Chapter 7

Chapter 7. Nonlinear Regression Models

7.1 Neural Networks

The outcome is modeled by an intermediary set of unobserved variables (hidden variables or hidden units). These hidden units are linear combinations of the original predictors, but, unlike PLS models, they are not estimated in a hierarchical fashion

Once the number of hidden units is defined, each unit must be related to the outcome. Another linear combination connects the hiddden units to the outcome

Early stopping: the iterative algorithms for solving for the regression equations can be permaturely halted

Weight decay: a penalization method to regularize model similar to ridge regression

Parameters: number of hidden layers, lambda (weight decay)

Predictors should be centered and scaled prior to modeling

7.2 Multivariate Adaptive Regression Splines

Like neural networks and partial least squares, MARS uses surrogate features instead of the original predictors

MARS creates two contrasted versions of a predictor to enter the model. Also, the surrogate features in MARS are usually a function of only one or two predictors at a time. The nature of the MARS features breaks the predictor into two groups and models linear relationships between the predictor and the outcome in each group.

Piecewise linear model: each new feature models an isolated portion of the original data

Hinge function

Two tuning parameters associated with the MARS model: the degree of the features that are added to the model and the number of retained terms

Advantages of MARS: First, the model automatically conducts feature selection, the model equation is independent of predictor variables that are not involved with any of the final model features. Second, interpretability. Each hinge feature is responsible for modeling a specific region in the predictor space using a piecewise linear model. When the MARS model is addictive, the contribution of each predictor can be isolated without the need to consider the others. Third, data transformation and the filtering of predictors are not needed.

The additive nature of the model allows each predictor to be viewed in isolation: changing the values of the other predictor variables will not alter the shape of the profile, only the location on the y-axis where the profile starts

7.3 Support Vector Machines

e-insensitive regression (minimize the effect of outliers on the regression equations.)

Given a threshold set by the user (denoted as e),data points with residuals within the threshold do not contribute to the regression fit while data points with an absolute difference greater than the threshold contribute a linear-scale amount

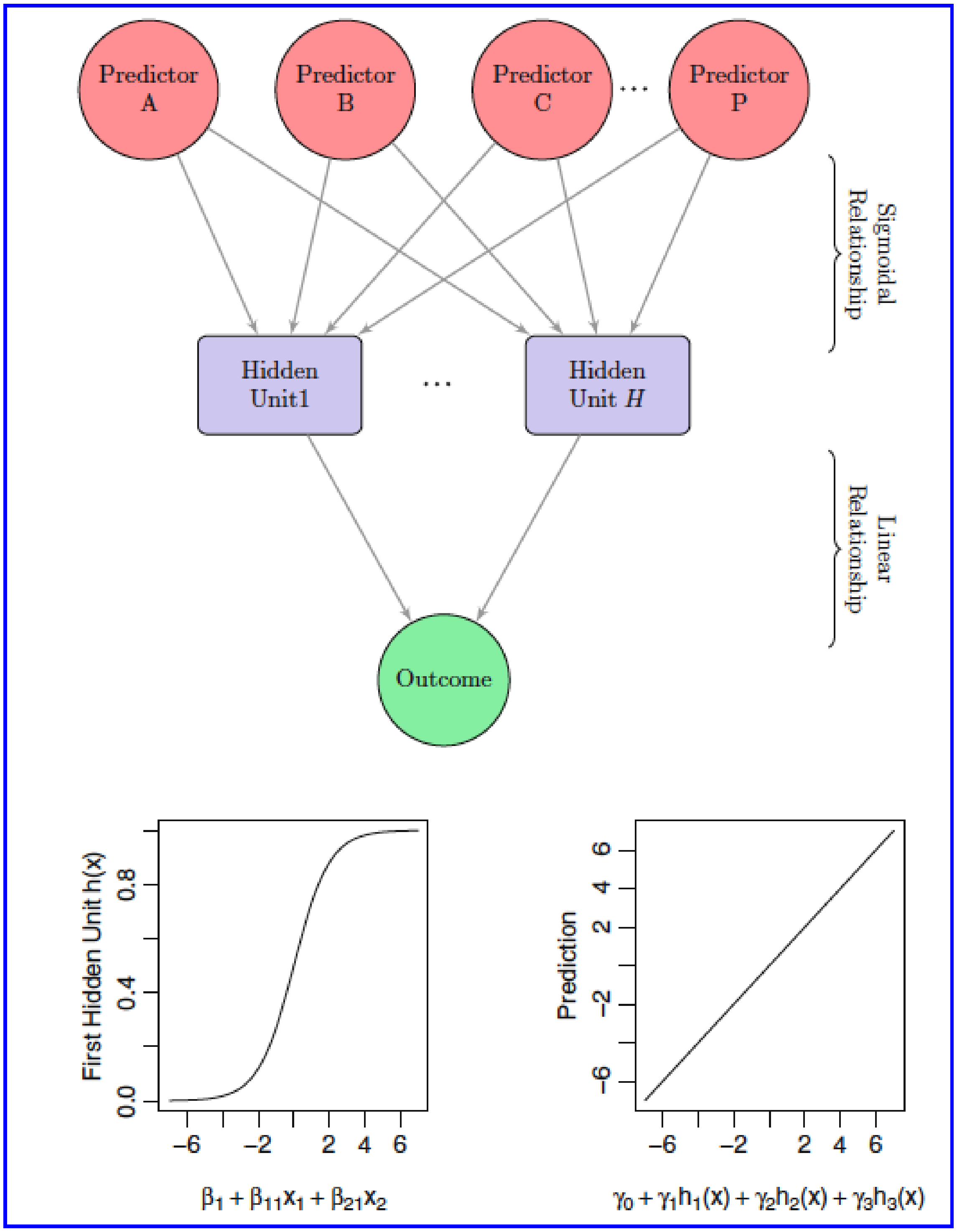

Fig. 7.1: A diagram of a neural network with a single hidden layer. The hidden units are linear combinations of the predictors that have been transformed by a sigmoidal function. The output is modeled by a linear combination of the hidden units

Figure 1: 1

As a consequence, only a subset of training set data points, where a /= 0, are needed for prediction. Since the regression line is determined using these samples, they are called the support vectors as they support the regression line.

kernel functions

Main tuning parameter: cost value

Centering and scaling the predictors prior to building an SVM model

7.4 K-Nearest Neighbours

To predict a new sample for regression, KNN identifies that sample's KNNs in the predictor space. The predicted response for the new sample is then the mean of the K neighbours' responses

Distance metrics: Euclidean distance, Minkowski distance

All predictors should be centered and scaled prior to performing KNN

Impute the missing values

Number of neighbours K can be determined by resampling

Computation complexity

The KNN method can have poor predictive performance when local predictor structure is not relevant to the response

7.5 Computing

Neural Networks

```r
library(nnet)
# nnetFit <- nnet(predictors, outcome, size = 5, decay = 0.01, linout = TRUE,
                # reduce the amount of printed output
                # trace = FALSE,
                # Expand the number of iterations to find parameter estimates
                # maxit = 500,
                # and the number of parameters used by the model
                # MaxNWts = 5 * (ncol(predictors)+1)+5+1)
```

```r
# nnetAvg <- avNNet(predictors, outcome, size = 5, decay = 0.01,
                  # specify how many models to average
                  # repeats = 5, linout = TRUE,
                  # reduce the amount of printed output
                  # trace = FALSE,
                  # expand the number of iterations to find parameter estimates
                  # maxit = 500,
                  # and the number of parameters used by the model
                  # MaxNWts = 5 * (ncol(predictors) + 1) + 5 + 1)
```

```r
# predict(nnetFit, newData)
# predict(nnetAvg, newData)
# The findCorrelation takes a correlation matrix and determines the column numbers that should be remov

library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
tooHigh <- findCorrelation(cor(solTrainXtrans),cutoff = 0.75)
```

```
## Error in is.data.frame(x): object 'solTrainXtrans' not found
```

```
trainXnnet <- solTrainXtrans[,-tooHigh]
```

```
## Error in eval(expr, envir, enclos): object 'solTrainXtrans' not found
```

```
testXnnet <- solTestXtrans[,-tooHigh]
```

```
## Error in eval(expr, envir, enclos): object 'solTestXtrans' not found
```

```
# create a specific candidate set of models to evaluate
nnetGrid <- expand.grid(.decay = c(0, 0.01, 0.1), .size = c(1:10),
                        # to use bagging instead of different random seeds
                        .bag = FALSE)

set.seed(100)
nnetTune <- train(solTrainXtrans, solTrainY, method = "avNNet", tuneGrid = nnetGrid, trControl = ctrl,
                  # automatically standardize data prior to modeling and prediction
                  preProc = c("center", "scale"), linout = TRUE, trace = FALSE,
                  MaxNWts = 10 * (ncol(trainXnnet)+1)+10+1, maxit = 500)
```

```
## Error in train(solTrainXtrans, solTrainY, method = "avNNet", tuneGrid = nnetGrid, : object 'solTrainX
```

```
nnetTune
```

```
## Error in eval(expr, envir, enclos): object 'nnetTune' not found
```

Multivariate Adaptive Regression Splines

```
library(earth)
```

```
## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```r
marsFit <- earth(solTrainXtrans, solTrainY)
```

```
## Error in earth(solTrainXtrans, solTrainY): object 'solTrainXtrans' not found
```

```r
marsFit
```

```
## Error in eval(expr, envir, enclos): object 'marsFit' not found
```

```r
summary(marsFit)
```

```
## Error in summary(marsFit): object 'marsFit' not found
```

```r
plotmo(marsFit)
```

```
## Error in plotmo(marsFit): object 'marsFit' not found
```

```r
# Define the candidate models to test
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

# Fix the seed so that the results can be reproduced
set.seed(100)
marsTune <- train(solTrainXtrans, solTrainY, method = "earth",
                  # explicitly declare the candidate models to test
                  tuneGrid = marsGrid, trControl = trainControl(method = "cv"))
```

```
## Error in train(solTrainXtrans, solTrainY, method = "earth", tuneGrid = marsGrid, : object 'solTrainX
```

```r
marsTune
```

```
## Error in eval(expr, envir, enclos): object 'marsTune' not found
```

```r
head(predict(marsTune, solTestXtrans))
```

```
## Error in predict(marsTune, solTestXtrans): object 'marsTune' not found
```

```r
varImp(marsTune)
```

```
## Error in varImp(marsTune): object 'marsTune' not found
```

Support Vector Machines

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# svmFit <- ksvm(x = solTrainXtrans, y = solTrainY, kernel = "rbfdot", kpar = "automatic", C = 1, epsil
```

```
svmRTuned <- train(solTrainXtrans, solTrainY, method = "svmRadial", preProc = c("center","scale"),
                  tuneLength = 14, trControl = trainControl(method = "cv"))
```

```
## Error in train(solTrainXtrans, solTrainY, method = "svmRadial", preProc = c("center", : object 'solT
```

```
svmRTuned
```

```
## Error in eval(expr, envir, enclos): object 'svmRTuned' not found
```

```
svmRTuned$finalModel
```

```
## Error in eval(expr, envir, enclos): object 'svmRTuned' not found
```

K-Nearest Neighbors

```
# remove a few sparse and unbalanced fingerprints first
knnDescr <- solTrainXtrans[,-nearZeroVar(solTrainXtrans)]
```

```
## Error in eval(expr, envir, enclos): object 'solTrainXtrans' not found
```

```
set.seed(100)
knnTune <- train(knnDescr, solTrainY, method = "knn",
                 # center and scaling will occur for new predictions too
                 preProc = c("center", "scale"),
                 tuneGrid = data.frame(.k=1:20),
                 trControl = trainControl(method = "cv"))
```

```
## Error in train(knnDescr, solTrainY, method = "knn", preProc = c("center", : object 'knnDescr' not fo
```

```
knnTune
```

```
## Error in eval(expr, envir, enclos): object 'knnTune' not found
```

```
plot(knnTune)
```

```
## Error in plot(knnTune): object 'knnTune' not found
```