Stat 133, Fall 14
Homework 8: Creating a Spam Filter
Due: Thursday Nov 20 at 11:55pm

For this assignment we will use some variables like those constructed in HW #6 to create a spam filter. You will use k-nearest neighbor to predict whether an email is either 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.
It contains 3 objects:

- Data frame called `Emails` with 2412 emails and 23 variables. The variables are described at the end of this document.
- Matrix called `distEmails` which is 2412 by 2412 and holds all distances between the emails in `Emails`.
- Vector of logicals called `isSpam` of length 2412 that indicates whether email is spam or not (this is also the first column of `Emails`).

Note that you should not use the function `knn()` or really worry too much about how the distances are being calculated. Just accept that the distances in `distEmails` are distances that make sense.

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 logical vector of length m which is a spam prediction for each of the m emails in the test set. The prediction is to be made using the k-nearest neighbor method. 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" to see 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).

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.

Ideally, your function can take in a **vector** *for k and make predictions for spam for each value of k for each email in the test set. If so, then the return value from your function would be an m by 20 (for 20 different values of k, see below) matrix of logicals, where each row corresponds to one of the m test emails and each column a value of k.*

To see how well the predictions do, you can compare to the truth as follows. For a set of m test emails, if you have the truth for them in the logical vector $spamTruth$ and you have the prediction for them in the logical vector $spamPred$, then

```
> assess = table(spamPred, spamTruth)
> errs = (assess[1,2] +  assess[2, 1]) / length(spamTruth)
```

The goal is to compare the predictive ability with respect to this error rate for k running from 1 to 20. We will use cross-validation to estimate the predictive ability of each of the 20 models (for $k = 1, ..., 20$).

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, but all data should be included in exactly one of the 10 parts. 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 and see what value of k produces the lowest misclassification rate.

1. **Write a function called `cvKnn`** that has input arguments:
   - `distEmails`,
   - `isSpam`,
   - `k` a vector of values for the k nearest neighbor, and
   - `vfold=10` a value for the number of folds in v-fold cross validation

   This function should returns a matrix with as many rows as in `distEmails` and as many columns as `k` values. We will call it with our `distEmails` and `k = 1:20`, and so the return value will be a 2412 by 20 logical matrix of the predictions for each email for k running from 1 to 20.

2. Recall from class that it's simplest to randomize the row numbers than the matrix. That is, the `sample()` function can produce a random permutation of the row indices and then you can use the first 1/10 of these as indices for the first test set, etc. Since the number of observations is not divisible by 10, you should put the extra data points in groups in a way that keeps the group sizes as similar as possible (in other words, 8 of the test groups should have 241 emails and 2 groups will have 242 emails).

3. **Plot**. Make a plot 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 misclassification rate.

TURN IN. Turn in code in a R script file, including the code for two functions: `predNeighbor` and `cvKnn`. Also turn in the code for plotting the error rates. Include a comment on the shape of the curve and your choice of k.

It's not necessary to work with `Emails` for this assignment, but in case you're curious, descriptions of the variables appear below. You could run some of your code on these variables to produce plots and see which ones seem to be more related to spam, but actually it doesn't matter for purposes of predicting spam which ones are being used in what way. The first column of the data frame is a variable called "isSpam" which is a logical that is TRUE if the email is spam, FALSE if not. After that, the variables are:

1. `isRe`: The subject is "Re: something or other."
2. `numLinesInBody`: The number of lines in the body of the email.
3. `bodyCharacterCount`: The number of characters in the body of the email.
4. `replyUnderline`: The Reply-To has an underline and numbers/letters.
5. `subjectExclamationCount`: The number of exclamation marks in the subject.
6. `subjectQuestionCount`: The number of question marks in the subject.
7. `numAttachments`: The number of attachments.
8. `priority`: The X-priority or X-Msmail-Priority set to high.
9. `numRecipients`: The number of recipients.
10. `percentCapitals`: The proportion of characters in the subject (excluding punctuation and numbers) that are capitals.
11. `isInReplyTo`: The email body contains the phrase "In Reply To:
12. `sortedRecipients`: The recipient list is sorted by address

13. `subjectPunctuationCheck`: The subject has punctuation or digits surrounded by characters, e.g. V?agra and pay1ng, but not New!

14. `multipartText`: The header states that the message is multipart, but it is mostly text or html.

15. `containsImages`: The email contains images.

16. `subjectSpamWords`: The subject contains one of the following words: viagra, pounds, free, buy, weight, guarantee, millions, dollars, credit, risk, prescription, generic, drug, money, credit.

17. `percentSubjectBlanks`: The fraction of blanks in the subject.

18. `messageIdHasNoHostname`: The Message-Id has no hostname or the Message-Id is missing

19. `fromNumericEnd`: The From: address ends in numbers, e.g.

    ```
    david gezi <davidgezi12@hotmail.com>
    ```

20. `isDear`: The body contains Dear something, such as DEAR SIR, or Dear Madam

21. `isWrote`: The body contains so-and-so Wrote:

22. `averageWordLength`: The average length of words in the body.