ITCS 6156 Fall 2016
Homework 1
Points: 100

This assignment is to be completed individually. No group work is allowed.

Description

For this homework, you will build two text categorization classifiers: one using naive Bayes and the other using decision trees. You will write general code for cross-validation that will apply to either of your classifiers.

Data and starter code: In the ITCS6156_HW1.zip archive, you should find the 20newsgroups data set (also available from the original source http://qwone.com/~jason/20Newsgroups/). This data set, whose origin is somewhat fuzzy, consists of newsgroup posts from an earlier era of the Internet. The posts are in different categories, and this data set has become a standard benchmark for text classification methods.

The data is represented in a bag-of-words format, where each post is represented by what words are present in it, without any consideration of the order of the words.

You are provided templates for the functions you need to program. These templates are provided as a prototype for you to follow. The templates are based on a MATLAB environment but you are not limited to this programming language. For programming tools options refer to the "Instructions for Programming Tools" on the course's online page. The functions you write should follow this prototype structure regardless of the software environment you adopt. You are allowed to create additional functions, files, or scripts, and you may also modify the included interfaces for existing functions if you prefer a different organization. Make sure to clearly document your work and justify your choices.

Examine the included files, which are described in Table 1. The **majorityTrain.m** and **majorityPredict.m** are the templates for the training and prediction functions. The training function takes a data set, labels, and some parameters as input and outputs a model, and the prediction algorithm takes a data example and a model and outputs a prediction. The functions you will write should follow this prototype.

We will use a binary representation of the word counts. The loading script **loadAllData.m** binarizes the word counts. You may change this if you want, but by making the observed values Bernoulli (binary) random variables, your classifiers will be quite a bit simpler than otherwise.

Your programs must be well documented. Use comments to explain the steps through out your code. If your code is not clear to the grader you will be asked to demonstrate you code and analysis.

Requirements

1. (15 points) Write the functions **naiveBayesTrain** and **naiveBayesPredict**. The training algorithm should find the maximum likelihood parameters for the probability distribution

$$\Pr(y_i = c|\boldsymbol{x}_i) = \frac{\Pr(y_i = c)\prod_{w\in W}\Pr(x_{iw}|y_i = c)}{\Pr(x_i)}$$

Make sure to use log-space representation for these probabilities, since they will become very small, and notice that you can accomplish the goal of naive Bayes learning without explicitly computing the prior probability Pr(x_i). In other words, you can predict the most likely class label without explicitly computing that quantity. Implement symmetric Beta regularization for your naive Bayes learner. One natural way to do this is to let the input parameter params simply be the prior count for each word. For a parameter α, this would mean your maximum likelihood estimates for any Bernoulli variable X would be

$$\Pr(X) = \frac{(\#\text{ examples where } X) + \alpha}{(\text{Total }\#\text{ of examples}) + 2\alpha}$$

Notice that if α = 0 , you get the standard maximum likelihood estimate.

2. (15 points) Write the function **computeInformationGain**. The function should take in training data and training labels and computes the information gain for each feature. That is, for each feature dimension, compute

$$\begin{aligned} G(Y, X_j) &= H(Y) - H(Y|X_j) \\ &= -\sum_y \Pr(Y = y)\log\Pr(Y = y) + \\ &\quad \sum_{x_j}\Pr(X_j = x_j)\sum_y \Pr(Y = y|X_j = x_j)\log\Pr(Y = y|X_j = x_j) \end{aligned}$$

Your function should return the vector

$$[G(Y, X_1), \ldots, G(Y, X_d)]^\top$$

You will use this function to do feature selection and as a subroutine for decision tree learning.

3. (15 points) Write the functions **decisionTreeTrain** and **decisionTreePredict**. You'll have to design a way to represent the decision tree in the model object. Your training algorithm should take a parameter that is the maximum depth of the decision tree, and the learning algorithm should then greedily grow a tree of that depth. Use the information-gain measure to determine the branches (hint: you're welcome to use your **computeInformationGain** function from the previous question).

   Algorithm 1 is abstract pseudocode describing one way to implement decision tree training. You are welcome to deviate from this somewhat; there are many ways to correctly implement such procedures.

Algorithm 1 Recursive procedure to grow a classification tree

```
1: function FITTREE(𝒟, depth)
2:     if not worth splitting (because 𝒟 is all one class or max depth is reached) then
3:         node.prediction ← arg max_c Σ_{(x,y)∈𝒟} I(y = c)
4:         return node
5:     w ← arg max_w G(Y, X_w)
6:     node.test ← w
7:     node.left ← FITTREE(𝒟_L, depth+1)        ▷ where 𝒟_L := {(x, y) ∈ 𝒟 | x_w = 0}
8:     node.right ← FITTREE(𝒟_R, depth+1)       ▷ where 𝒟_R := {(x, y) ∈ 𝒟 | x_w = 1}
9:     return node
```

   The pseudocode suggests building a tree data structure that stores in each node either (1) a prediction or (2) a word to split on and child nodes. The pseudocode also includes the formula for the entropy criterion for selecting which word to split on.

   The prediction function should have an analogous recursion, where it receives a data example and a node. If the node has children, the function should determine which child to recursively predict with. If it has no children, it should return the prediction stored at the node.

4. (15 points) Write the function **crossValidate**, which takes a training algorithm, a prediction algorithm, a data set, labels, parameters, and the number of folds as input and performs cross-fold validation using that many folds. For example, calling

```
params.alpha = 1.0;
score = crossValidate(@naiveBayesTrain, @naiveBayesPredict, trainData, ...
                      trainLabels, 10, params);
```

   will compute the 10-fold cross-validation accuracy of naive Bayes using regularization parameter α = 1.0.

   The cross-validation should randomly split the input data set into folds subsets. Then iteratively hold out each subset: train a model using all data except the subset and evaluate the accuracy on the held-out subset. The function should return the average accuracy over all folds splits.

5. (15 points) Write a script that performs cross validation on the provided 20newgroups training data (**trainData**, **trainLabels**). Plot the cross-validation accuracy score as you vary the regularization parameters for each of your classifiers (naive Bayes and decision tree). Choose a reasonable range of parameter settings so you can see the effect of the parameter values. Use the cross-validation accuracy to set a regularization parameter for training on the full training data **(trainData**, **trainLabels**) and test on the test set (**testData**, **testLabels**).

   You may base your script on **testPredictors.m**, but note that **testPredictors.m** DOES NOT do cross-validation.

6. Write a 1-2 page summary of your results. This summary should be short, but it should very briefly discuss any implementation decisions you made beyond the provided instructions and display and describe the plots you generated, any discoveries you made about the tuning process, which method worked better, and your hypotheses on why the results were as you saw. Be clear concise.

   One way to do this rather elegantly in MATLAB is to use the publishing mode. See http://www.mathworks.com/help/matlab/matlab_prog/publishing-matlab-code.html for more. Using this mode is not required, nor is it especially recommended, but it might be convenient to try.

Assignment Submissions

What to submit using Canvas (Email submissions will NOT be accepted):

1. **HW1_results.pdf** – PDF document with your write up for the results in question 6.
2. **HW1.zip** - An archive of the entire programming project stored in a standard ZIP file. Make sure to include all packages and libraries used to run your programs if any.
3. **README.txt** –This file should detail all the files in your project archive, libraries and packages used and any special setup you have used in your programming environment.
4. **INFO.pdf** – PDF document with the following assignment information:
   a) Explanation of status and stopping point, if incomplete.
   b) Explanation of additional functions and analysis, if any.
   c) Discuss the easy and challenging parts of the assignment. How did you overcome all or some of the challenges?

Table 1: Included files and brief descriptions.

File	Path Description
20news-bydate/data/test.data	Sparse representation of word counts in test data. The first column is the document ID, the second column is the word ID, and the last column is the count
20news-bydate/data/test.label	Labels (from 1 to 20) of each post's news group
20news-bydate/data/test.map	Mapping from label numbers to newsgroup names
20news-bydate/data/train.data	Sparse representation of word counts in training data
20news-bydate/data/train.label	Labels of each post's news group
20news-bydate/data/train.map	Mapping from label numbers to newsgroup names (this should be identical to test.map)
src/computeInformationGain.m	Function you will complete to compute the information gain for each feature.
src/crossValidate.m	Function you will complete to run k-fold cross-validation on data using a given training function and prediction function
src/decisionTreePredict.m	Function you will complete that takes a trained decision tree and a data example and predicts a label
src/decisionTreeTrain.m	Function you will complete that takes a data set, labels, and learning parameters, and returns a trained decision tree model
src/loadAllData.m	Script to load data into sparse matrices in MATLAB
src/majorityPredict.m	Function that takes a "trained" majority "model" and a data example and predicts the majority class
src/majorityTrain.m	Function that a data set, labels, and (empty) learning parameters, and returns a "model" that indicates the majority class from the labels
src/naiveBayesPredict.m	Function you will complete that takes a trained naive Bayes model and a data example and predicts a label
src/naiveBayesTrain.m	Function you will complete that takes a data set, labels, and learning parameters, and returns a trained naive Bayes model
src/testPredictors.m	Example script for running your predictors. This script does not do cross validation. It only uses hard-coded settings for regularization parameters