Data621 - Blog2

Amit Kapoor

4/3/2021

Non Linear Regression

Non-linear regression is a method to model a non-linear relationship between the dependent variable and independent variable(s). It is a regression technique in which the dependent variables are modeled as a non-linear function of one or more independent variables. Simple linear regression shows the relationship between two variables (X and y) with a straight line (y = aX + b), while nonlinear regression shows the relationship between the two variables in a nonlinear (or curved) relationship. In this blog we will cover 4 non linear regression models and their computing. We will use the package `mlbench` that contains a function called `mlbench.friedman1` that simulates the non linear data.

K-Nearest Neighbors

K-nearest neighbors is the simplest non linear model which is widely used. KNN regression tries to predict the value of the output variable by using a local average. It simply predicts a new sample using the K-closest samples from the training set. It's algorithm assumes that similar things exist in close proximity. In other words, this approach simply predicts a new sample using the K-closest samples from the training set. Here we will use training using knn method on training data and find the besttuned k value.

```
set.seed(317)
knnfit <- train(trainingData$x,
                trainingData$y,
                method = "knn",
                preProcess = c("center","scale"),
                tuneLength = 5,
                trControl = trainControl(method = "cv"))

knnfit
```

```
## k-Nearest Neighbors
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    5  3.504854  0.4917897  2.730587
##    7  3.468714  0.5263166  2.771600
##    9  3.417488  0.5690325  2.715001
##   11  3.403666  0.6073053  2.707929
```

```
##   13  3.397409  0.6193299  2.709256
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N being the number of features) that classifies the data points. Hyperplanes are decision boundaries to classify the data points. Data points that falls on either side of the hyperplane can be qualified for different classes. Support vectors are data points that are closer to the hyperplane and effect the position and orientation of the hyperplane. Using these support vectors, we do maximize the margin of the classifier.

```
set.seed(317)
svmfit <- train(trainingData$x,
                trainingData$y,
                method = "svmRadial",
                preProcess = c("center","scale"),
                tuneLength = 5,
                trControl = trainControl(method = "cv"))

svmfit
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  2.970943  0.7016496  2.356409
##   0.50  2.617538  0.7363843  2.063172
##   1.00  2.322752  0.7762193  1.801428
##   2.00  2.155949  0.8002262  1.613436
##   4.00  2.089039  0.8087624  1.595103
##
## Tuning parameter 'sigma' was held constant at a value of 0.05958105
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05958105 and C = 4.
```

Multivariate Adaptive Regression Splines

MARS creates a piecewise linear model which provides an intuitive stepping block into non-linearity after grasping the concept of multiple linear regression. MARS provided a convenient approach to capture the nonlinear relationships in the data by assessing cutpoints (knots) similar to step functions. The procedure assesses each data point for each predictor as a knot and creates a linear regression model with the candidate features.

```
set.seed(317)
marsGrid <- expand.grid(.degree=1:2, .nprune=1:5)
marsfit <- train(trainingData$x,
                trainingData$y,
```

```
                method = "earth",
                preProcess = c("center","scale"),
                tuneGrid = marsGrid,
                trControl = trainControl(method = "cv"))

marsfit
```

```
## Multivariate Adaptive Regression Spline
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 180, 180, 180, 180, 180, 180, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE      Rsquared   MAE
##   1       1       4.765652        NaN  3.891035
##   1       2       4.374986  0.2125511  3.698522
##   1       3       3.840402  0.3680465  3.105501
##   1       4       2.996576  0.6282644  2.355508
##   1       5       2.882511  0.6558138  2.352542
##   2       1       4.765652        NaN  3.891035
##   2       2       4.374986  0.2125511  3.698522
##   2       3       3.840402  0.3680465  3.105501
##   2       4       3.051692  0.6151892  2.395437
##   2       5       2.912588  0.6457529  2.372144
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 5 and degree = 1.
```

Neural Networks

Neural Networks are nonlinear regression techniques inspired by theories about how the brain works. The outcome is modeled by an intermediary set of unobserved variables (hidden variables). These hidden units are linear combinations of the original predictors. Neural networks contains node layers, having an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is more than specified threshold value, that node gets activated and sends data to the next layer of the network else it is not passed to the next layer.

Similar to earlier approach of choosing the number of hidden units and the amount of weight decay via resampling, here the train function is applied using method = "avNNet". avNNet Aggregate several neural network models.

```
set.seed(317)
nnetGrid <- expand.grid(.decay=c(0.05,.1),
                        .size=2,
                        .bag=FALSE)

nnetfit <- train(trainingData$x,
                 trainingData$y,
                 method = "avNNet",
                 tuneGrid = nnetGrid,
                 preProcess = c("center","scale"),
```

```
                linout = TRUE,
                trace = FALSE,
                MaxNWts =10 * (ncol(trainingData$x)+1) +10+1,
                maxit=500)

nnetfit
```

```
## Model Averaged Neural Network
##
## 200 samples
##  10 predictor
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   decay  RMSE      Rsquared   MAE
##   0.05   2.932520  0.6384131  2.279432
##   0.10   2.947952  0.6352441  2.280918
##
## Tuning parameter 'size' was held constant at a value of 2
## Tuning
##  parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 2, decay = 0.05 and bag = FALSE.
```

References

Applied Predictive Modeling by Max Kuhn and Kjell Johnson