Stats 201B: Statistical Modeling and Learning
Problem Set 5

Mar 3, 2016

Due 10 March

Question 1. Random Forest (20 points)

1. *(4 pts.)* In a few sentences, describe how "random forest" works. What parameter(s) would you tune to control the complexity? In your answer, explain how random forest addresses the problem of high-variance that we would encounter with a single tree model.

   Random forests generate B trees. For each tree, it draws a bootstrap sample of size N. Then it grows a tree to the bootstrap data by recursively repeating 3 steps. First, select m variables from p variables. Second, pick the best variable/ split-point among m. Third, split the node into two nodes. To predict, choose the majority votes for classification and take average for regression.
   I would tune the number of trees B, the number of selected variable, p with respect to m. By taking average of prediction from B trees, we reduce the variance. But if these trees are correlated, the convergent variance has positive relation with ρ. So random forests use bootstrapping to lower ρ without increasing the tree variance too much.

2. *(4 pts.)* On the course website, you'll find `Heart.csv`, a reduced and cleaned version of the South African heart dataset. The key outcome variable is "AHD", which is an indicator of heart disease. The other variables are measures thought to predict heart disease.

   Choose a random sample of 150 observations to be your training dataset, and hold-out the rest as test data. Using the `randomForest` package, make a model to predict `AHD` using the other variables in the dataset. Use `mtry`=3 and `ntree=1000`.

```
library(randomForest)
data = read.csv(file="E:\\Papers\\Statistics\\201B Statistical
 Modeling and Learning\\pset5\\Heart.csv", header = T)
#Divide dataset into training set and testing set
Index = sample(nrow(data), 150)
```

```
TrainSet = data[Index,]
TestSet = data[-Index,]
Training = randomForest(AHD~., data=TrainSet, mtry=3,         #Training
                        ntree=1000, na.action = na.omit)
Testing = predict(Training, newdata=TestSet, type="response",
                                na.action = na.omit)          #Testing
```

The last command will predict the results of the testing set.

3. *(4 pts.)* Compute the test error on the provided test set, and report it along side the "out-of-bag" error for your model. What does the out-of-bag error mean, and do you think it is a good measure of generalization error?

```
logic = Testing==TestSet[,"AHD"]
True = length(which(logic == T))
False = length(which(logic == F))
accuracy = True / (True+False)
Error = 1 - accuracy
> Error
[1] 0.2171053
ActualValue = data[as.numeric(names(Training$predicted)),"AHD"]
oobError = sum(ActualValue!=Training$predicted) / length(Training$predicted)
> oobError
[1] 0.137931
```

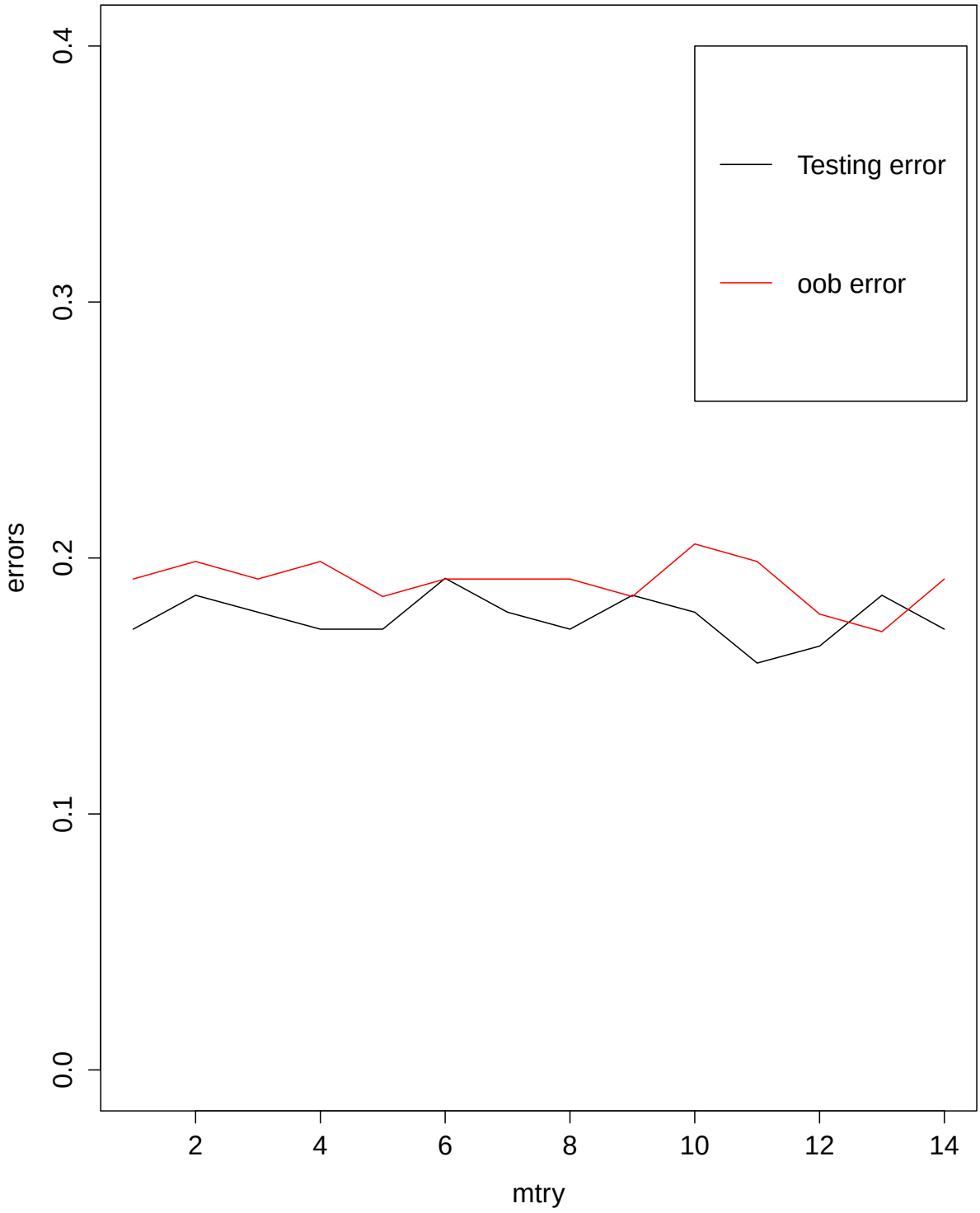
The testing error is 0.2171053 and out of bag erro is 0.137931. For each observation, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which the observation did not appear. Then calculate the observation error for this random forests. Finally take average of all errors. It is a good choice. The concept is samiliar with cross validation. The observation and the random forests are independent as long as all observations are independent.

4. *(4 pts.)* We now wish to see the effect of varying `mtry` on the out-of-bag and the test error. Write a function that will take your data and a value of `mtry` as arguments, and report back the out-of-bag error with $ntree = 500$ each time. Run the function using `mtry` ranging from 1 to the number of variables in the dataset. Plot the resulting out-of-bag and test-error (on the same plot) as functions of *mtry*.

```
RFplot <- function(data, ntry)
{
  Index = sample(nrow(data), 150)
```

```
  TrainSet = data[Index,]
  TestSet = data[-Index,]
  Test.error = c()
  oob.error = c()
  for(m in 1:ntry)
  {
    Training = randomForest(AHD~., data=TrainSet, mtry=3,
                            ntree=500, na.action = na.omit)
    Testing = predict(Training, newdata=TestSet, type="response",
                      na.action = na.omit)

    logic = Testing==TestSet[,"AHD"]
    True = length(which(logic == T))
    False = length(which(logic == F))
    accuracy = True / (True+False)
    Error = 1 - accuracy
    Test.error = c(Test.error, Error)
    ActualValue = data[as.numeric(names(Training$predicted)),"AHD"]
    oobError = sum(ActualValue!=Training$predicted) / length(Training$predicted)
    oob.error = c(oob.error, oobError)
  }
  plot(1:ntry, Test.error, type="l", col=1, ylim=c(0,0.4), ylab="errors",
       xlab="ntry")
  points(1:ntry, oob.error, type="l", col=2)
  legend(x= 10, y=0.4, lty=c(1,1), col=c(1,2),
         legend=c("Testing error", "oob error"))
}
RFplot(data, ncol(data))
```



From the graph, testing errors and oob errors are similar for every mtry. And the errors seeme to change with repesct to m. With smaller m, the correlation of two pairs of trees are smaller, so variance shinrks, and the errors are expected to be smaller. On the other side, with larger

numbero f variables, at each split, the chance can be small that the relevant variables will be selected. So I think the trade-off makes the lines in the plot are somehow horizontal.

5. *(4 pts.)* Finally, the lasso logit performed fairly well on this task in class. Compare the test error from the original random forest model (i.e. with `mtry=3`) to the test error you get with a lasso logit model (use `glmnet`, choosing λ by cross-validation using `cv.glmnet`).

```
Rdata = data
for(i in 1:14)
  Rdata = Rdata[which(!is.na(Rdata[,i])),]
Index = sample(nrow(Rdata), 150)
TrainSet = Rdata[Index,]
TestSet = Rdata[-Index,]
library(glmnet)
dataMat = model.matrix(AHD~.-1, data = TrainSet)
testMat = model.matrix(AHD~.-1, data = TestSet)
Respond = rep(1,nrow(TrainSet))
Respond[which(as.matrix(TrainSet[,"AHD"]) == "No")] = 0
tRespond = rep(1,nrow(TestSet))
tRespond[which(as.matrix(TestSet[,"AHD"]) == "No")] = 0
cv = cv.glmnet(x=dataMat, y=Respond,family="binomial")
lambda = cv$lambda.min
Training = glmnet(x=dataMat, y=Respond, family = "binomial", lambda = lambda)
predict = as.numeric(predict(Training, newx = testMat, type="class"))
error = 1 - mean(predict == tRespond)
> error
[1] 0.2040816
```

The error rate is 0.2040816. Both random forests and Lasso logit have similar effects.

Question 2. Who will vote for a bill? (18 points)

.

In this problem, we will work with a sample of speech data drawn from the 2005 congressional record. Your task is to build a classifier that predicts whether or not a member of congress will vote for a bill, using their speeches. In the file "pset5words.RData", there is a matrix (`wordMatrix`) comprised of dummy variables for whether or not a speech fragment contains one of 5,262 of the most frequently used words in the corpus. If you prefer to create your own set of predictors (using bigrams, for example), the raw speech data is in the list object `speeches`.

The outcome variable, `vote` is a variable that takes a value of 1 if the member voted for the bill and a -1 if the member voted against the bill.

1. *(10 pts.)* Your job is to try at least *six classification techniques.* At least one must be a support vector machine, and at least one must be an ensemble over the other methods you try, such as a weighted or unweighted average. For each model, write one sentence or so describing how or why this is a reasonable modeling approach for this problem. It is okay if some of your models are not seemingly ideal for this problem, but in those cases explain why they might not be ideal.

   (a)
   ```
   #linear knernel
   tune.out.linear=tune(svm, vote~., data=train ,kernel="linear")
   best.svm.linear=tune.out.linear$best.model
   class.test.linear=predict(best.svm.linear, test)
   table(class.test.linear, test$vote)
   class.test.linear   0   1
                   0 199  45
                   1  31  25
   ```
   The result is based on SVM with linear kernel. SVM is effective for classifying data.

   (b)
   ```
   #Gaussian kernel
   tune.out.radial=tune(svm, vote~., data=train,kernel="radial")
   best.svm.radial=tune.out.radial$best.model
   class.test.radial=predict(best.svm.radial, test)
   table(class.test.radial, test$vote)
   class.test.radial   0   1
                   0 230  70
                   1   0   0
   ```
   SVM with Gaussian kernel classifies all data to 0. It does have higher accuracy than linear kernel, but the model is obviously overfitted. There are more than 5000 variables but 1000 observation (actually, I use 700 for training). So the size of data set cannot support SVM to find the band and estimate kernel parameter.

   (c)
   ```
   #polynomial kernel
   tune.out.polynomial=tune(svm, vote~., data=train,kernel="polynomial")
   best.svm.polynomial=tune.out.polynomial$best.model
   class.test.polynomial=predict(best.svm.polynomial, test)
   table(class.test.polynomial, test$vote)
   class.test.polynomial   0   1
                       0 230  70
                       1   0   0
   ```
   SVM with polynomial kernel have the same results with Gaussian kernel. The reason is same. So it is better not to use SVM when dataset is small. If we have no choice, applying linear kernel is conservative and recommended.

   (d)
   ```
   #LASSO logit
   ```

```
library(glmnet)
cv = cv.glmnet(x=trainMat[,-vote.index], y=trainMat[,vote.index, drop=F],
               family="binomial")
lambda = cv$lambda.min
Training = glmnet(x=trainMat[,-vote.index], y=trainMat[,vote.index, drop=F],
                  family = "binomial", lambda = lambda)
predict = as.numeric(predict(Training, newx = testMat[,-vote.index],
                             type="class"))
table(predict, test$vote)
predict   0   1
      0 233  58
      1   3   6
```

This result is based on Lasso-logit. Since the number of variables is larger than the number of observations, so ordinary logistic regression cannot be used. For this result, the error rate is significantly less than the one by SVM. Also it is not overfitted.

(e) I tried LDQ and QDA but failed. When using LDA, there are errors suggested some columns are constant within groups. And the variable space is too sparse to calculate the variance for QDA. So LDA and QDA are not useful for sparse data.

(f)
```
#naive Bayes
library(e1071)
tune.NB = tune(naiveBayes, vote~., data=train)
best.NB=tune.NB$best.model
class.test.NB =predict(best.NB, test)
class.test.NB   0   1
            0   0   0
            1 234  66
```

The result is based on Naive Bayes. This is really a poor result, because the size of data is small compared with the number of variables. I believe the data is too sparse to calculate a proper posterior distribution.

(g)
```
#random forests
Training = randomForest(vote~., data=train[,-(1:4000)], mtry=2500, ntree=1000)
Testing = predict(Training, newdata=test, type="response")
table(Testing, test$vote)
Testing   0   1
      0 212  48
      1  22  18
```

The result is based on random forests. Because the calculation is so huge that it takes me infite time to operate for the 5000 features, I choose 1000 of them. The accuracy is not as good as previous results. But the model is not overfitted, so random forests may

peroform well for sparse data.

(h)
```
#ensemble
ensemble = (as.numeric(class.test.linear) + as.numeric(class.test.radial) +
              (predict+1) + as.numeric(Testing)) / 4
ensemble[which(ensemble<=1.4)] = 0
ensemble[which(ensemble>=1.6)] = 1
ensemble[which(ensemble==1.5)] = rbinom(1,1,0.5)
ensemble = as.factor(ensemble)
table(ensemble, test$vote)
ensemble   0   1
     0   221  53
     1    13  13
```

This is the ensemble effect with the same weight, not including LDQ, QDA, which are not able to be applied, SVM with polynomial method, which is similar with Gaussian method and naive Bayes. It is prone to misclassify most 1 to 0.

2. *(4 pts.) Report a reasonable measure of the test error* for each model in a table. Be sure you do not "cheat" and report a test error that is susceptible to over-fitting. Comment on the behavior of your ensemble estimator relative to the best individual model.

SVM(linear)	SVM(Gaussian)	SVM(polynomial)	LASSO logit
0.2467	0.22	0.22	0.2033
random forests	ensemble	LDA/QDA	naive Bayes
0.2333	0.23	-	-

SVM and LASSO logit both overfit the model. I think random forests are the optimal although the test error is not the smallest. The ensemble error is close to the random forests error, but it misclassify most 1 to 0.

3. *(4 pts.)* Once you have good estimates of the test error for each model, comment on any differences you see, and whether there are any systematic reasons you can think of why some types of models may have worked better than others.

   To be honest, all the errors are similar. As for performance, random forests is optimal while other models are more or less overfitted. Most of observations are 0 and data is unbalanced. SVM may be affected by classifying most covariates' space as 0. Random forests can simply split the data into several nodes so may not be affected.