General-Purpose Inductive Programming for Data Wrangling Automation

Lidia Contreras-Ochando, Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo and María José Ramírez-Quintana

Universitat Politècnica de València (UPV), Spain

{liconoc, fmartinez, cferri, jorallo, mramirez}@dsic.upv.es

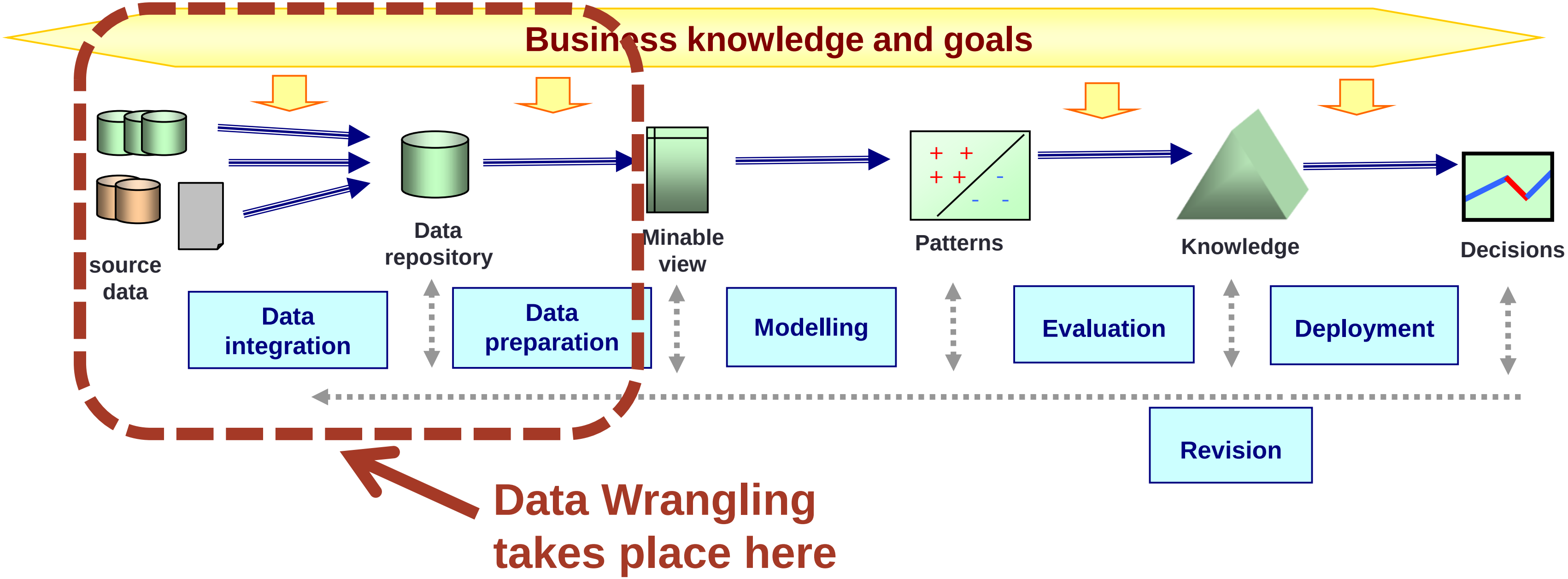
DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

NIPS 2016 Workshop on Artificial Intelligence for Data Science (AI4DataSci)

Data Wrangling in Data Science

- Part of the first stages of the process:



Data Wrangling

Data Scientist:
The Sexiest Job of the 21st Century

Meet the people who can coax treasure out of messy, unstructured data.
by Thomas H. Davenport and D.J. Patil

When Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the place still felt like a start-up. The company had just under 8 million accounts, and the number was growing quickly as existing members invited their friends and colleagues to join. But users weren't seeking out connections with the people who were already on the site at the rate executives had expected. Something was apparently missing in the social experience. As one LinkedIn manager put it, "It was like arriving at a conference reception and realizing you don't know anyone. So you just stand in the corner sipping your drink—and you probably leave early."

70 Harvard Business Review October 2012

Data Wrangling:
The Least Sexy Part of Data Science

- Data Wrangling is messy and unstructured.
- Data Wrangling is boring.
  - Because it is **repetitive.**

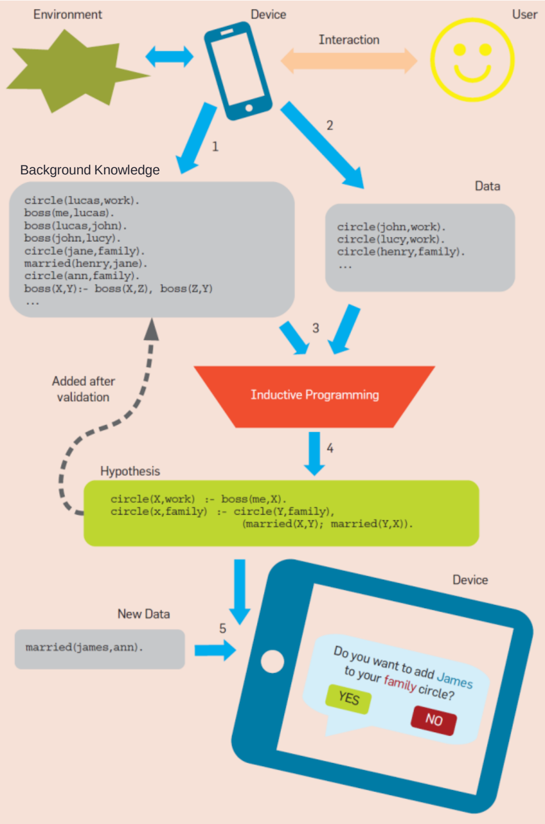
Why is Data Wrangling so Critical?

- It appears very early in data science projects
  - Sometimes even before having analysed the requirements.
- It depends on the **previous knowledge**.
  - No statistical technique is going to tell us that “male”, “man” and “m” are the same value for the attribute “gender”.
    - A great part of data preparation is about introducing knowledge into and checking constraints over the data through data cleansing and (feature) transformations.
- It takes 50-80% of the effort in data science projects.

(Semi-)Automating it would have a very significant impact

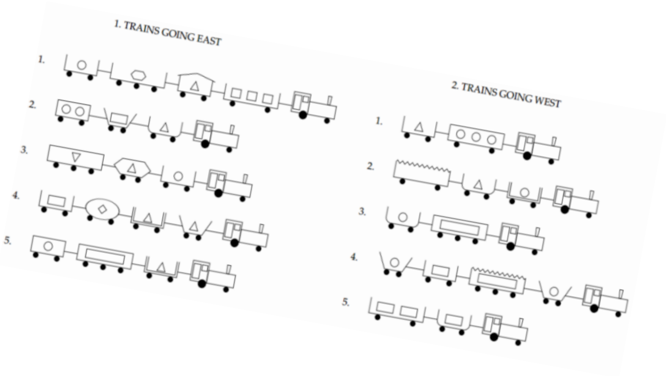
What's Inductive Programming?

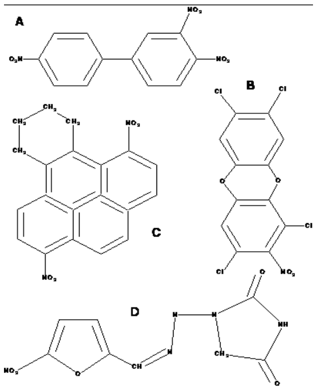
- An area with roots in the old vision of **machines programming themselves.**
  - Also known as or strongly overlapping with: programming by example, program synthesis, ILP, …
- Elements:
  - Input:
    - Data D (usually small sets of data)
    - Background Knowledge B (facts, functions, constraints, etc.)
  - Output:
    - Hypothesis h (a program)
  - D, B and h are usually represented in the same (declarative) language:
    - E.g., Prolog, Haskell, Erlang, etc.



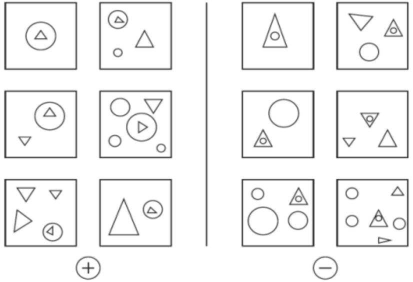
What’s Inductive Programming?

- From rich, but usually small data:

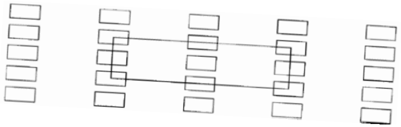





```
Problem domain:
puttable(x)
PRE: clear(x), on(x, y)
EFFECT: ontable(x), clear(y), not on(x,y)
Problem Descriptions:
: init-1 clear(A), ontable(A)
: init-2 clear(A), on(A, B), ontable(B)
: init-3 on(B, A), clear(B), ontable(A)
: init-4 on(C, B), on(B, A), clear(C), ontable(A)
: goal clear(a)
Problem Solving Traces/Input to IGOR2
fmod CLEARBLOCK is
  *** data types, constructors
  sorts Block Tower State .
  op table : -> Tower [ctor] .
  op __ : Block Tower -> Tower [ctor] .
  op puttable : Block State -> State [ctor] .
  *** target function declaration
  op ClearBlock : Block Tower State -> State [metadata "induce"] .
  *** variable declaration
  vars A B C : Block .
  var S : State .
  *** examples
  eq ClearBlock(A, A table, S) = S .
  eq ClearBlock(A, A B table, S) = S .
  eq ClearBlock(A, B A table, S) = puttable(B, S) .
  eq ClearBlock(A, C B A table, S) = puttable(B, puttable(C, S)) .
endfm
```



$qsort([3,4,1,8,5,6])=[1,3,4,5,6,8]$



Images from {Michie-etal1994trains} {Srinivasan-etal1994mutagenesis} {Schmid-etal2008analytical} (Olson1995incremental) (De raedt2011-encyclopedia)

What's Inductive Programming?

- To usually rich models:

$$qsort([],[]) \leftarrow$$
$$qsort([X|T],S) \leftarrow \quad part(X,T,L1,L2), qsort(L1,S1), qsort(L2,S2), app(S1,[X|S2],S)$$

delOdds(L,R) ← L=[], R=[]
delOdds(L,R) ← L=[HL|TL], odd(HL), delOdds(TL,TR), R=TR
delOdds(L,R) ← L=[HL|TL], ¬odd(HL), delOdds(TL,TR), R=[HL|TR]

$$\mathbf{B}_m \forall x.((group\_likes\ x) = if\ ((\wedge_3\ \blacklozenge \mathbf{B}_a likes\ \mathbf{B}_b likes\ \mathbf{B}_c likes)\ x)\ then\ \top\ else\ if\ ((\wedge_3\ \mathbf{B}_c likes\ (\vee_2\ \mathbf{B}_a likes\ \mathbf{B}_b likes)\ top)\ x)\ then\ \top\ else\ \bot).$$

1.0: likes(X,Y):- friendof(X,Y).
0.8: likes(X,Y):- friendof(X,Z), likes(Z,Y).
0.5: friendof(john,mary).
0.5: friendof(mary,pedro).
0.5: friendof(mary,tom).
0.5: friendof(pedro,tom).

prod(s(X0), X1) = sum(prod(X0, X1), X1)
prod(0, X0) = 0

```
% Bush syntactic model:
12 :: imperative_sentence(verb_phrase(verb,noun_phrase(noun)))
13 :: affirmative_sentence(noun_phrase(noun),verb_phrase(verb))
 3 :: imperative_sentence(verb_phrase(verb,noun_phrase(noun),
      prop_pred(prep,noun_phrase(det,noun))))
 3 :: imperative_sentence(verb_phrase(verb,noun_phrase(noun),
      prop_pred(prep,noun_phrase(noun))))
 4 :: affirmative_sentence(connector,
      affirmative_sentence(noun_phrase(noun),
      verb_phrase(verb(aux,verb,verb),noun_phrase(noun))))
 4 :: affirmative_sentence(noun_phrase(noun),
      verb_phrase(verb(aux,aux,verb)))
 4 :: imperative_sentence(verb_phrase(verb))
 7 :: affirmative_sentence(noun_phrase(noun),
      verb_phrase(verb,noun_phrase(det,noun)))
...
```

Examples from {Muggleton-Deraedt1994ilp} {Ferri-etal2001incremental} {Flener-Yilmaz1999inductive} {Castillo2012stochastic} {Deraedt-etal2007problog} {Lloyd-Ng2007modal}

What's Inductive Programming?

- Very different niche compared to other ML paradigms.

	Inductive Programming	Other Machine Learning Paradigms
Number of examples	Small	Large, for example, big data
Kind of data	Relational, constructor-based datatypes	Flat tables, sequential data,
Data Source	Human experts, software applications, HCI, and others	Transactional databases, Internet, sensors (IoT), and others.
Hypothesis Language	Declarative: general programming languages or domain-specific languages	Linear, non-linear, distance-based, kernel-based, rule-based, probabilistic, etc.
Search strategy	Refinement, abstraction operators, brute-force.	Gradient-descent, data partition, covering, instance-based, etc.
Representation learning	Higher-order and predicate/function invention	Deep learning and feature learning.
Pattern comprehensibility	Common.	Uncommon.
Pattern expressiveness	Usually recursive, even Turing-complete.	Feature-value, not Turing complete.
Learning bias	Using background knowledge and constraints	Using prior distributions, parameters and features.
Evaluation	Diverse criteria, including simplicity, comprehensibility	Oriented to error (or loss) minimisation.
Validation	Code inspection, divide-and-conquer debugging, background knowledge consistency	Statistical reasoning (only a few techniques are locally inspectable).

Inductive Programming for Data Wrangling

- Proof of concept with killer apps
  - Flash Fill:

	A	B
1	Email	Column 2
2	Nancy.FreeHafer@fourthcoffee.com	nancy freehafer
3	Andrew.Cencici@northwindtraders.com	andrew cencici
4	Jan.Kotas@litwareinc.com	jan kotas
5	Mariya.Sergienko@gradicdesigninstitute.com	mariya sergienko
6	Steven.Thorpe@northwindtraders.com	steven thorpe
7	Michael.Neipper@northwindtraders.com	michael neipper
8	Robert.Zare@northwindtraders.com	robert zare
9	Laura.Giussani@adventure-works.com	laura giussani
10	Anne.HL@northwindtraders.com	anne hl
11	Alexander.David@contoso.com	alexander david
12	Kim.Shane@northwindtraders.com	kim shane
13	Manish.Chopra@northwindtraders.com	manish chopra
14	Gerwald.Oberleitner@northwindtraders.com	gerwald oberleitner
15	Amr.Zaki@northwindtraders.com	amr zaki
16	Yvonne.McKay@northwindtraders.com	yvonne mckay
17	Amanda.Pinto@northwindtraders.com	amanda pinto

  - Flash Extract:

Ana Trujillo
357 21st Place SE
Redmond, WA
(757) 555-1634

Antonio Moreno
515 93th Lane
Renton, WA
(411) 555- 2786

Thomas Hardy
742 17th Street NE
Seattle, WA
(412) 555-5719

Christina Berglund
475 22th Lane
Redmond, WA
(443) 555-6774

Hanna Moos
785 45th Street NE
Puyallup, WA
(376) 555-2462

Frederique Citeaux
308 66th Place
Redmond, WA
(689) 555-2770

(a)

Label 1	Label 2	Label 3
Ana Trujillo	Redmond	(757) 555-1634
Antonio Moreno	Renton	(411) 555-2786
Thomas Hardy	Seattle	(412) 555-5719
Christina Berglund	Redmond	(443) 555-6774
Hanna Moos	Puyallup	(376) 555-2462
Frederique Citeaux	Redmond	(689) 555-2770

(b)

PairSeq SS ::= LinesMap(λx: Pair(Pos(x, p_1), Pos(x, p_2)), LS)
| StartSeqMap(λx: Pair(x, Pos($R_0[x]$, p)), PS)
LineSeq LS ::= FilterInt(init, iter, BLS)
BoolLineSeq BLS ::= FilterBool(b, split(R_0, '/n'))
PositionSeq PS ::= LinesMap(λx:Pos(x, p), LS)
| FilterInt(init, iter, PosSeq (R_0, rr))
Pred b ::= λx: {Starts,Ends}With(r,x) | λx: Contains(r, k, x)

(c)

But how do they work?

Inductive Programming using DSLs

- It is argued that general languages produce a vast search space.
- Idea:
  - Define IP systems with a domain specific language (DSL) that fits the domain perfectly:
    - Balance:
      - Expressive enough to cover the problems in the domain.
      - Restrictive enough to enable efficient search.
- It has been a success! (Gulwani 2011-2016)
  - FlashFill, FlashExtractText, FlashRelate, FlashNormalize, BlinkFill, …
- Limitations:
  - Systems (and the IP engines) must be redesigned for each domain.
    - FlashMeta proposed as a partial solution.
  - Lack of flexibility: how to include background knowledge, customisation, …

Inductive Programming using GPDLs

- We can use any general-purpose IP system:
  - GOLEM, Progol, (F)FOIL, ADATE, DIALOGS, FLIP, IGOR I/II, Aleph, MagicHaskeller, Metagol, gErl, …
- Users may edit the solutions written in languages such as Haskell, Erlang or Prolog.
- But, by using the built-in functions (BIF) or some particular background knowledge,

Would these systems be able to cope with general data wrangling problems?

Examples

- Feature wrangling:

	Gender	GenderOK	Birthdate	BirthdateOk	Postcode	PostcodeOK	Score	ScoreOk	Km	metresOK	Weight	WeightOK
#1	male	m	3 1 1971	1971 1 3	46 025	46025	5.5, 4.6, 5.8	5.3	5	5000	f "CAMP DRY DBL NDL 3.6 OZ"	"3.6 OZ"
#2	female	f	4 5 1993	1993 5 4	46225	46225	3.5	3.5	3	3000	f "DRY NDL 0.23 KG"	"0.23 Kg"
#3	…	…	…	…	…	…	…	…	…	…	…	…

- Can we automate these transformations with just one or two examples?
  - MagicHaskeller (Katayama 2004-today):
    - http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html
  - Metagol (Muggleton et al. 2014-today) .
    - https://github.com/metagol/metagol
  - gErl (Martínez-Plumed et al. 2012-today)
    - https://github.com/nandomp/gErl

Examples

	Gender	GenderOK
#1	male	m
#2	female	f
#3		

- MagicHaskeller
  - BK: standard library
  - Input: `( f "female" ~= "f" ) && ( f "male" ~= "m" ))`
  - Output: `f = (take 1)`
- Metagol
  - BK: several predicates, including “head”, Metarules: several.
  - Input: `Pos = [ f(['f','e','m','a','l','e'],'f'), f(['m','a','l','e'],'m') ]`
  - Output: `f(A,B):-head(A,B).`
- gErl
  - BK: Erlang BIFs for lists, Operators: to handle BK.
  - Input: `Pos = [ f(“Female”)->“F”, f(“Male”)->“M” ]`
  - Output: `f([A|_]) -> A.`

Examples

	Birthdate	BirthdateOk
#1	3 1 1971	1971 1 3
#2	4 5 1993	1993 5 4
#3	…	…

- MagicHaskeller
  - BK: standard library
  - Input: `( f [3, 1, 1971] ~= [1971, 1, 3] ) && ( f [4, 5, 1993] ~= [ 1993, 5, 4] )`
  - Output: `f = reverse`
- Metagol
  - BK: several predicates, including "reverse", Metarules: several.
  - Input: `Pos = [f([3,1,1971],[1971,1,3]), f([4,5,1993],[1993,5,4]), f([1,3,2013],[2013,3,1]) ]`
  - Output: `f(A,B):-reverse(A,B).`
- gErl
  - BK: Erlang BIFs for lists, Operators: to handle BK.
  - Input: `Pos = [f([3,1,1971])->[1971,1,3]), f([4,5,1993])->[1993,5,4], f([1,3,2013]) -> [2013,3,1]]`
  - Output: `f(A)-> reverse(A).`

Examples

	Postcode	PostcodeOK
#1	46 025	46025
#2	46225	46225
#3	…	…

- MagicHaskeller
  - BK: standard library
  - Input: `f "46 025" ~= "46025"`
  - Output: `f = (filter isDigit)`
- Metagol
  - BK: several predicates, including several char_X and delete, Metarules: several.
  - Input: `Pos = [ f(['4','6',' ','0','2','5'],['4','6','0','2','5']), f(['4','6','2','2','5'] ,['4','6','2','2','5']), f(['3','0','5',' ','2','3'],['3','0','5','2','3']) ]`
  - Output: `f (L1, L2):- char_space(X), delete(L1,X,L2).`
- gErl
  - BK: Erlang BIFs for lists, Operators: to handle BK.
  - Input: `Pos = [ f([“4”,“6”,“ ”,“0”,“2”,“5”]->“46025”, f([“4”,“6”,“2”,“2”,“5”]->“46225”, f([“3”,“0”,“5”,“ ”,“2”,“3”]->“30523”]`
  - Output: `f(A) -> flatten(A).`

Examples

	Score	ScoreOk
#1	5.5, 4.6, 5.8	5.3
#2	3.5	3.5
#3	…	…

- MagicHaskeller
  - BK: standard library
  - Input: `( f [5.5, 4.6, 5.8 ] ~= 5.3 ) && f [ 3.5 ] ~= 3.5`
  - Output: `f = (\a -> sum a / fromIntegral (length a))`
- Metagol
  - BK: several predicates, including sumlist, length, div, …, Metarules: several.
  - Input: `Pos = [ f([5.5, 4.6, 5.8 ],5.3), f([3.5],3.5) ]`
  - Output: `f(A, B):- sumlist(A, C), length(A, D), div(C, D, B).`
- gErl
  - BK: Erlang BIFs for lists, Operators: to handle BK.
  - Input: `Pos = [ f([5.5, 4.6, 5.8 ])->5.3), f([3.5])->3.5) ]`
  - Output: `f(A) -> sum(A)/length(A).`

Examples

	Km	metresOK
#1	5	5000
#2	3	3000
#3	…	…

- MagicHaskeller
  - BK: standard library
  - Input: `( f 5 ~= 5000 ) && ( f 3 ~= 3000 )`
  - Output: `f = (\a -> round (1000 * fromIntegral a))`
- Metagol
  - BK: would require predicates introducing any constant.
- gErl
  - Operators: very specific operators to cope with constant math operations.
  - Input: `Pos = [ f(5)->5000), f(3)->3000) ]`
  - Output: `f(A) -> A*1000.`

Examples

	Weight	WeightOK
#1	f "CAMP DRY DBL NDL 3.6 OZ"	"3.6 OZ"
#2	f "DRY NDL 0.23 KG"	"0.23 Kg"
#3	…	…

- MagicHaskeller
  - BK: standard library
  - Input: `( f "CAMP DRY DBL NDL 3.6 OZ" == "3.6 OZ" ) && ( f "DRY NDL 0.23 KG" == "0.23 Kg" )`
  - Output: `f = (dropWhile (\b -> not (isDigit b)))`
- Metagol
  - (not attempted).
- gErl
  - BK: Erlang lists BIFs, Operators: to handle BK.
  - Input: `Pos = [ f([“CAMP DRY DBL NDL 3.6 OZ”])->“3.6 OZ”, f([“DRY NDL 0.23 KG”])->“0.23 Kg”`
  - Output: `f(A) -> dropwhile(not is number,A)`

Modifying the background knowledge

- Can we use general background knowledge?
  - It works well for MagicHaskeller:
    - Solves many problems efficiently, without any other tuning.

Number of solved instances (from http://nautilus.cs.miyazaki-u.ac.jp/~skata/presentation/Haskell2013.html#(22)).

problems	MagicHaskeller	Flash Fill
First 20 of 99 Haskell Problems	11/20 (55%)	3/20 (15%)
10 Flash Fill Examples	3/10 (30%)	9/10 (90%)

    - It would do much better with specialised background knowledge.
  - Katayama presented MagicExceller:
    - Reduces and specialises the library to Excel functions.

We should define libraries for data wrangling in a more flexible, customisable way.

Next steps

- Analyse common transformation problems in data wrangling
  - From tutorials/books/users/systems:
    - openrefine.org, Trifacta, R tidyr/dplyr, DSL-based data wrangling tools: e.g., BlinkFill.
  - At different levels:
    - Feature transformations.
    - Row transformations.
    - Table transformations.
    - Integration from several tables.
    - Other kinds of formatting.
- Define a library (BK) that ***can*** solve these common problems.
  - Trade-off between efficiency and power (syntactic and semantic domain).
- Interaction:
  - Ask the user: Do you think this is a date? An address? Use the right BK for the task.

Currently

- With MagicHaskeller:
  - Identify the function library (LibTH.hs) and turn it into an easily editable format:
    - Modifiable by the user.
    - Possibly adding new functions.
- With Metagol:
  - Try to resolve some type problems (if a function in the BK uses lists of integers and the examples are not numbers, there is an error when the predicate is used).
  - Two “libraries” to be selected for Metagol:
    - Metarules.
    - Proper background knowledge.
- With gErl:
  - It must be configured with different (possibly user-defined) learning operators depending on the problem, data representation and the way the examples should be navigated.
  - Identify BIFs or new functions to be added in the background knowledge.
- With other IP systems (including those based on neural abstract machines!)

Conclusions

- Inductive programming is appropriate for data wrangling
  - Transformation not programmed or menu-picked, but indicated with a few examples.
  - The output is understandable and editable by the user.
- DSLs have been shown successful
  - But need to be adapted for each application and tool.
- IP based on GPDLs can be customised (power/efficiency trade-off)
  - Through domain-specific background knowledge.
- We are exploring this route with some of state-of-the-art IP systems.
  - We're in a preliminary stage:

Feedback very welcome!

Questions… and advertising!

- Next week here in Barcelona!

Data Wrangling Automation
ICDM 2016 (Monday 12th)
http://www.dsic.upv.es/~flip/DWA2016