Stat 133, Spr 15
Homework 8: Nearest Neighbor Methods and Cross-Validation
Due: Friday April 24

For this assignment we will use some of the variables constructed in HW #5 to select a nearest neighbor method for predicting spam. You will use the k-nearest neighbor method to predict whether an email is spam or not spam. The R objects that you need for this assignment can be found in
http://www.stat.berkeley.edu/users/nolan/data/EmailsDist.rda.
This binary file contains the R objects:

- Matrix called `distEmails` which is 2412 by 2412 and holds all distances between the emails based on the distance between the vector of values for the variables derived for each email, e.g., percent yelling in the subject line, presence of Re: in the subject, etc.
- Vector of logicals called `isSpam` of length 2412 that indicates whether an email is spam or not.

Part 1.
Write a function called `predNeighbor` which has 3 input arguments:

- k - the number of neighbors to consult in predicting whether or not an email is spam.
- $distMat$ - an m by n matrix which holds the distances between one set of m emails (the test set) and another set of n emails (the training set). This means that the first row of the matrix contains the distances from the first email in the test set to each of the n emails in the training set.
- $indSpam$ - a logical vector of length n that indicates whether the email in the training set is spam or not.

The return value for this function is a numeric vector of length m which is the proportion of k nearest neighbors that are spam for each of the m emails in the test set. That is, your function will take the m-by-n distance matrix and for each row (which represents an email in the test set), choose the k nearest observations among the n emails (the columns of the matrix represent emails in the training set). These k emails then "vote" as to whether the email in the test set should be predicted to be spam or ham. The vote is cast using $indSpam$ (the true classification for the emails in the training set). The fraction of the k votes for spam is reported as the vote.

Functions that will be very helpful are `order()` and `apply()`. (Recall that `apply` is for matrices). Also, be sure to use your subsetting skills here.

Part 2.
Cross-validation provides a way to estimate the predictive ability of a model based on your data. You will use 10-fold cross validation in this assignment. In other words, you will split the data set into 10 parts at random so that they are as close to equal size as possible. For each tenth of the data, predict which points are spam based only the other nine-tenths of the data. Use the function you wrote in Part 1 to do this. Do this for values of k from 1 to 20.

Write a function called `cvKnn` that has input arguments:

- `distances` - a square matrix of distances between emails,
- `isSpam` - a logical vector indicating whether the email is spam,
- `k` - a vector of values to use in finding the k nearest neighbor, and
- `vfold` - a numeric value for the number of folds in v-fold cross validation, with a default value of 5.

This function should return a matrix with as many rows as in `distances` and as many columns as `k` values. We will call it with our `distEmails` matrix, `k = 1:20`, andvfold = 10. So the return value will be a 2412 by 20 matrix of the predictions for each email for k running from 1 to 20 (these predictions are actually proportions of spam among the nearest neighbors).

Hint: It's simplest to randomize row numbers rather than the matrix. That is, use the `sample` function to produce a random permutation of the indices from 1 to the number of rows in `distances`. Then you can use the first 1/10 of these as indices for the first test set, etc. It may be easiest to convert this random permutation into a matrix with 10 columns of indices, one for each fold.

Write a function called plotErrorRates to see how well the prediction is, you can compare it to the truth as follows. The input for this function is the return value from the call to `cvKnn` and a cutoff, e.g. `cutoff = 0.6`.

This function computes the error rates as follows. For a set of emails, if you have the truth for them in the logical vector $spamTruth$, and you have the prediction for them in the numeric vector $spamPred$, then use the cut-off, say 0.6, where if $spamPred$ exceeds 0.6 then the prediction is spam.

```
spamPred = as.numeric(spamPred > 0.6)
assess = table(spamPred, spamTruth)
assess
         spamTruth
```

```
spamPred       FALSE TRUE
       FALSE   1751  190
       TRUE      43  428
errs1 = assess[2, 1] / sum(assess[ , 1])
errs2 = assess[1, 2] / sum(assess[, 2])
```

We see that we incorrectly identified 190 spam emails as not spam, and incorrectly identified 43 emails that were not spam as spam. If we normalized 190 by the number of spam $(190 + 428)$ and normalized the 43 by the number of not spam $(1751 + 43)$ then we have two types of error rates.

The function `plotErrorRates` computes these two error rates for all values of k. Then it makes a line plot of them, which shows the misclassification rate for different values of k. That is, the x-axis will be k, and the y axis will be the two types of misclassification rates.

TURN IN. Turn in your functions and code to compute the error rates and make the plot in a plain text .R file.

Include as a comment on the shape of the curve and your choice of k.