2D1431 Machine Learning
Lab 1: Concept Learning & Decision Trees

Frank Hoffmann
e-mail: hoffmann@nada.kth.se

November 8, 2002

1 Introduction

You have to prepare the solutions to the lab assignments prior to the scheduled labs, which are mainly for examination. In order to pass the lab you present your program and answers to the question to the assistent. Labs can be presented in groups of two, however both students need to fully understand the entire solution and answers. It is also assumed that you complete the assignment on your own and do not use parts of someone else's code or solution. Violation of these rules may result in failing the entire course.
Before you start this lab you should be familiar with programming in MATLAB. In particular you should understand how MATLAB performs matrix computations, simple graphics, simple programming constructs, functions and the help system in MATLAB. If you want to learn MATLAB or refresh your knowledge I recommend that you carefully study the online guide *Practical Introduction to MATLAB* [4] and the *MATLAB Primer* [5] to which you find links on the course webpage.
In this lab you will use some predefined functions for building decision trees and analyzing their performance, but you will also have to program some computations yourself. During the examination with the lab assistent, you will not only present your results but also the code that you wrote.
It is assumed that you are familiar with the basic concepts of decision trees and that you have read chapters 2 and 3 on concept learning and decision trees in the course book *Machine Learning* [2].

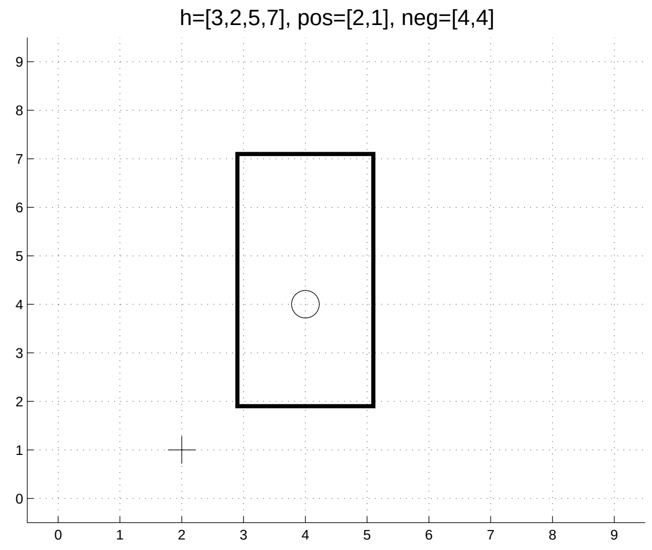

Figure 1: Example of a hypothesis $h = (3, 2, 5, 7)$ that is inconsistent with positive (+) $(2, 1)$ and negative (o) $(4, 4)$ examples.

2 Concept Learning

Concept learning is concerned with acquiring the definition of a general category given a sample of positive and negative examples of the category. The candidate elimination or version space algorithm searches trough a space of potential hypotheses, exploiting a general-to-specific ordering of hypotheses. The candidate elimination algorithm finds all potential hypotheses that are consistent with the observed training examples.

Definition: A hypothesis h is **consistent** with a set of training examples D if and only if $h(x) = c(x)$ for each example $< x, c(x) >\in D$.

Consider the instance space X consisting of integer points in the x-y plane $(0 \leq x \leq 9, 0 \leq y \leq 9)$ and the set of hypotheses H consisting of axes-parallel rectangles. More precisely, hypotheses are of the form $(a \leq x \leq c, b \leq y \leq d)$, where a, b, c and d are natural numbers in the range $\{0, ..., 9\}$. For example, a hypothesis $h = (a, b, c, d) = (3, 2, 5, 7)$ corresponding to a rectangle from lower left corner $(3, 2)$ to upper right corner $(5, 7)$ as shown in figure 1 would be inconsistent with a negative example $(4, 4)$ inside the rectangle $(3 \leq x \leq 5, 2 \leq y \leq 7)$. Vice versa, it is also inconsistent with a positive example $(2, 1)$ that lies outside the rectangle, as the hypothesis should encompass all positive examples.

Assignment 1:

Consider the version spaces with respect to a set of positive (+) and negative (O) training examples. Trace the S- (most specific) and G-(most general)-boundaries of the version space using the candidate-elimination algorithm for each new training instance. Write out the hypotheses that belong to the S- and G-boundary and plot them together with the set of examples using the MATLAB script `plot_clearn(pos, neg,sbound,gbound)` where `pos` and `neg` are sets of positive and negative examples specified by a vector of (x,y)-pairs, `sbound` and `gbound` are the S- and G-boundary specified by a vector of $(x_{min}, y_{min}, x_{max}, y_{max})$-quadruples that represent rectangles. In order to be able to use the predefined functions you have to add the directory /info/mi02/labs/lab1 to your MATLAB path.

```
>> addpath('/info/mi02/labs/lab1');
```

We start with no training examples at all. Obviously, the G-boundary (most general hypothesis) is the largest possible rectangle $G_0 = \{(0, 0, 9, 9)\}$. The most specific (least general) hypothesis is the empty set $S_0 = \{\emptyset\}$. In MATLAB we specify the examples and boundaries and visualize them.

```
>> pos=[];
>> neg=[];
>> gbound(1,:)=[0 0 9 9];
>> lbound=[];
>> plot_clearn(pos,neg,gbound,lbound);
```

Assume that our first instance is the point $(3, 3)$. We query an oracle for the classification of this example

```
>> oracle(3,3);
```

and obtain the answer that $(3, 3)$ is a positive example. For a positive example we expand the current (empty) S-boundary such that it includes the positive example. The new S-boundary $S_1 = \{(3, 3, 3, 3)\}$ is formed by the smallest rectangle that covers the single positive instance $(3, 3)$. Notice, that the hypothesis includes the corner points of the rectangle $(3, 3, 3, 3)$ even though its width and height are zero. As we have a positive example, the G-boundary $G_1 = \{(0, 0, 9, 9)\}$ does not change.

```
>> pos(1,:)=[3 3];
>> lbound(1,:)=[3 3 3 3];
>> plot_clearn(pos,neg,gbound,lbound);
```

The next query point $(5,5)$ is another positive example.

```
>> oracle(5,5)
```

Again we have to expand the S-boundary, so that it includes the new positive instance. The rectangle $S_2 = \{(3,3,5,5)\}$ is the smallest (most specific) rectangle that still contains both positive instances. Again, the G-boundary does not change.

```
>> pos(2,:)=[5 5];
>> lbound(1,:)=[3 3 5 5];
>> plot_clearn(pos,neg,gbound,lbound);
```

We continue with a new query point $(1,2)$ which turns out to be a negative instance.

```
>> oracle(1,2)
```

Therefore we have to reduce our overly optimistic assumption about the G-boundary. There are two maximally large rectangles $G_3 = \{(0,3,9,9),(2,0,9,9)\}$, that do cover the two positive examples, but do not contain the negative instance. The S-boundary does not change as it is not contradicted by a negative example.

```
>> neg(1,:)=[1 2];
>> gbound(1,:)=[0 3 9 9];
>> gbound(2,:)=[2 0 9 9];
>> plot_clearn(pos,neg,gbound,lbound);
```

Continue with the candidate elimination algorithm algorithm by querying the following sequence of instances $(4,8)$, $(7,2)$ and $(8,5)$. For each new instance specify the resulting G- and S-boundary $(G_4, S_4, \ldots, G_6, S_6)$ and verify that they are correct by plotting the instances and boundary.
$G_4 = ...$
$S_4 = ...$
$G_5 = ...$
$S_5 = ...$
$G_6 = ...$
$S_6 = ...$
Suppose that at this stage (G_6, S_6) you may suggest a new example and ask the oracle for its classification. Suggest a query point guaranteed to reduce the size of the version space, regardless of how the oracle classifies it.
`good query=...`

name	# train	# test	# attributes	# classes
monk1	124	432	6	2
monk2	169	432	6	2
monk3	122	432	6	2

Table 1: Characteristic of the three MONK datasets

Suggest a query that certainly will neither reduces the S- nor the G-boundary
`bad query=...`

Continue querying the oracle until the candidate elimination algorithm uniquely identifies the correct hypothesis. How do you know that the candidate elimination algorithm found the correct hypothesis? Try to use as few queries as possible.
`query points=...`
`final hypothesis=...`

Now assume that you are a teacher, attempting to teach a particular target concept (e.g. $3 \leq x \leq 5, 2 \leq y \leq 7$). What is the smallest set of training examples you can provide, such that the candidate elimination algorithm will perfectly learn the target concept?
`minimal set of examples=...`

3 MONK datasets

This lab uses the artificial MONK dataset from the UC Irvine repository [1]. It consists of three separate datasets monk1, monk2 and monk3. The MONK's problems are a collection of three binary classification problems monk1, monk2 and monk3 over a six-attribute discrete domain (see table 1). The attributes may take the following values:

attribute 1 : 1, 2, 3
attribute 2 : 1, 2, 3
attribute 3 : 1, 2
attribute 4 : 1, 2, 3
attribute 5 : 1, 2, 3, 4
attribute 6 : 1, 2
Therefore, each dataset consists of 432 instances.

The *true* concepts underlying each MONK's problem are given by:
MONK-1: (attribute_1 = attribute_2) or (attribute_5 = 1)
MONK-2: (attribute_n = 1) for EXACTLY TWO choices of n $\in \{1, 2, ..., 6\}$
MONK-3: (attribute_5 = 3 and attribute_4 = 1) or
(attribute_5 != 4 and attribute_2 != 3)

MONK-3 has 5% additional noise (misclassifications) in the training set. Can you guess which of the three problems is most difficult for a decision tree algorithm to learn?
Each dataset is divided into a training and test set, where the first one is used for learning the decision tree, and the second one to evaluate its classification accuracy. The datasets are available in the directory `/info/mi02/labs/datasets/` as `monks-1.train`, `monks-1.test`, `monks-2.train`, `monks-2.test`, `monks-3.train` and `monks-3.test`. Each row contains one instance. The first column contains the target class (0 or 1), and the next six columns the attributes. You can ignore the last column containing the string $data_i$ which simply enumerates the instances.
For the sake of convenience you find a MATLAB script `readmonks.m` that loads the datasets into MATLAB.

```
>> readmonks
>> who
```

The data sets are available in the MATLAB variables `monks_1_train`, `monks_1_test` etc. . Notice, that for compatibility reasons the class is stored in the last column and the first six columns contain the attributes.

4 Decision Trees

In order to decide on which attribute to split, the decision tree learning algorithms such as ID3 and C4.5 [3] use a statistical property called *information gain*. It measures how well a particular attribute distinguishes among different target classifications. Information gain is measured in terms of the expected reduction in the *entropy* or impurity of the data. The entropy of an arbitrary collection of examples is measured by

$$Entropy(S) = -\sum_i p_i \log_2 p_i \qquad (1)$$

in which p_i denotes the proportion of examples of class i in S. The monk dataset is a binary classification problem (class 0 or 1) and therefore equation

1 simplifies to

$$Entropy(S) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \qquad (2)$$

where p_0 and $p_1 = 1 - p_0$ are the proportions of examples belonging to class 0 and 1.

Assignment 2: Write a function `ent(data)` that computes the entropy of a collection of examples stored passed as the parameter `data`. Hint: Use the MATLAB function `log2` for computing the base 2 logarithm. Use this function to compute the entropy of the three monk training data sets.

```
ent(monks_1_train)=...
ent(monks_2_train)=...
ent(monks_3_train)=...
```

The information gain measures the expected reduction in impurity caused by partitioning the examples according to an attribute. It thereby indicates the effectiveness of an attribute in classifying the training data. The information gain of an attribute A, relative to a collection of examples S is defined as

$$Gain(S,A) = Entropy(S) - \sum_{k \in values(A)} \frac{|S_k|}{|S|} Entropy(S_k) \qquad (3)$$

where S_k is the subset of examples in S for the attribute A has the value k. For the purpose of extracting the subset of examples use the two following MATLAB functions provided in the course directory.
`values(data,i)` returns a vector of unique values for the i-th attribute in data.
`subset(data,i,j)` returns the subset of examples in data for which attribute i has value j.

Assignment 3: Write a function `gain(data)` that computes the information gain of all the attributes of a collection of examples. Use this function to compute the information gain of the six attributes in the three monk training data sets.

```
gain(monks_1_train)=...
gain(monks_2_train)=...
gain(monks_3_train)=...
```

Based on the results, which attribute should be used for the splitting the examples at the root node?
Split the data into subsets according to the selected attribute and compute the information gains for the nodes on the next level of the tree. Which attributes should be tested for these nodes?
For the *monks_1_train* data draw the decision tree up to the first two levels and assign the majority class of the subsets that resulted from the two splits to the leaf nodes. Use the predefined function `majority_class(data)` to obtain the majority class for a set of instances.
Now compare your results with that of a predefined routine for ID3. Use the function `T=build_tree(data)` to build the decision tree and the function `disp_tree(T)` to display the tree. Notice, that in the display a *no* corresponds to class 0, *yes* to class 1. If you set the global variable `max_depth=2`, you can limit the depth of the decision tree built. Do not forget to set this parameter back to its original value of 10 afterwards.
Assignment 4:
Build the full decision trees for all three Monk datasets using the some predefined routines for ID3.

`calculate_error(T,data)`
computes the classification error over some data.

For example to built a tree for monks_1, display it and compute the test set error for monks_1

```
>> T=build_tree(monks_1_train);
>> disp_tree(T);
>> error=calculate_error(T,monks_1_test);
```

Compute the train and test set errors for the three Monk datasets for the unpruned trees.

E(monks_1_train)=...
E(monks_1_test)=...
E(monks_2_train)=...
E(monks_2_test)=...
E(monks_3_train)=...
E(monks_3_test)=...

The idea of reduced error pruning (see section 3.7.1.1 [2]) is to consider each node in the tree as a candidate for pruning. A node is removed if

the resulting pruned tree performs no worse than the original tree over a validation set not used during training. In that case the subtree rooted at that node is replaced by a leaf node, to which the majority classification of examples in that node is assigned.
`T=prune_tree(T,data)`
prunes a tree previously generated with `build_tree` using the examples in `data`.
For the purpose of pruning, we have to partition our original training data into a training set for building the tree and a validation set for pruning tree. Notice, that using the test set for validation would be cheating as we then are no longer able to use the test set for estimating the true error of our pruned decision tree. Therefore, we randomly partition the original training set into training and validation set.

```
>> [n,m]=size(monks_1_train);
>> p=randperm(n);
>> frac=0.7;
>> monks_1_train_new=monks_1_train(p(1:floor(n*frac)),:);
>> monks_1_prune=monks_1_train(p(floor(n*frac)+1:n),:);
>> T1=build_tree(monks_1_train_new);
>> T1p=prune_tree(T1,monks_1_prune);
```

Where the fraction `frac` of instances is used for building the tree and the remaining examples for pruning it.

Assignment 5:
Evaluate the effect of pruning on the test error for the monk_1 and monk_3 datasets, in particular determine the optimal partition into training and pruning by optimizing the parameter `frac`. Plot the classification error on the test sets as a function of the parameter `frac` $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

E(monks_1_test) for frac {0.3,0.4,0.5,0.6,0.7,0.8} =...
E(monks_3_test) for frac {0.3,0.4,0.5,0.6,0.7,0.8}=...
optimal value of frac for monks_1=...
optimal value of frac for monks_3=...

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[2] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[3] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993.

[4] Mark S.Gockenback. A practical introduction to matlab, 1999. http://www.math.mtu.edu/~msgocken/intro/intro.html.

[5] Kermit Sigmon. Matlab primer, 1993. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/misc/matlab/common/www/matlab_primer.pdf.