Suggested Class Projects

1. Spam email filter – A comparison of Naïve Bayes and Decision Trees

A spam filter automatically classifies incoming email messages as 'spam' or 'non-spam'. First the spam filter must be trained to identify spam emails. Essentially, give the filter a pile of emails that are negative examples (non-spam) and a pile of emails that are positive examples (spam). The filter will then analyze the piles for clues as to what makes the spam and non-spam emails different. For example: different words, differences in the mailer headers and content style. The system then uses these clues to examine new, possibly unseen messages.

For this project you have to implement a spam filter that uses two different supervised machine learning approaches: Naive Bayes Classifier and Decision Tree Classifier (eg, C4.5) to classify a new given email as "legitimate" and "spam".

Data: http://spamassassin.org/publiccorpus
Processing:
- Build a feature vector based on a list of about 150 most frequently used words in a chunk of spam emails and transform each email into a vector of words or phrases with binary values.
- Split data into training and test set; make sure they don’t overlap;
- Apply each Machine Learning algorithm and compute Precision and Recall metrics;

Suggested bibliography:

2. *Konane – The Hawaiian Checkers*

Implement a program (in the programming language of your choice) to play the game of Konane (Hawaiian Checkers). The game is typically played on an 8X8 board of “white” and “black” pieces as shown (using X for “black” and O for “white”).

```
   1 2 3 4 5 6 7 8
  8  OXOXOXOXO
  7  OXOXOXOX
  6  XOXOXXOXO
  5  OXOXOXOX
  4  XOXOXOXO
  3  OXOXOXOX
  2  XOXOXOXO
  1  OXOXOXOX
```

First, the black player removes a black piece either at position 18, 81, 45, or 54. Next the white player removes a white piece adjacent to the space created by the first move. Then the players alternate moves, each jumping one of his/her own pieces over one horizontally or vertically adjacent opponent piece, landing in a space on the other side, and removing the jumped piece. If desired, this may be continued in a multiple move, as long as the same piece is moved in a straight line. For example, after the following moves:

<table>
<thead>
<tr>
<th>“black”</th>
<th>“white”</th>
</tr>
</thead>
<tbody>
<tr>
<td>removes &lt;5, 4&gt;</td>
<td>removes &lt;5, 5&gt;</td>
</tr>
<tr>
<td>moves &lt;3, 4&gt; to &lt;5, 4&gt;</td>
<td>moves &lt;5, 7&gt; to &lt;5, 5&gt;</td>
</tr>
<tr>
<td>moves &lt;3, 2&gt; to &lt;3, 4&gt;</td>
<td>moves &lt;5, 3&gt; to &lt;3, 3&gt;</td>
</tr>
<tr>
<td>moves &lt;7, 6&gt; to &lt;5, 6&gt;</td>
<td>moves &lt;6, 4&gt; to &lt;6, 6&gt;</td>
</tr>
<tr>
<td>moves &lt;7, 8&gt; to &lt;7, 6&gt;</td>
<td>moves &lt;2, 4&gt; to &lt;4, 4&gt;</td>
</tr>
</tbody>
</table>

the board looks like

```
   1 2 3 4 5 6 7 8
  8  XOXOXOXOXO
  7  OXOXOXO..|
  6  XOXO.OXO |
  5  OX..OX.X |
  4  XO..XOXO |
  3  O.O.OXOX |
  2  XOX.XOXO |
  1  OXOXOXOX |
```
Your program should use the MINIMAX algorithm with ALPHA-BETA pruning. For each game, your program should also provide the following information:

1. The number of times a static evaluation was done.
2. The average branching factor.
3. The number of cut offs that took place.

You are advised to used the same coordinate system as shown. Play the game by varying the depth of search (from 1 to 6) and plot the above results. If you like you may use a graphical front end, rather than the text version shown above. However, make sure that you complete the program as described. If time permits, you may work on the graphics part.

Extra credit will be given for the implementation of a graphical version.

Note: To play the game Konane, go to the following website:
http://www.palmgear.com/index.cfm?fuseaction=software.showsoftware&prodid=6708

Report Format and Source Code

At the end of the semester you have to provide a printed report (no later than May 7, 2004) and the project complete source code (on April 29, 2004). Your report must include the following:

(5 points) **Introduction**: A brief description of the problem that your system is supposed to solve and the behavior that it is intended to exhibit. Any additional information necessary to understand your project should also be included here. For example, for a game-playing program, a brief summary of the game rules should be included.

(15 points) **Approach**: An overview of the approach you took to the problem, describing the major data structures and algorithms used by your system. There should be enough information in this overview to understand how your system works at a high level without looking at the source code. For example, a detailed description of the static evaluator for a game-player or the rule representation for a rule-based system should be given.

(45 points) **Sample Runs**: Sample runs of your system showing each of its capabilities. There should be sufficient examples to illustrate the full range of behavior of your system. Sample runs should be clearly organized and
completely explained. For example, for a game-playing program, you should show some sample games (or highlights of games if large numbers of moves are involved) with commentary that emphasizes particularly good or bad moves made by your program and why they were made. In addition, the particular notation that you use to describe board positions should be explained.

(10 points) **Summary and conclusions:** What features of your system you think are worth emphasizing? What features do you think can/should be improved? What was easy and hard? What do you feel you’ve learned during the course of this project?

(25 points) **Source code:** A complete source code zipped file for your system must be included. Regardless of the programming language that you use, your code should be well-organized and clearly documented so that your implementation is easy to understand. You should also include a README file with information on how to run the program.