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

Problem Set 4a

This problem set is due Wednesday, November 8th 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, much like you did for earlier problem sets.

The code can be downloaded from the assignments section.

Your answers for the problem set belong in the main file `ps4.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. Nearest-Neighbors

In this part of the problem set, you will be implementing nearest-neighbors classification. You will be working with data that has the following form: `(features . class)`. (The dot in the middle means that this is a simple pair and not a list: the car contains the features, and the cdr contains the class.)

The first element in the pair is a feature vector. Feature vectors are represented as lists of values, each corresponding to a different variable. The second element is the class of the example. An example might look like this:

```
((blonde average light no) . "sunburned")
```

which represents a person who is blonde, average height, light in weight, didn't use sunscreen, and belongs in the "sunburned" class. You can find more sample data in the file data.scm. For this pset, the tester will be using data on congress-people, correlating their voting histories with which political party they belong to.

There are some helper functions that you may find useful:

- `(get-features example)` - Gets the features of an example
- `(get-class example)` - Gets the class of an example

- `(get-k-nearest k distance-metric features examples)` - Gets the k examples nearest a given feature set

1.1. Distance Metrics

In order to determine which neighbor is the nearest, it is necessary to define a distance metric.

1.1.1. Hamming Distance

Implement (hamming-distance x y). Given a pair of feature vectors, it should return the number of features that vary between the two. For example,

```
(hamming-distance '(1 1 1) '(1 1 5))
```

should return 1 because one feature is different.

1.1.2. Euclidean Distance

Hamming distance is a reasonable distance metric for discrete features, but does not perform as well with continuous data points. Implement (euclidean-distance x y). Given a pair of feature vectors, it should return the Euclidean distance between them. Recall that the formula for Euclidean distance is:

[(x1 - y1)^2 + (x2 - y2)^2 + ... + (xn - yn)^2] ^ (1/2)

If you want to test your Euclidean distance metric on congressional data, you'll have to first convert the feature vectors into numerical ones using this method:

`(congress-to-metric data)` - Converts an entire dataset from yes/no/maybe votes into numerical vectors

1.1.3. Angular Distance

An alternative to using plain distance is to find the angle between the vectors. This accounts for similar feature sets that differ mainly in magnitude. Implement (cos-theta x y), which returns the cosine of the angle between the two vectors. Recall that this can be calculated in the following way:

(x • y) / [|x||y|]

(x dot y over magnitude of x times magnitude of y)

1.2. Implementing Nearest-Neighbors

Now that we have working distance metrics, implement `(make-nn-classifier distance-metric training-examples)`. This sets up a nearest-neighbor classifier, which can then be used to classify feature sets in the following manner:

```
(define nn (make-nn-classifier distance-metric training-examples))
(nn feature-vector)
```

When given a feature vector, the nearest neighbor classifier should return the class of the neighbor with the least distance, as measured by `distance-metric`.

1.3. K-Nearest-Neighbors

To generalize, implement `(make-k-nn-classifier k distance-metric training-examples)`. This sets up a k-nearest-neighbors classifier, which can then be used to classify feature sets in the following manner:

```
(define k-nn (make-k-nn-classifier k distance-metric training-
examples))
(k-nn feature-vector)
```

When given a feature vector, the classifier should return the class most represented within the k nearest neighbors, defined by `distance-metric`.

When you have written code for nearest-neighbors and k-nearest-neighbors, you can use this procedure to test your classifiers on large datasets:

```
(validate-classifier classifier examples)
```

This tests a classifier against a set of labeled data, indicating the number of correct classifications.

The problem set file (ps4.scm) tells you how to use a hash table to store data when counting neighbors. You don't have to use a hash table, and the inner workings of hash tables are irrelevant to this problem. The information about hash tables is only there for your convenience.

2. Identification Trees

In this part of the problem set, you will be implementing code that works as part of an identification tree classification scheme. When creating an identification tree, it is desirable to order the attribute tests such that the tree is as minimal as possible. This implies ordering tests such that disorder decreases as fast as possible. In order to do this, we are using the heuristic of choosing as the next attribute to test the one that places the most elements in homogenous groups.

2.1. Basic Disorder Metric

Implement `(basic-disorder attribute-to-split classes examples)`, which returns the negative of the number of elements in homogenous groups (so that when there are more homogenous elements, the disorder metric decreases). The argument `attribute-to-split` is the attribute that is being tested. An attribute is a list containing the attribute name, a procedure to extract the attribute value from an example, and a list of possible values. An attribute for hair color might look like:

```
(list "Hair color" first '(blonde brown red))
```

The argument `classes` is a list of the possible classes, and `examples` is a list of examples to be split up.

Note: there is another common method used to measure disorder in the identification tree. The average disorder formula is as follows:

*sum over branches: (num in branch / num total) * disorder of branch*

Where disorder of branch =

*sum over classes: -(num of class in branch / num in branch) * lg (num of class in branch / num in branch)*

You don't have to implement this disorder metric for the problem set.

Once you have a working disorder formula, you can generate and use a decision tree in the following manner:

```
(define congress-id-classifier (make-idtree-classifier congress-
attribute-specs train-data-large))
(congress-id-classifier feature-vector)
```

You may also use validate-classifier to test your decision tree as you did with nearest-neighbors:

```
(validate-classifier congress-id-classifier test-data-large)
```

3. Neural Nets

These problems will be part of Problem Set 4b.

4. Survey

Please answer these questions at the bottom of your `ps4.scm` file:

- How many hours did this problem set take?
- Which parts of this problem set, if any, did you find interesting?
- Which parts of this problem set, if any, did you find boring or tedious?

(We'd ask which parts you find confusing, but if you're confused you should really ask a TA.)

When you're done, **run the tester** and submit your .scm files to your `6.034-psets/ps4` directory on Athena. If the tester dies with an error when it's run on your code, or if it stops and waits for user input, your submission will not count.

5. Errata

5.1. November 6, 2006

- The previous tester had several cases in it that depended on ties. These cases have been replaced. Download a new tester.scm.
- You should disregard "Problem 3". It shouldn't be there.
- Problem set 4b will be out soon, with a problem that involves training a neural net.