Self-Tuning Association Rules for KNIME

Yacaree: from Python to Java

Javier de la Dehesa

Universidad de Cantabria

October 21, 2011

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

From Python to KNIME

We want to introduce Yacaree into KNIME, so we need to:

1. Know Yacaree check
2. Know KNIME check
3. Port Yacaree to a KNIME node ??

Porting issues

Integrating data types How do we represent our data? Can we take advantage of the way KNIME handles data?

Structural design Should KNIME node follow the same structure that the Python program?

Memory management Which is the best way to ensure that we do not run out of memory and the closures queue does not grow too much?

Iterators How can we reproduce with Java the behaviour of the "magical" Python keyword `yield`?

Input/output How do we get our data and where do we put discovered rules?

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Representing transactions

KNIME data is encapsulated in a `BufferedDataTable` class with interesting features:

- Iterable.
- Cacheable to virtual memory if necessary.
- Easy to use and well documented.
- Handy!

But **one BIG drawback** (*for our purposes*):

- Does not allow random access.

`BufferedDataTable` is not suitable for Yacaree, so transactions are put into a Java `HashMap` that maps each row identifier (`RowKey`) to corresponding transaction as a `Set` of `String`.

Representing closures and rules

- Instead of storing a single global bidirectional relation between items and transactions, now we only keep track of the **transactions list** (transaction to items relation).
- **Item closures** are stored along with its support set (transactions containing it).
- **Rules** are stored as a couple of closures satisfying that antecedent is subset of consequent.

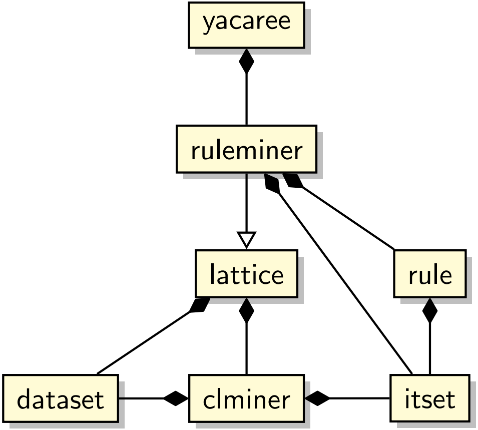
Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output

2. A quick demo

3. Conclusions

Python class diagram

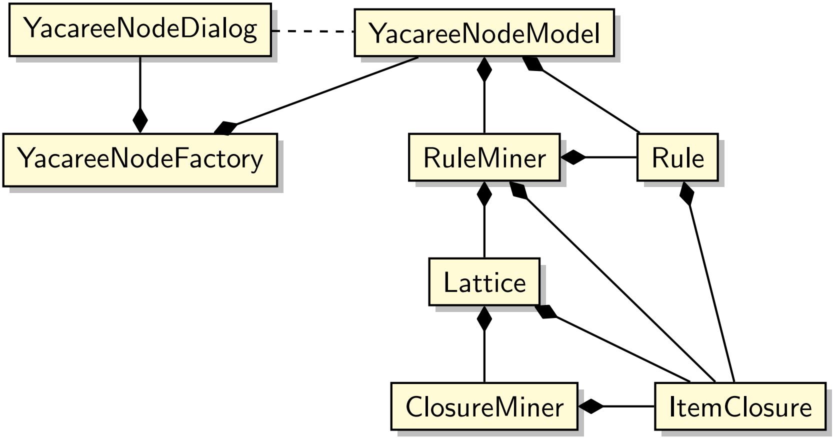


Rearranging the structure

Several changes have been made to the original structure:

- KNIME forces us to put some auxiliary classes, `YacareeNodeModel`, `YacareeNodeDialog` and `YacareeNodeFactory`.
- `ruleminer` does not inherit from `lattice` anymore to improve modularity.
- `dataset` is not be modeled as a class itself but as an instance of an existing class.
- As a consequence, item sets will be replaced by closures with its support set.

KNIME class diagram



Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Managing resources

Two thresholds to watch:

- Maximum heap space
- Closures queue size

In both cases, whenever the threshold is exceeded the closures queue is halved, so:

- Bigger $\Rightarrow$ More closures explored
- Lower $\Rightarrow$ Faster
- Bigger $\nRightarrow$ More rules

Setting thresholds

Heap space

- In Python, fixed to 1 GB.
- In KNIME, depends on memory assigned to JVM $\Rightarrow$ KNIME configuration.

Closures queue size

- In both cases fixed to 2^{14}.

Disclaimer: These thresholds has been set experimentally and have sensible effects on the execution. Their values are subject to discussion.

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Iterators in Python

Any function can return a value with `yield` saving program counter, variables values, etc. so next time it is called it resumes where we left off.

Example

```
def iterable_function():
    i = 1
    yield i
    yield i + 1
    yield i + 2

for i in iterable_function():
    print i
```

Amazing!

Iterators in Java

A conventional `Iterator` interface with `next()`, `hasNext()` and `remove()` methods that, in combination with `Iterable` interface, provides a tiny *syntactic sugar* that saves you a couple of lines of code.

Example

```
Iterable<E> iterableObject = new ArrayList<E>();
/* insert elements into list */

for (E element : iterableObject)
    System.out.println(E.toString());
```

Unamazing!

`ClosureMiner`, `Lattice` and `RuleMiner` inherit from `Iterator`, *but just for convention*.

Comparative implementation. Closure miner

Python version

Initializes closures queue to singletons.

While queue is not empty:

- Yield next closure in the queue.
- Combine closure with every singleton and enqueue.

Java version

Initializes closures queue to singletons.

`hasNext()` checks if closures queue is empty.

`next()` method:

- Combine next closure with every singleton and enqueue.
- Return closure.

Comparative implementation. Lattice

Python version

For every received closure:

- Add every valid predecessor to a `ready` queue.
- Yield every item in `ready` queue.

Java version

While there are closures:

- `hasNext()` fetchs next and adds every valid predecessor to a `ready` queue.
- `next()` returns every item in the `ready` queue.

Comparative implementation. Rule miner

Python version

For every candidate closure:

- For every predecessor:
  - Make rule and yield if valid.

Java version

While there are candidate closures:

- `hasNext()` fetchs next and adds every every valid rule made with predecessors to a `ready` queue.
- `next()` returns every item in the `ready` queue.

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Python Yacaree

Input A plain text file.

Example

```
bread_and_cake baking_needs coffee
prepared_meals frozen_foods small_goods
```

Output Human-readable text file.

Example

```
3/
    prepared_meals
         =>
              frozen_foods
[conf: 0.732;supp: 0.201; lift: 1.246; boost: 1.217]
```

KNIME Yacaree

Input KNIME provides input handling for different sources out of the box:

- Files
- Databases
- Web services
- Wherever

User just have to put data into a "Collection type" column in a table.

Output A KNIME table that can be connected to other nodes or written to a file.

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Outline

1. The porting problem
   - Integrating data types
   - Structural design
   - Memory management
   - Iterators
   - Input/output
2. A quick demo
3. Conclusions

Conclusions

- KNIME offers a solid platform to implement data mining algorithms.
- Porting $\neq$ "Translate"
- Memory thresholds are an open question - probably with no answer.
- Iterators, and specially `yield`, can be one the most challenging issues when porting from Python.
- The obtained node is fairly easy to use - *we would love to see you using it*.