Adaboost Example

AdaBoost (Adaptive Boosting) is an ensemble learning technique that combines the predictions of multiple weak classifiers to create a strong classifier. In this tutorial, we'll walk through the process of building an AdaBoost model using the adabag package in R. We'll also cover how to visualize and evaluate the model's performance.

Step 1: Data Preparation

Let's begin by generating some synthetic data. We'll create two features, X1 and X2, and a binary target variable, y.

```
#install.packages("adabag")
```

Now, load the necessary libraries:

```
library(adabag)
```

```
## Loading required package: rpart
## Loading required package: caret
## Loading required package: ggplot2
## Loading required package: lattice
## Loading required package: foreach
## Loading required package: doParallel
## Loading required package: iterators
## Loading required package: parallel
```

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

Step 2: Prepare the Data

We will use the iris data for this example

```
indexes=createDataPartition(iris$Species, p=.90, list = F)
train = iris[indexes, ]
test = iris[-indexes, ]
```

In this example, createDataPartition is used to randomly split the rows of the iris dataset based on the Species variable. It will return either a vector of indices (if list = FALSE) or a list containing training and testing set indices (if list = TRUE) that can be used to subset the original dataset into separate training and testing datasets for model training and evaluation. The training set will contain 90% of the data, and the testing set will contain 10%.

Step 3: Build the AdaBoost Model

```
model = boosting(Species~., data=train, boos=TRUE, mfinal=50)
```

the *model = boosting(Species~., data=train, boos=TRUE, mfinal=50)* statement creates an AdaBoost model that predicts the Species variable based on all other variables in the train dataset.

It uses AdaBoost boosting with a maximum of 50 iterations to build a strong ensemble classifier.

We can check the model properties

```
print(names(model))
```

```
## [1] "formula"    "trees"      "weights"    "votes"      "prob"
## [6] "class"      "importance" "terms"      "call"
```

```
print(model$trees[1])
```

```
## [[1]]
## n= 135
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 135 84 versicolor (0.29629630 0.37777778 0.32592593)
##   2) Petal.Length< 2.45 40  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 95 44 versicolor (0.00000000 0.53684211 0.46315789)
##     6) Petal.Length< 4.85 48  1 versicolor (0.00000000 0.97916667 0.02083333) *
##     7) Petal.Length>=4.85 47  4 virginica (0.00000000 0.08510638 0.91489362) *
```

Step 4: Make predictions

The model is ready and we can predict test data. Predicted data accuracy is also included in output data.

```
pred = predict(model, test)

print(pred$confusion)
```

```
##                Observed Class
## Predicted Class setosa versicolor virginica
##      setosa          5          0         0
##      versicolor      0          5         1
##      virginica       0          0         4
```

```
print(pred$error)
```

```
## [1] 0.06666667
```

We can also print the probability of each class in test data.

```
result = data.frame(test$Species, pred$prob, pred$class)
print(result)
```

```
##    test.Species         X1         X2         X3 pred.class
## 1        setosa 0.92192447 0.07807553 0.00000000     setosa
## 2        setosa 0.89893662 0.10106338 0.00000000     setosa
## 3        setosa 0.89893662 0.10106338 0.00000000     setosa
## 4        setosa 0.92192447 0.07807553 0.00000000     setosa
## 5        setosa 0.92192447 0.07807553 0.00000000     setosa
## 6    versicolor 0.02463799 0.97536201 0.00000000 versicolor
## 7    versicolor 0.00000000 0.85447695 0.14552305 versicolor
## 8    versicolor 0.03746574 0.96253426 0.00000000 versicolor
## 9    versicolor 0.02463799 0.97536201 0.00000000 versicolor
## 10   versicolor 0.02463799 0.97536201 0.00000000 versicolor
## 11    virginica 0.00000000 0.01472184 0.98527816  virginica
## 12    virginica 0.04123338 0.88218393 0.07658269 versicolor
## 13    virginica 0.00000000 0.04118992 0.95881008  virginica
## 14    virginica 0.00000000 0.09338611 0.90661389  virginica
```

```
## 15    virginica 0.00000000 0.23045100 0.76954900  virginica
```

Step 5: Classification with boosting.cv

he boosting.cv function provides cross-validation method. The training data is divided into multiple subsets to apply boosting and prediction is performed for the entire dataset. To train the model we use entire dataset and get prediction result. Here, v is cross-validation subsets numbers.

```
cvmodel = boosting.cv(Species~., data=iris, boos=TRUE, mfinal=10, v=5)
```

```
## i:  1 Sun Sep 24 19:30:28 2023
## i:  2 Sun Sep 24 19:30:29 2023
## i:  3 Sun Sep 24 19:30:29 2023
## i:  4 Sun Sep 24 19:30:29 2023
## i:  5 Sun Sep 24 19:30:30 2023
```

```
# check accuracy
print(cvmodel[-1])
```

```
## $confusion
##                Observed Class
## Predicted Class setosa versicolor virginica
##      setosa         50          0         0
##      versicolor      0         44         4
##      virginica       0          6        46
##
## $error
## [1] 0.06666667
```

You can compare the original and predicted classes.

```
# data.frame(iris$Species, cvmodel$class)
```