Using Provenance to Extract Semantic File Attributes

Daniel Margo and Robin Smogor
Harvard University

Semantic Attributes

- Human-meaningful data adjectives.
- Applications:
  - Search (Google Desktop, Windows Live)
  - Namespaces (iTunes, Perspective [Salmon, FAST'09])
  - Preference Solicitation (Pandora)
  - And more...
- Make data more valuable (like provenance!)
  - Only...

Where do Attributes Come From?

- Manual labeling - intractable.
- Automated content extraction:
  - Arguably, Google.
  - Visual extraction (*La Cascia et al., '98*)
  - Acoustic extraction (*QueST, MULTIMEDIA'07*)
- Problems:
  - Need extractors for each content type.
  - Ignorant of inter-data relationships: dependency, history, usage, provenance, *context.*

How Might Context Predict Attributes? Examples:

- If an application always reads a file in its directory, that file is probably a component.
- If an application occasionally writes a file outside its directory, that's probably content.
- Etc...
- Prior work:
  - Context search [Gyllstrom IUI'08, Shah USENIX'07]
  - Attribute *propagation* via context [Soules '04]

The Goal

- *File relationships → attribute predictions.*
- Begin with a provenance-aware system (PASS)
- Run some file-oriented workflow(s).
- Output per-file data into a machine learner.
- Train learner to predict semantic attributes.
  - Simple! Only...

The Challenge

- ...like fitting a square peg into a round hole!
- Provenance → graphs → quadratic scale.
- Typical learner handles ~hundreds of features.
- Needs relevant feature extraction.
  - Going to “throw out” a lot of data.

about:PASS

- Linux research kernel.
- Collects provenance at system call interface.
- Logs file and process provenance as a DAG.
- Nodes are *versions* of files and processes.
  - Must resolve many-to-one node to file mapping.

Resolving Nodes to Files

- Simple solution: discard version data.
  - Introduces cycles (false dependencies).
  - Increases graph density.
- Alternatively: merge nodes by file name.
  - Similar to above; introduces more falsity.
  - But guarantees direct mapping.
- More complicated post-processing?
  - Future work.

Graph Transformations

- File graph: reduce graph to just files.
  - Emphasizes data dependency, e.g. libraries.
- Process graph: reduce graph to just processes.
  - Emphasizes workflow, omits specific inputs.
- Ancestor and descendant subgraphs.
  - Defined as transitive closure.
  - On a per-file basis.

Statistics

- How to convert per-file subgraphs to statistics?
- Experiments with partitioning, clustering:
  - Graclus (partitioner), GraphClust.
  - Failure: graph sparsity, different structural assumptions produce poor results.
- Success with “dumb statistics”:
  - Node and edge counts, path depths, neighbors.
  - For both ancestor and descendant graphs.
  - Still a work in progress.

Feature Extraction: Summary

De-version	Merge Names	Provenance Graph	Ancestors	Node Count
	Don't Merge	File Graph	Descendants	Edge Count
		Process Graph		Max Depth
				Neighbors

- 2 ways to merge (by versions or path names).
- 3 graph representations (full, process, file).
- 4 statistics for both ancestors and descendants.
- Totals 48 possible features-per-file...
- ...plus 11 features from *stat* syscall.
  - Content-free metadata.

Classification

- Classification via decision trees.
  - Transparent logic: can evaluate, conclude, improve.
- Standard decision tree techniques:
  - Prune splits via lower bound on information gain.
  - Train on 90% of data set, validate on 10%.
  - k-means to collapse real-valued feature spaces.
- Requires labeled training data...

Labeling Problem

- First challenge: how to label training data?
  - Semantic attributes are subjective.
  - No reason provenance *should* predict any random attribute; must be well-chosen.

Labeling Solution

- Initial evaluation using *file extensions* as label.
  - Semantically meaningful, but not subjective.
  - Pre-labeled.
  - Intuitively, usage predicts “file type”.
  - Reverse has been shown: extension predicts usage [Mesnier ICAC'04].

What’s the Data Set?

- Second challenge: finding a data set.
  - Needs a “large heterogeneous file workflow”.
  - Still a work in progress.
- In interim, Linux kernel compile.
  - 138,243 nodes, 1,338,134 edges, 68,312 de-versioned nodes, 34,347 unique path names, and 21,650 files-on-disk (*manifest* files).
- Long brute-force analysis; used 23 features.

Precision, Recall, and Accuracy

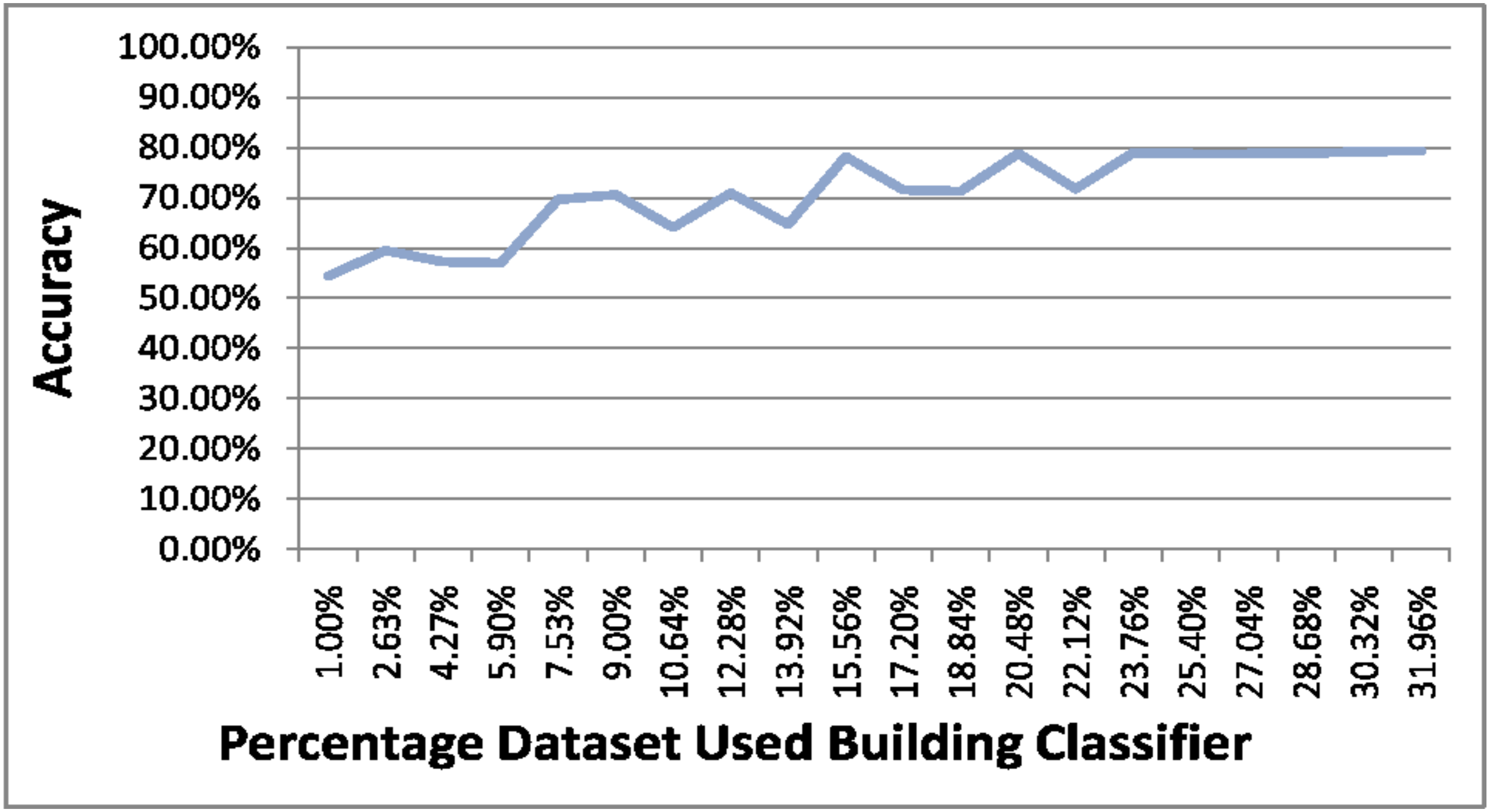
- Standard metrics in machine learning:
  - *Precision:* for a given extension prediction, how many predictions were correct?
  - *Recall:* for a given extension, how many files with that extension received the correct prediction?
  - *Accuracy:* how many of *all* the files received the correct prediction?

Results

ext	# in set	precision	recall
.h	8678	96.70%	72.65%
.c	8420	70.22%	96.94%
none	1869	80.26%	53.08%
.S	912	69.34%	27.52%
.o	829	99.28%	99.76%
.txt	415	59.39%	99.04%
.cmd	147	97.24%	95.92%
other	180	31.89%	15.00%
total	21450	82.55%	79.79%
.h+.c+.S	18010	98.76%	96.10%
total	21450	95.83%	91.87%

- 85.68% extension prediction accuracy.
- 79.79% on *manifest* files (present on disk).
  - Table at left.
  - Confuses “source files”.
  - If fixed, 94.08%.
- 93.76% on non-manifest objects.

Number of Records Needed



Talking Points

- Is "source file" confusion *wrong?*
  - .c/.h/.S have similar usage from PASS perspective.
  - "source file" may be right semantic level.
  - Can fix using 2nd-degree neighbors (object files).
- Other than this, high accuracy.
  - Especially on non-manifest objects – content-free.
  - Noteworthy features – ancestral file count, edge count, max path depth; descendant edge count

Future Work

- More feature extraction.
- Evaluate more attributes...
- ...on more data sets.
- More sophisticated classifiers (neural nets).
- Better understanding!