R ile Sınıflandırma Analizleri - I

Doç. Dr. Gökmen Zararsız

Mayıs 14, 2022

Contents

Kurs Öncesi Gereksinimler:

- R 4.2.0 sürümünün kullanılması önerilir.
- RStudio GUI Desktop
- MAC Kullanıcılar için XQuartz güncel sürümü.

1. Giriş

Kestirime yönelik modellerin oluşturulmasında sıklıkla kullanılan programlama dillerinden biri `R`'dir. `R` içerisinde bu amaçla geliştirilmiş çok sayıda kütüphane bulunmaktadır. `caret` bu kütüphaneler arasında sıklıkla kullanılan kapsamlı bir kütüphanedir.

Ayrıca, gerek sınıflandırma, gerek regresyon yöntemleri ile ilgili onlarca modeli içermesi, günlük hayattaki problemlerde bu kütüphanenin tercih edilmesine neden olmaktadır. Araştırmacılar, araştırma problemlerine ilişkin elde ettikleri verileri `R` ortamına taşıyarak, verilerini işleyebilmekte, onlarca farklı model oluşturabilmekte, her bir model için optimum parametre kestirimleri yapabilmekte, model performanslarını elde edebilmekte ve birbirleri ile karşılaştırabilmektedir. Özetle, araştırmacılar bu kapsamlı kütüphaneden faydalanarak, problemine ilişkin en iyi kestirim modelini elde edebilmekte ve kestirimler yapabilmektedir.

Elde edilen en iyi performansı veren model ile araştırmacılar yeni verilerini `R` içerisinde kestirebileceği gibi, kodlarını `R` içerisinde yer alan diğer kütüphaneler (örn. `shiny`) ile entegre ederek web servisleri de oluşturabilir. Bu sayede diğer kullanıcılar kodlama ihtiyacına gerek kalmaksızın, geliştirilen web servisi üzerinde verilerine ilişkin kestirimler yapabilmektedir. Konu ile ilgili basit iki uygulama aşağıda verilmiştir.

- **DDNAA** (http://www.biosoft.hacettepe.edu.tr/DDNAA/),
- **MLViS** (http://www.biosoft.hacettepe.edu.tr/MLViS/).

2. `caret` ile Model Oluşturma

`caret` kütüphanesinde model oluşturma süreci tek bir fonksiyon altında gerçekleştirilebilmektedir. Kullanıcılar verilerini `R` ortamına aktardıktan ve model oluşturma süreci ile ilgili ön işlemleri uyguladıktan sonra `train(...)` fonksiyonu ile belirledikleri bir yönteme ilişkin kestirim modeli oluşturabilirler. Bu ön işlemlere örnek olarak, eğitim ve test setlerinin oluşturulması, filtreleme, değişken seçimi, eksik değer ataması, standartlaştırma, veri dönüşümü verilebilir. Bu işlemlerin birçoğu `caret` içerisindeki diğer fonksiyonlar (örn. `createDataPartition`, `preProcess`, `rfe`, `gafs`, vs.) aracılığıyla yapılabilir.

Model oluşturma sürecinde eğitim verisi kullanılmaktadır. Bu veri `train(...)` fonksiyonu içerisinde tanımlanarak kestirime yönelik bir model oluşturulabilir. Model oluşturma süreci sonrasında test verileri kullanılarak model performansı değerlendirilebilir ya da farklı modellerin performansları karşılaştırılabilir. `train(...)` fonksiyonlarının önemli bazı argümanları aşağıda verilmiştir.

`x`: Satırları gözlemlerin, sütunları ise değişkenlerin oluşturduğu data.frame ya da matris sınıfındaki, sütun başlıklarına sahip, model oluşturmada kullanılacak veri.

`y`: Nümerik ya da faktör sınıfında vektör. Sınıf değişkenine ait değerleri içermeli ve vektörün uzunluğu, `x` verisinin satır sayısı ile aynı olmalıdır. `caret` kütüphanesi, bu vektör nümerik sınıfta ise regresyon modeli, faktör sınıfında ise sınıflandırma modeli oluşturacaktır.

`method`: Kullanılacak sınıflandırma ya da regresyon modeli. Karakter formatında oluşturulmalıdır. Örneğin `method = "rf"` olarak tanımlandığında, random forest modeli ile model oluşturulacaktır. Bu modellerin tümüne aşağıdaki adresten ulaşılabilir. http://topepo.github.io/caret/train-models-by-tag.html

`preProcess`: Model oluşturma süreci öncesi veri ön işlemesi yapılacaksa, `preProcess` içerisinde tanımlanabilir. Yapılacak ön işlemler karakter formatında bir vektör ile tanımlanabilir. Mevcut ön işlem yöntemleri `BoxCox`, `YeoJohnson`, `expoTrans`, `center`, `scale`, `range`, `knnImpute`, `bagImpute`, `medianImpute`, `pca`, `ica` ve

`spatialSign` yöntemleridir. Bu yöntemlerden hangileri uygulanacaksa belirtilen karakter vektörü içerisinde tanımlanmalıdır. Örneğin `preProcess = c("center", "scale", "knnImpute")`

`trControl`: `train(...)` fonksiyonuna ilişkin kontrol parametreleri liste biçiminde tanımlanabilir. Örneğin, parametre optimizasyonu için kullanılabilecek örnekleme yöntemi seçilebilir. Detaylar `trainControl(...)` fonksiyonu yardım sayfasında bulunabilir.

`metric`: Model parametrelerinin optimizasyonunda hangi ölçünün kullanılacağı seçilebilir. Bu ölçülerden hangisinin kullanılacağı karakter formatında tanımlanmalı ve regresyon modelleri için `RMSE` ve `Rsquared` ölçülerinden biri, sınıflandırma modelleri için `Accuracy` ve `Kappa` ölçülerinden biri seçilmelidir.

`tuneGrid`: Optimizasyon aşamasında denenecek model parametreleri data.frame sınıfında kullanıcı tanımlı olarak verilebilir. Data.frame sınıfındaki bu objenin sütun isimleri, optimize edilecek parametre ile aynı olmalıdır.

`tuneLength`: Optimize edilecek parametre için kullanılacak değer miktarı. Sayı olarak verilmelidir.

`form`: `y ~ x1 + x2 + ...` gibi tanımlanabilecek model formülü.

`weights`: Tanımlanacak olan, gözlem sayısı uzunluğunda nümerik vektör ile örnekler ağırlıklandırılabilir.

Bu fonksiyon ve model oluşturma süreci ile ilgili detaylı bilgilere aşağıdaki adresten ulaşılabilir:

https://topepo.github.io/caret/index.html

3. caret ile Test Verilerinin Kestirimi

Model oluşturma sürecinden sonra test verileri kullanılarak kestirim yapılabilir. Bu aşamada eğitim seti ile aynı formatta kullanılacak olan test setine ve oluşturulan model objesine ihtiyaç vardır. `caret` kütüphanesinde test verileri `predict(...)` fonksiyonu ile kestirilebilir. Bu fonksiyona ait önemli bazı argümanlar aşağıda gösterilmiştir:

`object`: Oluşturulan model objesi.

`newdata`: Kestirimi yapılacak olan test verisi. Satırları gözlemlerin, sütunları ise değişkenlerin oluşturduğu data.frame ya da matris sınıfında olmalıdır. Model oluşturma sürecinde kullanılan veri ile aynı sayıda sütuna ve aynı sütun başlıklarına sahip olmalıdır.

`type`: Sınıflandırma modelleri için kullanılabilir, `raw` ya da `prob` seçeneklerinden biri seçilmelidir.

4. caret ile Örnek Uygulama

Uygulamada istatistiksel modellerin uygulamalarında sıklıkla kullanılan `iris` verisinden faydalanılacaktır. `iris` verisi, üç farklı zambak türüne (`setosa`, `versicolor`, `virginica`) ait 150 gözlemden oluşmaktadır. Bu 150 gözlemin çanak yaprağı uzunluğu (`Sepal.Length`), çanak yaprağı genişliği (`Sepal.Width`), taç yaprağı uzunluğu (`Petal.Length`) ve taç yaprağı genişliği (`Petal.Width`) ölçülmüştür. `R` programlama dili içerisinde 150x4 boyutunda bir data.frame sınıfında saklanmaktadır.

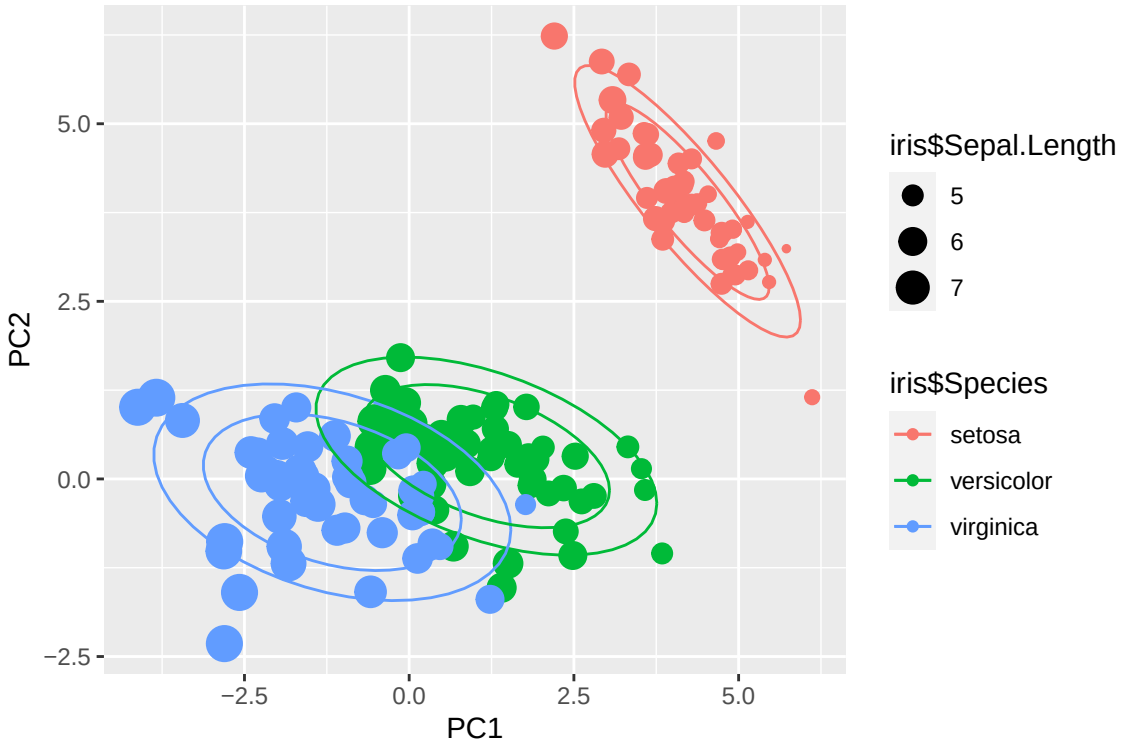
Acaba bu dört değişken kullanılarak, zambak türleri birbirinden ayırt edilebilir mi? Bir kestirim modeli oluşturularak 151. bir zambak örneğinin çanak ve taç yaprağı uzunluğuna ve genişliğine bakılarak, setosa türüne mi, versicolor türüne mi, ya da virginica türüne mi ait olduğu büyük doğrulukla kestirilebilir mi? `caret` kütüphanesini kullanarak bir random forest modeli oluşturalım ve probleme yanıt arayalım.

Öncelikle uygulamada gerekli olan iki kütüphaneyi, `caret` ve `ggplot2` kütüphanelerini çağıralım:

```
library(caret)
library(ggplot2)
```

Zambak örneklerinin bu dört değişkene göre nasıl saçıldığını görüntülemek için temel bileşenler analizi uygulayalım, ve ilk iki bileşen için saçılım grafiğini çizelim:

```
pca <- prcomp(iris[iris$Species %in% c("virginica","versicolor"), 1:4],
              retx = TRUE,  scale = TRUE, tol=0.4)
predicted <- predict(pca,iris[,1:4])
ggplot(data.frame(predicted)) +
  aes(x = PC1, y = PC2, color = iris$Species) +
  geom_point(aes(size = iris$Sepal.Length)) +
  stat_ellipse() +
  stat_ellipse(level = 0.8)
```



Elde edilen grafikten, setosa türünün diğer iki türe göre keskin biçimde ayrıldığı görülebilir. Versicolor ve virginica türlerinin de farklı dağılımlara sahip olduğu, fakat belli noktalarda iç içe geçtiği görülebilir. Random forest yöntemiyle bir öğrenme kuralı geliştirilebilir ve yeni örnekler bu kurala göre ilgili türe atanabilir. Öncelikle eğitim ve test setlerini oluşturalım. Örneklerin %70'ini eğitim setine ayıralım ve bu veri ile model oluşturalım, kalan %30'unu test setine ayıralım ve bu veri ile kestirim yaparak model performansını değerlendirelim.

```
set.seed(1881)

n <- nrow(iris)
p <- ncol(iris)
nTest <- ceiling(n*0.3)
ind <- sample(n, nTest, FALSE)

tr <- iris[-ind,]
ts <- iris[ind,]
```

```
tr.input <- tr[,-5]
ts.input <- ts[,-5]
tr.output <- factor(tr[,5])
ts.output <- factor(ts[,5])
```

Eğitim ve test verilerini standartlaştıralım. Bu amaçla `caret` kütüphanesinin `preProcess(...)` fonksiyonundan faydalanalım. Bu sayede test verilerini eğitim setinin dağılımına göre standartlaştıralım.

```
prep <- preProcess(tr.input, method = c("center", "scale"))
tr.input.transformed <- predict(prep, tr.input)
ts.input.transformed <- predict(prep, ts.input)
```

İzleyen süreci standartlaştırılmış eğitim ve test verileri üzerinden gerçekleştirelim. Parametre optimizasyonu için ve aşırı uyum problemini aşmak amacıyla 5-parça çapraz geçerlilik yönteminden faydalanalım ve bu süreci 3 kez tekrarlayalım. Bu işlem `trainControl(...)` fonksiyonu yardımıyla yapılabilir.

```
ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
```

Random forest yöntemi ile bir kestirim modeli oluşturalım. Ölçü olarak doğru sınıflandırma oranından faydalanalım.

```
fitRF <- train(y = tr.output, x = tr.input.transformed, method = "rf", trControl = ctrl,
fitRF
```

```
## Random Forest
##
## 105 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 84, 84, 84, 84, 84, 84, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9396825  0.9089194
##   3     0.9396825  0.9089194
##   4     0.9396825  0.9089194
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Elde edilen random forest modeli ile test seti için ayrılan 45 örneğin zambak türünü kestirelim.

```
predRF <- predict(fitRF, ts.input.transformed)
predRF
```

```
##  [1] versicolor virginica  versicolor versicolor setosa     versicolor
##  [7] setosa     setosa     versicolor setosa     versicolor virginica
```

```
## [13] virginica  virginica  versicolor virginica  setosa     versicolor
## [19] setosa     virginica  versicolor virginica  setosa     versicolor
## [25] versicolor setosa     versicolor virginica  versicolor virginica
## [31] virginica  setosa     versicolor setosa     virginica  setosa
## [37] virginica  versicolor virginica  versicolor virginica  virginica
## [43] virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

Kestirim sonuçlarını, test verilerinin gerçek sınıfları ile karşılaştıralım ve kestirim performansına ilişkin istatistikler hesaplayalım.

```
tblRF <- table(predRF, ts.output)
confusionMatrix(tblRF)
```

```
## Confusion Matrix and Statistics
##
##             ts.output
## predRF       setosa versicolor virginica
##   setosa         11          0         0
##   versicolor      0         15         1
##   virginica       0          0        18
##
## Overall Statistics
##
##                Accuracy : 0.9778
##                  95% CI : (0.8823, 0.9994)
##     No Information Rate : 0.4222
##     P-Value [Acc > NIR] : 8.826e-16
##
##                   Kappa : 0.966
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9474
## Specificity                 1.0000            0.9667           1.0000
## Pos Pred Value              1.0000            0.9375           1.0000
## Neg Pred Value              1.0000            1.0000           0.9630
## Prevalence                  0.2444            0.3333           0.4222
## Detection Rate              0.2444            0.3333           0.4000
## Detection Prevalence        0.2444            0.3556           0.4000
## Balanced Accuracy           1.0000            0.9833           0.9737
```

Test verileri %97.78 başarı ile doğru olarak sınıflanmıştır.