Machine Learning Fall 2015 Homework 1

Homework must be submitted electronically following the instructions on the course homepage. Make sure to explain you reasoning or show your derivations. Except for answers that are especially straightforward, you will lose points for unjustified answers, even if they are correct.

General Instructions

Submit your homework electronically on Canvas. We recommend using LaTeX, especially for the written problems. But you are welcome to use anything as long as it is neat and readable.

Include a README file that describes all other included files. Indicate which files you modified. You are welcome to create additional functions, files, or scripts, and you are also welcome to modify the included interfaces for existing functions if you prefer a different organization.

Since we will work on some of the homework in class, clearly indicate which parts of your submitted homework are work done in class and which are your own work.

Relatedly, cite all outside sources of information and ideas. List any students you discussed the homework with.

Written Problems

1. (5 points, Based on Murphy 2.2) Suppose a crime has been committed. Blood is found at the scene for which there is no innocent explanation. It is of a type that is present in 1% of the population. A suspect who has this rare blood type has been charged with the crime.

   The prosecutor claims: "There is a 1% chance that the defendant would have the crime blood type if he were innocent. Thus there is a 99% chance that he is guilty." This is known as the **prosecutor's fallacy**. What is wrong with this argument?

2. A Bernoulli distribution has the following likelihood function for a data set $\mathcal{D}$:

$$p(\mathcal{D}|\theta) = \theta^{N_1}(1-\theta)^{N_0}, \tag{1}$$

   where N_1 is the number of instances in data set $\mathcal{D}$ that have value 1 and N_0 is the number in $\mathcal{D}$ that have value 0. The maximum likelihood estimate is

$$\hat{\theta} = \frac{N_1}{N_1 + N_0}. \tag{2}$$

   (a) (5 points) Derive the maximum likelihood estimate above by solving for the maximum of the likelihood. I.e., show the mathematics that get from Equation (1) to Equation (2).

   (b) (5 points) Suppose we now want to maximize a posterior likelihood

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}, \tag{3}$$

   where we use the Bernoulli likelihood and a (slight variant[1] of a) symmetric Beta prior over the Bernoulli parameter

$$p(\theta) \propto \theta^{\alpha}(1-\theta)^{\alpha}. \tag{4}$$

   Derive the maximum posterior mean estimate.

[1] For convenience, we are using the exponent of α instead of the standard $\alpha - 1$.

3. (Based on Murphy 3.19) Let $x_{iw} = 1$ if word w occurs in document i and $x_{iw} = 0$ otherwise. Let θ_{cw} be the estimated probability that word w occurs in documents of class c. Then the log-likelihood that document $\boldsymbol{x}$ belongs to class c is

$$\log p(\boldsymbol{x}_i|c,\theta) = \log \prod_{w=1}^{W} \theta_{cw}^{x_{iw}} (1-\theta_{cw})^{1-x_{iw}} \tag{5}$$

$$= \sum_{w=1}^{W} x_{iw} \log \theta_{cw} + (1-x_{iw}) \log(1-\theta_{cw}) \tag{6}$$

$$= \sum_{w=1}^{W} x_{iw} \log \frac{\theta_{cw}}{1-\theta_{cw}} + \sum_{w} \log(1-\theta_{cw}), \tag{7}$$

where W is the number of words in the vocabulary. We can write this more succinctly as

$$\log p(\boldsymbol{x}_i|c,\theta) = \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\beta}_c \tag{8}$$

where $\boldsymbol{x}_i := (x_{i1}, \ldots, x_{iW})$ is a bit vector, $\boldsymbol{\phi}(\boldsymbol{x}) := (\boldsymbol{x}, 1)$, and

$$\boldsymbol{\beta}_c := \left(\log \frac{\theta_{c1}}{1-\theta_{c1}}, \ldots, \log \frac{\theta_{cW}}{1-\theta_{cW}}, \sum_{w} \log(1-\theta_{cw}) \right)^\top. \tag{9}$$

This is a linear classifier, since the class-conditional density is a linear function (an inner product) of the parameters $\boldsymbol{\beta}_c$.

(a) (4 points) Assuming $p(C=1) = p(C=2) = 0.5$, write an expression for the log posterior odds ratio, $\log_2 \frac{p(c_i=1|\boldsymbol{x}_i)}{p(c_i=2|\boldsymbol{x}_i)}$, in terms of the features $\boldsymbol{\phi}(\boldsymbol{x}_i)$ and the parameters $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$.

(b) (4 points) Ideally, words that occur in both classes are not very discriminative and therefore should not affect our beliefs about the class label. State the conditions on θ_{1w} and θ_{2w} (or equivalently the conditions on β_{1w} and β_{2w}) under which the presence or absence or w in a test document will have *no effect* on the class posterior. (Such a word will be ignored by the classifier.) Hint: using your previous result, figure out when the posterior odds ratio is $0.5/0.5$.

(c) (4 points) The posterior mean estimate of θ, using a $\mathrm{Beta}(1,1)$, is given by

$$\hat{\theta}_{cw} = \frac{1 + \sum_{i \in c} x_{iw}}{2 + n_c}, \tag{10}$$

where the sum is over the n_c documents in class c. Consider a particular word w, and suppose it always occurs in every document regardless of class. Consider a two-class problem with $n_1 \neq n_2$ (e.g., where we get more non-spam than spam, or other situations with class imbalance). If we use the above estimate for θ_{cw}, will word w be ignored by our classifier? Why or why not?

(d) (3 points) What other ways can you think of that encourage irrelevant words to be ignored?

Programming Assignment

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 HW1 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.

Your required tasks follow.

1. (0 points) Examine the included files, which are described in Table 1. Look at the template training and prediction functions `majorityTrain.m` and `majorityPredict.m`. 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.

   Notice that 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.

2. (5 points) Write function `computeInformationGain.m`. 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

$$\begin{aligned} G(Y, X_j) &= H(Y) - H(Y|X_j) \\ &= -\sum_y \Pr(Y = y) \log \Pr(Y = y) + \\ &\quad \sum_{x_j} \Pr(X_j = x_j) \sum_y \Pr(Y = y | X_j = x_j) \log \Pr(Y = y | X_j = x_j). \end{aligned} \tag{11}$$

   Your function should return the vector

$$[G(Y, X_1), \ldots, G(Y, X_d)]^\top. \tag{12}$$

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

3. (5 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 $\alpha = 0$, you get the standard maximum likelihood estimate.

4. (5 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.m` function). 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

function FITTREE($\mathcal{D}$, depth)
  if not worth splitting (because $\mathcal{D}$ is all one class or max depth is reached) **then**
    node.prediction $\leftarrow \arg\max_c \sum_{(\boldsymbol{x},y)\in\mathcal{D}} I(y = c)$
    return node
  $w \leftarrow \arg\max_w G(Y, X_w)$ ▷ See Equation (11)
  node.test $\leftarrow w$
  node.left $\leftarrow$ FITTREE($\mathcal{D}_L$, depth+1) ▷ where $\mathcal{D}_L := \{(\boldsymbol{x}, y) \in \mathcal{D} | x_w = 0\}$
  node.right $\leftarrow$ FITTREE($\mathcal{D}_R$, depth+1) ▷ where $\mathcal{D}_R := \{(\boldsymbol{x}, y) \in \mathcal{D} | x_w = 1\}$
  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.

5. (5 points) Write the function `crossValidate.m`, 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 $\alpha = 1.0$.

   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.

6. (5 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.

7. (5 points) 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.[2] Be concise!

[2]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.

Table 1: Included files and brief descriptions.

File Path	Description
20news-bydate/matlab/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/matlab/test.label	Labels (from 1 to 20) of each post's news group
20news-bydate/matlab/test.map	Mapping from label numbers to newsgroup names
20news-bydate/matlab/train.data	Sparse representation of word counts in training data
20news-bydate/matlab/train.label	Labels of each post's news group
20news-bydate/matlab/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.