R ile Sınıflandırma Analizleri - II

Doç. Dr. Gökmen Zararsız

Mayıs 14, 2022

Contents

1.Uygulama 2

1.1. Veri Seti

Veri seti UCI Machine Learning veritabanından alınmıştır (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)). Meme kanseri verisidir (Wisconsin Diagnostic Breast Cancer (WDBC)). 569 gözlem ve 30 değişkenden oluşmaktadır. Amaç memede tespit edilen kitleyi iyi huylu (benign - B) ve kötü huylu (malign - M) olarak sınıflamaktır.

```
data = read.table(file = "../data/wbdc2.txt", header = TRUE)
head(data[1:5])
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         M       17.99        10.38         122.80    1001.0
## 2         M       20.57        17.77         132.90    1326.0
## 3         M       19.69        21.25         130.00    1203.0
```

```
## 4         M       11.42        20.38          77.58     386.1
## 5         M       20.29        14.34         135.10    1297.0
## 6         M       12.45        15.70          82.57     477.1
```

```
dim(data)
```

```
## [1] 569  31
```

Değişken tiplerini görmek için `str` fonksiyonu kullanılabilir.

```
str(data[,1:5])
```

```
## 'data.frame':    569 obs. of  5 variables:
##  $ diagnosis     : chr  "M" "M" "M" "M" ...
##  $ radius_mean   : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean  : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean: num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean     : num  1001 1326 1203 386 1297 ...
```

1.2. Veri Görselleştirme

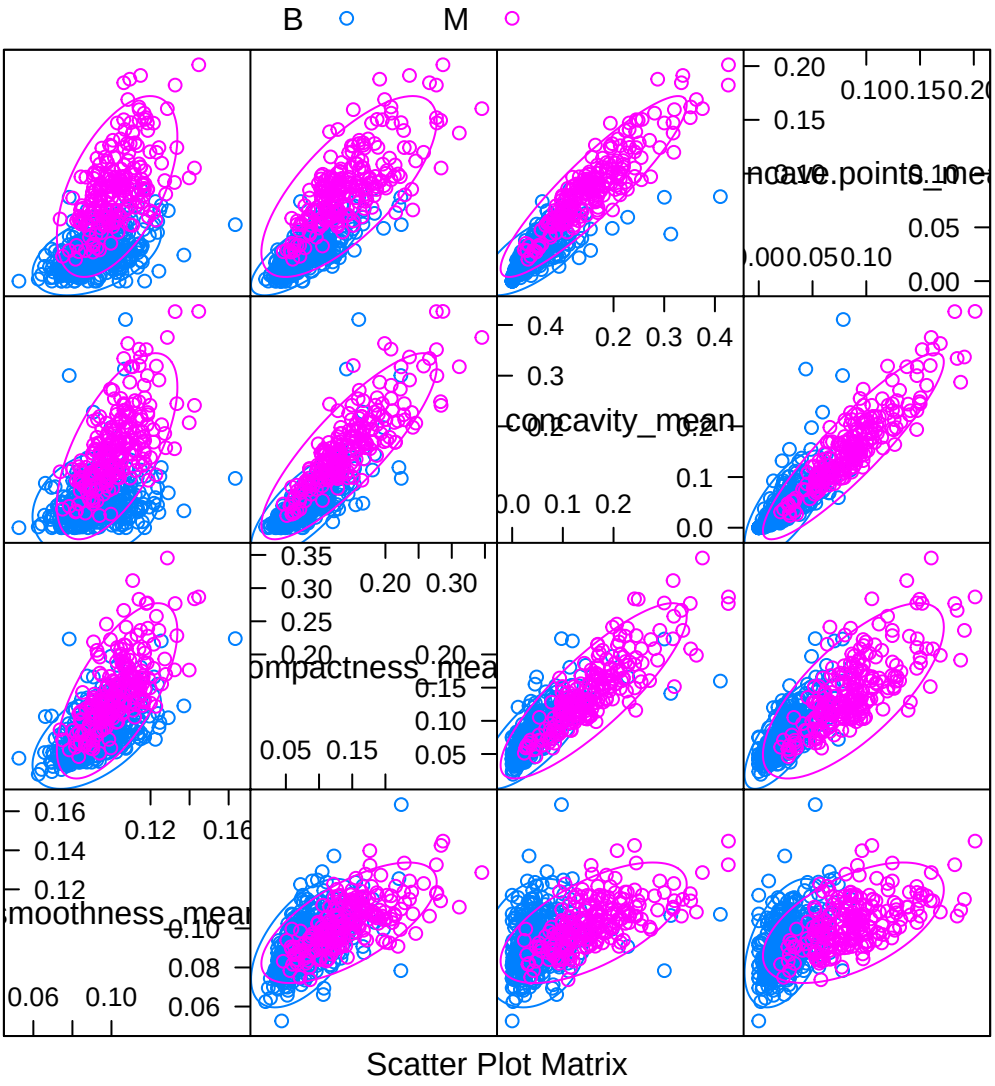
Herhangi bir modelleme adımı uygulanmadan önce verideki değişkenlerin sınıf (yanıt, çıktı, etiket) değişkeni ile ilişkilerini görsel olarak incelemek değişkenlerin modele katkısı hakkında öncül bilgiler verecektir.

`caret` paketinde bulunan `featurePlot` fonksiyonu kullanılarak çeşitli `lattice` grafikleri oluşturulabilir.

Scatterplot Matrix

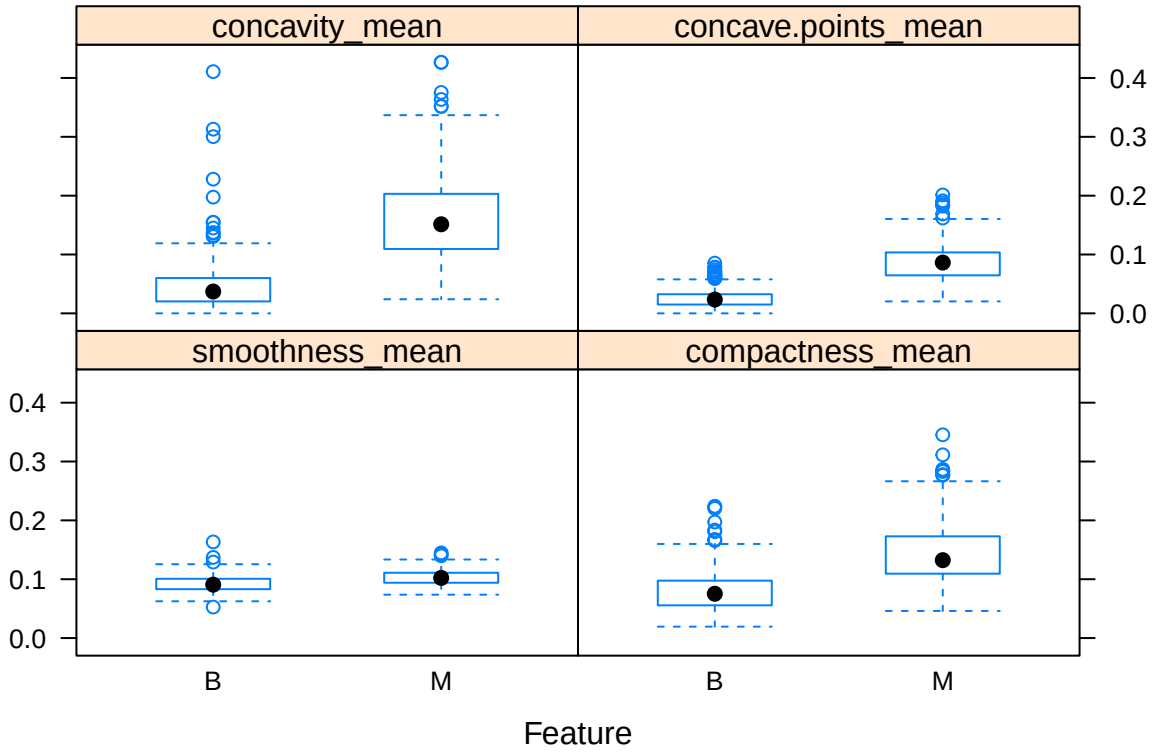
```
library(caret)

featurePlot(x = data[,6:9],
            y = factor(data$diagnosis),
            plot = "ellipse",
            auto.key = list(columns = 3))
```



Boxplot

```
library(caret)
featurePlot(x = data[,6:9],
            y = factor(data$diagnosis),
            plot = "box",
            auto.key = list(columns = 3))
```



1.3. Veri Ön-işleme

1.3.1. Sıfıra Yakın Varyanslı Değişkenlerin Tespiti Bazı durumlarda, veri üretme mekanizmasından kaynaklı olarak bazı nicel değişkenler sıfıra yakın varyansa sahip olabilirler. Bu durum, bir çok algoritmada (karar ağacı tabanlı algoritmalar hariç) sorunlara yol açabilir (hatalı kestirimler, tekil matris problemi, vb.)

Benzer şekilde, bazı nicel değişkenler sadece sıklığı az birkaç değerden oluşabilir. Böyle bir durumda çapraz geçerlilik veya bootstrap örnekleme yapıldığında bu değişkenlerde sıfıra yakın varyanslılık problemi ortaya çıkacaktır. Bu nedenle model kurma işlemine geçmeden önce, sıfıra yakın varysansa (near-zero-variance) sahip değişkenlerin tespit edilip veriden çıkarılması gerekir.

Sıfıra yakın varyanslı değişkenleri tespit etmek için aşağıdaki ölçüler hesaplanabilir:

Sıklık oranı (frequency ratio): en sık görülen değerin en sık görülen ikinci değere oranı. Bu değerin 1'e yakın olması iyi bir durum iken, bu değerin çok büyük olması sıfıra yakın varyanslılık probleminin bir işaretidir.

Tekil değerlerin yüzdesi (percent of unique values): değişkendeki tekil değerlerin sayısının toplam gözlem sayısına bölümü (x100) ile elde edilir. 100'e yakın olması istenir.

Eğer bir değişken için sıklık oranı belirli bir eşik değerinden büyükse ve tekil değerlerin yüzdesi belli bir eşik değerin altında ise, o değişken sıfıra yakın varyanslı değişken olarak adlandırılır ve veriden çıkarılır.

```
nzv <- nearZeroVar(data, saveMetrics= TRUE, freqCut = 10, uniqueCut = 10)
nzv
```

```
##                          freqRatio percentUnique zeroVar   nzv
```

```
## diagnosis                1.683962     0.3514938   FALSE FALSE
## radius_mean              1.333333    80.1405975   FALSE FALSE
## texture_mean             1.000000    84.1827768   FALSE FALSE
## perimeter_mean           1.000000    91.7398946   FALSE FALSE
## area_mean                1.500000    94.7275923   FALSE FALSE
## smoothness_mean          1.250000    83.3040422   FALSE FALSE
## compactness_mean         1.000000    94.3760984   FALSE FALSE
## concavity_mean           4.333333    94.3760984   FALSE FALSE
## concave.points_mean      4.333333    95.2548330   FALSE FALSE
## symmetry_mean            1.000000    75.9226714   FALSE FALSE
## fractal_dimension_mean   1.000000    87.6977153   FALSE FALSE
## radius_se                1.000000    94.9033392   FALSE FALSE
## texture_se               1.000000    91.2126538   FALSE FALSE
## perimeter_se             2.000000    93.6731107   FALSE FALSE
## area_se                  1.000000    92.7943761   FALSE FALSE
## smoothness_se            1.000000    96.1335677   FALSE FALSE
## compactness_se           1.000000    95.0790861   FALSE FALSE
## concavity_se             6.500000    93.6731107   FALSE FALSE
## concave.points_se        4.333333    89.1036907   FALSE FALSE
## symmetry_se              1.333333    87.5219684   FALSE FALSE
## fractal_dimension_se     1.000000    95.7820738   FALSE FALSE
## radius_worst             1.250000    80.3163445   FALSE FALSE
## texture_worst            1.000000    89.8066784   FALSE FALSE
## perimeter_worst          1.000000    90.3339192   FALSE FALSE
## area_worst               1.000000    95.6063269   FALSE FALSE
## smoothness_worst         1.000000    72.2319859   FALSE FALSE
## compactness_worst        1.000000    92.9701230   FALSE FALSE
## concavity_worst          4.333333    94.7275923   FALSE FALSE
## concave.points_worst     4.333333    86.4674868   FALSE FALSE
## symmetry_worst         188.666667     0.3514938   FALSE  TRUE
## fractal_dimension_worst  1.500000    94.0246046   FALSE FALSE
```

```
dim(data)
```

```
## [1] 569  31
```

```
nzv <- nearZeroVar(data, saveMetrics= FALSE, freqCut = 10, uniqueCut = 10)
names(data[nzv])
```

```
## [1] "symmetry_worst"
```

```
filteredData <- data[, -nzv]
dim(filteredData)
```

```
## [1] 569  30
```

1.3.2. Yüksek İlişkili Değişkenlerin Tespiti Bazı algoritmalar değişkenler arasındaki yüksek ilişkilere bağlı olarak geliştirilmiştir ve bu algoritmalar için değişkenler arasında yüksek korelasyon olması istenir (Partial Least Squares). Ancak, birçok makine öğrenimi algoritması değişkenler arasındaki yüksek ilişkiden olumsuz etkilenir ve değişkenler arasındaki korelasyon azaldığında performansları artar.

`caret` paketinde bulunan `findCorrelation` fonksiyonu kullanılarak istenilen kesim noktasına göre yüksek ilişkili değişkenler tespit edilebilir. Bu fonksiyon, her değişken için ortalama mutlak korelasyonu hesaplar

ve en büyük ortalama mutlak korelasyona sahip değişkeni veriden çıkarır. Başka bir deyişle, veriden hangi değişkenin çıkarılacağına karar verirken ilgili değişkenin verideki diğer değişkenlerle korelasyonuna bakılır ve en yüksek ortalama mutlak korelasyona sahip değişken modelden çıkarılır.

```
corr <-  cor(filteredData[,-1])
highlyCorr <- findCorrelation(corr, cutoff = 0.95)
removedVars <- names(filteredData[,-1])[highlyCorr]
removedVars
```

```
## [1] "perimeter_worst" "radius_worst"    "perimeter_mean"  "area_worst"
## [5] "radius_mean"     "perimeter_se"    "area_se"
```

```
filteredData <- filteredData[,!(names(filteredData)%in%removedVars)]
dim(filteredData)
```

```
## [1] 569  23
```

1.3.3. z Standartlaştırması Değişkenlerin standartlaştırılarak aynı birime indirgenmeleri model performansını arttırabilir. `caret` paketinde bulunan `preProcess` fonksiyonu kullanılarak z-standartlaştırması yapılabilir. Burada dikkat edilmesi gereken nokta standartlaştırmanın eğitim veri setine uygulanması ve test setine standartlaştırma uygulanırken eğitim setinden elde edilen ortalama ve standart sapma değerlerinin kullanılmasıdır.

```
## bu fonksiyon şu anda çalıştırılmayacak
## eğitim ve test setleri belirlendikten sonra çalıştırılacak
# preProcValues <- preProcess(training, method = c("center", "scale"))
# trainStandardized <- predict(preProcValues, training)
# testStandardized <- predict(preProcValues, test)
```

1.4. Eğitim ve Test Setlerinin Oluşturulması

Model oluşturmak için öncelikle algoritmanın eğitileceği eğitim (training) seti ve eğitilen modelin performansının test edileceği test seti oluşturulmalıdır. Eğitim ve test setlerinin hangi oranlarda oluşturulacağına ilişikin standart bir oran bulunmamakla birlikte %80 (eğitim)- %20 (test), %70 (eğitim) / %30 (test) sıklıkla kullanılan oranlardır. Bu oranlar veri setinin büyüklüğüne göre belirlenmelidir.

`caret` paketinde bulunan `createDataPartition` fonksiyonu kullanılarak eğitim ve test setleri belirlenebilir.

```
library(caret)
set.seed(1881)
inTrain <- createDataPartition(y = filteredData$diagnosis, ## sınıf değişkeni
p = 0.80, ## eğitim seti oranı
list = FALSE,
times = 1)

head(inTrain)
```

```
##      Resample1
## [1,]         1
## [2,]         2
## [3,]         4
```

```
## [4,]         7
## [5,]         8
## [6,]         9
```

```
training <- filteredData[ inTrain, ]
testing <- filteredData[-inTrain, ]
nrow(training)
```

```
## [1] 456
```

```
nrow(testing)
```

```
## [1] 113
```

Eğitim ve test setlerine z-standarlaştırmasının uygulanması.

```
preProcValues <- preProcess(training, method = c("center", "scale"))
trainStandardized <- predict(preProcValues, training)
testStandardized <- predict(preProcValues, testing)
```

1.5. Değişken Seçimi

caret paketinde bulunan `train` fonksiyonu ile eğitilen algoritmaların bir çoğu kendi içerisinde değişken seçimi yöntemlerine sahiptirler: `ada`, `AdaBag`, `AdaBoost.M1`, `adaboost`, `bagEarth`, `bagEarthGCV`, `bagFDA`, `bagFDAGCV`, `bartMachine`, `blasso`, `BstLm`, `bstSm`, `C5.0`, `C5.0Cost`, `C5.0Rules`, `C5.0Tree`, `cforest`, `chaid`, `ctree`, `ctree2`, `cubist`, `deepboost`, `earth`, `enet`, `evtree`, `extraTrees`, `fda`, `gamboost`, `gbm_h2o`, `gbm`, `gcvEarth`, `glmnet_h2o`, `glmnet`, `glmStepAIC`, `J48`, `JRip`, `lars`, `lars2`, `lasso`, `LMT`, `LogitBoost`, `M5`, `M5Rules`, `msaenet`, `nodeHarvest`, `OneR`, `ordinalNet`, `ORFlog`, `ORFpls`, `ORFridge`, `ORFsvm`, `pam`, `parRF`, `PART`, `penalized`, `PenalizedLDA`, `qrf`, `ranger`, `Rborist`, `relaxo`, `rf`, `rFerns`, `rfRules`, `rotationForest`, `rotationForestCp`, `rpart`, `rpart1SE`, `rpart2`, `rpartCost`, `rpartScore`, `rqlasso`, `rqnc`, `RRF`, `RRFglobal`, `sdwd`, `smda`, `sparseLDA`, `spikeslab`, `wsrf`, `xgbLinear`, `xgbTree`.

Çoğu zaman modellerin kendi değişken seçimi sonucuna göre oluşturdukları modelin performansı, harici bir değişken seçim yöntemi ile elde edilen model performansından daha iyidir.

Bu modeller dışında değişken seçimi için iki yöntem bulunmaktadır (John, Kohavi, ve Pfleger 1994):

Filtreleme yöntemleri: model performansı ile ilgilenmeyen, değişkenlerin tek değişkenli istatistiksel yöntemlerle değerlendirildiği ve belirli kriterlere uyan değişkenlerin modele alındığı değişken seçim yöntemleridir. Örneğin, t-testi, ANOVA, ki-kare testi, korelasyon analizi

`caret` paketinde bulunan `sbf` fonksiyonu ile filtreleme yöntemi ile değişken seçimi uygulanabilir.

```
set.seed(1881)
filterCtrl <- sbfControl(functions = caretSBF, method = "repeatedcv", number = 5, repeats = 1)
featureSelect <- sbf(x=trainStandardized[,-1], y=factor(trainStandardized[,1]), sbfControl = filterCtrl
featureSelect
```

```
##
## Selection By Filter
##
## Outer resampling method: Cross-Validated (5 fold, repeated 1 times)
##
```

```
## Resampling performance:
##
##  Accuracy  Kappa AccuracySD KappaSD
##    0.9649 0.9239    0.02117 0.04607
##
## Using the training set, 17 variables were selected:
##    texture_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean...
##
## During resampling, the top 5 selected variables (out of a possible 18):
##    area_mean (100%), compactness_mean (100%), compactness_se (100%), compactness_worst (100%), concav
##
## On average, 17.2 variables were selected (min = 17, max = 18)
```

```
featureSelect$variables$selectedVars
```

```
##  [1] "texture_mean"            "area_mean"
##  [3] "smoothness_mean"         "compactness_mean"
##  [5] "concavity_mean"          "concave.points_mean"
##  [7] "symmetry_mean"           "radius_se"
##  [9] "smoothness_se"           "compactness_se"
## [11] "concavity_se"            "concave.points_se"
## [13] "texture_worst"           "smoothness_worst"
## [15] "compactness_worst"       "concavity_worst"
## [17] "concave.points_worst"    "fractal_dimension_worst"
```

Wrapper yöntemler: iteratif bir süreç ile çoklu model kullanılarak model performansını maksimize eden en uygun değişken kombinasyonu bulunmaya çalışılır. Örneğin, recursive feature elimination, genetic algorithms, ve simulated annealing

`caret` paketinde bulunan `rfe` fonksiyonu ile 'Recursive Feature Elimination' yöntemi uygulanabilir.

```
set.seed(1881)

ctrl <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   number = 5,
                   repeats = 1,
                   verbose = FALSE)

featureSelect <- rfe(x=trainStandardized[,-1], y=factor(trainStandardized[,1]),
                  sizes = c(1:30),
                  metric = "Accuracy",
                  rfeControl = ctrl)

featureSelect
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (5 fold, repeated 1 times)
##
## Resampling performance over subset size:
##
```

```
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          1   0.8269 0.6289   0.041073 0.08450
##          2   0.9211 0.8308   0.021043 0.04585
##          3   0.9254 0.8398   0.021215 0.04550
##          4   0.9409 0.8739   0.027337 0.05707
##          5   0.9583 0.9104   0.009281 0.02038
##          6   0.9649 0.9241   0.014383 0.03167
##          7   0.9627 0.9192   0.016721 0.03685
##          8   0.9671 0.9288   0.011049 0.02455        *
##          9   0.9605 0.9149   0.012577 0.02786
##         10   0.9649 0.9241   0.012104 0.02679
##         11   0.9627 0.9196   0.009870 0.02198
##         12   0.9649 0.9241   0.012104 0.02679
##         13   0.9627 0.9194   0.009870 0.02187
##         14   0.9649 0.9241   0.009217 0.02046
##         15   0.9649 0.9241   0.009217 0.02046
##         16   0.9649 0.9245   0.009082 0.01939
##         17   0.9649 0.9241   0.009217 0.02046
##         18   0.9649 0.9241   0.009217 0.02046
##         19   0.9627 0.9192   0.012562 0.02780
##         20   0.9649 0.9241   0.009217 0.02046
##         21   0.9627 0.9192   0.012562 0.02780
##         22   0.9627 0.9192   0.012562 0.02780
##
## The top 5 variables (out of 8):
##    area_mean, concave.points_worst, concave.points_mean, concavity_worst, texture_worst
```

Rfe yöntemi ile seçilen en iyi 16 değişken:

```
    bestSubset <- featureSelect$optVariables
    bestSubset
```

```
## [1] "area_mean"            "concave.points_worst" "concave.points_mean"
## [4] "concavity_worst"      "texture_worst"        "concavity_mean"
## [7] "radius_se"            "texture_mean"
```

1.6. Model Kurma

Veri ön işleme, eğitim ve test setlerinin belirlenmesi ve değişken seçimi adımlarının ardından model kurma adımına geçilebilir.

Öncelikle bir önceki adımda rfe yöntemi ile seçilen değişkenleri kullanarak yeni eğitim ve test setimizi oluşturalım.

```
vars = c("diagnosis", bestSubset)
train = trainStandardized[,vars]
test = testStandardized[,vars]

train[1:5,1:4]
```

```
##   diagnosis  area_mean concave.points_worst concave.points_mean
## 1         M  0.9709914            2.2726962           2.5101371
## 2         M  1.8918683            1.0670532           0.5297482
```

```
## 4         M -0.7713078          2.1527392          1.4315163
## 7         M  1.0814966          1.1763810          0.6283429
## 8         M -0.2278487          0.6054468          0.2640831
```

```
dim(train)
```

```
## [1] 456   9
```

`caret` paketinde bulunan `train` fonksiyonu kullanılarak model eğitme işlemleri gerçekleştirilir. `train` fonksiyonu kullanılarak;

- parametre optimizasyonunun model performansı üzerindeki etkileri değerlendirilebilir,
- optimal model parametreleri seçilebilir,
- eğitim veri seti kullanılarak model performansı kestirilebilir,

`train` fonksiyonu içerisinde yüzlerce farklı makine öğrenimi algoritması eğitilebilir.

ÖRNEK: Destek vektör makineleri (support vector machines, SVM) algoritmasını kullanarak eğitim setimizi eğitelim.

```
set.seed(1881)
svmFit <- train(diagnosis ~ .,
data = train,
method = "svmRadial")

svmFit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 456 samples
##   8 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 456, 456, 456, 456, 456, 456, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.9638743  0.9218879
##   0.50  0.9681208  0.9310646
##   1.00  0.9700420  0.9350967
##
## Tuning parameter 'sigma' was held constant at a value of 0.2526579
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2526579 and C = 1.
```

1.6.1. Parametre Optimizasyonu Birçok makine öğrenimi algoritmasının optimize edilmesi gereken parametreleri bulunmaktadır.

SVM algoritmasının optimize edilmesi gereken iki parametresi bulunmaktadır: `sigma` ve `C` (cost)

`sigma` ve `C`: yanlış sınıflandırma için cezalandırma parametreleridir.

Model performansını arttırmak için SVM algoritmasının parametreleri olan sigma ve C parametrelerini optimize edelim. Bunu gerçekleştirebilmek için `train` fonksiyonundaki iki argümandan biri kullanılabilir: `tuneLength` ve `tuneGrid`

`tuneLength`: belirli bir uzunluktaki parametreler üzerinden optimizasyon gerçekleştirilir.

`tuneGrid`: kullanıcı tarafından belirlenen belirli bir aralıktaki parametreler üzerinden optimizasyon gerçekleştirilir.

```
set.seed(1881)
svmFit <- train(diagnosis ~ .,
 data = train,
 method = "svmRadial",
 tuneLength = 10)

 svmFit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 456 samples
##   8 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 456, 456, 456, 456, 456, 456, ...
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##     0.25  0.9638743  0.9218879
##     0.50  0.9681208  0.9310646
##     1.00  0.9700420  0.9350967
##     2.00  0.9698304  0.9346718
##     4.00  0.9676858  0.9299159
##     8.00  0.9641854  0.9223591
##    16.00  0.9577978  0.9086076
##    32.00  0.9540841  0.9004880
##    64.00  0.9485575  0.8885520
##   128.00  0.9463734  0.8839658
##
## Tuning parameter 'sigma' was held constant at a value of 0.2526579
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2526579 and C = 1.
```

caret paketinde bulunan `trainControl` fonksiyonu ile parametre optimizasyonu için bir takım düzenlemeler yapılabilir. Biz bu uygulamada örneklem seçim yöntemi olarak `repeatedcv` yöntemini kullanacağız. Bu yöntem çapraz geçerliliğin tekrarlı olarak uygulanmış halidir. Aynı fonksiyon içerisinde `repeatedcv` yöntemi için tekrar sayısı (`repeats`) ve çapraz geçerlilik için kat sayısı (`numbers`) belirlenebilir.

```
set.seed(1881)
ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

svmFit <- train(diagnosis ~ .,
data = train,
```

```
method = "svmRadial",
tuneLength = 10,
trControl = ctrl)

svmFit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 456 samples
##   8 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 365, 364, 365, 365, 365, 365, ...
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##     0.25  0.9605351  0.9158147
##     0.50  0.9649148  0.9251473
##     1.00  0.9649148  0.9251052
##     2.00  0.9685698  0.9328207
##     4.00  0.9671046  0.9292936
##     8.00  0.9641902  0.9231864
##    16.00  0.9605351  0.9154012
##    32.00  0.9517758  0.8964276
##    64.00  0.9459309  0.8841708
##   128.00  0.9444657  0.8810728
##
## Tuning parameter 'sigma' was held constant at a value of 0.2526579
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2526579 and C = 2.
```

`train` fonksiyonu içerisinde parametre optimizasyonu için kullanılan ölçü değiştirilebilir. Varsayılan olarak kullanılan `accuracy` ölçüsü yerine `ROC` ölçüsünü kullanalım.

```
set.seed(1881)
ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3,
                     classProbs = TRUE, summaryFunction = twoClassSummary)

svmFit <- train(diagnosis ~ ., data = train,
                method = "svmRadial",
                tuneLength = 10,
                trControl = ctrl,
                metric = "ROC")

svmFit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 456 samples
##   8 predictor
##   2 classes: 'B', 'M'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 365, 364, 365, 365, 365, 365, ...
## Resampling results across tuning parameters:
##
##   C       ROC        Sens       Spec
##     0.25  0.9928976  0.9662230  0.9490196
##     0.50  0.9924860  0.9708611  0.9549020
##     1.00  0.9927589  0.9708611  0.9568627
##     2.00  0.9927612  0.9743497  0.9588235
##     4.00  0.9921800  0.9755193  0.9588235
##     8.00  0.9908853  0.9720508  0.9509804
##    16.00  0.9901018  0.9720307  0.9392157
##    32.00  0.9879506  0.9651139  0.9333333
##    64.00  0.9843434  0.9592458  0.9294118
##   128.00  0.9791763  0.9546078  0.9176471
##
## Tuning parameter 'sigma' was held constant at a value of 0.2526579
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2526579 and C = 0.25.
```

1.6.2. Model Performansının Test Edilmesi ve Performans Ölçülerinin Elde Edilmesi

`caret` paketinde bulunan `predict` fonksiyonu kullanılarak test seti için sınıf kestirimleri yapılabilir.

```
svmClasses <- predict(svmFit, newdata = test)
```

`caret` paketinde bulunan `confusionMatrix` fonksiyonu kullanılarak test seti için performans ölçüleri hesaplanabilir.

```
confusionMatrix(data = svmClasses, factor(test$diagnosis), positive = "M")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 68  6
##          M  3 36
##
##                Accuracy : 0.9204
##                  95% CI : (0.8542, 0.9629)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : 9.656e-13
##
##                   Kappa : 0.827
##
##  Mcnemar's Test P-Value : 0.505
##
##             Sensitivity : 0.8571
##             Specificity : 0.9577
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 0.9189
```

```
##              Prevalence : 0.3717
##          Detection Rate : 0.3186
##    Detection Prevalence : 0.3451
##       Balanced Accuracy : 0.9074
##
##        'Positive' Class : M
##
```

`caret` paketinde bulunan `twoClassSummary` fonksiyonu kullanılarak test seti için eğri altında kalan alan (area under the ROC curve) değeri hesaplanabilir.

```
svmProbs <- predict(svmFit, newdata = test, type = "prob")
rocData <- data.frame(obs = factor(test$diagnosis), pred = svmClasses, svmProbs)
twoClassSummary(rocData, lev = levels(factor(test$diagnosis)))
```

```
##       ROC      Sens      Spec
## 0.9661301 0.9577465 0.8571429
```

1.6.3. Model Karşılaştırması Eğitim veri setini kullanarak bir random forest modeli eğitelim. Bunun için `train` fonksiyonu içerisine `method = "rf"` yazmak yeterli olacaktır.

RF algoritması için `mtry` parametresinin optimize edilmesi gerekir.

`mtry`: RF algoritmasındaki her ağaçta yer alan bölme için rastgele seçilen değişken sayısı. Sınıflandırma probleminde varsayılan olarak değişken sayısının karekökü kadar belirlenirken regresyon probleminde değişken sayısının üçte biri olarak belirlenir.

```
set.seed(1881)

ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3,
                     classProbs = TRUE, summaryFunction = twoClassSummary)


rfFit <- train(diagnosis ~ .,
data = train,
method = "rf",
tuneLength = 10,
trControl = ctrl,
metric = "ROC")
```

```
## note: only 7 unique complexity parameters in default grid. Truncating the grid to 7 .
```

```
rfFit
```

```
## Random Forest
##
## 456 samples
##   8 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
```

```
## Summary of sample sizes: 365, 364, 365, 365, 365, 365, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.9958157  0.9859851  0.9313725
##   3     0.9952816  0.9824965  0.9450980
##   4     0.9948863  0.9778383  0.9431373
##   5     0.9948151  0.9731801  0.9490196
##   6     0.9942867  0.9708409  0.9529412
##   7     0.9938893  0.9720105  0.9509804
##   8     0.9938555  0.9696915  0.9529412
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
rfProbs <- predict(rfFit, newdata = test, type = "prob")
rfClasses <- predict(rfFit, newdata = test)
confusionMatrix(rfClasses, factor(test$diagnosis), positive = "M")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 70  6
##          M  1 36
##
##                Accuracy : 0.9381
##                  95% CI : (0.8765, 0.9747)
##     No Information Rate : 0.6283
##     P-Value [Acc > NIR] : 1.718e-14
##
##                   Kappa : 0.8641
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.8571
##             Specificity : 0.9859
##          Pos Pred Value : 0.9730
##          Neg Pred Value : 0.9211
##              Prevalence : 0.3717
##          Detection Rate : 0.3186
##    Detection Prevalence : 0.3274
##       Balanced Accuracy : 0.9215
##
##        'Positive' Class : M
##
```

```
rocData <- data.frame(obs = factor(test$diagnosis), pred = rfClasses, rfProbs)
twoClassSummary(rocData, lev = levels(factor(test$diagnosis)))
```

```
##       ROC      Sens      Spec
## 0.9813883 0.9859155 0.8571429
```

Modellerin performanslarını karşılaştırmak için `caret` paketinde bulunan `resamples` ve `diffs` fonksiyonları kullanılabilir. `resamples` fonksiyonu ile her bir katta ve tekrarda eğitim seti için elde edilen performans ölçüleri elde edilir.

```
resamps <- resamples(list(svm = svmFit, rf = rfFit))
head(resamps$values)
```

```
##     Resample   svm~ROC  svm~Sens  svm~Spec    rf~ROC   rf~Sens   rf~Spec
## 1 Fold1.Rep1 0.9927761 0.9649123 0.9411765 0.9917441 0.9649123 0.9117647
## 2 Fold1.Rep2 0.9865841 0.9473684 0.8823529 0.9963880 0.9824561 0.8823529
## 3 Fold1.Rep3 0.9984520 0.9824561 0.9411765 0.9963880 0.9649123 0.9411765
## 4 Fold2.Rep1 0.9934077 0.9655172 0.9117647 0.9923935 0.9827586 0.8823529
## 5 Fold2.Rep2 0.9958720 0.9649123 1.0000000 0.9927761 0.9649123 0.9117647
## 6 Fold2.Rep3 1.0000000 1.0000000 1.0000000 0.9979360 1.0000000 0.9705882
```

diffs fonksiyonu ile elde edilen performans ölçüleri eşleştirilmiş t testi ile karşılaştırılır.

```
diffs <- diff(resamps)
summary(diffs)
```

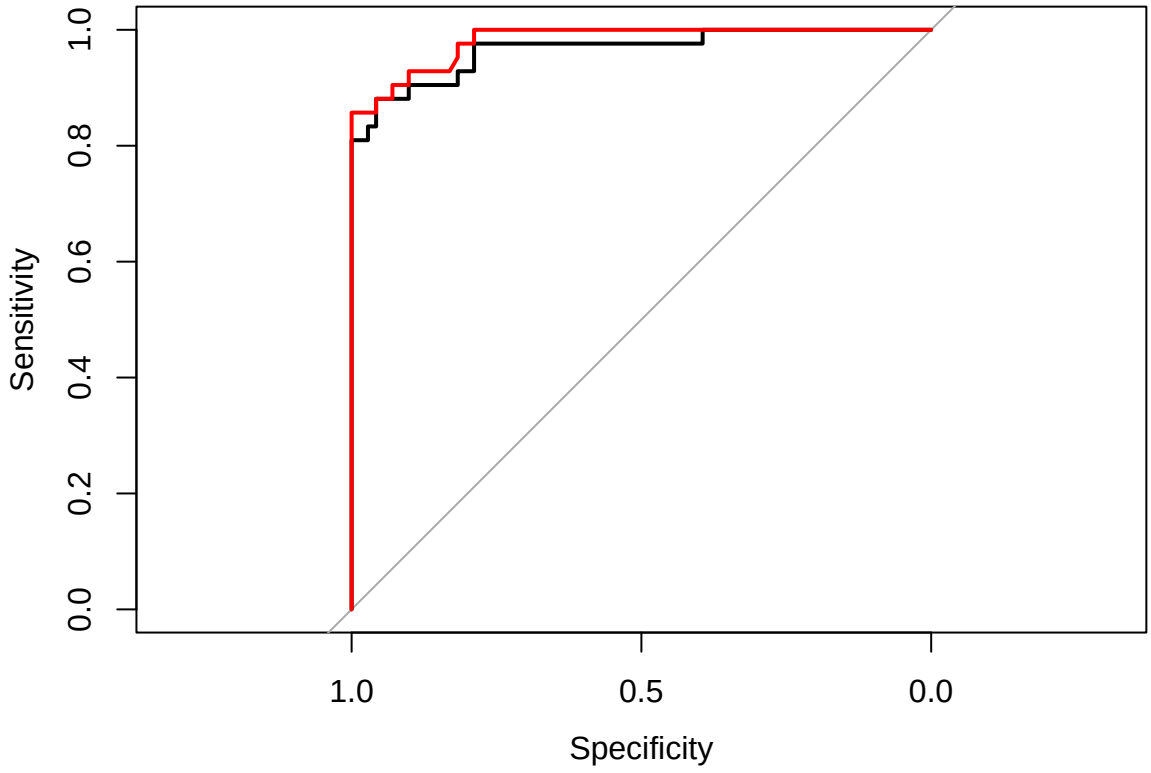
```
##
## Call:
## summary.diff.resamples(object = diffs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## ROC
##     svm    rf
## svm        -0.002918
## rf  0.0733
##
## Sens
##     svm      rf
## svm          -0.01976
## rf  0.009308
##
## Spec
##     svm     rf
## svm         0.01765
## rf  0.02299
```

İki modelin performanslarını kıyaslamak için ROC eğrileri de incelenebilir.

```
library(pROC)

svmRoc=roc(test$diagnosis ~ svmProbs[,1])
rfRoc=roc(test$diagnosis ~ rfProbs[,1])

{plot = plot(svmRoc)
plot(rfRoc, add=TRUE, col='red')}
```



1.7. Değişken Önemliliği

```
plot(varImp(rfFit, scale = TRUE))
```