Part VI

AI Workflows
Machine Learning Workflows
Simple Examples of Workflows Available for Experimentation in Wings/Pegasus

1. **Very specific, limited applicability**
   - **Weather-SM-2007-Data.csv** → **ID3-PREDICT** → **Weather-SM-2007-Model.csv**
   - **ClassIndex = 5**

2. **Very general, very broad applicability**
   - **Training Data** → **ID3-PREDICT** → **Model**
   - **Test Data** → **ID3-CLASSIFIER** → **Classification**

3. **Very general, very broad applicability**
   - **Training Data** → **DISCRETIZE** → **SAMPLE** → **BAYES-PREDICT** → **Model**
   - **Test Data** → **BAYES-CLASSIFIER** → **Classification**

4. **General but customized to a specific use**
   - **Weather-SM-2007-Data.csv** → **SAMPLE** → **Weather-SM-2007-DataSample.csv**
   - **Sampling Interval = 20**
   - **ClassIndex = 5**
   - **Weather test data** → **ID3-PREDICT** → **Weather-SM-2007-Model.csv**
   - **Weather prediction**
A Workflow for a New Method Combining Ensemble Machine Learning with k-Fold Cross-Validation [Caruana 07]
More Machine Learning Workflows

Beacon Warning:
Finding suspicious groups in the Hats simulator

Joshua Moody

Center for Research on Unexpected Events
USC Information Sciences Institute
Hats Simulator [Cohen et al 02]

- Hats is a society in a box.
  - hats: {benign | terrorist | covert}
  - organizations: groups of hats
  - capabilities: information or item(s)
  - beacons: ports, buildings, landmarks
  - meetings: {malicious | benign}
  - trades: exchange of information or item(s)

- Beacon Attack
  - a malicious meeting on a beacon
  - and the participants carry the required capabilities that match the vulnerabilities of the beacon.
Workflow Inputs

- Hats Meeting data
  - meeting participants
  - ~ 500 M b - ~ 5 G

- Watch List
  - known terrorists

- Capabilities

- Hats trade data
  - giver, taker, capability
  - ~500Mb - ~5G

- Beacon vulnerabilities
Components

■ Third party software
  • GDA/k-groups [Kubika et al 04]
  • BC [Neuman 00]
  • KOJAK [Adibi et al 04]

■ Glue components
  • merge, filter, translate
Hats Data to Warnings Workflow

- start
- hats tracks
- capabilities
- watch list
- CapTracker
- GBA
- filter
- make warning
- suspicious groups
- threatening groups
- GDA/KGroups
- KOJAK
- BC
- need members
- merge
- merge
- groups
- filter
- watch list
- watch list
- hats meetings
- RLP

Notes:
- RLP: Ranking through Label Propagation, Gil et al.
- GBA: Guilt by Association, Medsker
- BC: Rival/KC: Robustness Centrality, Newman
- CapTracker: Witvliet, Ganlyan, Morrison (ny)
- GDA/KGroups: Kubica
* requires paths to int-groups, gda-links, demographics, and to the file where the results (groups) are written.
Running Search Algorithms on Randomly Generated CSPs

Yaling Zheng and Berthe Chouery

University of Nebraska at Lincoln
Task

- A CSP is specified as:
  - Given:
    - A set of variables
    - Their domains
    - Constraints that restrict allowed combinations of values for variables
  - Find one (or more) solution, assignment of values to variables such that all constraints are satisfied

- Task: Run 12 backtrack-search algorithms on randomly generated CSP instances
Task (cont’d)

- A random CSP instance is characterized by
  - Number of variables (considered fixed, n=50)
  - Domain size of each variable (considered fixed, a=12)
  - Constraint density (p in [0.1, 0.2, …, 0.9], sometimes more)
  - Constraint tightness (t in [0.1, 0.2, …, 0.9], sometimes more)

- We generate about 400 instances for each combination of p and t

- We measure various output parameters reflecting search effort. Each experiment is repeated twice:
  - CPU time
  - All other parameters (Nodes visited, constraint checks, etc.)

- (Workflow constraint) When measuring CPU time, experiments must run on equivalent processors
Workflow Instance

Problem instance

A_1, A_2, ..., A_{12} → S_{11}, S_{12}, ..., S_{112} → SAS

A_1, A_2, ..., A_{12} → S_{21}, S_{22}, ..., S_{212} → SAS

A_1, A_2, ..., A_{12} → S_{11}, S_{12}, ..., S_{112} → SAS

A_1, A_2, ..., A_{12} → S_{21}, S_{22}, ..., S_{212} → SAS

Algorithm

SAS

Excel

g_1, g_2, ..., g_x

r_1, r_2, ..., r_x
## Computing Resources

<table>
<thead>
<tr>
<th>Cluster</th>
<th>PrairieFire.unl.edu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>128 Node Production-mode LINUX cluster</td>
</tr>
<tr>
<td>Processors</td>
<td>2 Opteron 248 (2.2GHz/64 bit) per node</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB PC2700 per node</td>
</tr>
<tr>
<td>Connection</td>
<td>Myrinet (2Gb/s) Gigabit Ethernet</td>
</tr>
<tr>
<td>Storage</td>
<td>1 TB SCSI raid (XFS over NFS) 6 TB SATA RAID (ReiserFS over NFS)</td>
</tr>
</tbody>
</table>

Submit jobs to PBS, currently available queues:
- 2 long jobs (maximum duration = 120 hours)
- 28 short jobs (maximum duration = 72 hours)
- Rest in the waiting queue
A Sample PBS File

#PBS –W group_list=consystlab
#PBS –N  fcp0.2cpu
#PBS –l nodes=1,walltime=72:00:00
#PBS –q csl_short
#PBS –M yzheng@cse.unl.edu
#PBS –m e
#PBS –j oe

cd /home/consystlab/yzheng/Codes/gredcpu
/util/opt/ACL/alisp -#! scripts/fc/script-fcp0.2.lisp > results-fcp0.2.txt
A Sample Script File

;; Repeated for every \( p \) value
(load "make.lisp")
(setf *variable-ordering-heuristic* 'select-singleton-or-var-degd)
(setf *global-gc-behavior* :auto)
(setf *timelimit* 3600000)
(dolist (tightness '(0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9))
  (solve-randomly-generated-csp \(<A_i>\) 50 12 \(<p>\) tightness 0 399 )

Algorithm
Instance Index
Parameters of Problem Instance
What SAS and Excel Do

- **SAS**
  - Analyzes the results using Friedman’s test (on ranked data) to compare 12 algorithms at each tightness for each given density

- **Excel**
  - Draws average /median values for tightness in [0.1..0.9] for each density
Experiences

■ Ideal situation
  • Each job reads one CSP instance and solves it and produces the result
  • Each job finishes in 10 msec, up to 1 hour
  • 46000 x 12 x 2 jobs, dream!

■ Current situation
  • Each job reads a group of CSP instances and solves them and produces the results
  • Write script for each job
  • Takes about 2 weeks to accomplish this
Chores

- Periodically (manually) I check the status of each job (when error, rewrite the script & re-submit)
- After understanding the priority policy on the queues, I re-organized the jobs to avoid being assigned a low priority
- After improving the code, I had to run again all the experiments
- In the future, when we have new algorithms or new problem instances, I have to repeat the procedure
- We need to automate the whole process
  - Be able to check on progress of experiments
  - Be able to easily rerun some/parts of the experiments
Tera-Scale Machine Translation

Jens Voeckler
and the Machine Translation group
at
USC Information Sciences Institute
What Is Painful Today?

- **training corpus**
- **LM training**
- **dev corpus**
- **rule extraction**
- **rules**
- **tuning**
- **weights**
- **test corpus**
- **decoding**
- **LM**
- **translation**
Control-Flow for Decoding
Rule Preparation 1

rule-prep-XX

Diagram showing the process of rule preparation with various steps and runtime durations.
Rule Preparation 2

rule-prep-XX

[Diagram with steps and time estimates]
Decoding

- unpack gar/tar
  - runtime: <1 min
- mini_decoder
  - runtime: 0.04\*w^2.63
- paste-byline 1-best
  - runtime: <1 min
- paste-byline n-best
  - runtime: <1 min
- 1-best extraction
  - runtime: 2 min
- n-best extraction
  - runtime: 10 min
Rule Pruning Workflow in Wings
What Researcher Might Like

- Assemble experiments
  - Tinker with the workflow structure.
    - Use existing workflow as template for new
  - Smart parallelization
  - Smart clustering
- "Set it and forget it" approach:
  - Assemble, start, go for lunch,
  - Results - or error - when back
- Avoid grid pains ;-P