Multipipes: Exploring Disjuntive Classifications in Hyperpipes

[In an exciting manner] *

Aaron Riesbeck
West Virginia University
100 Address Lane
Morgantown, WV 26505
ariesbeck@theriac.org

Adam Brady
West Virginia University
100 Fake St.
Morgantown, WV 26505
adam.m.brady@gmail.com

ABSTRACT

This paper explores classifiction with disjunctive sets using a modified form of HyperPipes called MultiPipes. Rather than apply HyperPipes it's intended sparse datasets, we find that it's application to non-sparse, many-class datasets typically results in several tied classification scores which we then union into a disjunction. This union presents interesting possibilities in it's high accuracy in containing the target class. Although we initially cannot predict single classes, we find that these disjunctions often eliminate large portions of possible classes. Essentialy we aren't certain what the class is, but we are very certain of what the class is not. The rest of the paper explores two alternative strategies with MultiPipes. The first involves methods of reducing the disjunctive sets to single classifications. The second considers growing the disjunctive sets to optimize the accuracy of containment vs. set size.

Keywords

HyperPipes, disjoint sets, LaTeX, multiple classes, indecisive learners

1. INTRODUCTION TO HYPERPIPES

Background on hyperpipes, description of algorithm, benefits, trade-offs, relevant applications (spare datasets)

1.1 Pseudocode for hyperpipes

1.2 The Problem with HyperPipes

On non-sparse datasets you get lots of ties, bra'h.

1.3 Patching HyperPipes

Plumbing reference

Program 1 HyperPipes Pseudo Code.

```
while X < numberOfTrials do
  tasks := generateTasks(size)
  teams := generateTeams(tasks, teamSize)
  teams.applyDependancy(interDependancy)
  teams.applyCriticality(criticality, modifier)
  stoppingAt := iterationsToComplete(2 to 6)
  while iteration < stoppingAt do
    for all team in teams do
      team.budget+=(TotalCost/6)
      AvailableTasks := null
      for all task in team do
        if noDependencies&&no Children then
          if notCompleted then
            AvailableTasks.append(task)
          end if
        end if
      end for
      AvailableTasks.applySortingPolicy(type)
      for all task in AvailableTasks do
        if budget − task.cost) < 0 then
          break
        else
          budget := budget − task.cost
          task.completed := true
        end if
      end for
      if budget > 0 then
        if AvailableTasks is empty then
          budget := 0
        end if
      end if
      team.discoverNewTasks(λ)
      for all task ∈ team do
        change := (N(0, σ) ∗ culture)
        task.value += (maxTaskValue ∗ change)
      end for
    end for
  end while
end while
```

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

Explain appending fix

2. NARROWING VS. CLASSIFYING
Why narrow when you can classify?

3. PRELIMINARY RESULTS
Description of results

- incremental learning
- batch learning
- weighted distance
- centroids via overlap
- increasing alpha

3.1 Disjoint Learning
nb vs. multipipes on >1 dataset (incremental) nb vs. multipies on >1 dataset (batch) (see menzies.us/iccle/?nb chart for dataset scores comparison)

size of sets returned relative to number of total classes

3.2 Breaking the Ties
Description of weighted distance measure

graph of weighted distance classification accuracy

Description of centroid acquisition from overlap

graph of centroid learning results

3.3 Casting a wider net
Description of alpha value

Purpose of alpha value for expanding class set

Results of expanding alpha (graph)

Analysis of growth in enclosure with alpha changes

4. PRELIMINARY CONCLUSIONS
WE CONCLUDE

4.1 References
Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.