Assignment 5:
Classification Rule Learning

Deadline: Wednesday, May 16, 2012

This assignment will contribute 12 points to your final grade.

Description
In this assignment, you will implement the Incremental Reduced Error Pruning (IREP) method of inferring classification rules, and apply it to a mushroom classification problem. The Mushroom dataset is available from the UCI Machine Learning repository\(^1\). This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms, each species being identified as edible or poisonous. Read the information on the website for more details. There is no simple rule for determining the edibility of a mushroom.

The performance of IREP often depends on what metric you use to evaluate the “goodness” of a pruning operation. You will implement two different metrics, and observe how they perform when applied to the Mushroom data. Further, the presence or absence of noise in the data can heavily impact the runtime of some rule induction algorithms. You will perform experiments to show that IREP is rather robust in that regard.

Task 1: Implementing IREP
Each instance in the data set is labeled as e (edible) or p (poisonous). Assume ‘e’ as the positive class. You will implement the two-class version of IREP to identify classification rules to predict the edibility of a mushroom, given the 22 features describing its physical structure. The pseudo-code for the method is as follows.

\[
\text{IREP ( PosEx, NegEx )}
\]
\[
RS \leftarrow \text{empty rule set}
\]
\[
\text{while } PosEx \text{ is not empty}
\]
\[
\text{split } PosEx \text{ into } PosGrow \text{ and } PosPrune \text{ in a 2:1 ratio}
\]
\[
\text{split } NegEx \text{ into } NegGrow \text{ and } NegPrune \text{ in a 2:1 ratio}
\]
\[
R \leftarrow \text{GrowARule (PosGrow, NegGrow)}
\]
\[
R_{\text{pruned}} \leftarrow \text{PruneARule (R, PosPrune, NegPrune)}
\]
\[
\text{if } \text{worth}(R_{\text{pruned}}, \text{PosPrune}, \text{NegPrune}) \leq \text{accuracy}_{\text{default}} \text{ return } RS
\]
\[
\text{add } R_{\text{pruned}} \text{ to } RS
\]
\[
\text{remove instances from } PosEx \text{ and } NegEx \text{ that are covered by } R_{\text{pruned}}
\]
\[
\text{return } RS
\]

PosEx and NegEx are the positive (edible mushroom) and negative (poisonous mushroom) instances in the data set. Write a function GrowARule that uses separate-and-conquer to create a rule to predict the positive class using PosGrow and NegGrow (detailed below). Your will also write another function PruneARule that will prune the rule generated by GrowARule using the pruning set—PosPrune and

\(^1\)http://archive.ics.uci.edu/ml/datasets/Mushroom
**NegPrune.** The function **worth** evaluates the goodness of a rule on the pruning set. You will implement two different variants of this function. Each variant takes as input a rule $R$, a pruning set $PS_{pos}$ with positive examples and a pruning set $PS_{neg}$ with negative examples.

1. **worth-A:** This is the simple accuracy measure. If $p$ is the number of instances covered by $R$ in $PS_{pos}$ and $n$ is the number of instances covered by $R$ in $PS_{neg}$, the worth of the rule is given as
   $$\frac{p}{p + n}.$$

2. **worth-B:** This measure accounts for the number of negative instances not covered by a rule. If $P$ is the total number of instances in $PS_{pos}$ and $N$ is the total number of instances in $PS_{neg}$, then the worth of a rule is given as
   $$\frac{p + (N - n)}{P + N}.$$

Note that $p$ and $n$ can both be zero (a rule with no coverage in the pruning set). In this case, use 0.5 as the value returned by **worth-A.** The accuracy default value is 0.5 when using **worth-A,** and $\frac{N}{P + N}$ when using **worth-B.**

**GrowARule** would use separate-and-conquer to generate a rule to predict the positive class. The pseudo-code for the function is as follows.

---

**GrowARule** ( $PosGrow$, $NegGrow$ )

Create a rule $R$ with an empty left-hand side

for each attribute $A$ not mentioned in $R$ and each value $v$ of the attribute that appears in $PosGrow$

\begin{align*}
&\text{consider adding the condition } A = v \text{ to the left-hand side of } R \\
&\text{select } A \text{ and } v \text{ to maximize the information gain, given as} \\
&p_a \left( \log \frac{p_a}{p_a + n_a} - \log \frac{p_b}{p_b + n_b} \right) \\
&\text{where } p_a \ (n_a) \text{ is the number of instances covered by rule } R \land (A = v) \text{ in } PosGrow \ (NegGrow), \text{ and } p_b \ (n_b) \text{ is the number of instances covered by rule } R \text{ in } PosGrow \ (NegGrow) \\
&R \leftarrow R \land (A = v) \\
&\text{remove the instances not covered by } R \text{ from } PosGrow \text{ and } NegGrow \\
\end{align*}

while $NegGrow$ is not empty and there are more attributes to use

return $R$

---

**PruneARule** would prune a rule based on it performance on the pruning set, and guided by one of the two worth functions. The pseudo-code for the function is as follows.

---

**PruneARule** ( $R$, $PosPrune$, $NegPrune$ )

while there is more than one test condition in $R$

$$R_{pruned} \leftarrow R \text{ with the last test condition removed}$$

if $\text{worth}(R_{pruned}, PosPrune, NegPrune) < \text{worth}(R, PosPrune, NegPrune)$

---
Again, **worth** can be one of the two variants. We will use the following naming in subsequent tasks.

- **IREP-A**: IREP with **worth-A** as the worth function
- **IREP-B**: IREP with **worth-B** as the worth function
- **PRISM**: IREP without the pruning operations – add the rule $R$ returned by `GrowARule` to $RS$ (no pruning step), remove instances from $PosEx$ and $NegEx$ that are covered by $R$, and then continue on the while loop

Split the data set into $PosEx$ and $NegEx$ by ‘e’ as the positive class and ‘p’ as the negative class. The class labels are in the first column of the data set.

**Task 2: Impact of training size**

Your first task is to run IREP-A, IREP-B and PRISM on training data sets of different sizes and observe the difference in performance. For this, you would use $frac$ fraction of the $PosEx$ and $NegEx$ sets as training data, and the remaining as testing data. Try values such as 0.01, 0.02, 0.05, 0.075, 0.1, 0.3, 0.5, 0.7 for $frac$.

For each value of $frac$ and a given method:

1. Create the rule set using the training part of the data. Note the runtime of the execution
2. Do a prediction on the test data (you will have to implement a small `predict` function for this)
3. Note the error rate
4. Repeat the above three steps at least 10 times
5. Average the time and error rate

Generate plots of the error and execution time of the three methods, with respect to the number of instances in the training data. Discuss the behavior in the report. Does any of the worth functions have an advantage in a specific case? Does training set size influence method performance (accuracy and time)?

**Task 3: Impact of noise**

In this part of the assignment, you will use a 60:40 split of the data for training:testing purposes. Add noise to the training data and perform steps 1-5 as discussed in Task 2. Try three different noise levels — 10%, 20% and 30%. For a given noise level, you would randomly choose that percentage of the training data and invert the class label ($e \leftrightarrow p$) on those instances. Generate plots of the error and execution time of the three methods, with respect to the noise level in the training data. Discuss the behavior in the report. Does any of the worth functions have an advantage in a specific case? Does training set size influence method performance?

**Report**

Your report is **NOT** a collection of statements corresponding to the discussion asked from you. It must be complete in itself, primarily including the problem you are solving, description of the data, the method employed, relevant R code, the experiments performed, the results you observed, informative (with labels, captions, and the remaining glitter!) and readable plots, and most importantly, a discussion along the lines of what is asked in the tasks. You are strongly encouraged to include your own observations and discussions outside of what is asked. Tell me a story!
Grading

This assignment will be graded out of 120 points. The points split is as follows:

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>25</td>
<td>25</td>
<td>30</td>
</tr>
</tbody>
</table>

The points you receive in each task will depend on how accurately you complete the task. In addition, your R code should be clean and well-commented. Sticking to directions is important! Additional explorations (outside of what is asked in the tasks) is most welcome. The report will be graded based on content, structure, formatting (I do not want to see half of a page empty because the plot does not fit there — reorganize!!), spelling, grammar, and other usual characteristics of a good scholarly document.

Submission

Submit the pdf report (not the Latex or Lyx source) and the R code into Blackboard by Wednesday, 16 May 2012, 11:59 PM. Refer to the late policy on the course website.

You must work alone on this assignment.