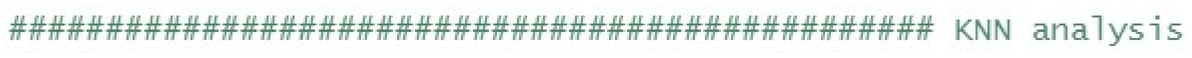
REPORT
Your report, which is submitted to Crowdmark, should answer these questions, as numbered below:

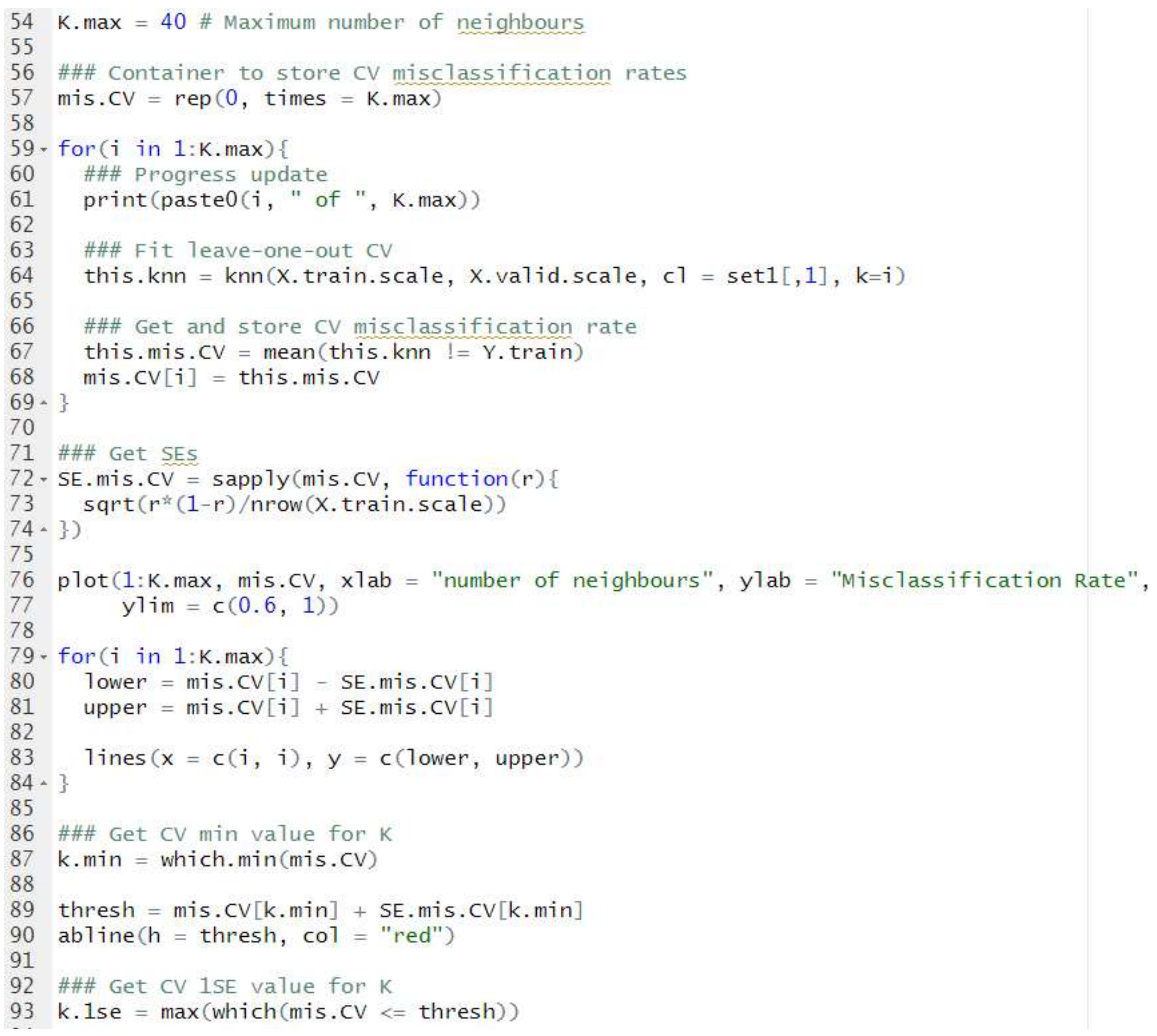
1. *What models or machines did you attempt to fit?* For each one, paste the R code from your program for the initial successful model fit. I want to see what you tried. For example, "fit1 = lm(y~., data=train)" is what I would list if I used multiple linear regression on all of the variables and my training data were called "train". **The answer should be a list (e.g., with bullets) of nothing but the code for each of these model fits.** Don't list code that did not run. If tuning was involved in the initial fitting process, you can can paste the function with variable names for the tuning parameters (e.g., your function might have "mtry=mm" if you looped over a variable called "mm").

A.



```r
################################################## KNN analysis
```

To get parameter tuning



```r
K.max = 40 # Maximum number of neighbours

### Container to store CV misclassification rates
mis.CV = rep(0, times = K.max)

for(i in 1:K.max){
  ### Progress update
  print(paste0(i, " of ", K.max))

  ### Fit leave-one-out CV
  this.knn = knn(X.train.scale, X.valid.scale, cl = set1[,1], k=i)

  ### Get and store CV misclassification rate
  this.mis.CV = mean(this.knn != Y.train)
  mis.CV[i] = this.mis.CV
}

### Get SEs
SE.mis.CV = sapply(mis.CV, function(r){
  sqrt(r*(1-r)/nrow(X.train.scale))
})

plot(1:K.max, mis.CV, xlab = "number of neighbours", ylab = "Misclassification Rate",
     ylim = c(0.6, 1))

for(i in 1:K.max){
  lower = mis.CV[i] - SE.mis.CV[i]
  upper = mis.CV[i] + SE.mis.CV[i]

  lines(x = c(i, i), y = c(lower, upper))
}

### Get CV min value for K
k.min = which.min(mis.CV)

thresh = mis.CV[k.min] + SE.mis.CV[k.min]
abline(h = thresh, col = "red")

### Get CV 1SE value for K
k.1se = max(which(mis.CV <= thresh))
```

```
knn.min = knn(X.train.scale, X.valid.scale, Y.train, k.min)
knn.1se = knn(X.train.scale, X.valid.scale, Y.train, k.1se)
```

B.

```
################################################# Logistic Rregression

fit.log.nnet = multinom(Y ~ ., data = data.train.scale, maxit = 200)
```

C.

```
################################################  LASSO version of logistic regression

fit.CV.lasso = cv.glmnet(X.train.scale, Y.train, family = "multinomial")
```

D.

```
#################################################   LDA

fit.lda = lda(X.train.DA, Y.train)
```

E.

```
#################################################   QDA

fit.qda = qda(X.train.DA, Y.train)
```

F.

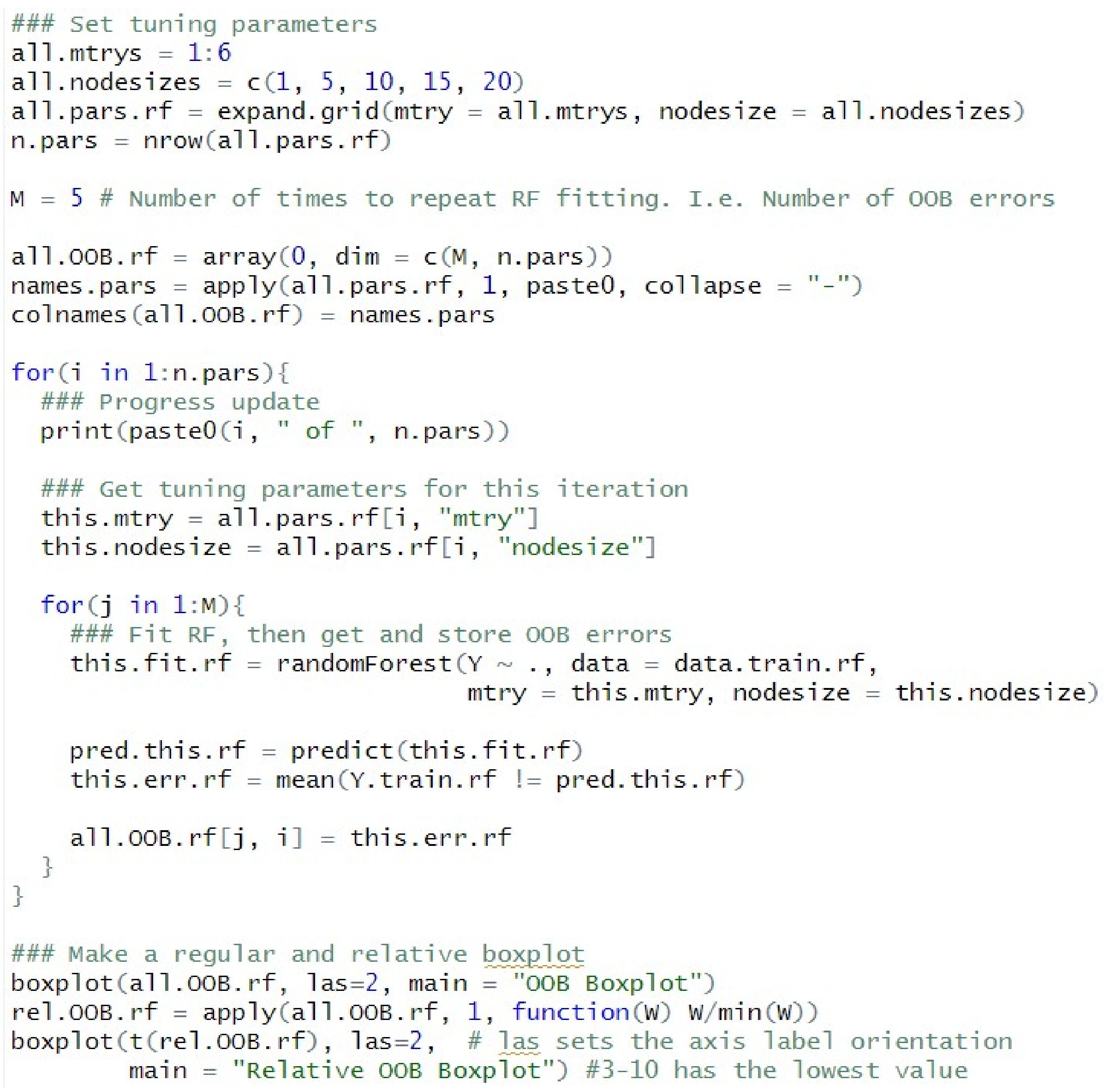
```
#################################################   Naive Bayes

fit.NB.userkernel = NaiveBayes(X.train, Y.train, usekernel = T)
fit.NB.notuserkernel = NaiveBayes(X.train, Y.train, usekernel = F)
```

G.

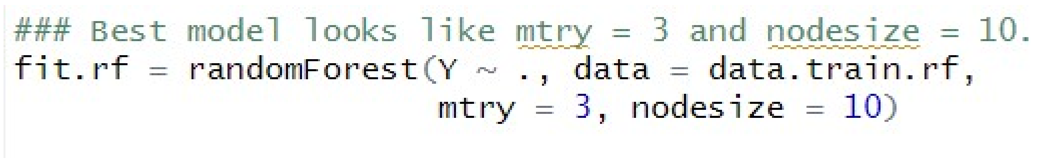
```
#################################################   Classification Trees

fit.tree.full = rpart(Y ~ ., data = data.train, method = "class",
                      cp = 0)
```

F.

```
#################################################   Random Forest
```



```r
### Set tuning parameters
all.mtrys = 1:6
all.nodesizes = c(1, 5, 10, 15, 20)
all.pars.rf = expand.grid(mtry = all.mtrys, nodesize = all.nodesizes)
n.pars = nrow(all.pars.rf)

M = 5 # Number of times to repeat RF fitting. I.e. Number of OOB errors

all.OOB.rf = array(0, dim = c(M, n.pars))
names.pars = apply(all.pars.rf, 1, paste0, collapse = "-")
colnames(all.OOB.rf) = names.pars

for(i in 1:n.pars){
  ### Progress update
  print(paste0(i, " of ", n.pars))

  ### Get tuning parameters for this iteration
  this.mtry = all.pars.rf[i, "mtry"]
  this.nodesize = all.pars.rf[i, "nodesize"]

  for(j in 1:M){
    ### Fit RF, then get and store OOB errors
    this.fit.rf = randomForest(Y ~ ., data = data.train.rf,
                               mtry = this.mtry, nodesize = this.nodesize)

    pred.this.rf = predict(this.fit.rf)
    this.err.rf = mean(Y.train.rf != pred.this.rf)

    all.OOB.rf[j, i] = this.err.rf
  }
}

### Make a regular and relative boxplot
boxplot(all.OOB.rf, las=2, main = "OOB Boxplot")
rel.OOB.rf = apply(all.OOB.rf, 1, function(W) W/min(W))
boxplot(t(rel.OOB.rf), las=2,  # las sets the axis label orientation
        main = "Relative OOB Boxplot") #3-10 has the lowest value
```



```r
### Best model looks like mtry = 3 and nodesize = 10.
fit.rf = randomForest(Y ~ ., data = data.train.rf,
                      mtry = 3, nodesize = 10)
```