6.034 Artificial Intelligence, Fall 2006
Prof. Patrick H. Winston

Problem Set 5

This is the last problem set in 6.034! It's due Wednesday, November 29th at 11:59 PM. If you have questions about it, ask the TA email list. Your response will probably come from a TA.

To work on this problem set, you will need to get the code:

The code can be downloaded from the assignments section.

Your answers for the problem set belong in the main file `ps5.scm`, as well as `boost.scm`.

Things to remember

- Avoid using DrScheme's graphical comment boxes. If you do, take them out or use *Save definitions as text...* so that you submit a Scheme file in plain text.
- If you are going to submit your pset late, see the problem set grading policy.

1. SVMs

These questions are going to ask you only to work out some SVM stuff on paper. Some of the answers are tested in the public tester, and some are tested in the hidden tester, so be careful which is which.

Vectors should be expressed as two-element lists. Decimals can be computed, as in `(sqrt 5)`, or rounded off -- but leave at least 3 significant digits there.

Fill these in in your ps5.scm as `answer-1.1` through `answer-1.5`.

1. A simple linear SVM, with equation $f(x) = \mathbf{w}\cdot\mathbf{u} + b$, is trained on only two data points: $\mathbf{x}_+ = <1, 1>$ and $\mathbf{x}_- = <-1, -1>$. The value of $\mathbf{w}$ that maximizes the width of the margin is used. What is the value of $\mathbf{w}$? (Express your answer as a two-element list.)
2. What is the margin width?
3. What is the value of $f(<-6, 10>)$? (Hidden answer.)
4. If the kernel is changed to $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$, maintaining the same values of $\mathbf{w}$ and b, are the two data points correctly classified? (Answer #t or #f. Hidden answer.)
5. If the kernel is changed to $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^3$, maintaining the same values of $\mathbf{w}$ and b, are the two data points correctly classified? (Answer #t or #f. Hidden answer.)

2. Boosting

The framework for a boosting classifier can be found in `boost.scm`. Your task is:

- Write the `reweight-data` procedure, which changes the weights of each datum based on whether it is classified correctly or incorrectly. In particular, it sets the new weight of datum *i*, $D_{t+1}(i)$, to:
  - $D_t(i)/2 \times 1/\varepsilon_t$ if the datum is incorrectly classified
  - $D_t(i)/2 \times 1/(1 - \varepsilon_t)$ if the datum is correctly classified

  The error rate, ε_t, is provided as a parameter to `reweight-data`.

- Write the main procedure, `make-boost-classifier`, which creates a boosting classifier based on a weighted combination of simple classifiers.
- Write a procedure called `hardest-example` that can analyze the results of boosting and return the training example that was hardest to classify.

2.1. Boosting notes

You'll probably want to refer to the formulas for boosting. The handout has them all on one page.

2.2. Data representation

2.2.1. The data

You'll be working with the same political data you used in problem set 4a. In that problem set, each training example was represented by a *pair* (not a list), whose car was a list of attribute values and whose cdr was the class it should be classified as.

We've extended this representation by adding a *weight* to every example. The resulting data structure is called a *weighted datum*. A weighted datum is an improper list containing a *weight* (a positive real number), some *info* (a list of attribute values), and a *class*, which is how this example should be classified. Because we want this to work with the boosting algorithm, the only allowable classes are **1** and **-1**. (Here, we use -1 to mean "Democrat" and 1 to mean "Republican").

You don't have to worry about the improper list representation: we've abstracted over it with these procedures:

- `(get-weight datum)`: gets the weight of a weighted datum.
- `(get-info datum)`: gets the list of attribute values.
- `(get-class datum)`: get the correct classification for this datum.
- `(make-datum weight info class)`: create a datum, given its weight, info, and class.

- `(reweight-datum datum factor)`: return a new datum that is like the input datum, except its weight has been *multiplied* by `factor`.

2.2.2. Attributes

An attribute is represented much like it was in problem set 4a: it consists of a name, an *extractor* procedure, and a list of possible *values*, and a numeric *id* (which could hypothetically be used for sorting attributes). Here are the important procedures for accessing attributes:

- `(attribute-name attr)` returns an English string describing what this attribute represents.
- `(attribute-extractor attr)` returns a procedure that will pull this attribute's value out of a list of values that comes from `(get-info datum)`.
- `(attribute-values attr)` returns a list of possible values for this attribute.

The attributes that we use in this problem set appear in the list `congress-attribute-specs`.

2.2.3. Classifiers

A *classifier* is a procedure that generally takes in a list of attribute values (which could come from the `get-info` procedure) and returns a classification. This is what classifiers will do by default.

We've provided the *make-base-classifier* procedure, which creates a very simple classifier: it tests whether a particular attribute has a particular value, and outputs either -1 or 1 based on that test.

A *weighted classifier* is a list of a positive real number and a classifier procedure. These will be used within the boosting procedure to make some decisions more important than others.

In order to analyze how boosting works, we'll want classifiers to be able to output some additional information:

- If a classifier is given `'describe` as its input instead of a list, it outputs a list describing what this classifier does. Base classifiers will do this already. Your boost classifier should do this too, by returning `(describe-classifiers weighted-classifiers)`, where `classifiers` is a list of weighted base classifiers *in the order they were created*.
- A non-base classifier (that is, your boost classifier) should also accept the input `'data`, which returns the final state of its weighted data after all the base classifiers within it have been created and tested.

2.3. Helpful procedures

These procedures are already written for you:

- `(classifier-error-rate classifier weighted-data)`: returns the error rate (scaled so that the maximum is 1.0) of a classifier on a list of weighted examples.
- `(reweight-data classifier weighted-data error-rate)`: Applies the formula that assigns new weights to the data, depending on whether the classifier gets them right or wrong.
- `(error-to-alpha error-rate)`: converts an error rate to an alpha value. The alpha value is used to give a weight to each classifier in boosting.
- `(normalize weighted-data)`: Given some weighted data, scales all the weights so that they add up to 1. You should do this to the output of your `reweight-data` function.
- `(better-classifier c1 c2 weighted-data)`: takes in two classifiers and some data, and returns the one that classifies the data better.
- `(find-best-classifier classifier-maker attributes weighted-data)`: given a procedure that makes classifiers and a list of attributes to try, this finds the (attribute, value, class) combination that gives the best base classifier, and returns that classifier. (Earlier attributes, values, and classes take precedence, in that order.)
- `(sign x)`: returns 1 if x > 0 and -1 if x < 0.
- `(describe-classifiers weighted-classifiers)`: given a list of weighted classifiers, output a description of what these classifiers are doing, in the form the tester expects.

2.4. You do the rest

These helper procedures should allow you to write `reweight-data`, `make-boost-classifier`, and `hardest-example`. Fill them in at the bottom of boost.scm, and take note of the comments describing what they should do.

3. Survey

You know the drill by now. Answer the questions at the bottom of `ps5.scm`.

4. Errata

4.1. Wednesday, November 29

A last minute hint:

Yes, the tester expects decimals in some places, and complains when you give it fractions. If your code is outputting fractions, the simplest thing to do is to multiply them by 1.0 to get a decimal. You can also use the `exact->inexact` function.

The hidden tester will be more tolerant.

4.2. Sunday, November 26

We should mention that the weighted data we give you as input to your boost function isn't normalized (the weights don't add up to 1). The output weights *are* supposed to add up to 1. If your reweight-data function is outputting data that's off by a constant factor, simply running your output through the `normalize` procedure will fix it.

The last SVM question said $(\mathbf{x}_1 \cdot \mathbf{x}_3)^3$ where it should have said $(\mathbf{x}_1 \cdot \mathbf{x}_2)^3$.

4.3. Friday, November 24

The skeleton for `ps5.scm` was accidentally left out of the problem set. It's included now.