macro.spec + spec.train

# virginica

sens.train <- conf.train['virginica','virginica'] / sum(conf.train[,'virginica'])

sens.train

[1] 0.9189189

spec.train <- sum(diag(conf.train[1:2,1:2])) / (sum(diag(conf.train[1:2,1:2])) + sum(conf.train['virginica',1:2]))

spec.train

[1] 0.9705882

ppv.train <- conf.train['virginica','virginica'] / sum(conf.train['virginica',])

ppv.train

[1] 0.9444444

npv.train <- sum(diag(conf.train[1:2,1:2])) / (sum(diag(conf.train[1:2,1:2])) + sum(conf.train[1:2,'virginica']))

npv.train

[1] 0.9565217

macro.sens <- macro.sens + sens.train

macro.spec <- macro.spec + spec.train

macro.sens <- macro.sens / 3

macro.spec <- macro.spec / 3

macro.sens

[1] 0.952771

macro.spec

[1] 0.9763072

### Test Set Metrics ###

macro.sens <- 0

macro.spec <- 0

# setosa

sens <- conf['setosa','setosa'] / sum(conf[,'setosa'])

sens

[1] 1

spec <- sum(diag(conf[2:3,2:3])) / (sum(diag(conf[2:3,2:3])) + sum(conf['setosa',2:3]))

spec

[1] 1

ppv <- conf['setosa','setosa'] / sum(conf['setosa',])

ppv

[1] 1

npv <- sum(diag(conf[2:3,2:3])) / (sum(diag(conf[2:3,2:3])) + sum(conf[2:3,'setosa']))

npv

[1] 1

macro.sens <- macro.sens + sens

macro.spec <- macro.spec + spec

# versicolor

sens <- conf['versicolor','versicolor'] / sum(conf[,'versicolor'])

sens

[1] 1

spec <- sum(diag(conf[c(1,3),c(1,3)])) / (sum(diag(conf[c(1,3),c(1,3)])) + sum(conf['versicolor',c(1,3)]))

spec

[1] 0.9642857

ppv <- conf['versicolor','versicolor'] / sum(conf['versicolor',])

ppv

[1] 0.9444444

npv <- sum(diag(conf[c(1,3),c(1,3)])) / (sum(diag(conf[c(1,3),c(1,3)])) + sum(conf[c(1,3),'versicolor']))

npv

[1] 1

macro.sens <- macro.sens + sens

macro.spec <- macro.spec + spec

# virginica

sens <- conf['virginica','virginica'] / sum(conf[,'virginica'])

sens

[1] 0.9230769

spec <- sum(diag(conf[1:2,1:2])) / (sum(diag(conf[1:2,1:2])) + sum(conf['virginica',1:2]))

spec

[1] 1

ppv <- conf['virginica','virginica'] / sum(conf['virginica',])

ppv

[1] 1

npv <- sum(diag(conf[1:2,1:2])) / (sum(diag(conf[1:2,1:2])) + sum(conf[1:2,'virginica']))

npv

[1] 0.969697

macro.sens <- macro.sens + sens

macro.spec <- macro.spec + spec

macro.sens <- macro.sens / 3

macro.spec <- macro.spec / 3

macro.sens

[1] 0.974359

macro.spec

[1] 0.9880952

### All Metrics ###

macro.sens <- 0

macro.spec <- 0

# setosa

sens.all <- conf.all['setosa','setosa'] / sum(conf.all[,'setosa'])

sens.all

[1] 1

spec.all <- sum(diag(conf.all[2:3,2:3])) / (sum(diag(conf.all[2:3,2:3])) + sum(conf.all['setosa',2:3]))

spec.all

[1] 1

ppv.all <- conf.all['setosa','setosa'] / sum(conf.all['setosa',])

ppv.all

[1] 1

npv.all <- sum(diag(conf.all[2:3,2:3])) / (sum(diag(conf.all[2:3,2:3])) + sum(conf.all[2:3,'setosa']))

npv.all

[1] 1

macro.sens <- macro.sens + sens.all

macro.spec <- macro.spec + spec.all

# versicolor

sens.all <- conf.all['versicolor','versicolor'] / sum(conf.all[,'versicolor'])

sens.all

[1] 0.96

spec.all <- sum(diag(conf.all[c(1,3),c(1,3)])) / (sum(diag(conf.all[c(1,3),c(1,3)])) + sum(conf.all['versicolor',c(1,3)]))

spec.all

[1] 0.96

ppv.all <- conf.all['versicolor','versicolor'] / sum(conf.all['versicolor',])

ppv.all

[1] 0.9230769

npv.all <- sum(diag(conf.all[c(1,3),c(1,3)])) / (sum(diag(conf.all[c(1,3),c(1,3)])) + sum(conf.all[c(1,3),'versicolor']))

npv.all

[1] 0.9795918

macro.sens <- macro.sens + sens.all

macro.spec <- macro.spec + spec.all

# virginica

sens.all <- conf.all['virginica','virginica'] / sum(conf.all[,'virginica'])

sens.all

[1] 0.92

spec.all <- sum(diag(conf.all[1:2,1:2])) / (sum(diag(conf.all[1:2,1:2])) + sum(conf.all['virginica',1:2]))

spec.all

[1] 0.98

ppv.all <- conf.all['virginica','virginica'] / sum(conf.all['virginica',])

ppv.all

[1] 0.9583333

npv.all <- sum(diag(conf.all[1:2,1:2])) / (sum(diag(conf.all[1:2,1:2])) + sum(conf.all[1:2,'virginica']))

npv.all

[1] 0.9607843

macro.sens <- macro.sens + sens.all

macro.spec <- macro.spec + spec.all

macro.sens <- macro.sens / 3

macro.spec <- macro.spec / 3

macro.sens

[1] 0.96

macro.spec

[1] 0.98

**Receiver Operating Characteristic (ROC) Curve**

The ROC Curve plots the sensitivity and specificity of a classification algorithm over varying levels of the discrimination threshold. For a problem with multiple classes, we traditionally plot the ROC for each individual class against the remaining classes.

Then, these are averaged using two techniques: The *macro-average* simply averages the smoothed sensitivity and specificity values for each ROC curve together. The *micro-average* combines the predicted results from each class, and calculates the ROC using the combined results. The micro-average is useful for detecting problems due to class imbalance but is more computationally expensive.

We plot the ROC below using **the test set** results.

probs <- predict(model, x.test, type="raw")

setosa.labels <- rep(0, length(y.test))

versicolor.labels <- rep(0, length(y.test))

virginica.labels <- rep(0, length(y.test))

for (i in 1:length(y.test)) {

if(y.test[i] == 'setosa') {

setosa.labels[i] <- 1

} else if (y.test[i] == 'versicolor') {

versicolor.labels[i] <- 1

} else if (y.test[i] == 'virginica') {

virginica.labels[i] <- 1

}

}

# Create a plot of ROC for each class label against the rest

setosa.roc <- roc(setosa.labels, probs[, 'setosa'], auc.polygon=TRUE, max.auc.polygon=TRUE, print.auc=TRUE, show.thres=TRUE)

setosa.smoothroc <- smooth(setosa.roc, method = "density")

plot(setosa.smoothroc, col = 'red', xaxt='n', xlab="False Positive Rate (1 - Specificity)", ylab = "True Positive Rate (Sensitivity)")

#par(new=T): make the second plot without cleaning the first

par(new=TRUE)

versicolor.roc <- roc(versicolor.labels, probs[, 'versicolor'], auc.polygon=TRUE, max.auc.polygon=TRUE, print.auc=TRUE, show.thres=TRUE)

versicolor.smoothroc <- smooth(versicolor.roc, method = "density")

plot(versicolor.smoothroc, col='blue', xaxt='n', xlab="", ylab = "")

par(new=TRUE)

virginica.roc <- roc(virginica.labels, probs[, 'virginica'], auc.polygon=TRUE, max.auc.polygon=TRUE, print.auc=TRUE, show.thres=TRUE)

virginica.smoothroc <- smooth(virginica.roc, method = "density")

plot(virginica.smoothroc, col='green', xaxt='n', xlab="", ylab = "")

# Averages

y.labels <- c(setosa.labels, versicolor.labels, virginica.labels)

y.probs <- c(probs[, 'setosa'], probs[, 'versicolor'], probs[, 'virginica'])

par(new=TRUE)

micro.roc <- roc(y.labels, y.probs, auc.polygon=TRUE, max.auc.polygon=TRUE, print.auc=TRUE, show.thres=TRUE)

micro.smoothroc <- smooth(micro.roc, method = "density")

plot(micro.smoothroc, col = 'black', lty = 'dotdash', xaxt='n', xlab="", ylab = "")

macro.sensitivity <- (setosa.smoothroc$sensitivities + versicolor.smoothroc$sensitivities + virginica.smoothroc$sensitivities) / 3

macro.specificity <- ((setosa.smoothroc$specificities + versicolor.smoothroc$specificities + virginica.smoothroc$specificities) / 3)

lines(macro.specificity, macro.sensitivity, type='l', xlim = rev(range(macro.specificity)), col='magenta', lty=4)

axis(1, at=(5:0) \* 0.2, labels=(0:5) \* 0.2, pos=c(-0.04,0))

legend(0.4, 0.5, legend = c('setosa','versicolor', 'virginica', 'micro-avg', 'macro-avg'), col = c('red', 'blue', 'green', 'black', 'magenta'), lty = c(1,1,1,4,4))

Chart

Description automatically generated

As we see, ROC hugging upper left corner is a good sign, indicating that the Naïve Bayes performs well on all classes. There is no class imbalance issue since the micro- and macro-averages are the same.

Let us now calculate the area under the receiver operating characteristic curve (AUC) for the test set, as well as the train set and the full dataset. We can do this using the **multiclass.roc** function. We find the AUC is similar for all three subsets of the data.

probs.train <- predict(model, x.train, type="raw")

multiclass.roc(y.train, probs.train)

#

Call:

multiclass.roc.default(response = y.train, predictor = probs.train)

Data: multivariate predictor probs.train with 3 levels of y.train: setosa, versicolor, virginica.

Multi-class area under the curve: 0.9948

#

multiclass.roc(y.test, probs)

#

Call:

multiclass.roc.default(response = y.test, predictor = probs)

Data: multivariate predictor probs with 3 levels of y.test: setosa, versicolor, virginica.

Multi-class area under the curve: 0.994

#

probs.all <- predict(model, iris[1:4], type="raw")

multiclass.roc(iris$Species, probs.all)

#

Call:

multiclass.roc.default(response = iris$Species, predictor = probs.all)

Data: multivariate predictor probs.all with 3 levels of iris$Species: setosa, versicolor, virginica.

Multi-class area under the curve: 0.9951

#

**Results Table**

The table below summarizes each metric across the testing, training, and full datasets. Overall, we see that Naïve Bayes performs exceptionally well on this particular dataset, and performs best for predicting the **setosa** class of irises.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Train | Test | All |
| Accuracy | | 0.95 | 0.98 | 0.96 |
| AUC | | 0.99 | 0.99 | 1.00 |
| Macro Sensitivity | | 0.95 | 0.97 | 0.96 |
| Macro Specificity | | 0.98 | 0.99 | 0.98 |
| Sensitivity | Setosa | 1.00 | 1.00 | 1.00 |
| Versicolor | 0.93 | 1.00 | 0.96 |
| Virginica | 0.92 | 0.92 | 0.92 |
| Specificity | Setosa | 1.00 | 1.00 | 1.00 |
| Versicolor | 0.96 | 0.96 | 0.96 |
| Virginica | 0.97 | 1.00 | 0.98 |
| PPV | Setosa | 1.00 | 1.00 | 1.00 |
| Versicolor | 0.91 | 0.94 | 0.92 |
| Virginica | 0.94 | 1.00 | 0.97 |
| NPV | Setosa | 1.00 | 1.00 | 1.00 |
| Versicolor | 0.97 | 1.00 | 0.98 |
| Virginica | 0.96 | 0.97 | 0.96 |