

CS446 Homework 1

- Due Wednesday, September 20th. Please by all means hand your hardcopy to the TA *at the beginning of the class when you enter the classroom* (since the TA will not be available after the class).
 - Late homework should be dropped in the box in 3332 SC. Slide under the door if it is not open.
 - The file for problem 3 is also due on the same day. It should be sent by email as an attachment to `cs446.fa06@gmail.com`. You will receive a confirmation email within 24 hours; if not, contact the TA immediately.
 - Feel free to talk to your classmates about the homework. We are more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, write down your solution yourself and master the material to solve similar problems unaided. Keep the solution brief and clear.
 - Please, no handwritten solutions. Be sure your name appears on the top of each page and that your pages are stapled together.
 - Please present your algorithms in both pseudocode and English. That is, give a precise formulation of your algorithm as pseudocode and also explain in one or two concise paragraphs what your algorithm does. Be aware that pseudocode is much simpler and more abstract than real code. Take a look at the textbook pseudocode (e.g. Table 2.5 on page 33) to get an idea about the appropriate level of abstraction.
1. [Representing Boolean Functions—10 points] (Based on Mitchell, exercise 3.1) Give decision trees to represent the following Boolean functions:
 - (a) $\neg A \vee \neg B \vee C$ [3 points]
 - (b) $A \wedge (\neg B \vee C)$ [3 points]
 - (c) $(A \oplus B) \vee (C \wedge D)$ [4 points]
 2. [Space Complexity of Decision Trees—15 points] Let x be a vector of n Boolean variables and k represent the number of relevant variables in the target function, ($k \leq n$).

- (a) Let D_k be the class of monotone k -disjunctions (disjunction on k of the n variables) over (x_1, x_2, \dots, x_n) . State the size of the smallest possible consistent decision tree for D_k in terms of n and k . Describe the shape of the resulting tree. [3 points]
 - (b) Let C_k be the class of monotone k -conjunctions (conjunction on k of the n variables) over (x_1, x_2, \dots, x_n) . State the size of the smallest possible consistent decision tree for C_k in terms of n and k . Describe the shape of the resulting tree. [3 points]
 - (c) Let P_k be the class of k -parity functions (parity function on k of the n variables) over (x_1, x_2, \dots, x_n) . The (odd) parity function evaluates to 1 if there are an odd number of 1's in the feature and evaluates to 0 if there are an even number of 1's in the feature vector. State the size of the smallest possible consistent decision tree for P_k in terms of n and k . [3 points]
 - (d) What do these results imply about the application of decision tree learning for learning functions in D_k , C_k , and P_k ? [6 points]
3. [Implementing Decision Trees—75 points] In this programming assignment, you will implement a simple ID3-like decision tree learning algorithm and test it on a data set. We will use a data set similar to the one from the Badges Game. You may use the programming language of your choice. Please note that your actual implementation of the decision tree algorithm should be independent from the feature extraction mechanism as we may use it as part of other assignments. In particular, we may be requiring you to reuse this generic decision tree code for rules extraction and boosting later this semester.

The data is available from the course web site:
<http://www.cs.uiuc.edu/class/fa06/cs446/>

The data is given as a list names preceded by a label '+' or '-'. It is split into two sets, **Train** and **Test**, consisting of 80% (235 examples) and 20% (59 examples) of the data, respectively. Altogether there are 134 positive examples (107 in **Train**) and 160 negative examples (128 in **Train**).

Part of your assignment is to pre-process the data and extract features from it. Use *22 features*. 20 of these features represent the characters in various positions in the two strings. For example, the feature $X(i, j)$ stands for the i th character in the j th string ($i = 1, 2, \dots, 10; j = 1, 2$). You should clean the data so that only the first and last names of each person are used. Ignore middle initials and names. Do not distinguish between lowercase and uppercase letters. You will need an additional symbol to represent whitespace and you should pad each string to a length of 10. Note that some strings are longer than 10 characters, in which case you should just ignore all characters beyond the tenth. In this way, these 20 features have 27 possible values. The remaining two features describe the length of the first and second strings (which can be greater than 10). You should decide how to handle these two features.

You are required to begin with this set of features as a baseline. However, if you would like to use different features, please do so as well. If you choose to do so, describe the set of features you use and compare the results you get from the basic set with those you get with the new set of features.

Your Program

You should write routines to do the following:

- Pre-process the data.
- Grow a decision tree using the information gain splitting heuristic.
- Display a decision tree.
- Evaluate a decision tree on a data set.

Each node in the decision tree corresponds to a feature. You may choose the nodes to be binary features of the form *feature = value* (for example, we might check whether feature $X(3, 1)$ has the value d) or you may allow them to accept more values. You should also handle the integer valued string length features by defining intervals. The choice of intervals, of course, is up to you.

Other than that, there are a few decisions you need to make in your implementation. Make sure you explain in your report what your algorithm does.

Before you run your program on the data, you may wish to test it on a small set of examples for which you can construct the tree yourself (e.g., the data from Mitchell, exercise 3.2) for debugging purposes. You may also consider testing out your decision tree on the original Badges Game data, which is found in the handouts.

Once your code is working, build a decision tree using `Train` and test your tree on `Test`. When running your code, try (at least) the following two options:

- Limit the depth of the tree. Do not allow the tree to be deeper than, e.g., 3.
- Construct a learning curve. Construct a series of random subsets of `Train`, containing 40, 60, 80, 120, 160, and 235 examples. For each of these sub-datasets construct a decision tree and test the accuracy of the tree on `Test`. You should report your results in a graph of number of examples vs. testing accuracy. You should average these values over multiple runs to smooth out sampling errors, and report their variances in a table. In general, avoid doing any repetitive tasks by hand; you may find the following resource on writing shell scripts useful if you've never done it before.

(You can automate your runs however you like, this is just FYI)
http://www.dartmouth.edu/~rc/classes/ksh/print_pages.shtml

It is sufficient to present your decision tree in this fashion:

```
feature 0 == x
  feature 1 == y
    feature 2 == z
      class = +
    feature 2 != z
      class = -
  feature 1 != y
    class = +
feature 0 != x
  feature 1 == r
    class = +
  feature 1 != r
    class = -
```

(Of course, use more descriptive feature names here so your output is comprehensible.)

Your routine for testing the accuracy of a decision tree should print the results in the following form.

| | Test Cases | True | False |
|---|------------|------|-------|
| + | 75 | 70 | 5 |
| - | 75 | 45 | 30 |

This says that:

- 70 test examples were predicted to belong to class + and actually did belong to class + (true positives).
- 5 examples were predicted to be in class + but were actually in class - (false positives).
- 45 test examples were predicted to belong to class - and actually did belong to class - (true negatives).
- 30 examples were predicted to be in class - but were actually in class + (false negatives).

Finally, report the *error rate*. The error rate is the sum of the errors (here, $30 + 5$) divided by the total number of examples (here, 150), in this case 23%.

What to turn in

- Include your name and email in a file called `README`.
- Create a file called `SCRIPT` containing a log of the runs used in the final discussion. This can be generated using the `script(1)` command on Unix. If people are coding on Windows, cut and paste a log of the relevant runs in a file called `SCRIPT`.
- Submit the report as *hardcopy*. Limit the text of the report to a reasonable length. Please include a printout of your source code.
- Create the tarball so that it will unpack into a new directory named after your NetID. For example, if my NetID is `jdoe`, I copy all the source files, `README`, and `SCRIPT` into a directory called `jdoe-hw1`. Then archive:

```
tar cvf jdoe-hw1.tar jdoe-hw1
```

Then verify:

```
tar tvf jdoe-hw1.tar
```

Exclude executables and object files from the submission.

- The tarball will be submitted electronically by email as an attachment to `cs446.fa06@gmail.com`.

You must include the following in your report:

- For the complete `Train` set, display at least three decision trees, two limiting the depth as described above (trying different depths). For each one, run the evaluation routine and present the error information.
- For the learning curve experiments, display a table of pairs (number of training examples, error rate).
- If you opted to use other sets of features, compare the results.

Grading

- Pre-process the data [10 points]
- Implementation to grow tree using the information gain splitting heuristic [30 points]
- Display Tree [10 points]
- Evaluation [20 points]
- Report - explain implementation, language used, instructions to run [5 points]