CIS 501 (Fall 2013)
Lab 4

Wei Lee Woon, CIS Program,

Masdar Institute, Abu Dhabi, UAE

1 Introduction

Today's laboratory will guide you through the implementation of a simple decision tree by developing then assembling the basic building blocks:

1. Entropy
2. Information Gain
3. Recursion + termination criteria

As described in lectures, the information gain for a feature a, w.r.t. a class variable X is given by:

$$IG(a) = H(X) - H(X|a)$$

where $H(X)$ is the entropy of the class variable X:

$$H(X) = -\sum_x p(x) log_2 p(x)$$

and $H(X|a)$ is the expected entropy of X given knowledge of a:

$$H(X|a) = -\sum_{a_i \in a} \frac{1}{n_{a_i}} \sum_x p(x|a_i) log_2 p(x|a_i) \quad (1)$$

where a_i is a particular value for the feature a, and n_{a_i} is the number of instances for which $a = a_i$.

Furthermore, to make things simple, we will only deal with binary class labels and nominal features (the principle generalizes easily to the case of multi-valued class; similarly, continuous labels can be converted into binary ones via thresholding.

2 Entropy

As an intermediate step, first build a function which calculates the entropy of a distribution. A suitable function interface is as follows:

```
function result=dm_entropy(inputlabels)
%
% inputlabels -> vector of labels
% returns -> entropy of labels
%
```

Once you have this function in place, test your function with a series of test label vectors, as shown:

```
ent=zeros(1,11);
for count=0:10
ent(count+1)=dm_entropy([repmat(1,1,count) repmat(0,1,10-count)]);
end
```

Be sure that you understand what the above few lines of code are doing. Finally, create a graph of the resulting entropy values:

```
plot(0:10,ent);
xlabel('Number of 1s')
ylabel('Entropy')
axis tight
```

and confirm that it looks like this:

3 Information Gain

Once the entropy function is ready, proceed to implement the information gain function. This function should have the following interface:

```
function result=dm_infogain(features,labels)
%
% features,labels -> vectors of features and corresponding labels
%
% Returns -> information gain
%
```

Implement this function. Broadly, it should perform the following steps:

1. Accept two vectors: one a list of instance features values, and another a list of labels corresponding to the each instance.
2. Calculate the overall entropy of the class labels.
3. Split the label vector into subsets based on the feature values, and calculate the entropy in each subset.
4. Calculate the expected entropy of the class labels given this feature, using the weighted averaging term shown in equation (1).
5. Return the difference between the two entropy values (which is the information gain)

Finally, create a simple test set as follows:

```
% Simple test set
features=[0 0 0 0 0 0 1 1 1 1 1 1;
0 1 0 1 0 1 0 1 0 1 0 1;
0 0 0 1 0 0 1 0 0 0 1 1];
labels=[1 1 1 0 1 1 1 0 0 0 1 1];
```

Use your function to test the information gain of all three features, and verify that they are $[0.093285, 0.093285, 0.011580]$ respectively.

Before proceding further, satisfy yourself that these information gain values match the relationships between the feature in question and the class labels. You can create other feature sets to check if the information gain works as expected. For example, what is the information gain when the class labels themselves are used as the features?

4 Decision tree

4.1 Introduction

Finally create a function for decision tree induction which uses the information gain function created above. Your function should have the following interface:

```
function result=dm_learntreeid3(features,labels)
%
% features=(dim x n) matrix of (dim x 1) feature vectors
% labels=vector of labels (1 x n or n x 1)
%
% Returns a tree structure
% only binary feature values supported for now... make it zero or one
%
```

As hinted above, the tree induction will be based on the ID3 algorithm, described below:

1. Given a set of records (each is a vector of feature values), calculate the information gain for each feature.
2. Choose the feature with the highest information gain, divide the set of records into subsets based on their values of this feature
3. For each subset repeat this entire procedure.

Termination criteria:

1. If all features have been used up, terminate (create a leaf labelled with the most common class label in the remaining records).
2. If remaining subset is "pure" (all class labels are the same), terminate and create leaf with that class label.

To facilitate your implementation, I have created a "skeleton" file - this is a rudimentary (but working) implementation which has had several chunks removed, and is included as part of the lab package (*dm_learntreeid3_skel.m*). Please study it and ensure that you understand all the parts.

There are also a number of octave features and commands that will be unfamiliar to you, in particular:

- Handling of a variable number of inputs via *varargin*.
- Octave "structures" (similar to structures in other programming languages, for e.g. *C*). In this case, the function returns a structure which can have up to three pieces of information: *result.feature* which is the feature to branch on; *result.branch0* which is the "left" branch and *result.branch1* which is the "right" branch. In some cases the return value can also be a numerical value (where it is a leaf).
- The idea of recursive programming (i.e. a function which calls itself) - this shold be familiar to most of you.
- The command *isnumeric* which allows the type of a variable to be tested. You may also want to investigate other similar commands such as "isbool" and "ischar".

The attached file has been divided into 5 blocks (clearly labelled in the code itself):

Block 1 This processes the inputs and checks if a list of "used" features has been provided.

Block 2 Calculates the information gain for each feature

Block 3 Picks the feature with the largest information gain, also checks for termination criterion (1).

Block 4 Split the available records based on the feature values, propagate down each branch (note the pair of recursive function calls). Also check for termination criterion (2).

Block 5 Tidies things up by (i) creating the struct which is to be returned by the function, and (ii) checking for an additional situation where both branches are leaves, and both have the same value.

4.2 Tasks

Your task then is to complete this skeleton function by filling out all the missing parts. These are labelled in the code (where it says "Implement code here". There are three tasks in total:

Task #1 Calculate the information gain for all the features, and store in a (1 x dim) vector "infogains" (you should use the function dm_infogain implemented previously).

Task #2 (A and B) Check for termination criteria (2) → i.e. if the resulting subset is already "pure", create a leaf. "A' and "B" refers to the left and right branch versions of this (should be identical).

Task #3 (A and B) If out of features, terminate and create a leaf with the most common class label.

Finally, to test that your function works, build a decision tree using the test data generated in section 3 earlier. You should see:

```
octave:> dm_learntreeid3(features,labels)
ans =
{
  feature =  1
  branch0 =
  {
    feature =  3
    branch0 =  1
    branch1 = 0
  }
```

```
  branch1 =
  {
    feature =  3
    branch0 = 0
    branch1 =  1
  }

}
```

Please study and ensure that you understand this output.

5 Comparison using WEKA

Start WEKA, and:

1. Load the data file "lab4_testdata.arff", which is provided in the lab package (feel free to view the file to satisfy yourself that it is the same data set).
2. Click on the "classify" tab, then on the "classifier" button and choose "trees → Id3".
3. Click on the "Start" button to initiate the training.

Look in the classifier output pane (you may need to scroll up if your window is small, etc., you should see this:

```
=== Classifier model (full training set) ===

Id3

Feature1 = 0
|  Feature3 = 0: yes
|  Feature3 = 1: no
Feature1 = 1
|  Feature3 = 0: no
|  Feature3 = 1: yes
```

Confirm that this is the same tree as was produced by your script.

6 Assignment

6.1 Tasks

For this assignment, you need to produce the following three functions/files:

- dm_entropy.m

- dm_infogain.m
- dm_learntreeid3.m

They need to follow the interfaces desribed in this document. *Please check that the output of your tree is correct before submission* (as described in section 4.2).

6.2 Submission guidelines

The deadline for submitting this assignment is **12pm, 20th of November 2013**. Late submissions will be rejected.

As usual, only electronic submissions will be accepted. The following is the submission procedure (which is almost the same as the previous laboratory):

1. Submit *only* the M-files required. These should be compressed into a single zip file, and sent as an attachment in your submission e-mails (along with any additional support functions which are required to run your code).
2. Send your solutions via e-mail to the TA, Bikash Joshi and *CC a copy to me*.
3. Format your subject line as follows: [CIS501] Fall 2013, Assignment #4 solution. Name: <Your name>
4. If you do not get an *acknowledgement e-mail* from the TA, please re-send the assignment.