project09

jason grahn

3/8/2019

Use TWO of the supervised learning predictive modeling methods (e.g., neural network, support vector machine, and naïve Bayes) you choose to do the prediction on the Iris Data Set.

## Data

#load data  
data <- iris   
  
# i dont like Camel case headers, so I'll correct those too  
data <- data.table::setnames(data, tolower(names(iris[1:5])))  
  
#show head of the data  
head(data,5) %>%   
 knitr::kable()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sepal.length | sepal.width | petal.length | petal.width | species |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |

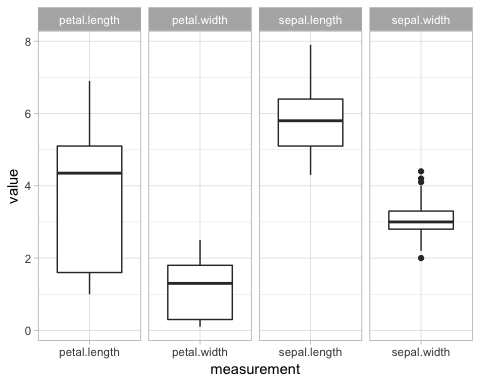
## Exploration

summary(data)

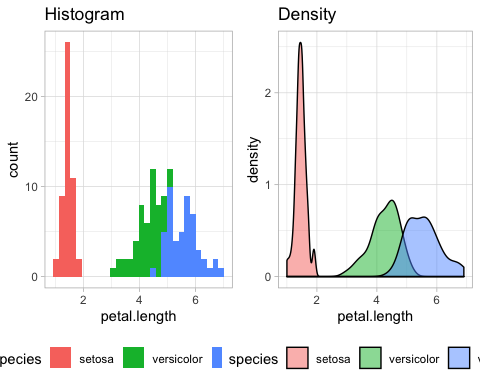
## sepal.length sepal.width petal.length petal.width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

### Visualization

# I like a common theme to my plots  
common\_theme <- theme\_light() +  
 theme(legend.position = "bottom")  
  
data %>% select(species, 1:4) %>% gather(-species, key = measurement, value = value) %>%  
 ggplot(aes(x = measurement, y = value)) +  
 geom\_boxplot() +  
 facet\_grid(~ measurement, scales = "free") +  
 theme\_light()



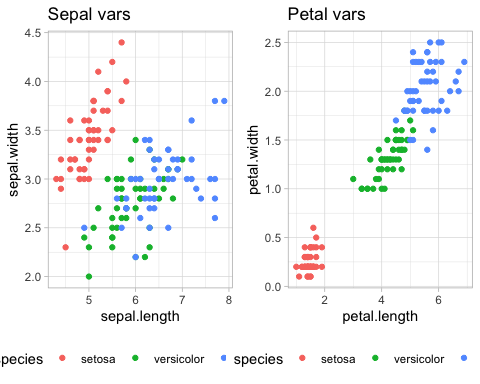
# histograms for each   
histogram <- iris %>%   
 ggplot() +  
 geom\_histogram(aes(x = petal.length, fill=species)) +  
 labs(title = "Histogram") +  
 common\_theme  
  
# density plots for each  
density <- iris %>%   
 ggplot() +   
 geom\_density(aes(x=petal.length, fill=species), alpha = 0.5) +  
 labs(title = "Density") +  
 common\_theme  
  
#put these together into a pretty picture  
cowplot::plot\_grid(histogram, density)



The histogram shows setosa clearly sits on it’s own with very short petal lengths; but virginica and veriscolor have at least some degree of overlap.

Density plotting reinforces the histogram. It does a better job showing the overlap between virginica and versicolor.

sepal.scatter <- iris %>% ggplot() +   
 geom\_point(aes(x = sepal.length, y = sepal.width, color = species)) +  
 labs(title = "Sepal vars") +  
 common\_theme  
  
petal.scatter <- iris %>% ggplot() +   
 geom\_point(aes(x = petal.length, y = petal.width, color = species)) +  
 labs(title = "Petal vars") +  
 common\_theme  
  
cowplot::plot\_grid(sepal.scatter, petal.scatter)



Scatterplotting for sepal variables shows similar behaviors to what we saw in the histogram and density plots. Clustering of the setosa lengths and widths and mixtures of versicolor and viginica points. The petal variables are a bit more defined. Setosa certainly sits on it’s own, and now we can see better definitions between versicolor and virginica species; with virginica having (generally) longer and wider petals than versicolor.

## Building sample vs training data

### Use two thirds of data of each class (i.e., Setosa, Versicolour and Virginica) as training set and do the prediction on the rest.

The first step in making our train v test dataset would be to assign random values to each row, then pull those out. I dont know how to do that in one full swoop, so I’ll subset each species, build some 1’s and 2’s, assign *those* to each subset, put them all back together, then filter out train vs test with the 1’s and 2’s. I’m 100% positive there’s a better way to do this, probably with one simple package; but this is what I *know*, so that’s what I’ll run with. We **should** be able to use the train vs test data for both methods.

set.seed(666)  
  
setosa.sub <- data %>% filter(species == "setosa")  
versicolor.sub <- data %>% filter(species == "versicolor")   
virginica.sub <- data %>% filter(species == "virginica")   
  
set.seed(666)  
  
ind <- sample(2,   
 nrow(setosa.sub),   
 replace=TRUE,  
 prob=c(0.67, 0.33))  
  
#tibble(ind) %>% group\_by(ind) %>% summarize(n = n())  
setosa.sub <- setosa.sub %>% mutate(ind = ind)   
versicolor.sub <- versicolor.sub %>% mutate(ind = ind)   
virginica.sub <- virginica.sub %>% mutate(ind = ind)  
  
data.joined <- setosa.sub %>%   
 full\_join(versicolor.sub) %>%   
 full\_join(virginica.sub)  
  
# make training set and labels  
data.training <- data.joined %>%   
 filter(ind == 1) %>%   
 dplyr::select(sepal.length, sepal.width, petal.length, petal.width)  
  
data.trainLabels <- data.joined[ind==1,5]  
# make testing set and labels  
data.test <- data.joined %>%   
 filter(ind == 2) %>%   
 dplyr::select(sepal.length, sepal.width, petal.length, petal.width)  
  
data.testLabels <- data.joined[ind==2, 5]

# K-Nearest Neighbors

-Nearest Neighbors uses the neighbors of a given datapoint in order to classify a new datapoint. It’s called -nearest because the value determines how many neighbor points are used in the classification. If too little neighbors are used, we may over-fit the data; while using too many neighbors overly “smooths” out the data and leads to more generalization, resulting in less classifications.

To execute KNN, we can use the class package. It’s good to reinforce the education we’ve received so we use this this package because it requires minimal inputs and we’ve been trained on it already. Class contains the knn function which requires, quite simply, your training dataset, your testing dataset, what your selected *k-value* is, and how to label the output.

data.knn <- knn(train = data.training,  
 test = data.test,   
 k = 5,  
 cl = data.trainLabels)

table(data.testLabels, data.knn)

## data.knn  
## data.testLabels setosa versicolor virginica  
## setosa 17 0 0  
## versicolor 0 17 0  
## virginica 0 1 16

The table shows the KNN output against the known labels for the iris data. I see I mispredicted 1 *virginica* as a *versacolor*. Otherwise we have accurate predictions across the board.

accuracy.table <- data.joined %>%   
 filter(ind == 2) %>%   
 mutate(data.knn = data.knn,  
 knn.true = if\_else(species == data.knn, TRUE, FALSE))   
  
knn.accuracy.rate <- accuracy.table %>%   
 group\_by(knn.true) %>%   
 summarize(count = n(),  
 rate = count / nrow(accuracy.table) \* 100,  
 rate = round(rate,2))  
  
knn.accuracy.rate %>%   
 knitr::kable()

|  |  |  |
| --- | --- | --- |
| knn.true | count | rate |
| FALSE | 1 | 1.96 |
| TRUE | 50 | 98.04 |

Nice! We have 98.04 percent accuracy with our nn!

# Support Vector Machine (SVM)

With Support Vector Machines, we first have to define the prediction model. We use the joined data table build previously, filtering to keep the training dataset. Then we build the svm model against each of the dimensions of the flowers.

To build the SVM training model, I use the e1071 package containing the svm function. I’m telling the model that we have a classification problem by setting the method to “classification” and setting the kernal to “linear” in order to draw linear boundaries around the classifications, and because it’s the simplest model.

set.seed(666)  
model <- data.joined %>%   
 filter(ind == 1) %>%   
 svm(species ~ sepal.length + sepal.width + petal.length + petal.width,   
 data = .,  
 method="C-classification",  
 kernel="linear")  
  
summary(model)

##   
## Call:  
## svm(formula = species ~ sepal.length + sepal.width + petal.length +   
## petal.width, data = ., method = "C-classification", kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.25   
##   
## Number of Support Vectors: 25  
##   
## ( 2 11 12 )  
##   
##   
## Number of Classes: 3   
##   
## Levels:   
## setosa versicolor virginica

The model summary shows that it has found 25 support vector points to use for delinating iris classifications of setosa, versicolor, and virginica. Now we apply this model to the test dataset in order to predict classifications.

pred <- predict(model, data.test)  
table(pred, data.testLabels)

## data.testLabels  
## pred setosa versicolor virginica  
## setosa 17 0 0  
## versicolor 0 17 0  
## virginica 0 0 17

When we compare the predictions against the testLabels, we are wrong on **zero** predictions!

# Write the SVM accuracy back to the accuracy table so we can compare results  
accuracy.table <- accuracy.table %>%   
 mutate(data.svm = pred,  
 svm.true = if\_else(species == data.svm, TRUE, FALSE))   
  
svm.accuracy.rate <- accuracy.table %>%   
 group\_by(svm.true) %>%   
 summarize(count = n(),  
 rate = count / nrow(accuracy.table) \* 100,  
 rate = round(rate,2))  
  
svm.accuracy.rate %>%   
 knitr::kable()

|  |  |  |
| --- | --- | --- |
| svm.true | count | rate |
| TRUE | 51 | 100 |

That *is* a surprise! We have 100 percent accuracy with our SMV!

# Comparing Performance (accuracy)

svm.ac <- svm.accuracy.rate %>% filter(svm.true == TRUE) %>% select(rate)  
knn.ac <- knn.accuracy.rate %>% filter(knn.true == TRUE) %>% select(rate)

The difference in accuracy between the two methods SVM and KNN are fairly close. SVM came in at 100 percent accuracy, while KNN came in at 98.04 percent. This is a difference of 1.96.