ExtracTable

Extracting Tables from Plain Text

Leonardo Hübscher

Prof. Felix Naumann
Lan Jiang

12.02.2021

Data preparation

Tasks	Collecting data	Cleaning data	Normalizing data
Benefits	Analysis	Machine learning	Data warehouses
Downside	Tedious tasks	Time consuming	Error prone

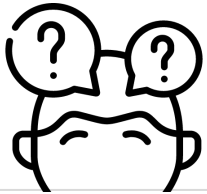
[1] https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/

Tables are great, right?

- Structured
- Store information in high densities
- Often used for data sharing
- Interpretable by humans and machines
- Tables look similar, yet they are so different...

```
2/28/2008,1:00:00,NaN
2/28/2008,2:00:00,NaN
2/28/2008,3:00:00,NaN
```

```
    X        Y        Z
--------  --------  --------
    0.00      0.00      0.00
```

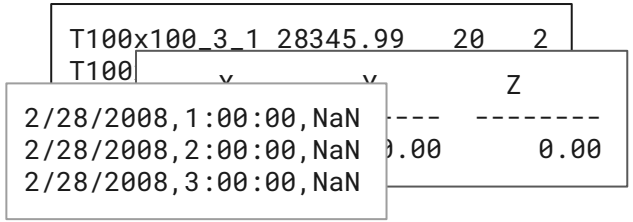


```
T100x100_3_1 28345.99   20   2
T100x100_3_2 29580.17   23   3
T100x100_3_3 27062.23   20   2
```

Vision

- Data preparation fuses information from different sources
- Valuable information is often stored in tables
- Tables are represented differently
- Manual pre-processing is time consuming

We want to decrease the time data scientists spend with data wrangling by extracting tables from plain text files automatically.



All kinds of table formats

2/28/2008	10:00:00	NaN
2/28/2008	11:00:00	83
2/28/2008	12:00:00	70

Standardized table format

Outline

1. Introduction
2. Background
3. Mission
4. Demo
5. Algorithm
6. Evaluation
7. Outlook

Parsing Instructions

CSV

```
“name“,“age“,“comment”
“Max Mustermann“,20,“max says \“hello\““
Peter Pan,30,
```

delimited by character

ASCII

name	age	comment
Max Mustermann	20	Max says “hello”
Peter Pan	30	

delimited by layout

Parsing Instructions

CSV

```
“name“,“age“,“comment“
“Max Mustermann“,20,“max says \“hello\““
Peter Pan,30,
```

delimited by character

Dialect

- Delimiter character — required
- Quotation character — optional
- Escape character — optional

ASCII

name	age	comment
Max Mustermann	20	Max says “hello”
Peter Pan	30	

delimited by layout

Indexes

19 30

Related Work

CleverCSV[2]
2019

Gertjan van den Burg
Postdoctoral researcher at The Alan Turing Institute

Dialect detection of CSV files based on row-patterns and cell data types

Both

- Work on file-level
- Expect single table per file
- Detect dialect of CSV tables

Hypoparsr[3]
2016

Till Döhmen
Research Assistant at Fraunhofer FIT

Improve quality by deferring decision-making for sub-problems like encoding & dialect detection

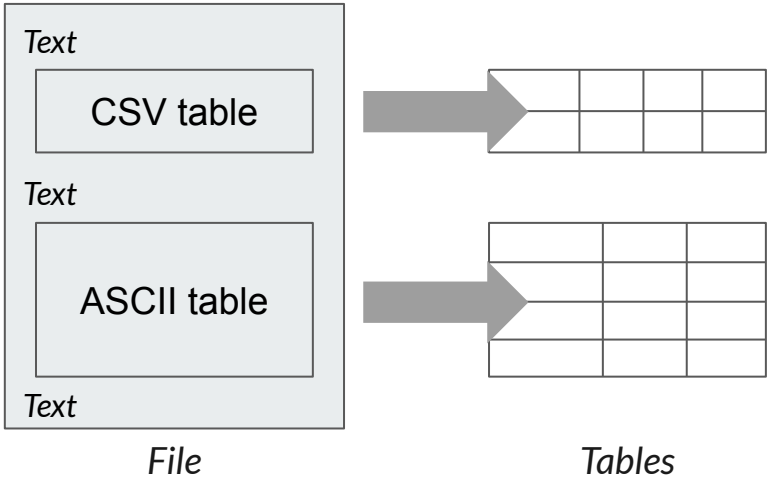
[2] van den Burg, G. J. J. (2019), ‘Wrangling messy CSV files by detecting row and type patterns’
[3] Döhmen, T. (2016), ‘Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files’

Mission

We want to develop an algorithm that is able to extract tables from more *complex* files compared to existing solutions.

Key features

- ASCII table support
- Support for more dialects (e.g. multi-character delimiters)
- Detection of multiple tables within a file (of different dialects)
- Support files with surrounding text



Demo

Algorithm Workflow

Input Lines

Tasks

1. Detect valid dialects for CSV table candidates (line-based)
2. Detect valid split indexes for ASCII table candidates (table-based)

Output Lines incl. parsing instructions

Example

2/28/2008,11:00:00,83

Detected parsing instructions

delimiter: / quotation: none escape: none
delimiter: , quotation: none escape: none
delimiter: : quotation: none escape: none
(no valid layout parsing instruction)

Algorithm Workflow

Read file → Detect parsing instr. → **Build solution spaces** → Extract line features → Build table candidates → Select tables

Input Lines incl. parsing instructions

Tasks

1. Apply parsing instructions to build solution space

Output Solution space for each line, containing one interpretation per parsing instruction

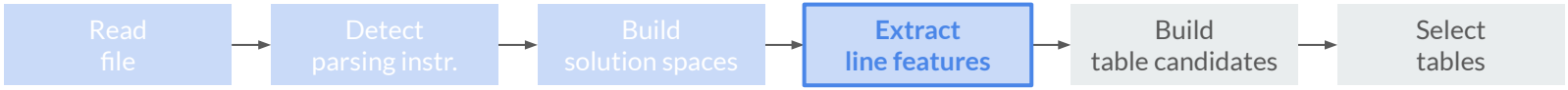
Example

2/28/2008,11:00:00,83

Solution space

2	28	2008,11:00:00,83
2/28/2008	11:00:00	83
2/28/2008,11	00	00,83

Algorithm Workflow



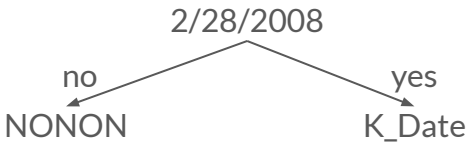
Input Solution space per line

Tasks

1. Describe table cell content by:
   - Recognizing known data types (RegEx-based)
   - Splitting into atomar components (String, *Number*, *Other*)

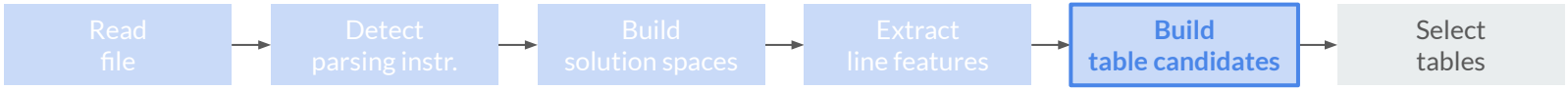
Output Solution space for each line, including features

Does a data type pattern match?



Number covers all kinds of numbers (int, signed number, floats, scientific)
Other matches everything that isn't a string or number component (similar to non-alphanumeric)

Algorithm Workflow



Input Solution space for each line, including features

Tasks

1. Group solution spaces into bins, use column count and parsing instruction as key
2. Split bins into compatibility blocks using some pattern-based consistency score

Output One table candidate per compatibility block

Algorithm Workflow

Task 1 Group solution spaces into bins, use column count and parsing instruction as key

date,time,humidity
2/28/2008,11:00:00,83
2/28/2008,12:00:00,70

Bin 1 3 columns, CSV, [delimiter=/]			**Bin 2** 3 columns, CSV, [delimiter=,]			**Bin 3** 3 columns, CSV, [delimiter=:]		
			date	time	humidity			
2	28	2008,11:00:00,83	2/28/2008	11:00:00	83	2/28/2008,11	00	00,83
2	28	2008,12:00:00,70	2/28/2008	12:00:00	70	2/28/2008,12	00	00,70

Algorithm Workflow

Task 2 Split bins into compatibility blocks using a column consistency score

Bin 1 3 columns, CSV, [delimiter=/]			**Bin 2** 3 columns, CSV, [delimiter=,]			**Bin 3** 3 columns, CSV, [delimiter=:]		
			date	time	humidity			
2	28	2008,11:00:00,83	2/28/2008	11:00:00	83	2/28/2008,11	00	00,83
2	28	2008,12:00:00,70	2/28/2008	12:00:00	70	2/28/2008,12	00	00,70

Bin 1 3 columns, CSV, [delimiter=/]			**Bin 2** 3 columns, CSV, [delimiter=,]			**Bin 3** 3 columns, CSV, [delimiter=:]		
			K_Text	K_Text	K_Text			
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN

Algorithm Workflow

Task 2 Split bins into compatibility blocks using a column consistency score

Bin 1 3 columns, CSV, [delimiter=/]			Bin 2 3 columns, CSV, [delimiter=,]			Bin 3 3 columns, CSV, [delimiter=:]		
			date	time	humidity			
2	28	2008,11:00:00,83	2/28/2008	11:00:00	83	2/28/2008,11	00	00,83
2	28	2008,12:00:00,70	2/28/2008	12:00:00	70	2/28/2008,12	00	00,70

Bin 1 3 columns, CSV, [delimiter=/]			Bin 2 3 columns, CSV, [delimiter=,]			Bin 3 3 columns, CSV, [delimiter=:]		
			K_Text	K_Text	K_Text			
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN

Algorithm Workflow

Bin 1 3 columns, CSV, [delimiter=/]			Bin 2 3 columns, CSV, [delimiter=,]			Bin 3 3 columns, CSV, [delimiter=:]		
			K_Text	K_Text	K_Text			
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN
N	N	NONNONN	K_Date	K_Time	N	NONON	NN	NN

N	N	NONNONN
N	N	NONNONN

K_Text	K_Text	K_Text
K_Date	K_Time	N
K_Date	K_Time	N
K_Text	K_Text	K_Text
K_Date	K_Time	N
K_Date	K_Time	N

NONON	NN	NN
NONON	NN	NN

Algorithm Workflow

Read file → Detect parsing instr. → Build solution spaces → Extract line features → Build table candidates → **Select tables**

Input Table candidates

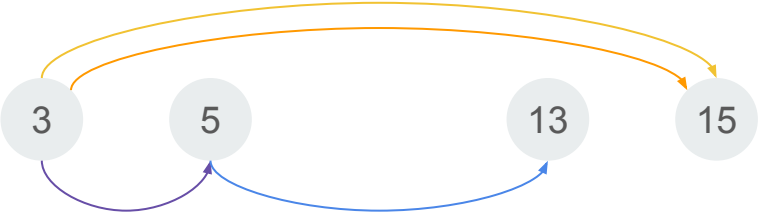
Tasks

- map table selection problem to multi-edge DAG
  V = line indexes E = table candidate
  distances are based on consistency and size
- use shortest path to find optimal solution

Output Selection of standardized tables

Table candidates

Table 1	start: 3	end: 15	distance: -135.2	✔
Table 2	start: 3	end: 5	distance: -9.0	✖
Table 3	start: 5	end: 13	distance: -56.7	✖
Table 4	start: 3	end 15	distance: -101.4	✖



Insight: ASCII Table Detection

Read file → **Detect parsing instr.** → Build solution spaces → Extract line features → Build table candidates → Select tables

Task Find the correct split indexes for a given ASCII table

```
1: Topic                                                Student            Supervision
2: Extracting Plain Tables from Text                    Leonardo Hübscher  Felix Naumann, Lan Jiang
3: Distributed Duplicate Detection on Streaming-Data    Jakob Köhler       Thorsten Papenbrock
4: Multi-Aspect Embeddings for Fiction Novels           Lasse Kohlmeyer    Ralf Krestel, Tim Repke
5: Generating Rap Lyrics with Flow and Rhythm           Noel Danz          Ralf Krestel, Tim Repke
```

Idea Detect split indexes by finding vertical lines of white spaces

Insight: ASCII Table Detection

```
1: Topic                                                Student            Supervision
2: Extracting Plain Tables from Text                    Leonardo Hübscher  Felix Naumann, Lan Jiang
3: Distributed Duplicate Detection on Streaming-Data    Jakob Köhler       Thorsten Papenbrock
4: Multi-Aspect Embeddings for Fiction Novels           Lasse Kohlmeyer    Ralf Krestel, Tim Repke
5: Generating Rap Lyrics with Flow and Rhythm           Noel Danz          Ralf Krestel, Tim Repke
```

transform to bitmap
1 = white space

```
1: Topic                                                Student            Supervision
1: 00000111111111111111111111111111111111111111111111111000000011111111111000000000001111111111111
```

Insight: ASCII Table Detection

```
1: Topic                                                Student            Supervision
2: Extracting Plain Tables from Text                    Leonardo Hübscher  Felix Naumann, Lan Jiang
3: Distributed Duplicate Detection on Streaming-Data    Jakob Köhler       Thorsten Papenbrock
4: Multi-Aspect Embeddings for Fiction Novels           Lasse Kohlmeyer    Ralf Krestel, Tim Repke
5: Generating Rap Lyrics with Flow and Rhythm           Noel Danz          Ralf Krestel, Tim Repke
```

transform to bitmap
1 = white space

```
1: 000001111111111111111111111111111111111111111111111000000011111111111000000000001111111111111
2: 000000000010000010000001000010000111111111111111111000000001000000001100000100000000100010000
3: 000000000001000000000100000000010010000000000000111000001000000111111100000000100000000011111
4: 000000000000100000000001000100000001000000111111111000001000000000111100001000000001000100001
5: 000000000001000100000100001000010001000000111111111000010000111111111100001000000001000100001
```

Insight: ASCII Table Detection

```
1 : 0000011111111111111111111111111111111111111111111111000000011111111111100000000000111111111111
2 : 0000000000100000100000010000100001111111111111111111000000001000000001100001000000001000100000
3 : 0000000000100000000001000000000100100000000000000111000001000000111111100000000100000000000011111
4 : 0000000000001000000000001000100000001000000111111111100000100000000011110001000000001000100000 1
5 : 0000000000100010000001000010000100010000001111111111000010000111111111100001000000001000100000 1
```

Idea Group consecutive vertical lines into blocks

Block width (number of characters)
height (number of rows)

Insight: ASCII Table Detection

```
1 : 0000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
2 : 0000000000100000100000010000100001111111111111111111000000001000000001100000100000000100001000000
3 : 0000000000010000000001000000000100100000000000000111000001000000111111100000000100000000000011111
4 : 0000000000001000000000010001000000010000001111111111000001000000000111100001000000001000100000 1
5 : 0000000000100010000001000010000100010000001111111111000010000111111111100001000000001000100000 1
```

When is a table started? If a block of length equal to T_{min_rows} (2) appears

Insight: ASCII Table Detection

```
1 : 0000011111111111111111111111111111111111111111111111111111000000001111111111110000000000001111111111111
2 : 0000000000100000100000010000100001111111111111111111111000000000100000000110000010000000001000100000
3 : 0000000000010000000001000000000100100000000000000111000001000000111111100000000100000000000011111
4 : 0000000000001000000000001000100000001000000111111111100000100000000011110001000000000100010000001
5 : 0000000000100010000001000010000100010000001111111111000010000111111111110001000000000100010000001
```

What happens if a block is continued only partially? Split the block

What happens if a block is discontinued?

- If width = 1 and height < $T_{essential}$ <u>or</u> block is leading/ trailing
  a. Remove block from table t
- Otherwise:
  a. Copy table t and remove block from copy
  b. Add table t to final table set

Insight: ASCII Table Detection

```
1: 00000111111111111111111111111111111111111111111111111111000000011111111111110000000000011111111111111
2: 00000000001000001000000100001000011111111111111111111111000000001000000001100000100000000100010000 0
3: 00000000000100000000010000000000100100000000000000111100000100000011111110000000010000000000011111
4: 00000000000010000000000010001000000010000001111111111100000100000000011110000100000000100010000001
5: 00000000001000100000010000100001000100000011111111110000100001111111111000010000000010001000001
```

- Existing blocks are continued
- A new table appeared (we will ignore it for now)

Insight: ASCII Table Detection

```
1 : 0000011111111111111111111111111111111111111111111111000000011111111111100000000000111111111111
2 : 0000000000100000100000010000100001111111111111111111000000001000000000110000010000000001000100000
3 : 0000000000010000000001000000000010010000000000000111000000100000011111110000000010000000000011111
4 : 0000000000001000000000001000100000001000000011111111110000010000000000111100001000000000100010000001
5 : 0000000000010001000000100001000010001000000011111111110000100001111111111100001000000000100010000001
```

We reached the final line: all blocks are discontinued

What happens if a block is discontinued?

- If width = 1 and height < $T_{essential}$ <u>or</u> block is leading/ trailing
  a. Remove block from table t
- Otherwise:
  a. Copy table t and remove block from copy
  b. Add table t to final table set

Insight: ASCII Table Detection

```
1: 0000011111111111111111111111111111111111111111111111000000011111111111100000000000011111111111111
2: 0000000000100000100000010000100001111111111111111111000000001000000000110000010000000001000100000
3: 0000000000010000000001000000000010010000000000000111000000100000011111110000000010000000000011111
4: 0000000000001000000000010001000000010000001111111111000000100000000001111000010000000001000100001
5: 0000000000010001000000100001000010001000000111111111000010000111111111110000100000000100010000001
```

	Topic	Student	Supervision
1:	Topic	Student	Supervision
2:	Extracting Plain Tables from Text	Leonardo Hübscher	Felix Naumann, Lan Jiang
3:	Distributed Duplicate Detection on Streaming-Data	Jakob Köhler	Thorsten Papenbrock
4:	Multi-Aspect Embeddings for Fiction Novels	Lasse Kohlmeyer	Ralf Krestel, Tim Repke
5:	Generating Rap Lyrics with Flow and Rhythm	Noel Danz	Ralf Krestel, Tim Repke

Evaluation

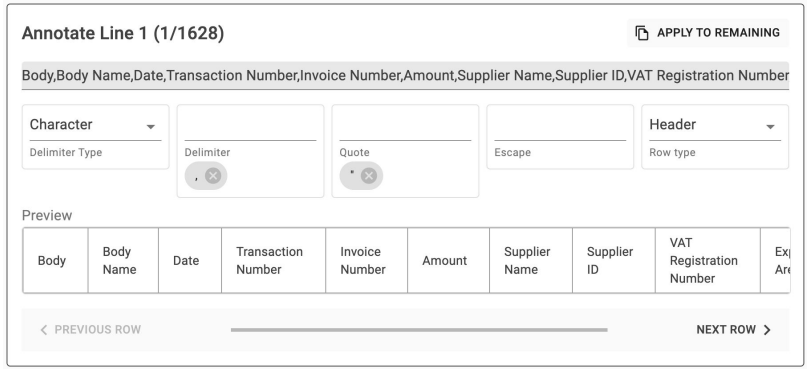
Dataset 1,000 annotated files from Mendeley, UKData, GitHub

Annotations

- Parsing instructions (line-level)
- Table ranges
- Row types

Complexity levels of files

- Simple single: 1 Table, w/o surrounding text
- Complex single: 1 Table, with surrounding text
- Complex multi: >1 Table



Screenshot of line annotation

Evaluation

Line classification
(table/ non-table)

File complexity	Accuracy	Precision	Recall	F1	Balanced accuracy
simple	0.999	1.000	0.999	1.000	1.000
complex-single	0.995	0.996	0.998	0.997	0.979
complex-multi	0.919	0.917	0.997	0.955	0.697

Evaluation

Line classification (table/ non-table)

File complexity	Accuracy	Precision	Recall	F1	Balanced accuracy
Simple single	0.999	1.000	0.999	1.000	1.000
Complex single	0.995	0.996	0.998	0.997	0.979
Complex multi	0.919	0.917	0.997	0.955	0.697

Parsing correctness (line-level)

Table type	Correct (cleaned)*
CSV	**88.5 %**
ASCII	**71.8 %**

**Cleaned: Filter on (1) files that did not time out (2) lines that have been classified correctly*

What's Next?

Next Steps

- Continue with evaluation (comparison to existing solutions)
- (Work on improvements as indicated by evaluation)
- If we have some spare time left: demo website
- Deadline: 22.04 → start writing in two weeks

Future Work

- Add support for spanning rows/ columns

Overview

Read file → Detect parsing instr. → Build solution spaces → Extract line features → Build table candidates → Select tables

Key features

- ASCII table support
- Support for more dialects (e.g. multi-character delimiters)
- Detection of multiple tables within a file (of different dialects)
- Support files with surrounding text