Bios664 HW1

Meng-Ni Ho

2/11/2019

For this homework, use the R code in the file "simple classification.R" to generate a set of training data.

Generate simulated training data

```
set.seed(40)
generate_data = function(){
  f<-function(x){
    return(0.2 + x - 0.5*x^2 + 0.1*x^3 - 0.5*x^4)
  }
  #xv = seq(0,1,0.001)
  #yv = f(xv)

  dx = runif(500)
  dy = runif(500) # true y

  boundry = f(dx) # estimated y
  label = (dy>boundry)+0
  x_value = dx
  y_value = dy + rnorm(length(dy),sd=0.1)
  training_data = cbind(y_value, x_value, label)
  return(training_data)

}

training_data = generate_data()
```

Question 1: linear classifier

Implement a bootstrap method to estimate the prediction error (EPE) of the linear classifier that we used in the class and compare it to the K-fold cross-validation results for K = 2, 5 and 10.

1. Validate with K-fold cross validation

```
# `validate_batch`: function for validate a single batch---------
  # data = training data
  # batch = rows from training data that are set for validate (answer)
  # test_batch_id = rows from training data that are set to validate (test)
  # procedure: split the training set to validate and testing, perform probit regression with validate


validate_batch = function(data, batch, test_batch_id){
  # validate set
  d = data[batch!=test_batch_id,]

  # fit the probit regression
```

```
  y = d[,1]
  x = d[,2]
  l = d[,3]

  fit = glm(l~y+x,family=binomial(link="probit"))
  beta = matrix(ncol=1, fit$coef)

  # testing set (do not need label)
  test_d = data[batch==test_batch_id,1:2]

  if(!is.null(dim(test_d))){
    test_d = matrix(ncol=3,cbind(rep(1,dim(test_d)[1]), test_d))
  }else{
    test_d = matrix(ncol=3,c(1,test_d))
  }

  # output fitted values
  pred = test_d%*%beta
  pred_label= (pred>=0)+0
  true_label = data[batch==test_batch_id,3]
  # compare fitted label with true label,
  # return error (counts of predicted label apart from true label)
  return(length(which(true_label!=pred_label)))
}


#`K_fold_CV`: perform `validate_batch()` and output prediction error
K_fold_CV = function(data, K){
  N = dim(data)[1] #nrow
  batch = rep(1:K, ceiling(N/K))[1:N]
  # total error across each fold
  total_error = sum(sapply(1:K, function(x) validate_batch(data,batch,x)))
  # prediction error
  EPE = total_error/N
  return(EPE)
}
```

EPE when 2 fold:

```
K_fold_CV(training_data,2)
```

```
## [1] 0.096
```

EPE when 5 fole:

```
K_fold_CV(training_data,5)
```

```
## [1] 0.09
```

EPE when 10 fold:

```
K_fold_CV(training_data,10)
```

```
## [1] 0.096
```

2. Validate with Bootstrap: using `boot()` in `boot` package

```r
# `get_epe`: get the prediction error from bootstrap
#   data = training_data
#   indices: number of sampling time
get_epe = function(data, indices) {

  N = dim(data)[1]
  d = data[indices,]

  # fit the probit regression
  y = d[,1]
  x = d[,2]
  l = d[,3]
  fit = glm(l~y+x,family=binomial(link="probit"))
  pred = predict(fit, type = 'link')
  pred_label= (pred>=0)+0
  total_error = length(which(l!=pred_label))
  EPE = total_error/N
  return(EPE)
}
```

Resample 20 times:

```r
bootstrap20 = boot(data = training_data, statistic = get_epe, R = 20)
# bootstrap20$t: ERE for each replicates
mean(bootstrap20$t)
```

```
## [1] 0.0891
```

Original EPE:

```r
# the observed value of ERE applied to data.
bootstrap20$t0
```

```
## [1] 0.092
```

Resample 50 times

```r
bootstrap50 = boot(data = training_data, statistic = get_epe, R = 20)
mean(bootstrap50$t)
```

```
## [1] 0.0915
```

Resample 100 times

```r
bootstrap100 = boot(data = training_data, statistic = get_epe, R = 20)
mean(bootstrap100$t)
```

```
## [1] 0.0859
```

Question 2: knn classifier

(a) Implement a cross-validation scheme select the tuning parameter k, i.e., the "optimal" number of nearest neighbors.
(b) Estimate the EPE for your optimal k using the training data
(c) Simulate new data according to the true generative model and re-estimate the EPE for the estimated optimal k.

```r
# Randomly shuffle the data
training_data = training_data[sample(nrow(training_data)),]

# perform knn
vote = function(test, K, train){
  dist = apply(train,1, function(x) (x[1]-test[1])^2+(x[2]-test[2])^2)
  # find the first k-ranked points
  index = which(rank(dist)<=K)
  rst = 1
  if(sum(train[index,3])<K/2){
    rst = 0
  }
  return(rst)
}

# Create 10 equally size folds-------------
folds  =  cut(seq(1,nrow(training_data)),breaks=10,labels=FALSE)

# Perform knn with 10 fold cross validation---------
knn_epe = function(k){
  for(j in 1:10){
    # Segement your data by fold using the which() function

    testIndexes = which(folds == j,arr.ind=TRUE)
    testData = training_data[testIndexes, ]
    trainData = training_data[-testIndexes, ]
    N = dim(testData)[1]
    est_rst = apply(testData, 1, function(x) vote(x, trainData, K=k))
    error = length(which(testData[,3]!=est_rst))
    EPE = error/N
  }
  return(EPE)
}

# testing k from 1~20, each using 10 fold cross-validation
result = vector("numeric", 20)
for (i in 1:20){
  epe = knn_epe(i)
  result[i] = epe
}
```

Find the optimal K:

```r
# find the k with smallest epe
which(result == min(result))
```

```
## [1] 15 17
```

Use the optimal K to generate new estimation and compute EPE:

```r
est_rst = apply(training_data, 1, function(x) vote(x, training_data, K=13))
error = length(which(training_data[,3]!=est_rst))
EPE = error/dim(training_data)[1] # 0.065
EPE
```

```
## [1] 0.074
```

Question 3: caret

Find an online tutorial on the R package "caret", study the relevant features and usages of the package (a) Use caret package to determine the optimal k value for the simple classification example. (b) Compare the knn classifier to the naive Bayes classifier implemented in the caret package. Given a brief summary on your conclusions.

```
# split  to training and testing dataset
train = training_data[1:350,]
test = training_data[351:500,]
```

1. knn classifier

```
knnFit = train(label ~ .,
                data = train,
                method = "knn",
                trControl = trainControl(method="cv", number = 10),
                preProcess = c("center","scale"),
                tuneLength = 20)
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to do
## classification? If so, use a 2 level factor as your outcome column.
```

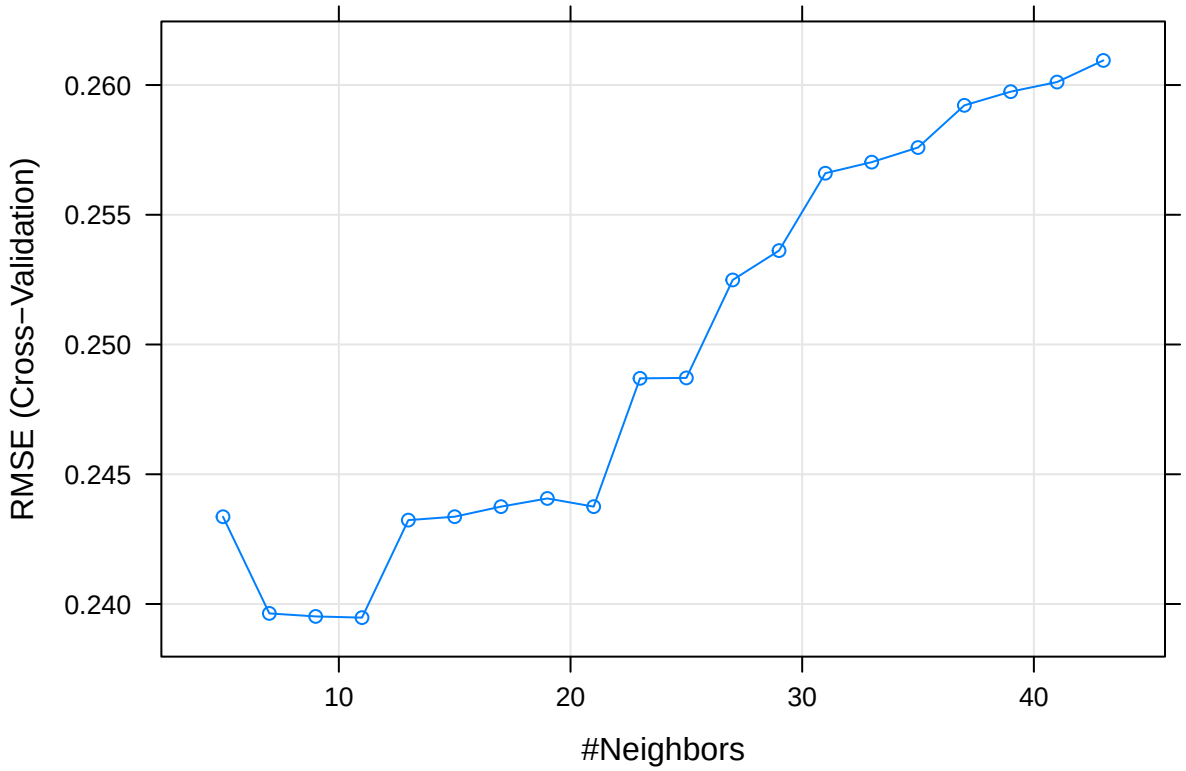
optimal K:

```
knnFit
```

```
## k-Nearest Neighbors
##
## 350 samples
##   2 predictor
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 315, 315, 315, 315, 315, 315, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  0.2433629  0.7579558  0.1108571
##    7  0.2396368  0.7646897  0.1122449
##    9  0.2395206  0.7633903  0.1140952
##   11  0.2394745  0.7658541  0.1184416
##   13  0.2432324  0.7597392  0.1226374
##   15  0.2433644  0.7618852  0.1257143
##   17  0.2437522  0.7600058  0.1285714
##   19  0.2440675  0.7587814  0.1303684
##   21  0.2437519  0.7606205  0.1330427
##   23  0.2486962  0.7523635  0.1400000
##   25  0.2487124  0.7530607  0.1414857
##   27  0.2524879  0.7460882  0.1465571
##   29  0.2536187  0.7448209  0.1487685
##   31  0.2566019  0.7402588  0.1527765
##   33  0.2570277  0.7403429  0.1543723
##   35  0.2575912  0.7405054  0.1562245
##   37  0.2592174  0.7388441  0.1588417
##   39  0.2597445  0.7383790  0.1604103
```

```
##   41  0.2601145  0.7376791  0.1615962
##   43  0.2609474  0.7366240  0.1632558
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 11.
```

```
plot(knnFit)
```



Accuracy:

```
# need to factor into same level (0, 1) in order to compute confusionMatrix
knnReal = factor(test[,3])
knnPred = predict(knnFit, test)
knnPred_label = factor((knnPred>0)+0)
confusionMatrix(knnPred_label, knnReal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40  0
##          1 21 89
##
##                Accuracy : 0.86
##                  95% CI : (0.794, 0.9112)
##     No Information Rate : 0.5933
##     P-Value [Acc > NIR] : 1.037e-12
##
##                   Kappa : 0.6933
##
##  Mcnemar's Test P-Value : 1.275e-05
```

```
##
##             Sensitivity : 0.6557
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8091
##              Prevalence : 0.4067
##          Detection Rate : 0.2667
##    Detection Prevalence : 0.2667
##       Balanced Accuracy : 0.8279
##
##        'Positive' Class : 0
##
```

2. Bayes classifier

```
# generate a Naive Bayes model, using 10-fold cross-validation: (method="cv", number = 10)
x = train[,1:2]
y = factor(train[,3])  # NaiveBayes is a classifier so convert y to factor
nbfit = train(x = x,
              y = y,
              method = "nb",
              trControl = trainControl(method="cv", number = 10))

nbReal = factor(test[,3])
nbpred = predict(nbfit, test)
```

Accuracy:

```
confusionMatrix(nbpred, nbReal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 51  5
##          1 10 84
##
##                Accuracy : 0.9
##                  95% CI : (0.8404, 0.9429)
##     No Information Rate : 0.5933
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7901
##
##  Mcnemar's Test P-Value : 0.3017
##
##             Sensitivity : 0.8361
##             Specificity : 0.9438
##          Pos Pred Value : 0.9107
##          Neg Pred Value : 0.8936
##              Prevalence : 0.4067
##          Detection Rate : 0.3400
##    Detection Prevalence : 0.3733
##       Balanced Accuracy : 0.8899
##
##        'Positive' Class : 0
```

##

By comparing accuracy, it seems that Bayes classifier has a higher accuracy than knn classifier.