

CS 221, Autumn 2009

Problem Set #2 Programming Assignment

Due by 9:30am on Tuesday, October 27. Please see the course information page on the class website for late homework submission instructions. SCPD students can also fax their solutions to (650) 725-1449. We will not accept solutions by email or courier.

Programming part (35 points)

1 Overview

For this problem, your programming team (1 to 3 people) will use an ensemble of decision trees to build a handwritten digit recognizer. The set of handwritten characters is supplied for you. They come from the post office in Buffalo, NY, where a classifier like yours would greatly speed up delivery. The goal of the recognizer is to get 100% of the characters correctly classified as the appropriate digit, but the problem with real-world data is that it is noisy. Some of these digits could not even be pre-classified by people, so getting as close as possible to 100% is a more feasible goal.

The overall structure of the problem set is as follows: We consider three types of classifiers for the data. The first is a single decision tree. The second is an ensemble of decision trees constructed using the Bagging algorithm. The last is an ensemble of decision trees constructed using the AdaBoost algorithm. In this problem set, we will explore varying the classifiers along different dimensions: the size of the training set, the depth of the decision tree(s), and the number of elements in the ensemble.

Note that this project involves running computationally intensive experiments, so you might want to start early to give yourself plenty of time to optimize your code and to obtain all the results.

1.1 Data Set

The data set consists of images of centered hand-written digits. Each image is a 14×14 matrix where each pixel's intensity is in the range $(0, 255)$, which we have normalized to be in the range $(0, 1)$. For now, let's just consider the case of distinguishing 0s from 1s; we'll label 1 the positive class.

Note that we are also providing you with your own independent test set, which you will use to evaluate your algorithm. **You must not use your test data to train or tune the parameters of your model.** The test data is there to gauge how well your classifier performs on unseen data. If you use it to train, your graphs will not show the desired behaviors and you won't get full credit.

1.2 Decision Trees

Since our data involves continuous variables, we need to extend our notion of splits in a decision tree. We use the following approach to build a decision tree on this input: Each node in the tree

chooses a single pixel and a threshold value. If the value of that pixel in an image is greater than the threshold, it sends the digit down the left branch. Otherwise, it sends it down the right.

We use decision trees in two ways. The decision tree learning algorithm, as described in class, assumes that each leaf in the tree is annotated with the distribution of positive/negative instances that reach that leaf.

1.3 Bagging

Building on this algorithm, we use *bagging* to generate several such trees, and choose the output predicted by the largest number of these trees. Given a training set of size m , bagging generates several training sets by sampling m training examples *with replacement* from the original training set.¹ We will call the number of training sets sampled as the number of *bags* and denote it by B . A base learning algorithm (decision tree in our case) is applied to each of these B training sets to produce B different classifiers. Given a new test example to classify, the algorithm classifies the example using each of the B classifiers, and picks the output label chosen by the majority of the classifiers.

1.4 Multiclass Classification

So far, we have assumed that we have only two classes. However, our task is to classify our inputs into one of 10 classes, corresponding to the 10 digits. To generalize the approach for 10 classes, we will use the “One-Against-Many” strategy. For each digit, we will train a classifier to recognize whether an image is that digit or not. For example, the classifier f_0 decides whether an image is a 0 or not; it will be trained on data where the positive instances are 1’s, and the negative instances are all the rest. The classifier f_1 decides whether the image is a 1 or not, and so on. For this assignment, each of these classifiers will either be a single decision tree, or an ensemble of decision trees created by bagging or boosting.

To select the final classification for a new digit, we run all 10 classifiers on it. We then select as our class label k the classifier k which was most confident in its prediction:

- For individual decision tree classifiers, we will use the probability assigned by the k ’th tree to the positive class (representing label k) as the measure of confidence.
- With bagging, there will be several decision tree classifiers with digit k as the positive class. We will measure the confidence for label k using the number of decision tree classifiers that assign the digit to the positive class (i.e., returns a probability of greater than 0.5).
- With boosting, there are several *weighted* decision tree classifiers with digit k as the positive class, and we will measure the confidence for label k using a weighted voting scheme. Each decision tree classifier (for label k) outputs either +1 if it assigns this digit to the positive class (i.e., outputs a probability of greater than 0.5), and -1 otherwise. This vote gets multiplied by the weight assigned to the decision tree by the AdaBoost algorithm. This weighted sum of all the votes corresponds to the confidence of the k ’th classifier.

¹In statistics, such a sample is called a bootstrap sample.

2 Tasks and Deliverables

2.1 Assignment

1. [17 points] Growing Decision Trees.

We will start by implementing the decision tree algorithm we studied in class: Implement the entropy learning rule for decision trees. That is, for each node in the tree, choose a pixel and a threshold value that maximizes the information gain. If there is tie, prefer lower pixel numbers. If there is a tie between thresholds for the same pixel, prefer lower thresholds. For the thresholds, consider only the values $0.1, 0.2, \dots, 0.9$. The decision tree should output probabilities.

We have already provided most of the code needed. For this question, you will need to write the splitting rule for the decision tree, and the one-against-many classifier for decision trees. This requires implementing the following routines:

- `initializeProbabilities`, which initialize probabilities at a leaf node of the decision tree
- `choosePixelAndThreshold`, which uses entropy to decide which pixel to split on and at which threshold
- `positiveConfidence`, which outputs the confidence (probability) of assigning a given digit to the positive class according to the given decision tree, and
- `decisionTreeAccuracy`, which evaluates the accuracy of the decision tree classifier set consisting of 10 trees. “Accuracy” is the percent of correctly classified test set digits. Each digit is classified by all 10 decision trees, and is assigned a label corresponding to the decision tree which returns the highest probability, as described in section 1.4.

Throughout the code, assume that each example is also assigned a *weight*—think of this as just meaning that the example is repeated that many times. For the single tree, all example weights will simply be 1, but we will set the weight to other non-negative values in later parts.

Since other parts of this assignment rely on the functions you implement here, we provide you with some debugging output to compare your implementation of each function against ours. There is also some debugging code in `pa2_matlab_a` which you can use to verify that all parts of the pipeline are working together correctly.

`pa2_matlab_a` will run your decision tree classifier with tree depths of 4, 6, 8, \dots , 14. It will generate a plot of training and test set accuracy against depth. Run this test for two different training set sizes, 1k and 10k instances (you will need to modify the top line of `pa2_matlab_a` to switch between the training sets).

First, what do you notice about each graph? Second, how does the behavior change as we train on a larger data set? Explain concisely but clearly (using formal terminology) what’s happening.

2. [7 points] Bagging

We will now try bagging with decision trees. In the multiclass setting with B bags, we will use bagging to train B decision trees to classify each digit as the positive class (i.e., there will be $10 \cdot B$ decision trees trained in total).

The code will “add” bags one-by-one in the following loop: at each bagging iteration, for each of the 10 digits, we sample a training set and train a decision tree considering that digit

as the positive class. For a digit x , let $a_k(x)$ be the number of decision trees with positive class representing digit k that classify the digit into the positive class (representing label k); then, the output label should be $\arg \max_k a_k(x)$, breaking ties by picking the lowest digit possible. We iterate the loop upto a maximum number of bags (100) or until the test set accuracy converges (to limit the runtime of the experiments). After each bagging iteration, we compute the training and test error to analyze the results later.

Again, we have provided most of the code to implement bagged decision trees. You will need to implement:

- `addBaggedDecisionTree`, which creates one decision tree using bagging to add to the ensemble, and
- `baggedDecisionTreeAccuracy`, which evaluates the accuracy of bagged decision tree classifier set, as described above.

To represent the number of times a training example is chosen, the code assigns a non-negative weight to each example. Thus, if an example is not chosen in the random sample, its weight is set to 0; or, if it is chosen twice, its weight is 2. Your decision tree code should already take these weights into consideration appropriately.

`pa2_matlab_b` will run your code with tree depths from 4 to 14 using the data set with 1k instances. The resulting plot shows training and test set accuracy against depth for the bagged decision tree classifier.

How is the performance of the bagged classifier different from the performance of the single decision tree? Briefly explain this difference.

3. [11 points] AdaBoost

Finally, implement the AdaBoost algorithm with decision trees, as described in Lecture Notes 7. Be sure to carefully read the comments in the code to understand exactly what is being asked. In particular, you do not need to implement the sampling step explicitly when doing an iteration of AdaBoost, since along the way (e.g., in `initializeProbabilities` and `choosePixelAndThreshold` functions) you were already considering the weights of the training examples that reach each node, which is essentially the same as sampling. You will need to implement the following functions:

- `addAdaBoostDecisionTree`, which runs one iteration of AdaBoost and returns one more weighted decision tree classifier to add to the ensemble, and
- `adaBoostAccuracy`, which evaluates the current set of 10 AdaBoost classifiers as described in section 1.4.

`pa2_matlab_c` will run your code with tree depths of 4, and plot training and test set accuracy (percentage of the training/test set classified correctly) as a function of ensemble size (number of trees) using a dataset of 1k instances.

Does the training set accuracy monotonically improve as we add more trees? Does the test set accuracy? Briefly explain this behavior.

2.2 Deliverables

The programming assignment can be done in groups of up to 3 students. Please turn in just one copy per group. Remember, the **only** files you should modify to complete this assignment are

the one listed below. Your code should be clear and commented. You should print out and turn in the following:

1. The 8 functions discussed in the previous section: `initializeProbabilities.m`, `choosePixelAndThreshold.m`, `positiveConfidence.m`, `decisionTreeAccuracy.m`, `addBaggedDecisionTree.m`, `baggedDecisionTreeAccuracy.m`, `addAdaBoostDecisionTree.m` and `adaBoostAccuracy.m`
2. Any other code (e.g., helper functions) that you write
3. The two plots generated by `pa2_matlab_a`, and one plot each from `pa2_matlab_b` and `pa2_matlab_c`.
4. The answers to the short questions at the end of each of the three parts of the assignment.

3 Data and Code

3.1 Data set

The data set comes from a large database (over 60k) of handwritten digits known as MNIST. There is a relevant webpage on the data at <http://yann.lecun.com/exdb/mnist/index.html>. We have created our own subset of this data comprising 10k training examples and 1k test examples.

- `data/training-*k-images.idx3` are your training sets which contain the image data.
- `data/training-*k-labels.idx1` are your training sets which contain the image labels (digits 0-9).
- `data/test-1k-images.idx3` is your test set images.
- `data/test-1k-labels.idx1` is your test set labels.

Don't worry about having to learn the `.idx` file format. A friendly TA has already gone about and done this for you. All you need to know is that each digit is represented as a 14x14 array of pixel values which range from 0 to 1. You will access this array as a vector (in row-major format) with 14^2 elements.

To visualize the digits, you can first load them in using the provided `loadDigits` function (see `pa2_matlab*.m` for usage), and then use the `displayDigit` and `displayDigits` functions available in the `data` directory. The former displays a single digit stored as an array of pixels, and the latter displays a random subset of 20 training examples along with their corresponding labels. This is not necessary for the assignment, but would give you an idea of the difficulty of the dataset.

3.2 Code Infrastructure

The Matlab files for this assignment are available at cs221.stanford.edu/code/pa2.zip. It will unpack into a folder called `cs221_pa2`. Be sure to read all the comments in the code, and to use the provided debugging output.

The data structures are explained in the code. In particular, be sure to read over the comments in `initializeDecisionTreeNode` for a description of the data structure used to represent a decision tree.

Finally, remember to write optimized code for training your decision trees, especially in the `choosePixelAndThreshold` function. This is the inner loop of all parts of this assignment, and implementing it inefficiently might mean that your experiments will take a lot longer to run than they should. See the comment at the bottom of `choosePixelAndThreshold.m` for optimization ideas. Each of the three parts of the assignment can be implemented to run in under a couple of hours.

3.3 Etiquette and Unix Tips

This project involves running computation intensive training experiments. The Leland machines are a shared resource for all of Stanford which should not be abused. To be fair and limit the load on the cluster, we ask that each group use a maximum of six processes total on at most three machines. If you are running a long experiment, you should nice the processes, like this:

```
% nice +10 myprog parameters
```

Exceeding the above recommendations will be considered a violation of class rules.