Prediction and Classification in R Using the `caret` Package

Nate Magee

November 29, 2017

Introduction: Purpose and Methods

The purpose of this post is to introduce machine learning techniques in R using the `caret` packages, along with a few visualizations using `ggplot2` and the base `plot` functions. To do this (and to enourage reproducibility), we'll use a preloaded dataset that comes with R, the `iris` dataset. This contains numerical data on three species of iris flower, and is a perfect introductory dataset to use for machine learning because of its relatively easy categorizations. Using `caret`, we'll try to predict the species of a flower using just a few physical measurements of its flower features. The general flow of the post is based off of this extremely useful demo. First we'll load and look at the data, then we'll split it into training and test sets, and finally we'll use plotting techniques and the `caret` package to build and evaluate prediction models.

First, install the necessary packages and dependencies, then load them into our session

```
install.packages(c("caret", "ggplot2"), dependencies=c("Depends", "Suggests"),
                 repos = 'http://cran.us.r-project.org')
```

```
##
## The downloaded binary packages are in
##  /var/folders/zl/mszdc_492wbc3fmgq62tvdtr0000gn/T//Rtmpvv6ZJK/downloaded_packages
```

```
# load our packages
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

First looks at the data

Now, we'll pull our data and print the first few rows to get a feel for how it looks

```
# load the iris dataset and store it
data(iris)

dat <- iris

head(dat)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Creating a validation set

A necessary step in machine prediction projects is to partition your data into a test set and a training set. The training set is what your model will use to "learn", whereas the test set is how it validates its prediction accuracy. We'll segment the `iris` data into quarters, taking 75% of it to train and the remainder to test.

> Note that in our dataset, rows are clustered by species. This means that simply taking the first 114 rows as our training set and the last 36 as our test set would skew the measurements of our model. Therefore, we use the `createDataPartition` function to take a random sample of the data for each set.

```
# create a validation set and a training set
training <- createDataPartition(dat$Species, p=0.75, list=FALSE)

# select 25% of the data for validation
validation_set <- dat[-training,]

# use the remaining 75% to train and test
dat <- dat[training,]
```

Now that we've split our dataset, we can take summary statistics of the training set. Notice that species type is at a perfect 38/38/38 split; this was ensured by using `createDataPartition` with `dat$Species` as an argument. This way, we know that we have an even balance of data on all three flower species.

```
# summarize the data
summary(dat)
```

```
##   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.:1.525   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.862   Mean   :3.068   Mean   :3.739   Mean   :1.196  
##  3rd Qu.:6.400   3rd Qu.:3.400   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :38  
##  versicolor:38  
##  virginica :38  
##                 
##                 
## 
```

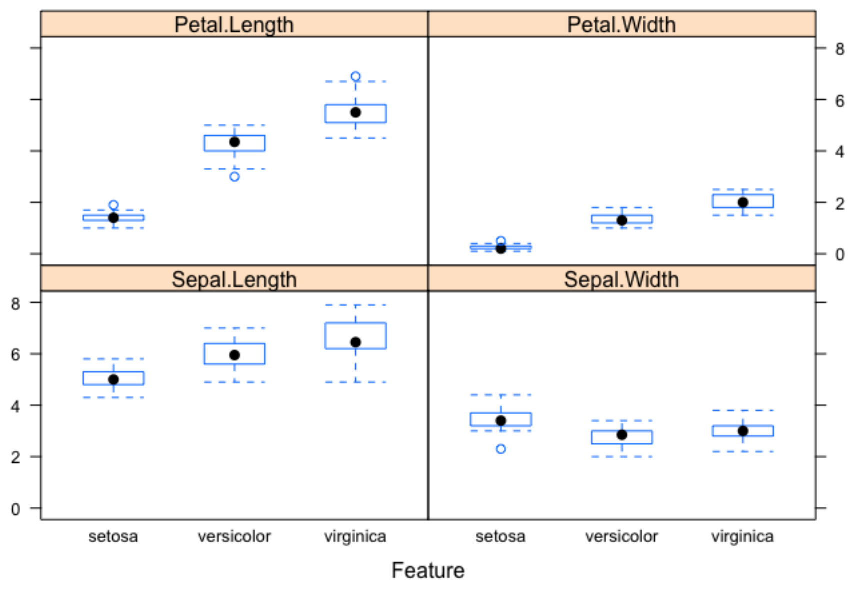
Visualizing elements of the dataset

To get a cursory idea of what kinds of relationships we can find in our data, let's first create objects for flower features and species type, then make some plots. We'll use boxplots and scatterplots to get this first glance.

```
# create objects for flower features and species type
features <- dat[, 1:4]
species <- dat[, 5]
```

First, let's plot each of the four features of our data with boxplots, separating them out for species.

```
# create a box plot giving flower features for each species
featurePlot(features, species, plot="box")
```



From this, we can already get a sense of which features will be useful in predicting species. The distribution of petal length appears to have nice separation between each species, and petal width and sepal length aren't far behind.

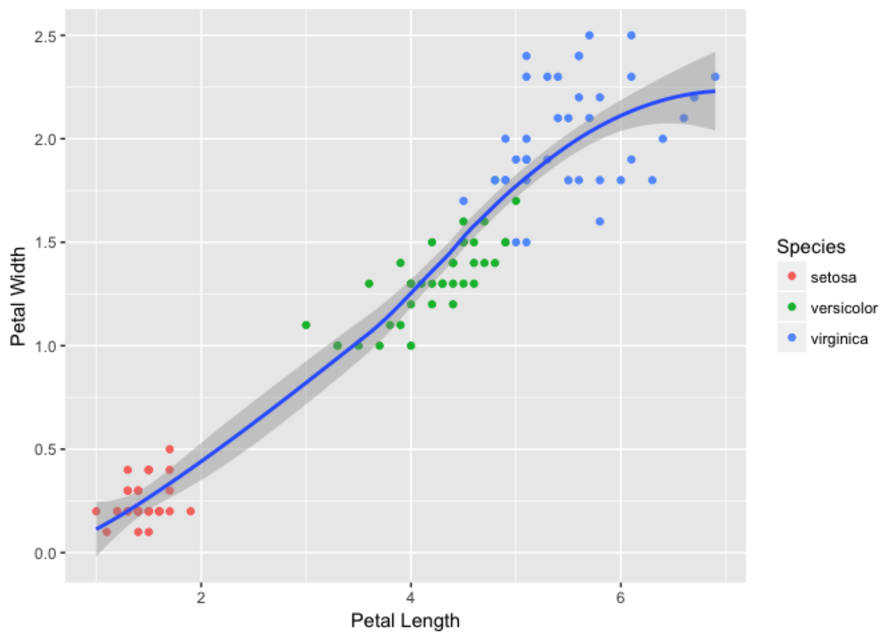
To get numerical evidence to back up our initial observations, we'll compute a correlation matrix on the features set to get an idea of which features have a strong correlation, then we can see if there are clear gradations between species.

```
corr_matrix <- cor(features)
corr_matrix
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.0816226    0.8625508   0.8023688
## Sepal.Width    -0.0816226   1.0000000   -0.4086722  -0.3571309
## Petal.Length    0.8625508  -0.4086722    1.0000000   0.9643537
## Petal.Width     0.8023688  -0.3571309    0.9643537   1.0000000
```

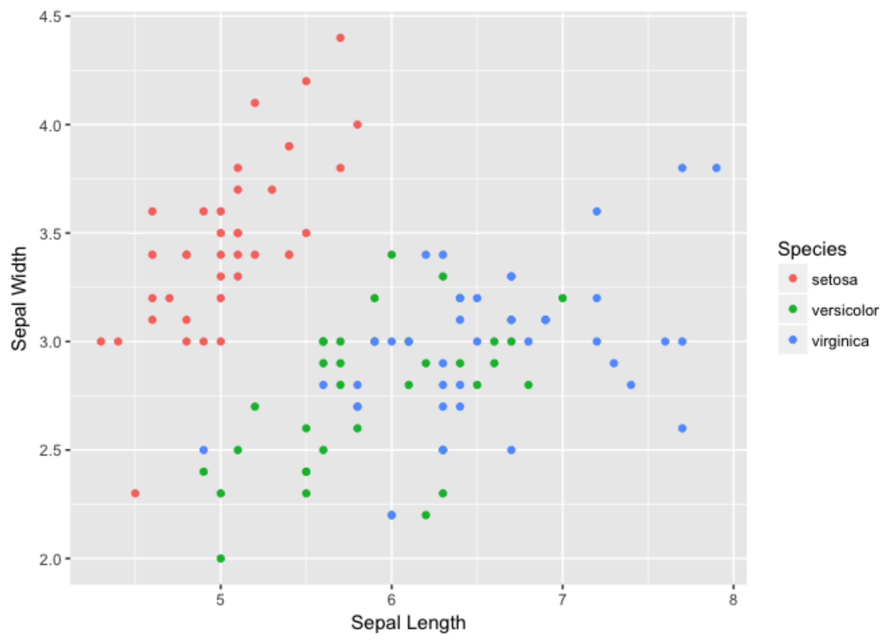
The evidence of collinearity between petal width and petal length should produce a nice linear scatter plot. We'll color the points by species to try and determine a clear pattern.

```
base <- ggplot(dat)

petal_length_width <- base + geom_point(aes(dat$Petal.Length, dat$Petal.Width, color=dat$Species)) +
  geom_smooth(method="loess", aes(dat$Petal.Length, dat$Petal.Width)) +
  labs(x="Petal Length", y = "Petal Width", color = "Species")

petal_length_width
```



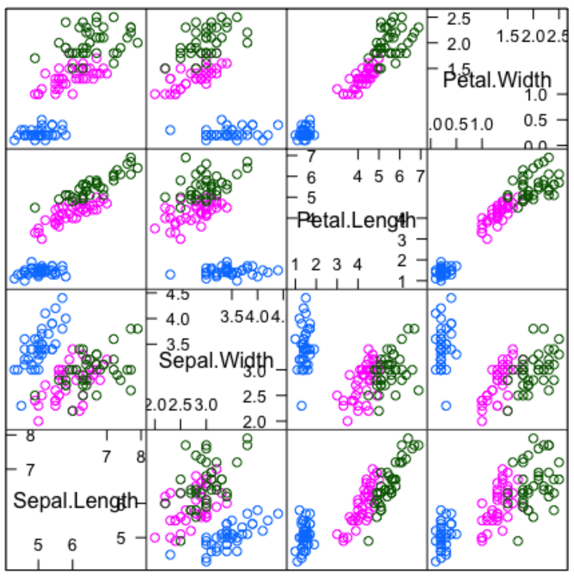
It's hard to argue against the obvious clustering by species in the above plot. Conversely, we can look at two relatively uncorrelated variable with not much distinction between species, like sepal width and sepal length, and find that they might not be effective predictors of species.

```
sepal_length_width <- base + geom_point(aes(dat$Sepal.Length, dat$Sepal.Width, color=dat$Species)) +
  labs(x="Sepal Length", y = "Sepal Width", color = "Species")

sepal_length_width
```



Another useful type of visualization is a scatterplot matrix, which can be achieved using the `featurePlot` function we called to generate the boxplot. This gives us a grid of scatterplots, plotting every variable in `features` against every other variable, and coloring the dots by species. From this matrix of plots, it's easy to pick out where predictive relationships might exist.

```
featurePlot(features, species, plot="pairs")
```



Scatter Plot Matrix

## The classification phase

Now we can take advantage of `caret`'s vast array of resources for model building. Embedded in the package are several hundred different machine learning models that can tackle a wide variety of problems. For our sake, we'll use a few of the most popular classification models and take a look at which one outperforms the rest.

To make classification predictions on our dataset, we'll specify our control values and testing metric. The `trainControl` method takes bootstrap samples of the testing set, and then performs an n-fold validation on the model. Specifying `metric` will allow us to pull the accuracy of each model's predictions, verified by the validation process, after the classifications have been made.

```
control <- trainControl(method = "boot", number = 10)
metric <- "Accuracy"
```

Training the models

We'll use five popular classification models (LDA, KNN, SVM, Random Forest, and Partial Least Squares) on our `iris` data. Each of these calls to `train` trains and validates the model using a different method, specified by the `method` parameter. After the classifications are all run and validated, we will be able to evaluate their accuracy against one another.

```
# linear discriminant analysis
set.seed(7)
lda <- train(Species~., data = dat, method="lda", metric = metric, trControl = control)

# k nearest neighbors
set.seed(7)
knn <- train(Species~., data = dat, method="knn", metric = metric, trControl = control)

# support vector machines
set.seed(7)
svm <- train(Species~., data = dat, method="svmLinear", metric = metric, trControl = control)

# random forest
set.seed(7)
rf <- train(Species~., data = dat, method="rf", metric = metric, trControl = control)

# partial least squares
set.seed(7)
pls <- train(Species~., data = dat, method="pls", metric = metric, trControl = control)
```

Looking at accuracy

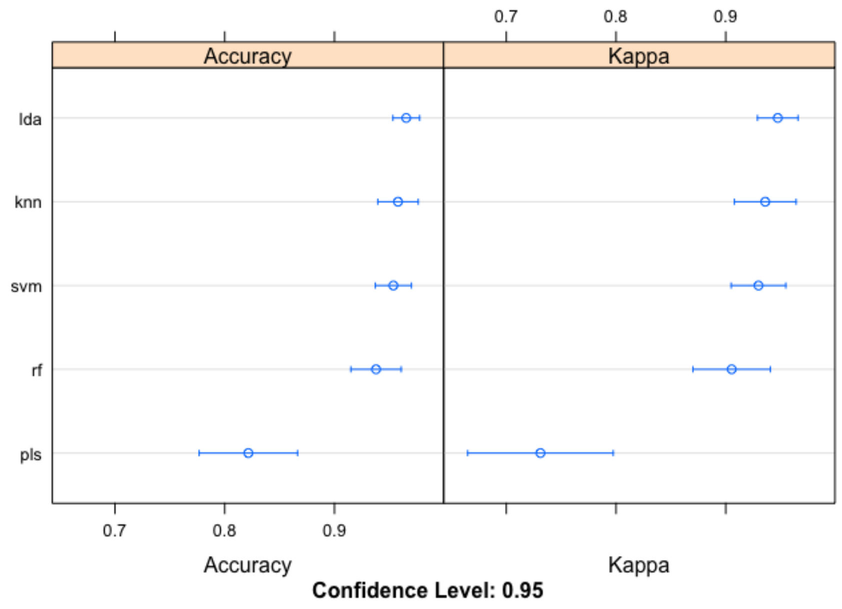
Now we can take a look at how accurate each model was in its predictions. Calling summary on the results will give us a nice view of which model's accuracy was consistently the highest.

```
# summarize accuracy of models
model_results <- resamples(list(lda=lda, knn=knn, svm=svm, rf=rf, pls=pls))
summary(model_results)
```

```
##
## Call:
## summary.resamples(object = model_results)
##
## Models: lda, knn, svm, rf, pls
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.9473684 0.9523810 0.9621528 0.9654279 0.9761905 1.0000000    0
## knn 0.9047619 0.9420139 0.9740216 0.9579167 0.9760453 0.9767442    0
## svm 0.9230769 0.9297619 0.9605655 0.9536913 0.9751284 0.9767442    0
## rf  0.9047619 0.9099507 0.9309524 0.9378960 0.9532115 1.0000000    0
## pls 0.6666667 0.8155933 0.8333333 0.8215235 0.8533835 0.8888889    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda 0.9201681 0.9265985 0.9423121 0.9473925 0.9637070 1.0000000    0
## knn 0.8564103 0.9106511 0.9607803 0.9359959 0.9636461 0.9651257    0
## svm 0.8839286 0.8936976 0.9397118 0.9298237 0.9626074 0.9651257    0
## rf  0.8505338 0.8624960 0.8960797 0.9054026 0.9295114 1.0000000    0
## pls 0.5071249 0.7228761 0.7490294 0.7311226 0.7790865 0.8320896    0
```

We can see now that while the Partial Least Squares model was inconsistent and inaccurate, both SVM and LDA had high levels of accuracy in predicting flower species. Linear Discriminant Analysis, though, had a minimum accuracy of 92% compared to SVM's 80% minimum. Due to its high prediction accuracy and small variance in accuracy across all resamples, it's a safe conclusion that the Support Vector Machine model was our best predictor of flower species. We can visualize model accuracy using a dotplot.

```
dotplot(model_results)
```



Conclusion

As we have seen, the `caret` package makes machine learning problems incredibly accessible with just a few lines of code. However, simply plugging in a model and using the `train` function isn't an informed or reliable method of undertaking a project. As with all statistical analysis, the first steps should be to acquaint yourself with your data, make some visualizations, and try to pick out a signal from the noise. Then, you can hone in on your goal and use the models at your disposal to answer a question with great accuracy.

References

https://www.rdocumentation.org/packages/caret/versions/6.0-77/topics/trainControl

https://pythonprogramming.net/pandas-statistics-correlation-tables-how-to/

https://rdrr.io/cran/caret/man/models.html

https://machinelearningmastery.com/machine-learning-in-r-step-by-step/

https://rstudio-pubs-static.s3.amazonaws.com/4991_14f069b28daf4d3187f3b5ae40051473.html

https://cran.r-project.org/web/packages/caret/vignettes/caret.pdf

https://www.analyticsvidhya.com/blog/2016/12/practical-guide-to-implement-machine-learning-with-caret-package-in-r-with-practice-problem/