Chapter 7 Nonlinear models

Load the required packages and data

|  |
| --- |
| #Required packages  library(AppliedPredictiveModeling)  library(caret)  library(earth)  ### Load the data  data(solubility)  ### Create a control function that will be used across models. We  ### create the fold assignments explicitly instead of relying on the  ### random number seed being set to identical values.  set.seed(100)  indx <- createFolds(solTrainY, returnTrain = TRUE)  ctrl <- trainControl(method = "cv", index = indx) |

Neural Networks

|  |
| --- |
| ################################################################################  ### Neural Networks  #Create a grid for tuning parameters  nnetGrid <- expand.grid(decay = c(0, 0.01, .1),  size = c(1, 3, 5, 7),  bag = FALSE)  #It takes time run  ##The following codes takes more than 6,000 seconds to run.  ## Your running time may be different depending  ## on your cpu.  ptm <- proc.time() #takes more than 6,000 seconds to run in my computer  set.seed(100)  nnetTune <- train(x = solTrainXtrans, y = solTrainY,  method = "avNNet",  tuneGrid = nnetGrid,  trControl = ctrl,  preProc = c("center", "scale"),  linout = TRUE,  trace = FALSE,  MaxNWts = 13 \* (ncol(solTrainXtrans) + 1) + 13 + 1,  maxit = 1000,  allowParallel = FALSE)  nnetTune  Model Averaged Neural Network  951 samples  228 predictors  Pre-processing: centered (228), scaled (228)  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results across tuning parameters:  decay size RMSE Rsquared MAE  0.00 1 0.8279833 0.8411337 0.6351877  0.00 3 0.8415406 0.8320163 0.6362154  0.00 5 0.7879532 0.8518015 0.5818827  0.00 7 0.8497575 0.8325912 0.6314441  0.01 1 0.7255300 0.8766697 0.5318100  0.01 3 0.8237462 0.8456118 0.6105706  0.01 5 0.8124533 0.8517685 0.5963451  0.01 7 0.7930880 0.8551410 0.5678490  0.10 1 0.7369941 0.8724582 0.5393379  0.10 3 0.7854472 0.8560870 0.5544581  0.10 5 0.6950386 0.8879509 0.5015236  0.10 7 0.6769793 0.8928230 0.4863267  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 = 7, decay = 0.1 and bag = FALSE.  proc.time() - ptm  ### Stop the clock  plot(nnetTune)    #save the predicted values into testResults  testResults <- data.frame(obs = solTestY,  NNet = predict(nnetTune, solTestXtrans))  head(testResults)  # obs NNet  #20 0.93 0.6531965  #21 0.85 0.7620576  #23 0.81 0.2121431  #25 0.74 1.1017603  #28 0.61 -0.4494221  #31 0.58 1.1281280 |

MARS

|  |
| --- |
| ### Multivariate Adaptive Regression Splines  ptm <- proc.time() #takes 163 seconds to run in my computer  set.seed(100)  marsTune <- train(x = solTrainXtrans, y = solTrainY,  method = "earth",  tuneGrid = expand.grid(degree = 1, nprune = 2:38),  trControl = ctrl)  marsTune  Multivariate Adaptive Regression Spline  951 samples  228 predictors  No pre-processing  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results across tuning parameters:  nprune RMSE Rsquared MAE  2 1.5390057 0.4375015 1.1745616  3 1.1044691 0.7125038 0.8408548  4 1.0354182 0.7457278 0.7977694  5 0.9920646 0.7659750 0.7521363  6 0.9324996 0.7918923 0.7176847  7 0.9123327 0.8000754 0.7047421  8 0.8714228 0.8191121 0.6793606  9 0.8622010 0.8221868 0.6668435  10 0.8559876 0.8237726 0.6530190  11 0.8397856 0.8298046 0.6347597  12 0.8328426 0.8316524 0.6287339  13 0.8021653 0.8443921 0.6147170  14 0.7790891 0.8537616 0.5946793  15 0.7744492 0.8551611 0.5927687  16 0.7584420 0.8611298 0.5806881  17 0.7436889 0.8665585 0.5673168  18 0.7440275 0.8670957 0.5695748  19 0.7347737 0.8696648 0.5614749  20 0.7367187 0.8690113 0.5625564  21 0.7256314 0.8725350 0.5533448  22 0.7227027 0.8733250 0.5480017  23 0.7148578 0.8762923 0.5469065  24 0.7149982 0.8763551 0.5494001  25 0.7135022 0.8772182 0.5491729  26 0.7116974 0.8775205 0.5475208  27 0.7074705 0.8792338 0.5435797  28 0.7015355 0.8814607 0.5369847  29 0.6961520 0.8831530 0.5322974  30 0.6938606 0.8839511 0.5293593  31 0.6930248 0.8840253 0.5291532  32 0.6887240 0.8855800 0.5254246  33 0.6844724 0.8872491 0.5239397  34 0.6782811 0.8894273 0.5192552  35 0.6781610 0.8893081 0.5185573  36 0.6772147 0.8895992 0.5181628  37 0.6778717 0.8893533 0.5185987  38 0.6774625 0.8893991 0.5176233  Tuning parameter 'degree' was held constant at a value of 1  RMSE was used to select the optimal model using the smallest value.  The final values used for the model were nprune = 36 and degree = 1.  proc.time() - ptm  plot(marsTune)    #Check the importance of each predictor  marsImp <- varImp(marsTune, scale = FALSE)  plot(marsImp, top = 25)    #save the predicted values into testResults  testResults$MARS <- predict(marsTune, solTestXtrans) |

SVM with radial basis function

|  |
| --- |
| ### Support Vector Machines  ## In a recent update to caret, the method to estimate the  ## sigma parameter was slightly changed. These results will  ## slightly differ from the text for that reason.  #SVM with the radial basis function function  ptm <- proc.time() # Takes 72.23 seconds in my computer  set.seed(100)  svmRTune <- train(x = solTrainXtrans, y = solTrainY,  method = "svmRadial",  preProc = c("center", "scale"),  tuneLength = 14,  trControl = ctrl)  svmRTune  Support Vector Machines with Radial Basis Function Kernel  951 samples  228 predictors  Pre-processing: centered (228), scaled (228)  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results across tuning parameters:  C RMSE Rsquared MAE  0.25 0.8082294 0.8636665 0.6028126  0.50 0.7158045 0.8849736 0.5315914  1.00 0.6666914 0.8966605 0.4957241  2.00 0.6363044 0.9037211 0.4692192  4.00 0.6216719 0.9070653 0.4543005  8.00 0.6077567 0.9109752 0.4430939  16.00 0.6031571 0.9123541 0.4393460  32.00 0.6007413 0.9132360 0.4383832  64.00 0.5981252 0.9142755 0.4362848  128.00 0.5958107 0.9151748 0.4353230  256.00 0.5957323 0.9155000 0.4377823  512.00 0.5997785 0.9145462 0.4415741  1024.00 0.6038872 0.9136810 0.4440797  2048.00 0.6079735 0.9127263 0.4476578  Tuning parameter 'sigma' was held constant at a value of 0.00265531  RMSE was used to select the optimal model using the smallest value.  The final values used for the model were sigma = 0.00265531 and C = 256.  proc.time() - ptm  plot(svmRTune, scales = list(x = list(log = 2)))    ##save the predicted values into testResults  testResults$SVMr <- predict(svmRTune, solTestXtrans) |

SVM with polynomial basis function

|  |
| --- |
| ptm <- proc.time() # takes 313.55 second to run  #SVM with the polynomial basis function  svmGrid <- expand.grid(degree = 1:2,  scale = c(0.01, 0.005, 0.001),  C = 2^(-2:5))  set.seed(100)  svmPTune <- train(x = solTrainXtrans, y = solTrainY,  method = "svmPoly",  preProc = c("center", "scale"),  tuneGrid = svmGrid,  trControl = ctrl)  svmPTune  Support Vector Machines with Polynomial Kernel  951 samples  228 predictors  Pre-processing: centered (228), scaled (228)  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results across tuning parameters:  degree scale C RMSE Rsquared MAE  1 0.001 0.25 1.0523656 0.7929664 0.7860298  1 0.001 0.50 0.8983142 0.8327514 0.6759869  1 0.001 1.00 0.7985992 0.8561235 0.6049115  1 0.001 2.00 0.7422340 0.8714661 0.5636232  1 0.001 4.00 0.7093366 0.8810412 0.5374689  1 0.001 8.00 0.6977103 0.8842710 0.5279487  1 0.001 16.00 0.7008511 0.8833591 0.5277940  1 0.001 32.00 0.7059424 0.8818421 0.5277085  1 0.005 0.25 0.7751826 0.8625192 0.5884955  1 0.005 0.50 0.7302206 0.8748949 0.5550970  1 0.005 1.00 0.7021792 0.8831516 0.5319944  1 0.005 2.00 0.6974319 0.8843542 0.5270265  1 0.005 4.00 0.7019578 0.8829595 0.5263159  1 0.005 8.00 0.7096212 0.8807520 0.5301593  1 0.005 16.00 0.7243731 0.8760338 0.5377313  1 0.005 32.00 0.7337179 0.8728955 0.5428281  1 0.010 0.25 0.7302433 0.8748873 0.5551126  1 0.010 0.50 0.7021596 0.8831565 0.5319791  1 0.010 1.00 0.6974200 0.8843556 0.5270316  1 0.010 2.00 0.7020198 0.8829434 0.5263865  1 0.010 4.00 0.7096941 0.8807390 0.5301754  1 0.010 8.00 0.7247330 0.8759239 0.5378933  1 0.010 16.00 0.7338415 0.8728477 0.5428373  1 0.010 32.00 0.7429545 0.8701441 0.5472151  2 0.001 0.25 0.8779985 0.8404121 0.6596468  2 0.001 0.50 0.7745472 0.8655917 0.5856184  2 0.001 1.00 0.7098113 0.8830178 0.5373961  2 0.001 2.00 0.6666038 0.8952678 0.5002856  2 0.001 4.00 0.6417386 0.9021108 0.4775214  2 0.001 8.00 0.6266796 0.9062763 0.4610715  2 0.001 16.00 0.6289768 0.9054599 0.4587719  2 0.001 32.00 0.6330373 0.9048566 0.4614501  2 0.005 0.25 0.6565240 0.8988443 0.4916291  2 0.005 0.50 0.6354794 0.9037161 0.4717091  2 0.005 1.00 0.6212316 0.9073044 0.4543243  2 0.005 2.00 0.6194414 0.9082701 0.4520047  2 0.005 4.00 0.6177324 0.9092338 0.4487707  2 0.005 8.00 0.6197750 0.9089279 0.4512670  2 0.005 16.00 0.6247663 0.9079007 0.4568642  2 0.005 32.00 0.6294370 0.9069237 0.4613274  2 0.010 0.25 0.6293745 0.9049443 0.4635494  2 0.010 0.50 0.6226344 0.9069243 0.4557601  2 0.010 1.00 0.6148168 0.9095818 0.4491489  2 0.010 2.00 0.6146942 0.9099031 0.4497460  2 0.010 4.00 0.6175615 0.9094128 0.4530769  2 0.010 8.00 0.6207772 0.9088222 0.4563315  2 0.010 16.00 0.6228678 0.9086508 0.4579929  2 0.010 32.00 0.6282422 0.9072207 0.4635977  RMSE was used to select the optimal model using the smallest value.  The final values used for the model were degree = 2, scale = 0.01 and C = 2.  proc.time() - ptm  plot(svmPTune,  scales = list(x = list(log = 2),  between = list(x = .5, y = 1)))    ##save the predicted values into testResults  testResults$SVMp <- predict(svmPTune, solTestXtrans) |

KNN

|  |
| --- |
| ### K-Nearest Neighbors  ### First we remove near-zero variance predictors  knnDescr <- solTrainXtrans[, -nearZeroVar(solTrainXtrans)]  ptm <- proc.time() # takes 39.86 seconds to run  set.seed(100)  knnTune <- train(x = knnDescr, y = solTrainY,  method = "knn",  preProc = c("center", "scale"),  tuneGrid = data.frame(k = 1:20),  trControl = ctrl)    knnTune  k-Nearest Neighbors  951 samples  225 predictors  Pre-processing: centered (225), scaled (225)  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results across tuning parameters:  k RMSE Rsquared MAE  1 1.197488 0.6864408 0.8844173  2 1.108861 0.7149867 0.7985735  3 1.052025 0.7393821 0.7788221  4 1.044642 0.7397692 0.7822507  5 1.029508 0.7460239 0.7723175  6 1.049528 0.7356698 0.7902894  7 1.052394 0.7341858 0.7913727  8 1.053536 0.7339818 0.7952906  9 1.061929 0.7306908 0.8048726  10 1.068142 0.7279470 0.8139883  11 1.067823 0.7284678 0.8154235  12 1.075813 0.7243982 0.8262795  13 1.080953 0.7220706 0.8276790  14 1.096012 0.7139975 0.8393113  15 1.103495 0.7106689 0.8469115  16 1.112635 0.7059419 0.8565096  17 1.117501 0.7041408 0.8613976  18 1.121126 0.7021762 0.8647754  19 1.129509 0.6978970 0.8701995  20 1.135813 0.6951420 0.8759220  RMSE was used to select the optimal model using the smallest value.  The final value used for the model was k = 5.  proc.time() - ptm  plot(knnTune)    testResults$Knn <- predict(knnTune, solTestXtrans[, names(knnDescr)]) |

Print out the predicted values for the response variable based on different models

|  |
| --- |
| #print out the predicted values based on different models  head(testResults)  obs NNet  20 0.93 0.6531965  21 0.85 0.7620576  23 0.81 0.2121431  25 0.74 1.1017603  28 0.61 -0.4494221  31 0.58 1.1281280 |

Performance comparison of the four nonlinear models

|  |
| --- |
| #Performance of nonlinear models  set.seed(100)  Nnet.pred = predict(nnetTune, solTestXtrans)  MARS.pred <- predict(marsTune, solTestXtrans)  SVMr.pred <- predict(svmRTune, solTestXtrans)  SVMp.pred <- predict(svmPTune, solTestXtrans)  Knn.pred <- predict(knnTune, solTestXtrans[, names(knnDescr)])  data.frame(rbind(NNET=postResample(pred=Nnet.pred,obs = solTestY),  MARS=postResample(pred=MARS.pred ,obs = solTestY),  SVMr=postResample(pred=SVMr.pred,obs = solTestY),  SVMp=postResample(pred=SVMp.pred,obs = solTestY),  KNN=postResample(pred=Knn.pred,obs = solTestY) ))  RMSE Rsquared MAE  NNET 0.7254278 0.8801212 0.5313776  MARS 0.7311925 0.8767131 0.5496563  SVMr 0.6073453 0.9148340 0.4536504  SVMp 0.6039573 0.9158389 0.4486317  KNN 1.0782867 0.7336572 0.8115053 |