#Required Packages

|  |
| --- |
| library(AppliedPredictiveModeling)  library(lattice)  library(caret)  library(corrplot)  library(e1071)  library(pls)  library(elasticnet)  library(earth)  library(pls)  library(tree)  library(rpart)  library(partykit)  library(Cubist) |

#Access Data

|  |
| --- |
| #Access the solubility data  data(solubility)  #http://127.0.0.1:27533/library/AppliedPredictiveModeling/html/solubility.html  #This data contain traing data  str(solTrainY)  dim(solTrainX)  ##Test data  head(solTestY)  head(solTestX) |

#Modeling Building (Use the same train control for comparison purposes.

|  |
| --- |
| set.seed(100)  #Create a series of test/training partitions  #default is 10, the funtion below creates 10 folder  indx <- createFolds(solTrainY, returnTrain = TRUE)  #control the computational nuances of the train function  ctrl <- trainControl(method = "cv", index = indx) |

#################################################

### Chapter 6 Linear Regression and Its Cousins

#################################################

|  |
| --- |
| ##############################  ### Linear Regression  ##############################  set.seed(100)  lmTune0 <- train(x = solTrainXtrans, y = solTrainY,  method = "lm", preProc = c("center", "scale"),  trControl = ctrl)  lmTune0 > lmTune0  Linear Regression  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:  RMSE Rsquared MAE  0.7170016 0.8792751 0.5298775  Tuning parameter 'intercept' was held constant at a value of TRUE  summary(lmTune0) #provide regression coefficients  ### Save predicted results of the test set in a data frame  testResults <- data.frame(obs = solTestY, LM= predict(lmTune0, solTestXtrans))  ##############################  ### Partial Least Squares (PLS)  ##############################  #library(pls)  set.seed(100)  plsTune <- train(x = solTrainXtrans, y = solTrainY,  method = "pls", reProc = c("center", "scale"),  tuneGrid = expand.grid(ncomp = 1:50),  trControl = ctrl)  plsTune  plot(plsTune)  ### Save the test set results into testResults as follows  testResults$PLS =predict(plsTune, solTestXtrans)  ##############################  ### Principal Component Regression (PCR)  ##############################  set.seed(100)  pcrTune <- train(x = solTrainXtrans, y = solTrainY,  method = "pcr", preProc = c("center", "scale"),  tuneGrid = expand.grid(ncomp = 1:50),  trControl = ctrl)  pcrTune  plot(pcrTune)  ### Save the test set results into testResults as follows  testResults$PCR =predict(pcrTune, solTestXtrans)  ##############################  ### Penalized Models  ##############################  ### Ridge Regression  #you may need to try different ranges of values for lambda  ridgeGrid <- expand.grid(lambda = seq(0, .1, length = 10))  #library(elasticnet)  set.seed(100)  ridgeTune <- train(x = solTrainXtrans, y = solTrainY,  method = "ridge",  tuneGrid = ridgeGrid,  trControl = ctrl,  preProc = c("center", "scale"))  plot(ridgeTune)  #prediction for test data  testResults$Ridge <- predict(ridgeTune, solTestXtrans)  ### ENET  #you may need to try different ranges of values for lambda1 and lambda2  enetGrid <- expand.grid(lambda = c(0, 0.01, .1),  fraction = seq(.05, 1, length = 20))  set.seed(100)  enetTune <- train(x = solTrainXtrans, y = solTrainY,  method = "enet",  tuneGrid = enetGrid,  trControl = ctrl,  preProc = c("center", "scale"))  plot(enetTune)    #prediction for test data  testResults$ENET <- predict(enetTune, solTestXtrans) |

#################################################

### Chapter 7 Nonlinear Regression Models

#################################################

|  |
| --- |
| ##############################  ### 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)  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  plot(nnetTune)    #prediction for test data  testResults$NNet <- predict(nnetTune, solTestXtrans)  ##############################  ### Multivariate Adaptive Regression Splines (MARS)  ##############################  set.seed(100)  marsTune <- train(x = solTrainXtrans, y = solTrainY,  method = "earth", preProc = c("center", "scale"),  tuneGrid = expand.grid(degree = 1, nprune = 2:38),  trControl = ctrl)  marsTune  plot(marsTune)    #save the predicted values into testResults  testResults$MARS <- predict(marsTune, solTestXtrans)  ##############################  ### Support Vector Machines (SVM) with Radial Kernel  ##############################  # 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  # You may use other kernels  set.seed(100)  svmRTune <- train(x = solTrainXtrans, y = solTrainY,  method = "svmRadial",  preProc = c("center", "scale"),  tuneLength = 14,  trControl = ctrl)  svmRTune  plot(svmRTune, scales = list(x = list(log = 2)))    ##save the predicted values into testResults  testResults$SVMr <- predict(svmRTune, solTestXtrans)  ##############################  ### Support Vector Machines (SVM) with polynomial basis function kernel  ##############################  svmGrid <- expand.grid(degree = 1:2,  scale = c(0.01, 0.005, 0.001),  C = 2^(-2:5))  set.seed(100)  svmPTune <- train(x = solTrainXtrans, y = solTrainY,  method = "svmPoly",  preProc = c("center", "scale"),  tuneGrid = svmGrid,  trControl = ctrl)  svmPTune  plot(svmPTune,  scales = list(x = list(log = 2),  between = list(x = .5, y = 1)))    ##save the predicted values into testResults  testResults$SVMp <- predict(svmPTune, solTestXtrans)  ##############################  ### K-Nearest Neighbors  ##############################  ### First we remove near-zero variance predictors  knnDescr <- solTrainXtrans[, -nearZeroVar(solTrainXtrans)]  set.seed(100)  knnTune <- train(x = knnDescr, y = solTrainY,  method = "knn",  preProc = c("center", "scale"),  tuneGrid = data.frame(k = 1:20),  trControl = ctrl)  knnTune  plot(knnTune)    ##save the predicted values into testResults  testResults$Knn <- predict(knnTune, solTestXtrans[, names(knnDescr)]) |

#################################################

### Chapter 8 Nonlinear Regression Models

|  |
| --- |
| ################################################# ##############################  ### Basic Regression Trees  ##############################  set.seed(100)  cartTune <- train(x = solTrainXtrans, y = solTrainY,  method = "rpart",  tuneLength = 25,  trControl = ctrl)  cartTune  ### Plot the tuning results  ### Cross-validated RMSE profile for the regression tree  plot(cartTune, scales = list(x = list(log = 10)))  ### Save the test set results in a data frame  testResults$CART = predict(cartTune, solTestXtrans)    ##############################  ### Bagged Trees  ##############################  set.seed(100)  treebagTune <- train(x = solTrainXtrans, y = solTrainY,  method = "treebag",  nbagg = 50,  trControl = ctrl)  treebagTune > treebagTune  Bagged CART  951 samples  228 predictors  No pre-processing  Resampling: Cross-Validated (10 fold)  Summary of sample sizes: 856, 855, 857, 856, 856, 855, ...  Resampling results:  RMSE Rsquared MAE  0.904804 0.8077096 0.6883977  ### Save the test set results in a data frame  testResults$Bagged <- predict(treebagTune, solTestXtrans)  ##############################  ### Boosting  ##############################  gbmGrid = expand.grid( interaction.depth = seq( 1, 7, by=2 ),  n.trees = seq( 100, 1000, by=100 ),  shrinkage = c(0.01, 0.1),  n.minobsinnode = 10 )  set.seed(100)  gbmTune <- train(x = solTrainXtrans, y = solTrainY,  method = "gbm",  tuneGrid = gbmGrid,  trControl = ctrl,  verbose = FALSE)  gbmTune  plot(gbmTune, auto.key = list(columns = 4, lines = TRUE))    ### Save the test set results in a data frame  testResults$Boosting <- predict(gbmTune, solTestXtrans)  ##############################  ### Random Forests  ##############################  mtryGrid <- data.frame(mtry = floor(seq(10, ncol(solTrainXtrans), length = 10)))  ### Tune the model using cross-validation  set.seed(100)  rfTune <- train(x = solTrainXtrans, y = solTrainY,  method = "rf",  tuneGrid = mtryGrid,  ntree = 200,  importance = TRUE,  trControl = ctrl)  rfTune  plot(rfTune)    ### Save the test set results in a data frame  testResults$RF <- predict(rfTune, solTestXtrans)  ##############################  ### use the model using the OOB estimates  ##############################  ctrlOOB <- trainControl(method = "oob")  set.seed(100)  rfTuneOOB <- train(x = solTrainXtrans, y = solTrainY,  method = "rf",  tuneGrid = mtryGrid,  ntree = 200,  importance = TRUE,  trControl = ctrlOOB)  rfTuneOOB  plot(rfTuneOOB)    ### Save the test set results in a data frame  testResults$RFOOB <- predict(rfTuneOOB, solTestXtrans)  ##############################  ### Cubist  ##############################  #library(Cubist)  cbGrid <- expand.grid(committees = c(1:10, 20, 50, 75, 100),  neighbors = c(0, 1, 5, 9))  set.seed(100)  cubistTune <- train(solTrainXtrans, solTrainY,  "cubist",  tuneGrid = cbGrid,  trControl = ctrl)  cubistTune  plot(cubistTune, auto.key = list(columns = 4, lines = TRUE))    ### Save the test set results in a data frame  testResults$Cubist <- predict(cubistTune, solTestXtrans) |

# Model Comparisons

|  |
| --- |
| #Prediction based on linear models from Chapter 6  LM.pred = predict(lmTune0, solTestXtrans)  PLS.pred <- predict(plsTune, solTestXtrans)  PCR.pred <- predict(pcrTune , solTestXtrans)  Ridge.pred <- predict(ridgeTune, solTestXtrans)  ENET.pred <- predict(enetTune, solTestXtrans)  #Prediction based on nonlinear models from Chapter 7  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)]) #drop nearzerovariance  ### Prediction based on tree-based models from Chapter 8  set.seed(100)  Cart.pred <- predict(cartTune, solTestXtrans)  Bagged.pred <- predict(treebagTune, solTestXtrans)  Boosting.pred <- predict(gbmTune, solTestXtrans)  RF.pred <- predict(rfTune, solTestXtrans)  RFOOB.pred <- predict(rfTuneOOB, solTestXtrans)  Cubist.pred <- predict(cubistTune, solTestXtrans)  #Model comparisons with respect to RMSE, Rsquared, and MAE  #Which model has the best predictive ability?  Results = data.frame(rbind(LM=postResample(pred=LM.pred,obs = solTestY),  PLS=postResample(pred=PLS.pred,obs = solTestY),  PCR=postResample(pred=PCR.pred ,obs = solTestY),  Ridge=postResample(pred=Ridge.pred,obs = solTestY),  ENET=postResample(pred=ENET.pred,obs = solTestY),  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),  CART=postResample(pred=Cart.pred,obs = solTestY),  Bagged=postResample(pred=Bagged.pred,obs = solTestY),  Boosting=postResample(pred=Boosting.pred,obs = solTestY),  RF=postResample(pred=RF.pred,obs = solTestY),  RFOOB=postResample(pred=RFOOB.pred,obs = solTestY),  Cubist=postResample(pred=Cubist.pred,obs = solTestY) ))  Results  > Results  RMSE Rsquared MAE  LM 0.7455802 0.8722236 0.5497605  PLS 0.7411940 0.8733263 0.5496657  PCR 0.8761147 0.8221764 0.6687375  Ridge 0.7215616 0.8801309 0.5369285  ENET 0.7072226 0.8841315 0.5300267  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  CART 0.8644157 0.8277231 0.6659295  Bagged 0.8500198 0.8355628 0.6188157  Boosting 0.6215909 0.9105978 0.4299896  RF 0.6535905 0.9012634 0.4645375  RFOOB 0.6360412 0.9067003 0.4521137  Cubist 0.6059198 0.9148408 0.4349645  >  #We can draw barplot to visulize the predictive performance in terms of R-squared and RMSE  #Sort Results with respect to Rsquared in ascending order  NewResults1 = Results[order(Results$Rsquared),]  Rsquared = NewResults1$Rsquared  names(Rsquared)= rownames(NewResults1)  barplot(Rsquared,las=2,cex.names=0.8, xlab="Models", ylab="Rsquared",  main="Barplot of the predictive performance based on R-squared")    ############  #In terms of R-squared, SVM with polynomial basis function kernel performs the best.  ############  #Sort Results with respect to RMSE in ascending order  NewResults2 = Results[order(Results$RMSE, decreasing = T),]  RMSE = NewResults2$RMSE  names(RMSE)= rownames(NewResults2)  barplot(RMSE,las=2,cex.names=0.8, xlab="Models", ylab="RMSE", col='blue',  main="Barplot of the predictive performance based on RMSE")    ############  #In terms of RMSE, SVM with polynomial basis function kernel performs the best.  #KNN performs the worst.  ############ |