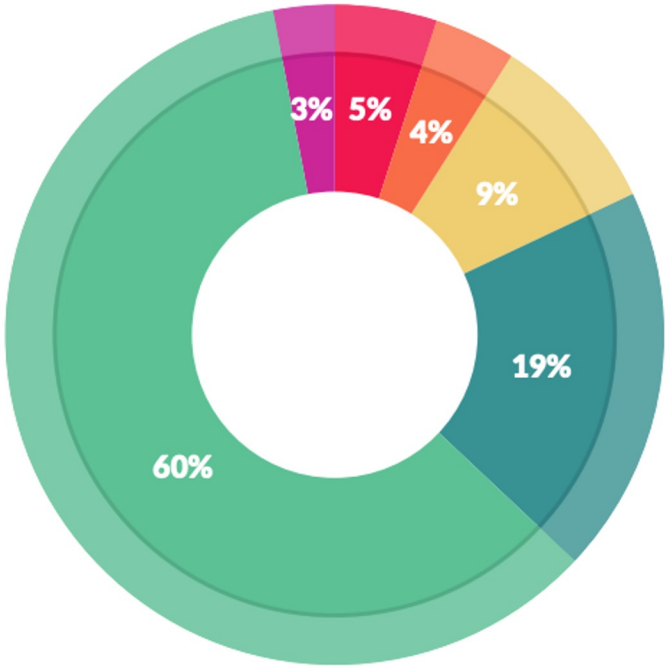
Learning Over Dirty Data Without Cleaning

Jose Picado, John Davis, Arash Termehchy, Ga Young Lee

Data cleaning is a major obstacle in machine learning

Real-world datasets are dirty: duplicates, inconsistencies, …



What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

("CrowdFlower")

Dirty data significantly reduces the accuracy of learning

IMDb

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt3832730	They Hunger (2015)	Spain	2015

+

highGrossing
positive examples

highGrossing
negative examples

Predict whether a movie will be **high grossing**

ML algorithm

highGrossing(x)

Dirty data significantly reduces the accuracy of learning

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt3832730	They Hunger (2015)	Spain	2015

+

OMDb API
The Open Movie Database

omdbMovie		
id	title	distributor
13	The Hunger Games	Lionsgate
16	They Hunger	Aurum

+

highGrossing
positive examples

highGrossing
negative examples

Predict whether a movie will be **high grossing**

ML algorithm

highGrossing(x)

Dirty data significantly reduces the accuracy of learning

imdbMovie

id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt3832730	They Hunger (2015)	Spain	2015

+

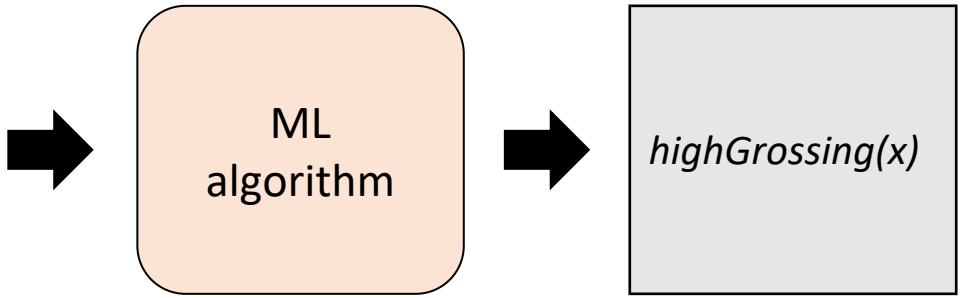
omdbMovie

id	title	distributor
13	The Hunger Games	Lionsgate
16	They Hunger	Aurum

+

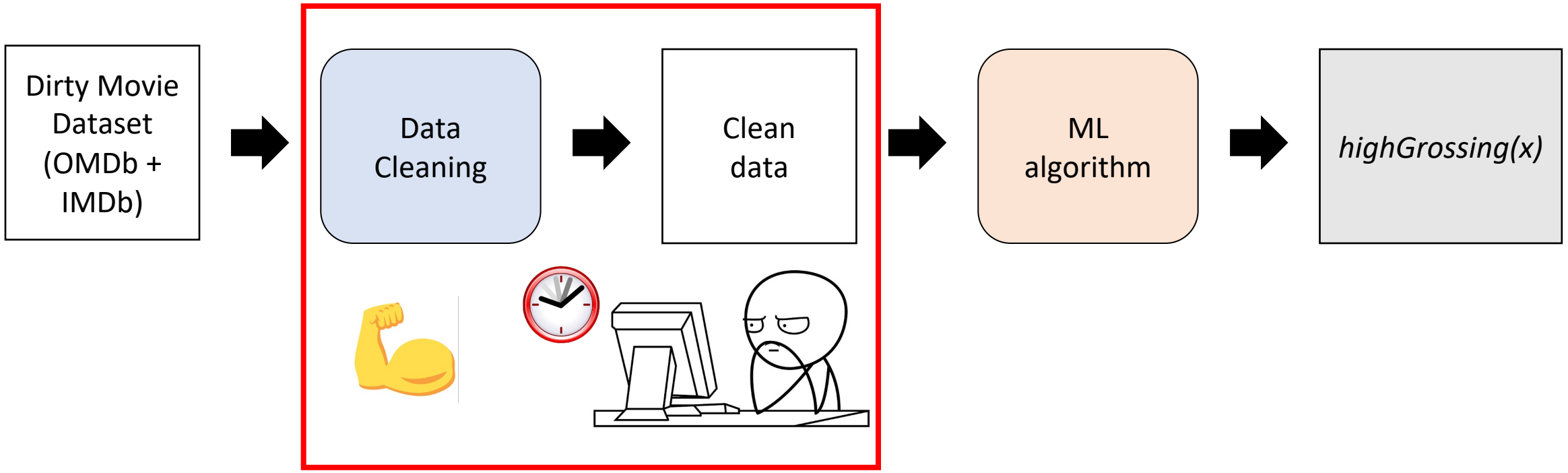
highGrossing	highGrossing
positive examples	negative examples

Predict whether a movie will be **high grossing**



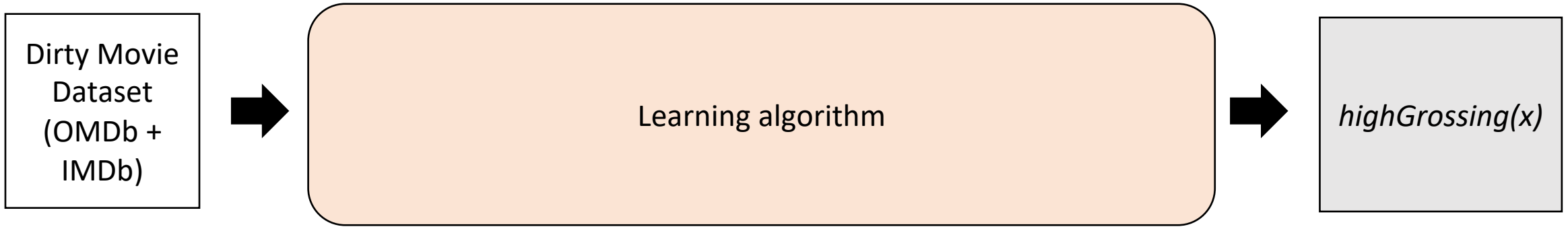
Current approaches to data cleaning are difficult and time-consuming

Preprocessing is a separate step



Significant manual effort, computational and financial resources

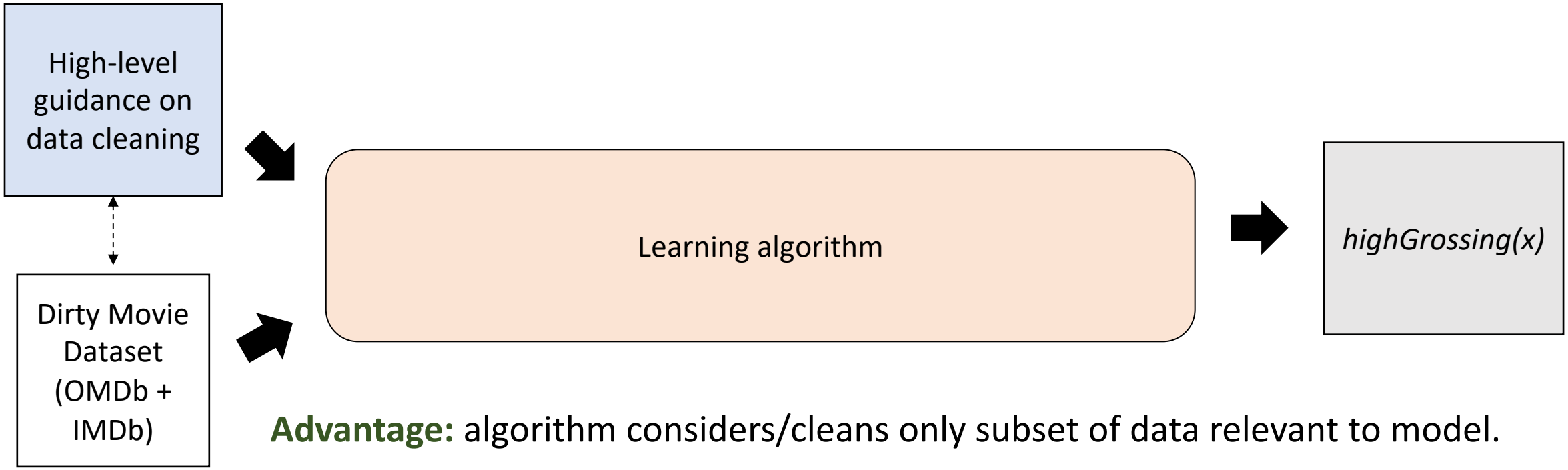
Our question: can we eliminate the data cleaning step?



Our goal:

Learn accurate models directly over dirty data

Our approach: guide the learning algorithm to deal with dirty data

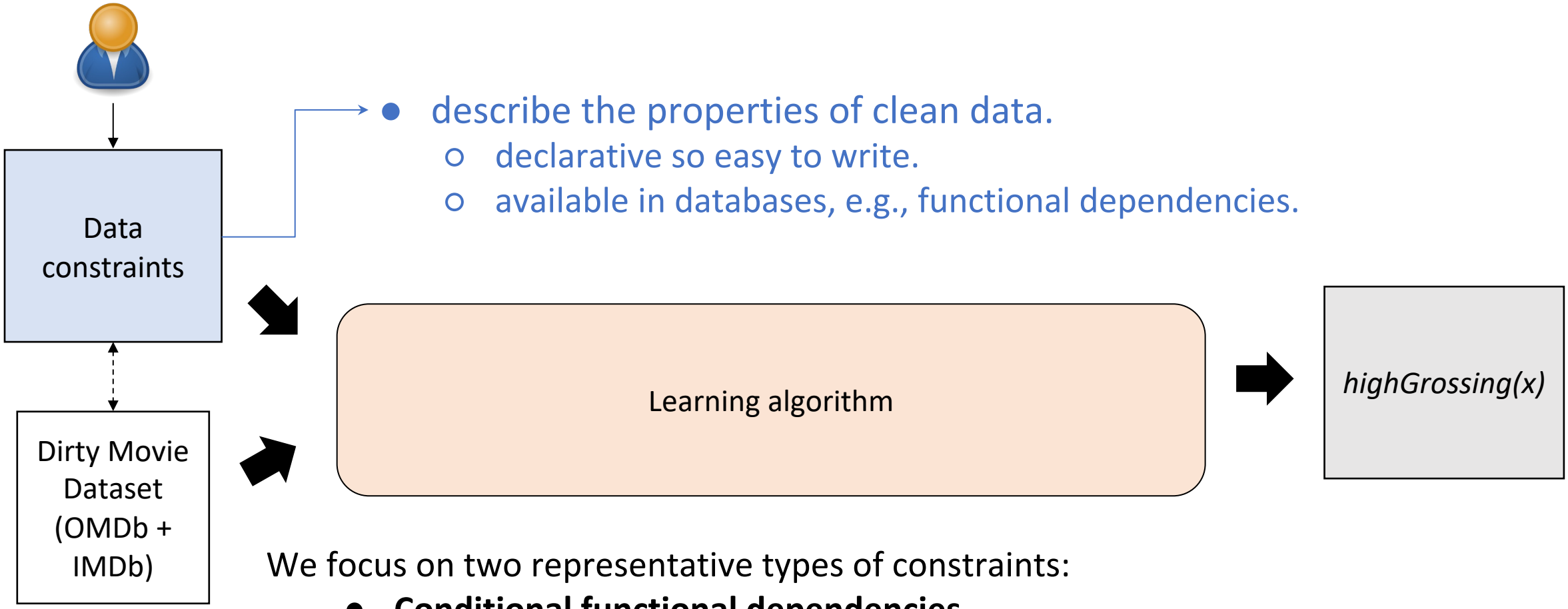


Advantage: algorithm considers/cleans only subset of data relevant to model.

Challenges on guidance

- ⚠ must be easy to provide.
- ⚠ integration in the learning algorithm.

Use data constraints to guide learning



- describe the properties of clean data.
  - declarative so easy to write.
  - available in databases, e.g., functional dependencies.

We focus on two representative types of constraints:

- **Conditional functional dependencies**
- **Matching dependencies**

Conditional functional dependency (CFD)

Generalizes functional dependency

→ “movies with the same title must be made in the same year if they were made in the same country”

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	Hunger Games, The	USA	2013
tt2365997	Hungover Games, The (2012)	USA	2012
tt3832730	They Hunger (2015)	Spain	2015
tt2387500	War Games (2012)	USA	2012
tt2338317	Hunger Games Project, The (2011)	France	2011

Dirty data violates CFD

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	Hunger Games, The	USA	2013
tt2365997	Hungover Games, The (2012)	USA	2012
tt3832730	They Hunger (2015)	Spain	2015
tt2387500	War Games (2012)	USA	2012
tt2338317	Hunger Games Project, The (2011)	France	2011

Titles and countries are the same, however:
2012 != 2013

CFD: Title → Distributor | Country

Challenge: hard to pinpoint the clean version of data

- Several possible repairs to resolve a violation of CFD.

- **not** clear which repair is accurate.

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	Hunger Games, The	USA	2013
tt2365997	Hungover Games, The (2012)	USA	2012
tt3832730	They Hunger (2015)	Spain	2015
tt2387500	War Games (2012)	USA	2012
tt2338317	Hunger Games Project, The (2011)	France	2011

Possible repairs:

2012 → 2013

or

2013 → 2012

or

"Hunger Games, The" → new title (XYZ)

….

Numerous possible repairs for a large dirty database

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2013
tt1392174	Hunger Games, The	USA	2013
	...		

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	Hunger Games, The	USA	2012
	...		

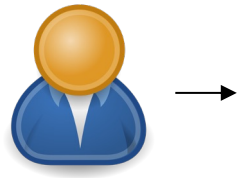
imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	xyz	USA	2013
	...		

Matching dependency (MD)

Resolves duplicates, connects entities.

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012
tt1392174	Hunger Games, The	USA	2013
tt2365997	Hungover Games, The (2012)	USA	2012
tt3832730	They Hunger (2015)	Spain	2015
tt2387500	War Games (2012)	USA	2012
tt2338317	Hunger Games Project, The (2011)	France	2011

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony
13	The Hunger Games: Catching Fire	Lionsgate
14	Younger Games	Warner
15	The 47th Hunger Games	Fox
16	They Hunger	Aurum



"If movies across two tables have **sufficiently similar** title, they have the same title."

$$imdbMovie[title] \approx omdbMovie[title] \rightarrow imdbMovie[title] \rightleftharpoons omdbMovie[title]$$

Same challenge: numerous /infinitely many repairs

Matching each of these similar titles satisfy the data constraint -> not clear **which one** is correct

imdbMovie			
id	title	country	year
tt1392170	Hunger Games, The	USA	2012

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Fox
13	The Hunger Games: Catching Fire	Lionsgate
14	Younger Games	Warner
15	The 47th Hunger Games	Sony
16	They Hunger	Aurum

$imdbMovie[title] \approx omdbMovie[title] \rightarrow$
$imdbMovie[title] \leftrightharpoons omdbMovie[title]$

not clear what the final correct title is, thus, infinitely many repairs

Current cleaning systems materialize repaired DB for learning

- **Challenge: manage many repaired DBs**

- generate only one or minimally repaired DB ➔ shown be inaccurate
- allow violations of constraints ➔ leads to inaccurate learning as data is dirty
- support only attributes with finite domains ➔ does not cover most real-world data

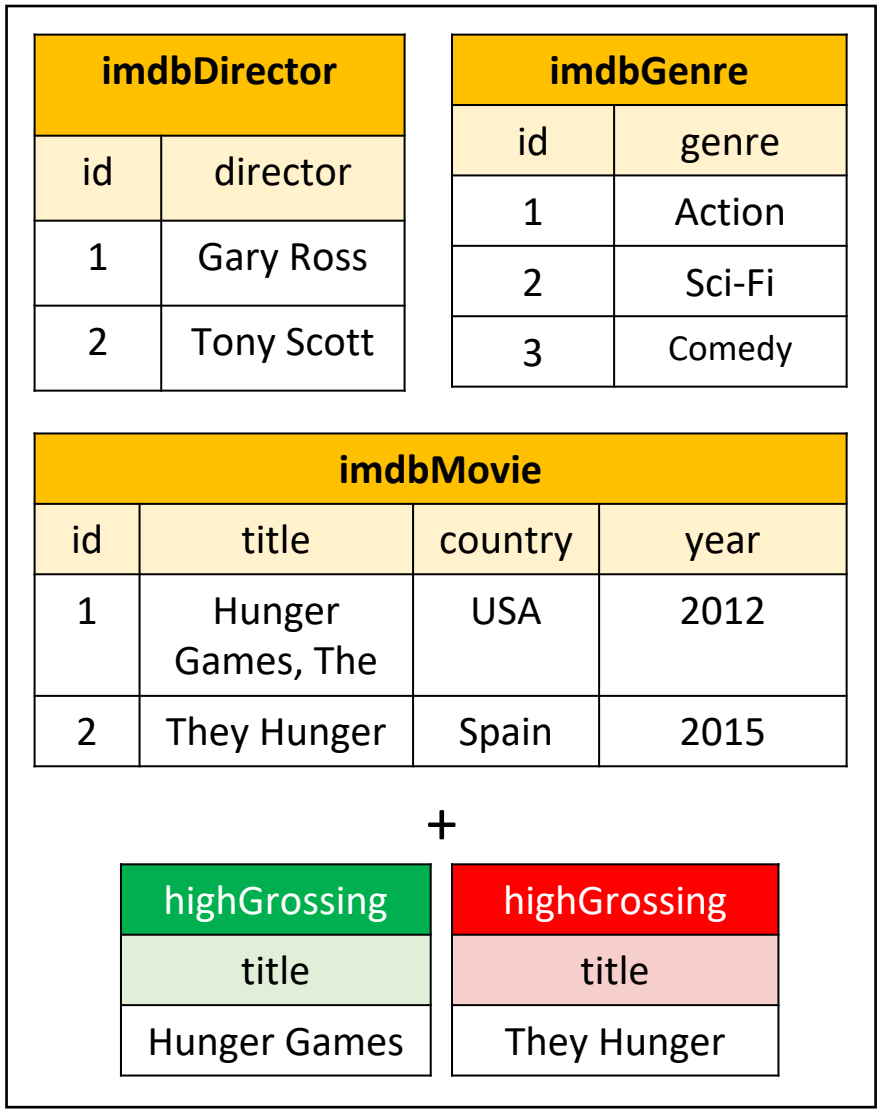
- **Challenge: again** when the original data changes: lengthy process

Our solution: producing & generalizing repairs while learning

Addresses problem of having too many repaired DBs:

- encode repairs in hypothesis language of learner
  - instead of producing repaired DBs.
- produces only repairs relevant to model
- generalization in learning reduces variation in model
  - will reduce repairs, too

We focus on relational machine learning



imdbDirector

id	director
1	Gary Ross
2	Tony Scott

imdbGenre

id	genre
1	Action
2	Sci-Fi
3	Comedy

imdbMovie

id	title	country	year
1	Hunger Games, The	USA	2012
2	They Hunger	Spain	2015

+

highGrossing
title
Hunger Games

highGrossing
title
They Hunger

- Non-relational ML, e.g., linear regression/NN, assume data is **IID**, i.e. no dependency in data
  - does **not** hold in relational data.
- (Statistical) relational models learn over non-IID data.

- Other benefits:
  - automatic feature extraction, interpretable
  - used with non-relational models

Example of Relational Learning

imdbDirector	
id	director
1	Gary Ross
2	Tony Scott

imdbGenre	
id	genre
1	Action
2	Sci-Fi
3	Comedy

imdbMovie			
id	title	country	year
1	Hunger Games, The	USA	2012
2	They Hunger	Spain	2015

+

highGrossing
title
Hunger Games

highGrossing
title
They Hunger

Relational learning algorithm

head literal

body literals

highGrossing(x) :- imdbMovie(y, x, USA, w), imdbGenre(y, Action)

highGrossing(x) :- imdbMovie(y, x, w, z), imdbDirector(y, Gary Ross)

(Weighted) Datalog definition

Prediction: Classify a new example by checking if learned definition **covers** the example

imdbDirector	
id	director
1	Gary Ross
2	Tony Scott

imdbGenre	
id	genre
1	Action
2	Sci-Fi
3	Comedy

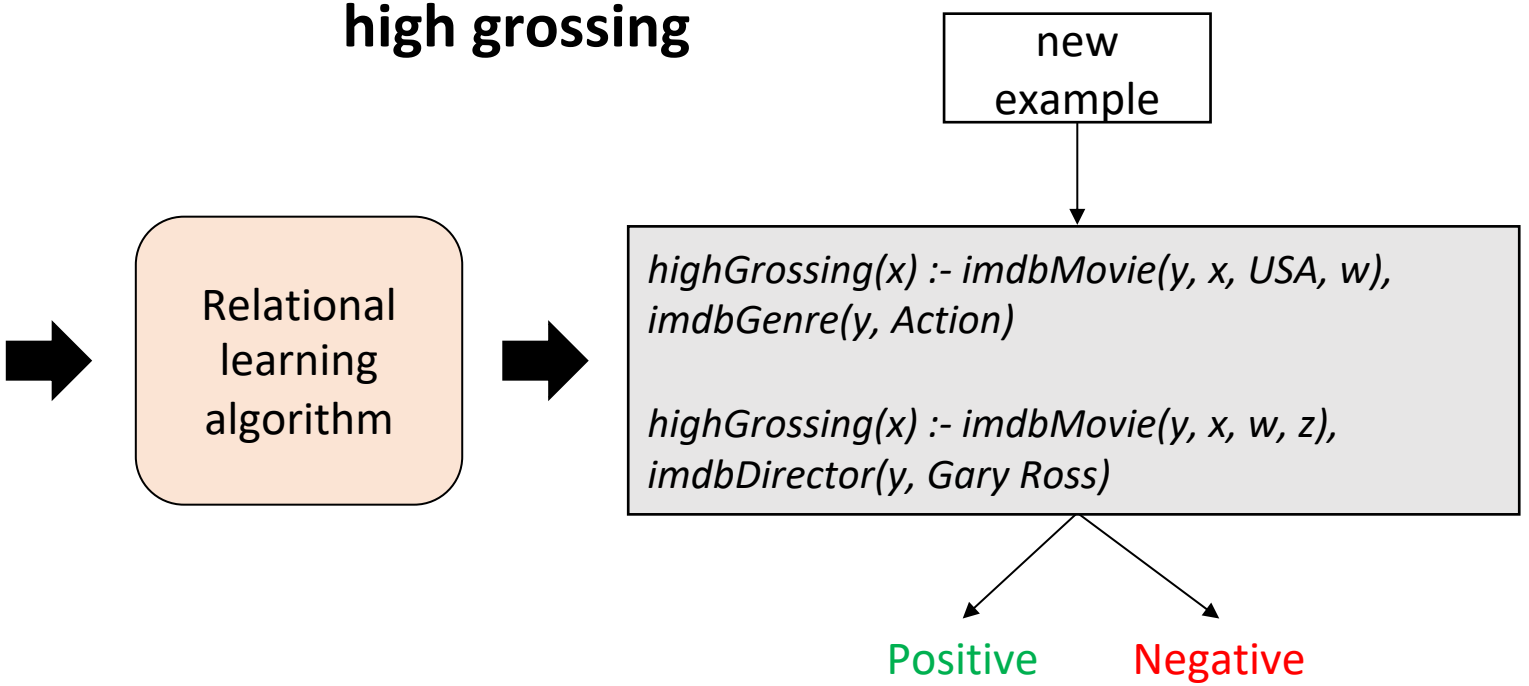
imdbMovie			
id	title	country	year
1	Hunger Games, The	USA	2012
2	They Hunger	Spain	2015

+

highGrossing
title
Hunger Games

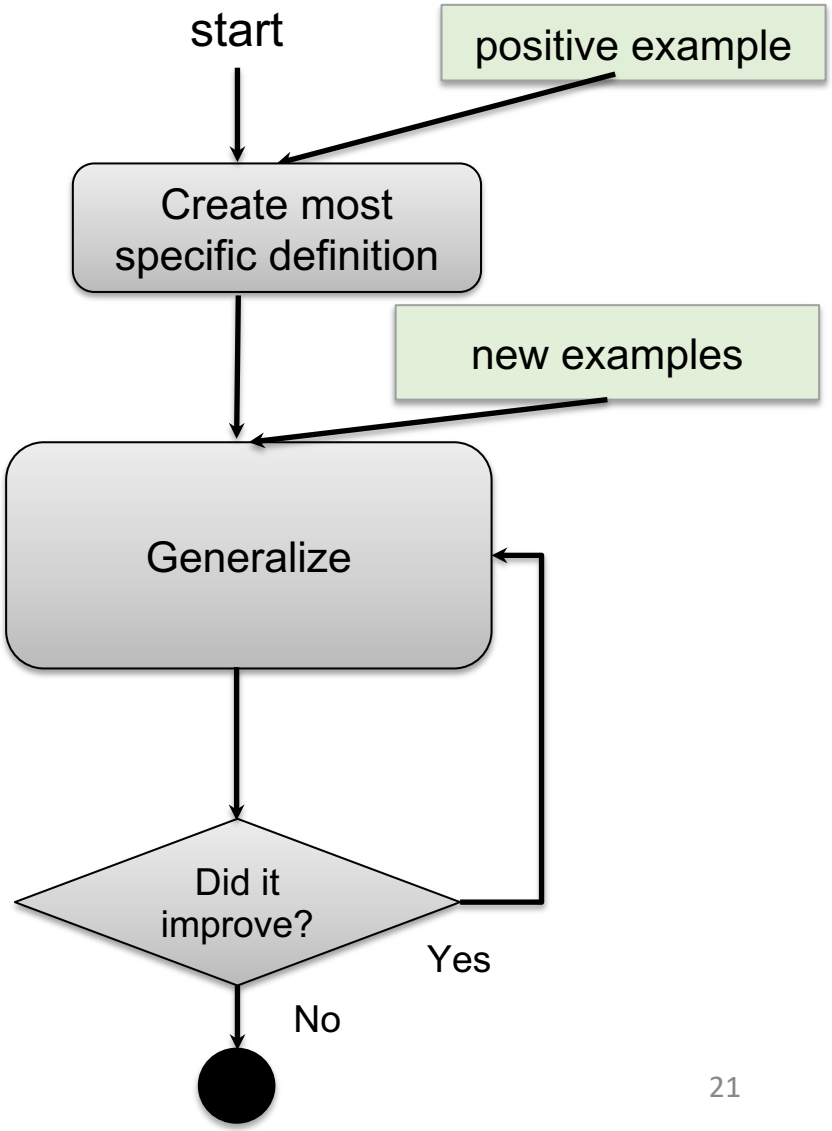
highGrossing
title
They Hunger

Predict whether a movie will be **high grossing**



Our Learning algorithm: DirtyLearn (**DLearn**)

- Extends current relational learning algorithm
- Bottom-up approach
  - find a specific Datalog definition that covers one positive
  - generalize to cover more positives
  - ensure not to cover too many negatives



Step 1. Create most specific definition for an example

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony

ϕ: *imdbMovie[title]* $\approx$ *omdbMovie[title]* $\rightarrow$
imdbMovie[title] $\leftrightharpoons$ *omdbMovie[title]*

positive example

tt1392170

Create most specific definition

highGrossing(v_0) :- .

Step 1. Create most specific definition

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony

ϕ: *imdbMovie[title]* ≈ *omdbMovie[title]* → *imdbMovie[title]* ⇋ *omdbMovie[title]*

positive example

tt1392170

Create most specific definition

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), *imdbGenre*(v_0,*Action*).

Extend hypothesis language: definition compactly encodes possible repairs of data

- For each constraint φ, we introduce a repair literal $\mathbf{m}_{\phi}$

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony

ϕ: *imdbMovie[title] ≈ omdbMovie[title] →*
imdbMovie[title] ⇋ omdbMovie[title]

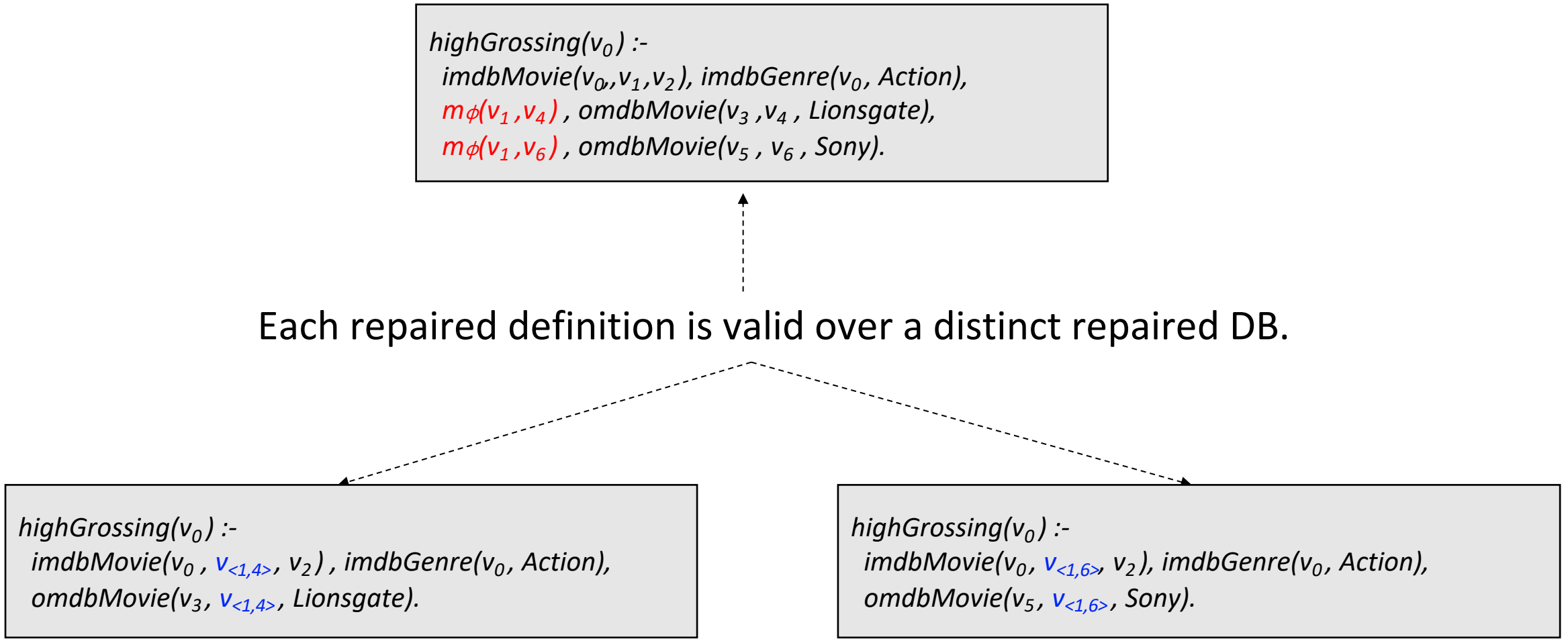
positive example

tt1392170

Create most specific definition

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
mϕ(v_1,v_4) , omdbMovie(v_3 ,v_4 , Lionsgate),
mϕ(v_1,v_6) , omdbMovie(v_5 , v_6 , Sony).

Each definition compactly represents multiple *repaired definitions*



Step 2: Generalize definition to cover other examples

- Change the definition to cover other positive examples
- By dropping literals, … for clean data.

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012
tt2911666	John Wick	2014

omdbMovie		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony
20	John Wick	Lionsgate

another example

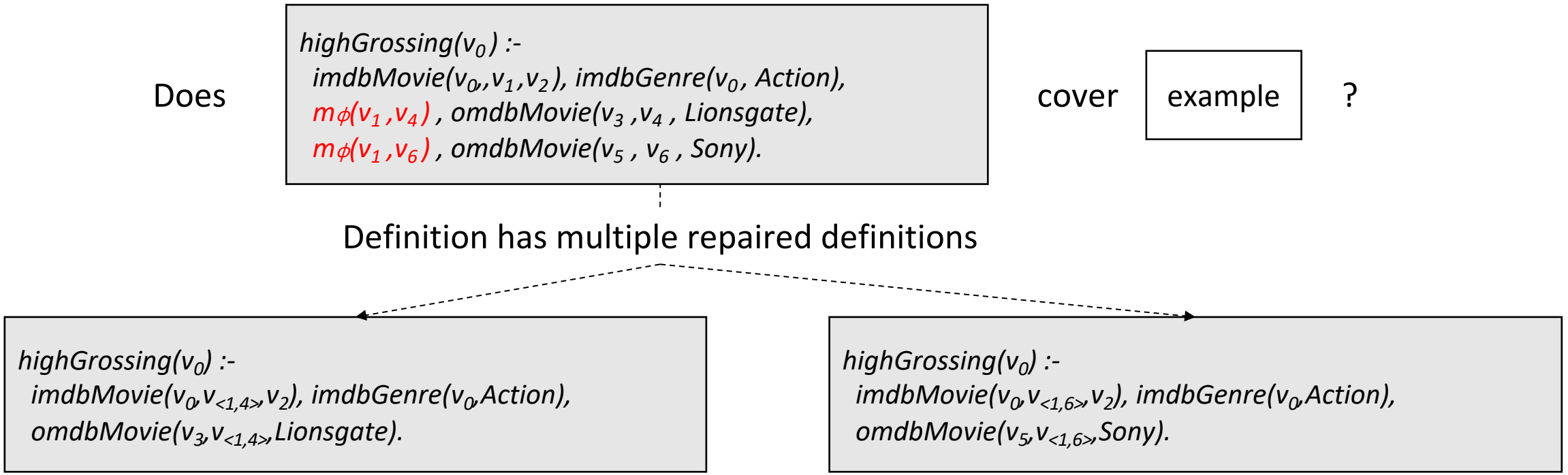
tt2911666

highGrossing(v_0) :-
…

Generalize to cover new example

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
$m\phi(v_1, v_4)$, omdbMovie(v_3 ,v_4 , Lionsgate),
$m\phi(v_1, v_6)$, omdbMovie(v_5 , v_6 , Sony).

How to define coverage for definitions with repair literals?



All / some repaired definition must cover the example in all/ some repaired DBs?

We need a semantic that is not too restrictive and not too loose.

We hit a middle-ground

Does

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
m_ϕ(v_1,v_4) , omdbMovie(v_3 ,v_4 , Lionsgate),
m_ϕ(v_1,v_6) , omdbMovie(v_5 , v_6 , Sony).

cover positive example ?

Definition covers a positive example if **each** repaired definition covers the example in some repaired database

highGrossing(v_0) :-
imdbMovie(v_0,$v_{<1,4>}$,v_2), imdbGenre(v_0,Action),
omdbMovie(v_3,$v_{<1,4>}$,Lionsgate).

highGrossing(v_0) :-
imdbMovie(v_0,$v_{<1,6>}$,v_2), imdbGenre(v_0,Action),
omdbMovie(v_5,$v_{<1,6>}$,Sony).

We hit a middle-ground

Does

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
$m\phi(v_1, v_4)$, omdbMovie(v_3 ,v_4 , Lionsgate),
$m\phi(v_1, v_6)$, omdbMovie(v_5 , v_6 , Sony).

cover negative example ?

Definition covers a negative example if **at least one**
of its repaired definitions covers the example in some repaired database

highGrossing(v_0) :-
imdbMovie(v_0, $v_{<1,4>}$, v_2), imdbGenre(v_0,Action),
omdbMovie(v_3, $v_{<1,4>}$, Lionsgate).

highGrossing(v_0) :-
imdbMovie(v_0, $v_{<1,6>}$, v_2), imdbGenre(v_0,Action),
omdbMovie(v_5, $v_{<1,6>}$, Sony).

Step 2: Generalize definition

- We extend current generalization method, e.g., dropping literals, to satisfy our semantic.
- Generalization also drops repair literals => reduce possible repairs

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012
tt2911666	John Wick	2014

omdbDistributor		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony
20	John Wick	Lionsgate

tt2911666

cover the new example by dropping these literals

highGrossing(v_0) :-
...

Generalize to cover new example

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
m_ϕ(v_1,v_4) , omdbMovie(v_3 ,v_4 , Lionsgate),
~~*m_ϕ(v_1,v_6) , omdbMovie(v_5 , v_6 , Sony).*~~

Step 2: Generalize definition

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012
tt2911666	John Wick	2014

omdbDistributor		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony
20	John Wick	Lionsgate

tt2911666

highGrossing(v_0) :-
…

Generalize to cover new example

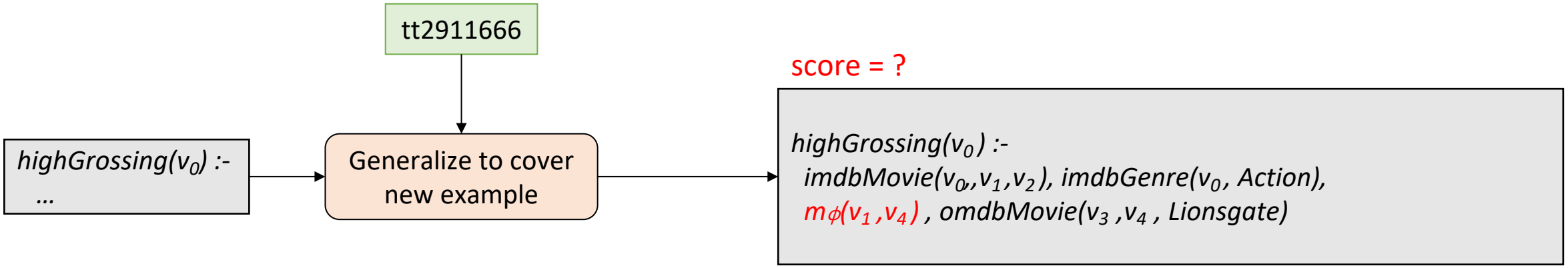
highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
$m_\phi(v_1, v_4)$, omdbMovie(v_3 ,v_4 , Lionsgate)

When to stop generalizing?

- Compute a score to ensure a definition covers enough positives and not too many negatives
  . e.g., number of covered positives minus number of negatives

imdbMovie		
id	title	year
tt1392170	Hunger Games, The (2012)	2012
tt2911666	John Wick	2014

omdbDistributor		
id	title	distributor
11	The Hunger Games	Lionsgate
12	The Hungover Games	Sony
20	John Wick	Lionsgate



Challenging to compute coverage score of a definition over many examples

How many examples does

$highGrossing(v_0)$:-
$imdbMovie(v_0, v_1, v_2)$, $imdbGenre(v_0, Action)$,
$m_\phi(v_1, v_4)$, $omdbMovie(v_3, v_4, Lionsgate)$

cover ?

Naive approach:

produce all its repaired definitions and compute coverage over repaired DBs.

Very time-consuming:

- too many repaired definitions.
- has to produce repaired databases.

We extend and use subsumption checking between Datalog definitions

How many examples does

highGrossing(v_0) :-
imdbMovie(v_0,v_1,v_2), imdbGenre(v_0, Action),
m_ϕ(v_1 ,v_4) , omdbMovie(v_3 ,v_4 , Lionsgate)

cover ?

1. Create the most specific definition for every example e, i.e., *ground definition* g_e
   - done once per example

2. Check if the definition *D* subsumes g_e
   - if *D* subsumes g_e , *D* covers *e*
   - we extend subsumption checking for definitions with repair literals.

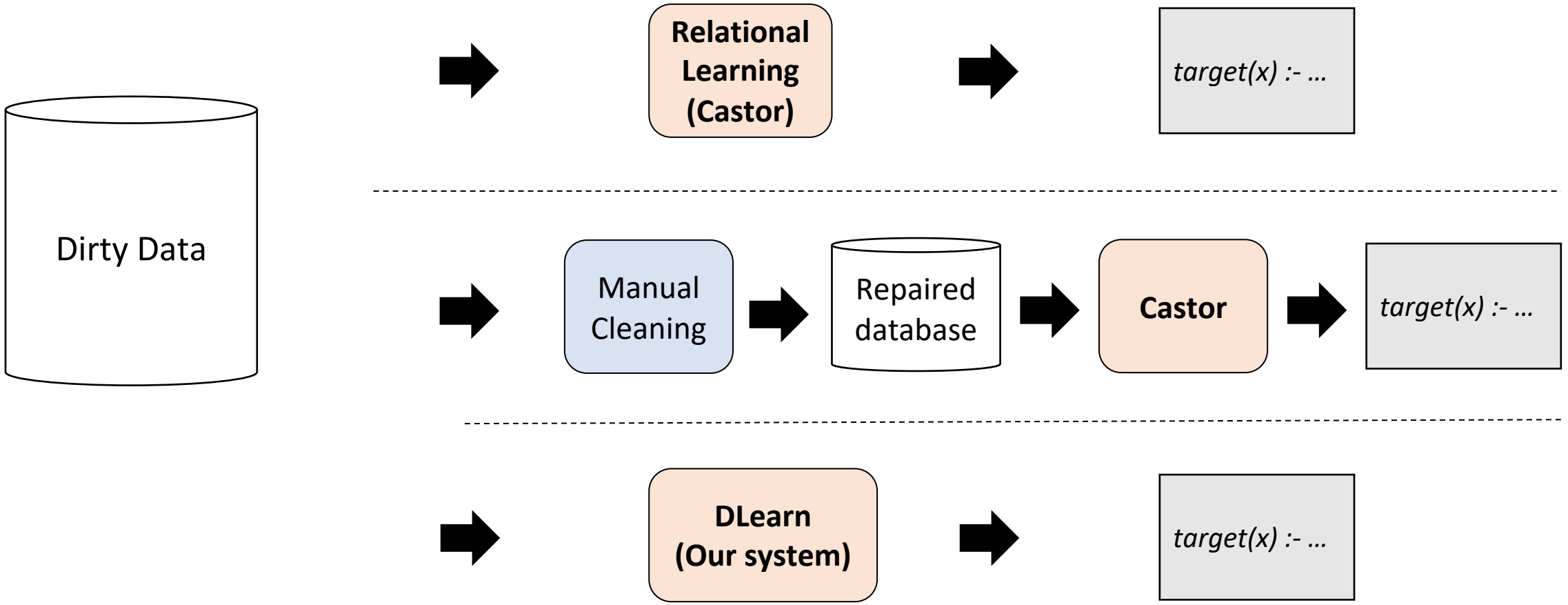
- does **not** generate repaired definitions/ databases -> **computes score efficiently**
- **Proofs and other interesting results in the paper!**

Empirical study – implementation/ datasets

We implement our system on top of an in-memory RDBMS (VoltDB).

Data	# of MDs	# of CFDs	Target relation
DBLP+Google Scholar	2	2	*GoogleScholarPaperYear(gsId, dblpYear)*
Walmart + Amazon	1	6	*upcComputersAccessories(upc)*
IMDB + OMDB	3	4	*dramaRestrictedMovies(imdbId)*

Empirical study – systems



Experiments Result Only Considering MDs

Data	Metric	Castor	Repaired data	DLearn
DBLP + Google Scholar	F1-score Time (m)	0 2.5	0.61 3.1	**0.71** 2.7
Walmart + Amazon	F1-score Time (m)	0.39 0.09	0.61 0.13	**0.63** 0.13
IMDB + OMDB	F1-score Time (m)	0.47 0.12	0.86 0.21	**0.93** 25.87

- DLearn is more effective than others.
- DLearn is reasonably efficient
  - does *not* need manual cleaning as opposed to learning over repaired data.

Experiments Result Considering MDs & CFDs

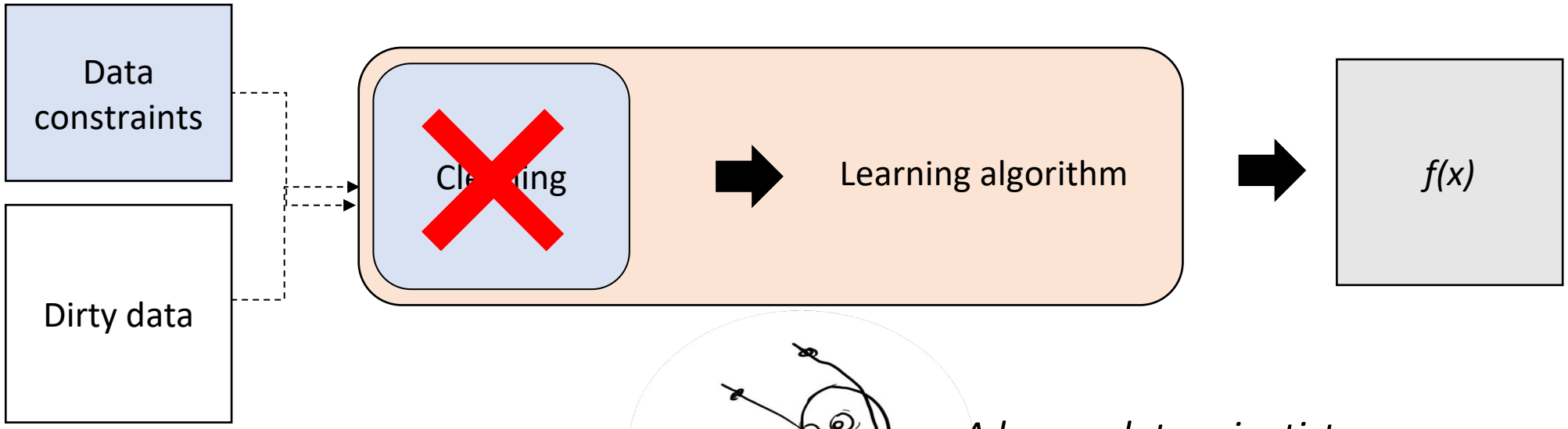
randomly injected CFD violations p= proportion of tuples having CFD violations

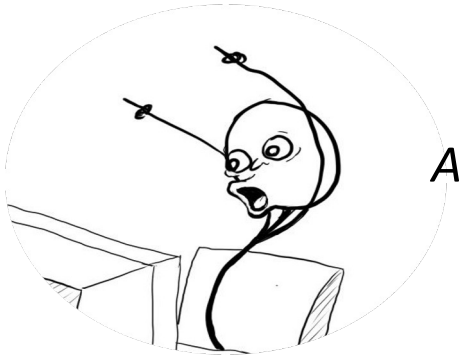
Data	Metric		DLearn p = 0.05	DLearn p = 0.10	DLearn p = 0.20		Repaired data p = 0.05	Repaired data p = 0.10	Repaired data p = 0.20
DBLP + Google Scholar	F1-score Time (m)		**0.79** 5.92	**0.68** 7.04	**0.47** 8.57		0.73 2.51	0.55 2.6	0.23 6.51
Walmart + Amazon	F1-score Time (m)		**0.64** 0.17	**0.61** 0.2	0.54 0.23		0.49 0.18	0.52 0.18	**0.56** 0.19
IMDB + OMDB (Three MDs)	F1-score Time (m)		**0.79** 11.15	**0.78** 16.26	**0.73** 26.95		0.76 5.70	0.73 12.54	0.50 22.28

Castor results inferior to other methods.

Conclusions & Future Work

- Use constraints in learning to eliminate cleaning and learn effective models efficiently.





A happy data scientist

- **Next step:** extend to other learning models and types of errors.