STAT 675 – Homework 4
Due: Dec. 3, 2015

For this assignment, let's attempt to make a spam filter[1]. Usually, this would involve a lot of text processing on a huge number of emails. In this case, someone has created a feature matrix, $\mathbb{X}$, for us. $\mathbb{X}$ has rows given by individual emails and columns given by the number of each word or character that appears in that email, as well as three different numerical measures regarding capital letters (average length of consecutive capitals, longest sequence of consecutive capitals, and total number of capital letters). The response, Y, is given by the user supplied label marking that email as either spam ($Y = 1$) or not ($Y = 0$).

Here is a function that may be useful for this assignment:

```
miss.class =function(pred.class,true.class,produceOutput=FALSE){
  confusion.mat = table(pred.class,true.class)
  if(produceOutput){
    return(1-sum(diag(confusion.mat))/sum(confusion.mat))
  }
  else{
    print('miss-class')
    print(1-sum(diag(confusion.mat))/sum(confusion.mat))
    print('confusion mat')
    print(confusion.mat)
  }
}
# this can be called using:
#     (assuming you make the appropriately named test predictions)
miss.class(Y.hat,Y_0)
```

Read in the `R` data set `spam.Rdata` and read the documentation file `spambase.Documentation`. Note that I've flipped the training and test sets so this homework doesn't take too long to run.

```
train = !spam$train

test  = !train

X     = spam$XdataF[train,]
X_0   = spam$XdataF[!train,]
Y     = spam$Y[train]
Y_0   = spam$Y[!train]

#Split data as in 2-fold CV, but without the 'cross' part
cvTrain = sample(c(T,F),size=sum(train),replace=T,prob=c(.5,.5))
Xcv   = X[cvTrain,]
Xcv_0 = X[!cvTrain,]
Ycv   = Y[cvTrain]
```

[1] It has been estimated that spam (that is, unsolicited bulk) emails cost businesses $\approx$ \$10 billion a year

```
Ycv_0 = Y[!cvTrain]
```

For doing classification with the `neuralnet` package, set the `err.fct = 'ce'` for 'cross-entropy' and `linear.output = F`, to use the softmax function (which is the same as the sigmoid). Also, `neuralnet` requires a numeric response, so I've removed the `as.factor`.

1. Let's look at SVMs for the two class classification problem and compare it with neural networks. Let's write the neural network as

$$\begin{aligned} Z_k &= \sigma(\alpha_{0k} + \alpha_k^\top X) \\ W &= \beta_0 + \beta^\top Z \\ \pi(X) &= \frac{e^W}{1+e^W} \end{aligned}$$

   Now, we classify $\hat{Y} = 1$ if $\pi(X) > 1/2$, and $\hat{Y} = -1$ otherwise.

   Write down the form for the SVM (you don't need to write down the optimization problem, just how we can go from a hyperplane after transformation by a feature map to a classifier). Using the notation and terminology (such as activation function, number of hidden units, etc.) for neural networks, carefully show how the SVC can be seen as a particular instance of a neural network.

2. Let's look at the sensitivity with respect to the starting values. Run a simple neural network and examine the fitted values.

```
nRep = 3
model.out = as.formula(paste("Ycv ~ ",paste(names(Xcv),collapse='+')))
nn.out = neuralnet(model.out,data=Xcv, hidden=c(5), threshold=0.01,rep=nRep,err.fct='ce',
                         linear.output=F)
nn.out$net.result
```

   (a) Qualitatively, how different are the fitted values for each run?

   (b) Looking at the predictions on `Xcv_0`, what are the confusion matrices for each rep? What are the missclassification rates? Note, you need to form classifications out of the probabilities formed by

   ```
   compute(nn.out,Xcv_0,rep=?)$net.result
   ```

   (c) What is the confusion matrix when using the average of all the probabilites over the runs instead?

3. We need to specify the complexity of the NN. Unfortunately, `neuralnet` doesn't allow for penalization of the loss function directly, and hence we must use the `threshold` parameter for regularization. Note that the `nnet` package does, but is limited to single layer NNs. Form a large NN by choosing the number of hidden layers and hidden units. Then, we can select the `threshold` parameter using 2-fold CV (technically, we are just doing a single test/training

split instead of a full 2-fold CV. Feel free to do the full CV if you'd like). Be sure to average over starting values to produce the probability estimates that go into the classifications.

Since this involves a grid search, this problem is well suited to using parallelism over the cores on your computer. A good reference for parallel in R:

`http://journal.r-project.org/archive/2009-1/RJournal_2009-1_Knaus+et+al.pdf`

For this question, (i) describe to me the general idea of parallelism as it relates to this problem. (ii) give me the 2-fold CV estimate of the risk over the included grid (again, by 2-fold CV I mean the single validation set) (iii) report the threshold that minimizes that estimate of the risk.

```
cvF = function(threshold,Xcv,Ycv,Xcv_0,Ycv_0,nRep){
  model.out = as.formula(paste("Ycv ~ ",paste(names(Xcv),collapse='+')))
  #here, you need to choose the complexity of the neural net.
  nn.out    = neuralnet(model.out,data=Xcv, hidden=c(5,5), threshold=threshold,
                        rep=nRep,err.fct='ce',linear.output=F)
  nn.results = matrix(0,nrow=nrow(Xcv_0),ncol=nRep)
  for(reps in 1:nRep){
    nn.results[,reps] = compute(nn.out,Xcv_0,rep=reps)$net.result
  }
  probHat    = apply(nn.results,1,mean)
  YcvHat     = rep(0,nrow(Xcv_0))
  YcvHat[probHat > .5] = 1
  test.error = mean(YcvHat != Ycv_0)
  return(test.error)
}
library(snowfall)
#set cpus equal to number of cores on your processor
sfInit(restore = TRUE,parallel=TRUE, cpus=2, type="SOCK")

thresholdGrid =  10^c(-5,-3,-1,0,1)

sfLibrary(neuralnet)#this loads the package on each core

result <- sfLapply(thresholdGrid, cvF,Xcv,Ycv,Xcv_0,Ycv_0,nRep)
```

(note that if there are ties, pick the smaller threshold)

4. Using your conclusions from the above, fit a neural network to the training data and get prediction for the test data. Using your code from the previous homework assignment, refit an SVM with radial kernel (be sure to choose the parameters using CV with the function `tune`). Get the confusion matrices and missclassification rates for each method and compare them.