Detecting CSV File Dialects by Table Uniformity Measurement and Data Type Inference

Wilfredo García[1] [a,*]
[a] *CEO office, ECP Solutions, Santiago, República Dominicana*
E-mail: wilfredo_garcia@outlook.es

Abstract. The human-readable simplicity with which the CSV format was devised, together with the absence of a standard that strictly defines this format, has allowed the proliferation of several variants in the dialects with which these files are written. The latter has meant that the exchange of information between data management systems, or between countries and regions, requires human intervention during the data mining and cleansing process. This has led to the development of various computational tools that aim to accurately determine the dialects of CSV files, in order to avoid data loss at data loading stage in a given system. However, the dialect detection is a complex problem and current systems have limitations or make assumptions that need to be improved and/or extended. This paper proposes a method for determining CSV file dialects through table uniformity, a statistical approach based on table consistency and records dispersion measurement along with the detection of data type over each field. The new method has a 100% accuracy on a dataset with 148 CSV files composed of samples coming from a data load testing framework and some others added as verification of the parsing routines. In tests on truly messy data, the proposed solution outperforms the state-of-the-art tool by achieving an improvement of about 10% in the accuracy with which dialects are detected. Furthermore, the proposed method is accurate enough to determine dialects by reading only ten records, requiring more data to disambiguate those cases where the first records do not contain the necessary information to conclude with a dialect determination.

Keywords: Comma Separated Values, CSV dialect detection, Data Mining, Data Wrangling

1. Introduction

The CSV files are a special kind of tabulated plain text data container widely used in data exchange, currently there is no defined standard for CSV file's structure and a multitude of implementations and variants. Notwithstanding the foregoing, there are specifications such as RFC-4180 that define the basic structure of these files, while a useful addendum to this is defined in the specifications of the USA Library of Congress (LOC) [13]. According to the LOC specifications the CSV simple format is intended for representing a rectangular array (matrix) of numeric and textual values. “It is a delimited data format that has fields/columns separated by the comma character %x2C (Hex 2C) and records/rows/lines separated by characters indicating a line break. RFC 4180 stipulates the use of CRLF pairs to denote line breaks, where CR is %x0D (Hex 0D) and LF is %x0A (Hex 0A). Each line should contain the same number of fields. Fields that contain a special character (comma, CR, LF, or double quote), must be "escaped" by enclosing them in double quotes (Hex 22). An optional header line may appear as the first line of the file with the same format as normal record lines. This header will contain names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file. CSV commonly employs US-ASCII as character set, but other character sets are permitted” [9]. Furthermore, so far to the specifications, in a file may exist: commented or empty records; the tab character (\t) or semicolon (;) as field delimiter; one or more, in exceptional cases, of the characters CRLF, CR, and LF as a record delimiter; quote character escaped by preceding it with a backslash (Unix style).

Given that many public administration portals use CSV files to share information of public interest[1], coupled with the reality that the process of manipulating the information contained in them requires structuring the data in tables and correcting data quality errors, it is necessary to automate tasks as much as possible to reduce the time and effort required to deal with messy CSV data [10, 14]. The automation problem focuses on seeking the delimiters (also called dialect sniffing) of a given file. Dialect sniffing requires that the field delimiter, record delimiter and escape character be determined [15].

[*]Corresponding author. E-mail: wilfredo_garcia@outlook.es.
[1]An analysis of a 413 GB data body found CSV files available for download on 232 portals.

```
Acme Ltd.;£1.800,80;£5.400,50
Global Corp.;£2.100,00;£3.020,30
```

Fig. 1. CSV that cannot be disambiguated by a simple delimiter count

Acme Ltd	;£1	800,80;£5	400,50
Global Corp	;£2	100,00;£3	020,30

Acme	Ltd.;£	1.800,80;£	5.400,50
Global	Corp.;£	2.100,00;£	3.020,30

Fig. 2. misinterpreted data using the "most frequent char" strategy

This problem seems straightforward, but it is by no means simple. If one opts to implement a simple field delimiter counter to choose the one with the most occurrences in the entire file, it is very likely that disambiguation will become impossible if the algorithm is confronted with data that have two or more delimiters with the same number of matches.

A CSV file with a structure as shown in Figure 1 is at risk of being misinterpreted, this is illustrated in [4]. If delimiters are counted, the period or space will be selected as field delimiters because of their three constant occurrences, generating four fields, in the records, as opposed to the two occurrences and three fields generated by the comma and semicolon. Although a well-defined file should have a header row, there are many files on the Internet that do not [14].

It is a fact that systems that work with CSV files may require the user to set the configuration with which they want the file to be processed, however, when the intention is to analyze data coming from different sources, it is very beneficial to implement a methodology that allows to automatically infer CSV dialects with minimal user intervention.

In this sense, CSV file dialect inference is a fundamental part of data mining, data wrangling and data cleansing environments [14]. Moreover, dialect detection has the potential to be embedded in systems designed for the new paradigm with the NoDB philosophy, under which it is proposed to make databases systems more accessible to users [1, 8]. These trends suggest that the traditional practice of considering CSV files outside of database systems is tending to change [7].

2. Related work

Dialect detection in CSV files is an understudied field, and there are few sources on the subject. In 2017, T. Döhmen proposed the ranking decision method based on quality hypotheses for parsing CSV files. A similar method is implemented in the DuckDB system [5]. Another treatment, based on the discovery of the table structures once the information is loaded into the RAM, is ad-dressed by C. Christodoulakis et. al. [3]. This latter methodology uses the classification of records present in CSV files with a specific heuristic applied to discover and interpret each line of data.

In 2019, G. van den Burg et al, developed the CleverCSV system as a culmination of his research, in which he demonstrated that the methodology significantly improved the accuracy for dialects determination problem compared to tools such as Python's csv module, or the intrinsic functions of the Pandas package, also in the Python programming language. The implementation of CleverCSV is based on detection of patterns in the structure of CSV records, in addition to data types inference over the fields that compose each record. In this way, the utility applies necessary heuristics to seek the potential dialect for a given CSV file through mathematical and logical operations devised to discern between possible dialects [15].

In 2023, Leonardo Hübscher et al, presented a research project that led to the development of a software application capable of detecting tables in text files. This research considers the dialect determination of CSV files as a subproblem to be solved in order to seek the dialect that produces the best table [6].

3. Problem formulation

Properly formulating the dialect detection problem requires establishing certain fundamental definitions.

Definition 1. *(CSV content). Given a CSV file Υ, its content is defined as $\xi\{\xi_1, \xi_2, ..., \xi_n\}$, where $\xi_i \epsilon \Omega$ and Ω represents a character set encoded using a given encoder.*

As per the CSV content definition, there is a real possibility that a single CSV file contains characters encoded in more than one encoder. For the purposes of this document, it is assumed that all characters share the same encoding.

Given that each file Υ originates from a table Γ to which a format $\Psi(\Gamma, \rho)$ and the helper function $W(\xi)$ have been applied to produce and write a sequence of human readable characters separated by lines; then from each CSV content ξ is possible to obtain a table Γ_δ so that we can verify $\Gamma_\delta = \Psi^{-1}(\xi_\delta \leftarrow R(\Upsilon), \rho_\delta)$.

Definition 2. *(CSV table). A table Γ_δ is defined as a set of records composed of a given set of fields, which share data types between corresponding fields across their records. This table can be represented as a data array of fields and records. Thus, its records are defined as $\Phi\{\varphi_1, \varphi_2, ..., \varphi_n\}$; i.e. a set of fields φ_i; $i \in [1, 2, ..., k]$. Then, the table can be expressed as $\Gamma_\delta\{\Phi_1, \Phi_2, ..., \Phi_n\}$; i.e. a set of records Φ_i; $i \in [1, 2, ..., n]$.*

The function $R(\Upsilon)$ is in charge of reading content from the file Υ, while the function $\Psi^{-1}(\xi_\delta, \rho_\delta)$ parses and transforms the CSV content ξ_δ into a table Γ_δ. The parsing and transformation processes is clearly out of this study scope, so in the following it is assumed that the selected implementation is able to process the tables obtained by parsing a CSV file with the selected tool.

Definition 3. *(CSV dialect). Let Γ be the data table from which the content ξ_δ of file Υ is generated, the dialect ρ is defined as the formatting rule to be applied to produce the output data stream.*
So that, by the dialect definition, the following statement is verified:
$\Upsilon \leftarrow W(\xi \leftarrow \Psi(\Gamma, \rho)); \rho\{\upsilon_d, \upsilon_q, \upsilon_e, \upsilon_r\} \in \Omega$.

Definition 4. *(CSV dialect determination). Given a CSV file Υ determining the dialect is the act of seeking the dialect ρ_δ that satisfies the statement $\Gamma \approxeq \Gamma_\delta \leftarrow \Psi^{-1}(\xi_\delta \leftarrow R(\Upsilon), \rho_\delta)$.*

Thus, it can be concluded that for a CSV file Υ, created using a dialect ρ, there exists a dialect ρ_δ that verifies the condition $\Gamma \approxeq \Gamma_\delta$. Therefore, it is verifiable that the content of a CSV file is a function of its dialect.

3.1. Potential dialect boundaries

It should be noted that multiple potential dialects can produce similar table outputs that are equal or approximately equal to the source table Γ. Furthermore, ρ_δ shares the same character set as the contents ξ for the CSV file Υ. That is, an element from ρ_δ can be practically any character within Ω domain. Thus, it is necessary to reduce the range of candidate characters involved in dialect detection to streamline the process.

For the purposes of this research, the potential dialect is restricted to
$\rho_\delta\{$
$\upsilon_d[\text{"},\text{" ";" } TAB \text{"}|\text{" " : " } SPACE],$
$\upsilon_q[\text{"""" "'" " } \sim \text{"}],$
$\upsilon_e[\upsilon_q \text{"}\backslash\text{"}],$
$\upsilon_r[CRLF\ CR\ LF]\}^2$

4. Table uniformity

The table uniformity approach is proposed to solve the problem of dialect determination. The method is based on consistency measurement over a table Γ_δ, which has been returned by parsing a CSV file with a dialect ρ_δ, and the dispersion of records along with the inference of raw data types from fields.

Definition 5. *(Table consistency). Let Γ_δ be a table generated when parsing a CSV file Υ, using a dialect ρ_δ, the table consistency, denoted by τ_0, is a ratio that describes how uniform a table is across its k fields and its n records.*

Definition 6. *(Records dispersion). Let Φ be the sets of records from table Γ_δ, generated when parsing a CSV file Υ using a dialect ρ_δ, the records dispersion, denoted by τ_1, is a measure describing the magnitude of the change in the records composition throughout the table.*

[2] In most applications the record delimiter υ_r is not considered, as modern systems handle new lines discrepancies internally.

These definitions are based on the fact that tables, in general, have a defined structure with persistent k fields in its n records.

The two measurements that define the table uniformity parameter $\tau\{\tau_0, \tau_1\}$ are related to the structure of records Φ from a table Γ_δ. Where τ_0 is a direct function of the standard deviation of fields, and τ_1 is a function measuring the weighted dispersion in records structures as a factor of the statistical segmented mode[3].

$$\tau_0 = \frac{1}{1+2\sqrt{\sigma}}; \; \tau_1 = 2 \cdot R(\alpha^2 + 1)(\frac{1-\beta}{M})$$

Where, for a given table Γ_δ, σ is the number of fields standard deviation across records; α represents the count of times number of fields changes between records; R is the statistical range for the number of fields over records; M is the segmented mode, describing the largest number of times the record structure is sequentially preserved within the table, and $\beta = \frac{M}{n}$ is the records variability factor.

The definitions provided propose a concept diametrically opposed to that used in most solutions, since it discourages data dispersion, i.e. records with a higher number of fields/columns are only favored if their record structure is uniform.The parameter τ_0 indicates the degree of consistency for the records in a table, while τ_1 is a fine-grained measure of the dispersion and inconsistency within the records. This quality allows the new method to discern between data tables by inferring uniformity in two senses: consistent and invariant records with little dispersion in their structure. The parameter τ_0 ranges from $0 \leqslant \tau_0 \leqslant 1$, being 1 for those tables with consistent records; while τ_1 ranges from $0 \leqslant \tau_1 < \infty$, being 0 for those tables with invariant record structure and without dispersion.

5. Type detection

Data type detection is the core basis of the implemented methodology. Recognition of data types over fields from each record allows us to collect information about the contents of a given table. In this context, the records scoring, denoted as λ, is computed as

$$\lambda = \frac{(\sum_{i=1}^{k} S_i)^2}{100 \cdot k^2}$$

Where S_i is a score for the ith field φ in $\Phi\{\varphi_1, \varphi_2, ..., \varphi_n\}$ from the table Γ_δ. If the type of the ith field φ is known, $S_i = 100$, $S_i = 0.1$ otherwise.

For the purposes of this paper, the following field types are generally considered to be known:

- *Time and date*: matching regular dates and time format, as well stamped ones like MM/DD/YYYY[YYYY/MM/DD] HH:MM: SS +/- HH:MM.
- *Numeric*: matching all numeric data supported by the implementation language selected.
- *Percentage*.
- *Alphanumeric*: matching numbers, ASCII letters and underscore.
- *Currency*.
- *Especial data*: like “n/a” or empty strings.
- *Email*.
- *System paths*.
- *Structured scripts data types*: matching JSON arrays and data delimited by parentheses, curly and square brackets.
- *Numeric lists*: matching fields with numeric values delimited with common separator character.
- *URLs*.
- *IPv4*.

Al other fields will be scored as unknown type.

6. Table scoring

Once table uniformity $\tau\{\tau_0, \tau_1\}$ for records $\Phi\{\varphi_1, \varphi_2, ..., \varphi_n\}$ from the table $\Gamma_\delta\{\Phi_1, \Phi_2, ..., \Phi_n\}$, which has been generated by reading a CSV file Υ using a dialect ρ_δ, and the score λ are computed, the table score, denoted as ϖ, is computed as

[3]Segmented mode refers to the use of sample segments, which are defined as the data undergoes dispersion.

```
title,description,url,group,...
sample title,"###
# ||abc - abc||
||def -|| def
||ghi-|| ghi
||jkl-|| sdf
||def:|| jkl
||abc:|| mno
### def: pqr",https://example.com/,group 1,...
...
```

Fig. 3. Messy CSV file preview.

$$\varpi = (\frac{\tau_0}{\Delta} + \frac{1}{\tau_1 + n}) \cdot \sum_{i=1}^{n} \lambda_i); \forall n > 1$$

Where Δ is a threshold indicating the expected number of records to be imported from the CSV file Υ which contains a number of records m. For $m > n$, and an appropriate selection of ρ_δ, $\Psi^{-1}(\xi_\delta \leftarrow R(\Upsilon), \rho_\delta)$ will generate a table where $\Delta = n$; therefore, by the definition stated, the table score is in the range $0 < \varpi \leqslant 200$.

In the case $n = 1$ we have

$$\varpi = \lambda \cdot \frac{\eta + \frac{1}{k}}{k - \lfloor \eta \cdot k \rfloor + 1}$$

Where $\eta = \frac{\sqrt{\lambda}}{10}$ is a discriminant to ensure the exclusion of false positives with a single record.

7. Determining CSV file dialects

Algorithm 1 Dialect Determination

```
Input: CSV content $\xi$, expected number of records to import $\Delta$
Output: the dialect $\rho_\delta$ the that produces the more accurate table
1: function DETERMINE($\xi, \Delta$)
2:     $P \leftarrow$ StartDialects()
3:     for $\rho \in P$ do
4:         $\Gamma_\delta \leftarrow \Psi^{-1}(\xi, \rho)$                ▷ Parsing
5:         $\aleph(\varpi, \rho) \leftarrow$ TScore($\Gamma_\delta, \Delta$)
6:     return GetBestDialect($\aleph$)
```

This section shows the core algorithms on which the methodology presented in this research is based, complementary algorithms are listed in the appendix.

The main pseudocode for dialect determination is listed in Algorithm 1. At line 2 the set of predefined dialects are initialized; then, in line 4, a table `$_\delta$ is created by parsing the CSV content ξ with each ρ dialect.

At this point, it becomes clear that the selection of a robust parser is of utmost importance in order to obtain the best results even on messy files. In line 5, the output table `$_\delta$ is scored and this result is saved within the current dialect in the collection $\aleph$. At line 6, the dialect that gets the highest scored table is selected.

The table uniformity procedure is outlined in Algorithm 2 pseudocode. The method uses a set of sentinels to measure table inconsistency through monitoring table changes over parsed records.

The parameter τ_0 is derived from the standard deviation that indicates how uniformly the fields count are grouped around the average number of fields contained in the parsed records, resulting in an appropriate measure to qualify the structure of a table [2]. However, when there are two or more dialects with a small variance, the τ_0 parameter is not decisive. It is in this situation where the τ_1 parameter provides support by penalizing tables with variations in its records structures, and whose structure resembles sparse data that do not maintain consistency.

The Figure 3 shows a preview from the modified content of one file used during the testing phase. It was published in the CleverCSV repository on GitHub[4]. The star character has been replaced by the vertical bar

[4]https://github.com/alan-turing-institute/CleverCSV/issues/99

"|" to include in the detection a potential dialect with this character. As the author points out, the CSV file is comma delimited, using double quotes as the quote and escape character, then this file is compliant with RFC-4180 specifications. When running dialect detection, CleverCSV gets the vertical bar "|" as the delimiter because this field pattern gets a $P = 93.6395$ score vs a $P = 37.647059$ from patterns with the "," character as delimiter. This behavior is because the implemented logic heavily weights the delimiter count over the detected data types, where dialects containing the comma as delimiter obtain a type score of $T = 0.942647$ against the type score of $T = 0.843074$ obtained by dialects with the vertical bar as delimiter.

By executing the algorithms presented in this research, we get the following for dialects with the vertical bar as the delimiter $\lambda = 448.2243$, $\tau_0 = 0.2056$, $\tau_1 = 12$, and $\varpi = 29.5883$. For the comma we get $\lambda = 897.3315$, $\tau_0 = 1$, $\tau_1 = 0$, and $\varpi = 179.4663$. Then the comma "," character is selected as delimiter.

Algorithm 2 Table Uniformity

```
Input: CSV table $\Gamma_\delta$ with $n$ records containing $k_i$ fields
Output: the table uniformity factors $\tau_0, \tau_1$
 1: function TUniformity($\Gamma_\delta$)
 2:     $\varphi \leftarrow$ AverageFields($\Gamma_\delta$)
 3:     for $i \leftarrow 0$ to $n - 1$ do
 4:         $\mu \leftarrow \mu + (k_i - \varphi)^2$                ▷ Deviations
 5:         if i=0 then
 6:             $c \leftarrow c + 1$                              ▷ Sentinel 1
 7:         else
 8:             if $k_{i-1} \neq k_i$ then
 9:                 $\alpha \leftarrow \alpha + 1$                ▷ Sentinel 2
10:                 if $c > M$ then
11:                     $M \leftarrow c$
12:                 $c \leftarrow 0$
13:             else
14:                 $c \leftarrow c + 1$
15:                 if $i = n - 1$ then
16:                     if $c > M$ then
17:                         $M \leftarrow c$
18:     if $n > 1$ then
19:         $\sigma \leftarrow \sqrt{\frac{\mu}{n-1}}$
20:     else
21:         $\sigma \leftarrow \sqrt{\frac{\mu}{n}}$
22:     $\tau_0 \leftarrow \frac{1}{1+2\cdot\sigma}$
23:     $R \leftarrow k_{max} - k_{min}$                          ▷ Range
24:     if $\alpha > 0$ then
25:         $\beta \leftarrow \frac{M}{n}$
26:     $\tau_1 \leftarrow 2 \cdot R((\alpha)^2 + 1)(\frac{1-\beta}{M})$
27:     return $\tau_0, \tau_1$
```

8. Experiments

It was decided to code the new method and integrate it with CSV Interface[5], a VBA CSV file parser. Thus, the new CSV dialect determination method will be available in a widespread programming language without over-investing efforts. Additionally Python code has been written to run the tests for CleverCSV. The code repository is currently available on GitHub[6].

The new solution was tested on two datasets, both on GitHub: the one provided by Gerardo Vitagliano et al, and available in the Pollock framework repository; the other provided by G. van den Burg in the CleverCSV repository. For the first dataset, one or two polluted CSV file per pollution case are included for testing, all the

[5]https://github.com/ws-garcia/VBA-CSV-interface
[6]https://github.com/ws-garcia/CSVsniffer

Method	Success Rate %	Erroneous Rate %
Actual (10R)	99.32	0.68
Actual (25R)	99.32	0.68
Actual (50R)	***100.00***	***0.00***
CleverCSV	94.59	5.41

Table 1

Accuracy on dialect detection in simple Pollock testing dataset. An erroneous detection implies that the method has failed to infer either the delimiter or quote character, or both.

99 survey having at least one pollution case as described in the aforementioned study (excluding empty ones by the fact infinite dialects can be produce no payload files [16]). In addition, the dataset was enriched with data from the OpenRefine[7] testing, CleverCSV failure cases and other files used at development phase serves as testing samples. In total, the solution was tested against 148 CSV files (104 MB of data) for the simple Pollock testing.

The second dataset is composed of the 256 CSV files that CleverCSV could not accurately determine when conducting the research that led to the tool development [11]. At the time of this research, 244 of these files were available online. A filter was applied to exclude from the dataset all files with a structure that did not visually look like a CSV. After filtering, the dataset ended up with 179 CSV files (79 MB of data), which were used as a ground truth of our dialect detection method. Additionally, these files were subdivided to extract from them a set of CSVs that we can call "messy"; the structure of these being unconventional and whose dialect is much more difficult to infer. This last step is required since the dataset contains files that fall under the “normal forms” classification implemented in CleverCSV, which refers to CSV files with such a simple structure that they allow the determination of their dialects using only data inference[8].

To set up the tests, all files were manually annotated, using a separated set of annotation files, in order to verify the validity of detected dialects. In this context, we define the accuracy of dialect detection as the ratio of correctly detected dialects to the total number of test files with no error after execution.

8.1. Dialect detection accuracy

The Table 1 shows the results after running the dialect detection tests over the simple Pollock testing dataset. It can be seen that the new proposed heuristic gets a perfect score when using a table with a threshold of fifty records (50R) to be imported from the target CSV file.

When using tables of ten or twenty-five records (10R, 25R) for dialect determination, the proposed method was not able to determine dialect of the *"dd_Wickenburg_nobmp_623.csv"* file for the testing dataset. This file has been selected to show the variation of certainty as the considered table size increases across computations. As can be seen in the Figure 4, when the proposed heuristic is applied, it is settled that delimiter is the equal sign "=", since the dialects containing it divide each record into known data types: an alphanumeric field/column and a field with structured data delimited by square brackets. Increasing the table size to twenty-five (25R) induces the heuristic begins to highlight the semicolon ";" as a possible field delimiter character. Finally, the semicolon is correctly detected as a delimiter when the threshold of fifty records (50R) in the table is specified. This behavior demonstrates that the proposed methodology is strongly related to changes in the structure of tables used in dialect inference.

The results obtained after running the tests over dataset from CleverCSV are shown in Table 2. In this dataset the percentage of incorrectly detected dialects became approximately 10%. This metric indicates the presence of CSV files with unconventional structures. Notwithstanding the foregoing, dialect detection improves by 9.81% compared to CleverCSV.

CleverCSV running in verbose mode indicates that the tool failed to read 37 of the test files with errors related to the file encoding. These files, along with ones listed as "normal forms", were excluded from the dataset, producing a really messy subset of CSV files. Executing the tests over this selective filtered subset yields the results shown in Table 3. For this subset of files there is a slight increase in the rate of incorrect detections, preserving the 10% improvement of the new methodology over CleverCSV. On average, the heuristic proposed in this research shows an improvement of 7.51% compared to CleverCSV, outperforming the latter with 10% when handling messy CSV files.

[7]An open-source tool for working with messy data: https://openrefine.org/

[8]https://clevercsv.readthedocs.io/en/latest/source/clevercsv.html#module-clevercsv.normal_form

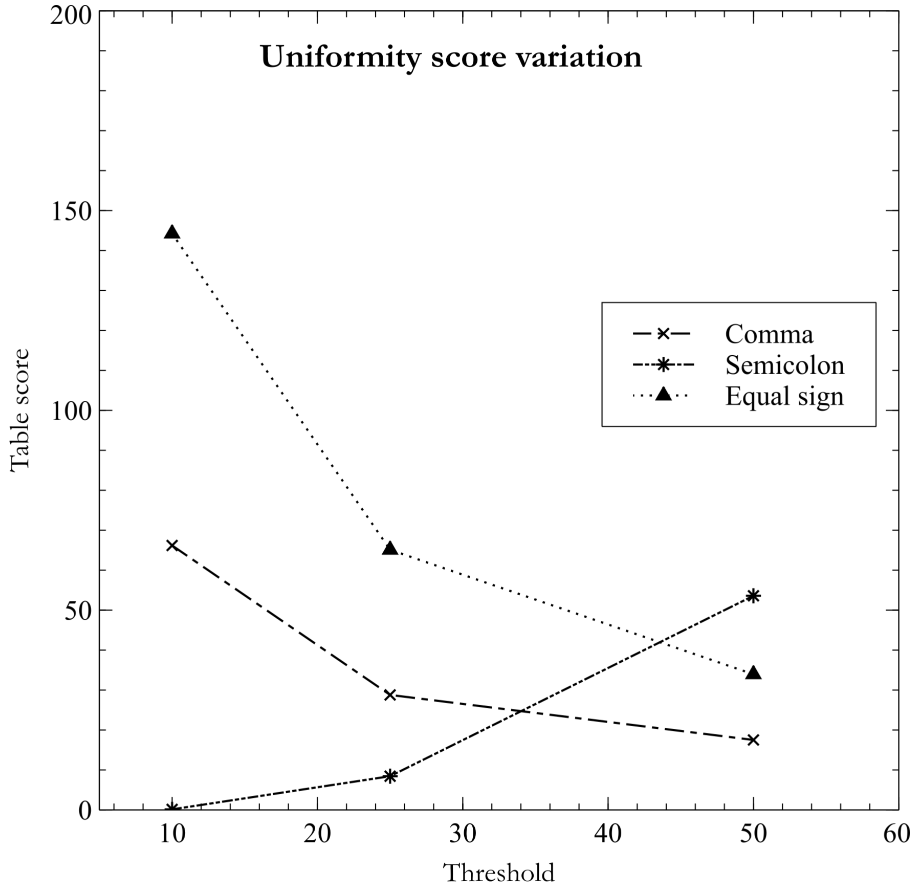

Fig. 4. scoring variation of three different delimiters and their dialects when applying the uniformity heuristic over tables from the dd_Wickenburg_nobmp_623.csv file.

Method	Success Rate %	Erroneous Rate %
Actual (10R)	88.83	11.17
Actual (25R)	***89.39***	***10.61***
Actual (50R)	88.83	11.17
CleverCSV	79.58	20.42

Table 2

Accuracy on dialect detection in the failed CleverCSV dataset. An erroneous detection implies that the method has failed to infer either the delimiter or quote character, or both.

Method	Success Rate %	Erroneous Rate %
Actual (10R)	86.51	13.49
Actual (25R)	***87.30***	***12.70***
Actual (50R)	***87.30***	***12.70***
CleverCSV	76.98	23.02

Table 3

Accuracy on dialect detection over really messy CSV files. An erroneous detection implies that the method has failed to infer either the delimiter or quote character, or both

9. Discussion

By looking closely at the results obtained, it can be deduced that there are two main categories that influence the certainty of determined dialects: the type of heuristics used, the CSV file parser behavior while producing tables using a certain dialect. In this section both categories are discussed in order to briefly qualify the experiments results.

9.1. Heuristic

In contrast to CleverCSV, in whose heuristic the detection of data types serves as a factor to scale down the score obtained by a certain pattern; the table consistency method uses data detection as a base score to be

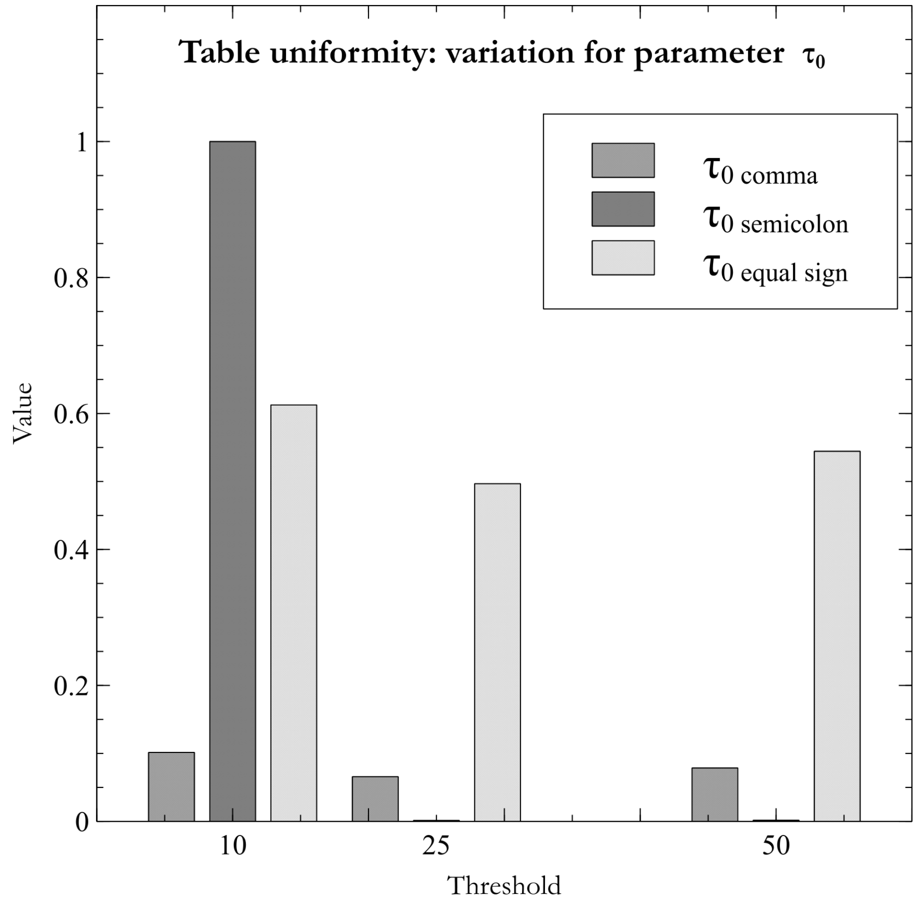

Fig. 5. uncertainty caused by analyzing tables with a single field across all their records.

narrowed using the table consistency and data dispersion parameters. The results therefore indicate that the factors obtained are not commutative.

Since data type detection is a fundamental part of both methods, it is necessary to include a wide range of known data typologies. This factor is undoubtedly determining in dialect detection. According to Mitlohner's research [10] [4], with a base of 104,826 CSV files, the vast majority of data commonly stored in this type of files are numeric, tokens (words separated by spaces), entities, URLs, dates, alphanumeric fields and general text, so these data types must be recognized. Additionally, in the field of programming, there are other types of data frequently dumped in CSV files, namely: structured data with the Regex pattern $(([a - zA - Z] + [\backslash([a - zA - Z] + [\backslash[\{][\wedge\backslash]] * [\backslash]\}])[\{][\wedge\backslash]] * [\backslash]\}])$, numerical lists, tuples, arrays among others.

It is worth mentioning that dialect detection is prone to failure when the CSV file is composed of unknown data types. In these cases, the table uniformity tends to select dialects that produce registers with a single field. When reviewing the cases where CleverCSV was not able to determine the dialect, it has been observed that the common denominator has been the high count of a potential delimiter with more occurrences than the expected delimiter. In this sense, both solutions have poor performance when the space character appears in the list of potential delimiters.

There are files where the threshold of records in the target table is decisive; however, tests have found that the dialect of some files is determined incorrectly as the value of this parameter is increased and more records are loaded into the table. This peculiarity allows us to conclude that the first records can adequately describe the structure of CSV files, avoiding, to a certain extent, the need to read the whole file. In this particular, it was found that CleverCSV had a running time of approximately 19 minutes before completing the tests it was subjected to. The results obtained lead to conclude that the default option when detecting dialects in CSV files should be to read only a sample of the file instead of reading its entire contents.

As pointed out earlier, the table uniformity method prefers grouped data over those that appear to be sparse data. In these cases, detection tends to depend exclusively on the data types detected in the records. This fact is evidenced by plotting the values of the uniformity parameter τ_0.

Looking at Figure 5, it can be seen that, even though the score obtained by the semicolon dialect is very close to zero, the value of τ_0 is maximum. In contrast, this value fluctuates to nearly zero for the dialect containing semicolon; it remains almost unchanged among the dialects using other fields delimiters characters. In these cases, the dialect determination is relegated to data type detection and fine-grained monitoring of changes in table structures through the τ_1 parameter. It is noted that the parameters τ_0 and τ_1 work together for well-defined tables, selectively overriding each other when processing tables with poorly defined data structures.

9.2. CSV parser basis

The accuracy of dialect determination is intimately related to the way CSV parsers behave when confronted with atypical situations. This is because heuristics use these results to infer the configuration that returns the most suitable data structures.

One of the capabilities required for dialect determination is the recovery of data after the occurrence of a critical error. This is the case when import CSV files where there is no balanced quotation count. This situation breaks the RFC-4180 specifications and causes an import error in almost all solutions intended to work with CSV files. In this sense, the recovery of this error should include a specific message after which the loading of information should continue until the whole file is processed.

Since the determination of dialects can be done with a few records received from a CSV file, there is a probability that some of the parameters that compose the dialect cannot be determined properly. Given this reality, it is preferable that CSV parsers be able to convert between one escaping mechanism and another instead of making the escape character mutually exclusive as established in the most relevant proposals on these topics [12]. This results in the correct interpretation of escape sequences that use the "\" for those files in which a quote character has been detected as part of their dialect.

Appendix A. Algorithms pseudocode

Algorithm 3 Table Score

```
Input: CSV table $\Gamma_\delta$ with $n$ records, threshold $\Delta$
Output: the score $\varpi$ for given table
1: function TSCORE($\Gamma_\delta, \Delta$)
2:     $\lambda \leftarrow$ SUMSCORE($\Gamma_\delta$)
3:     if $n > 1$ then
4:         $(\tau_0, \tau_1) \leftarrow$ TUNIFORMITY($\Gamma_\delta$)
5:         return $\lambda \cdot (\frac{\tau_0}{\Delta} + \frac{1}{(\tau_1 + n)})$
6:     else
7:         $\eta \leftarrow \frac{\sqrt{\lambda}}{10}$
8:         return $\lambda \cdot \frac{\eta + \frac{1}{k}}{k - \lfloor \eta \cdot k \rfloor + 1}$
```

Algorithm 4 Sum of Records Score

```
Input: CSV table $\Gamma_\delta$ with $n$ records containing $k_i$ fields
Output: the sum of records score for the given table
1: function SUMSCORE($\Gamma_\delta$)
2:     for $i \leftarrow 0$ to $n - 1$ do
3:         for $j \leftarrow 0$ to $k_i - 1$ do
4:             if KNOWNDATATYPE($\Gamma_\delta[i, j]$) then
5:                 $\Lambda \leftarrow \Lambda + 100$
6:             else
7:                 $\Lambda \leftarrow \Lambda + 0.1$
8:         $\chi \leftarrow \chi + (\frac{\Lambda^2}{100 \cdot k_1^2})$
9:     return $\chi$
```

References

[1] Ioannis Alagiannis et al. “NoDB: efficient query execution on raw data files”. In: Communications of the ACM 58.12 (Nov. 23, 2015), pp. 112–121. issn: 0001-0782, 1557-7317. doi: 10.1145/2830508. url: https://dl.acm.org/doi/10.1145/2830508 (visited on 07/24/2021).

[2] Mohammad Fraiwan Al-Saleh and Adil Eltayeb Yousif. “Properties of the Standard Deviation that are Rarely Mentioned in Classrooms”. In: Austrian Journal of Statistics 38.3 (Apr. 3, 2016). issn: 1026-597X. doi: 10.17713/ajs.v38i3.272. url: https://www.ajs.or.at/index.php/ajs/article/view/vol38

[3] Christina Christodoulakis et al. “Pytheas: pattern-based table discovery in CSV files”. In: Proceedings of the VLDB Endowment 13.12 (Aug. 2020), pp. 2075–2089. issn: 2150-8097. doi: 10.14778/3407790.3407810. url: https://dl.acm.org/doi/10.14778/3407790.3407810 (visited on 07/23/2021).

[4] Till D¨ohmen, Hannes M¨uhleisen, and Peter Boncz. “Multi-Hypothesis CSV Parsing”. In: Proceedings of the 29th International Conference on Scientific and Statistical Database Management. SSDBM ’17: 29th International Conference on Scientific and Statistical Database Management. Chicago IL USA: ACM, June 27, 2017, pp. 1–12. isbn: 978-1-4503-5282-6. doi: 10.1145/3085504.3085520. url: https://dl.acm.org/doi/10.1145/3085504.3085520 (visited on 07/23/2021).

[5] Dutch Stichting DuckDB Foundation. DUCKDB. Version 0.9.2. Amsterdam NL, 2023. url: https://duckdb.org/docs/archive/0.9.2/ (visited on 02/04/2024).

[6] Leonardo H¨ubscher, Lan Jiang, and Felix Naumann. “ExtracTable: Extracting Tables from Raw Data Files”. In: (2023). ISBN: 9783885797258 Publisher: Gesellschaft f¨ur Informatik e.V. doi: 10.18420/BTW2023-20. url: https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/publications/PDFs/2023_huebscher_extractable.pdf (visited on 02/10/2024).

[7] S. Idreos et al. “Here are my data files. Here are my queries. Where are my results?” In: Proceedings of 5th Biennial Conference on Innovative Data Systems Research. Biennial Conference on Innovative Data Systems Research (CIDR 2011). Asilomar, California, USA, Jan. 9, 2011, pp. 57–68. url: https://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper7.pdf (visited on 07/24/2021).

[8] Manos Karpathiotakis et al. “Adaptive query processing on RAW data”. In: Proceedings of the VLDB Endowment 7.12 (Aug. 2014), pp. 1119–1130. issn: 2150-8097. doi: 10.14778/2732977.2732986. url: https://dl.acm.org/doi/10.14778/2732977.2732986 (visited on 07/24/2021).

[9] Library of Congress. CSV, Comma Separated Values (RFC 4180). LOC. Feb. 11, 2020. url: https://www.loc.gov/preservation/digital/formats/fdd/fdd000323.shtml.

[10] Johann Mitlohner et al. “Characteristics of Open Data CSV Files”. In: 2016 2nd International Conference on Open and Big Data (OBD). 2016 2nd International Conference on Open and Big Data (OBD). Vienna: IEEE, Aug. 2016, pp. 72–79. isbn: 978-1-5090-4054-4. doi: 10.1109/OBD.2016.18. url: http://ieeexplore.ieee.org/document/7573692/ (visited on 07/23/2021).

[11] Tomas Petricek et al. “AI Assistants: A Framework for Semi-Automated Data Wrangling”. In: IEEE Transactions on Knowledge and Data Engineering 35.9 (Sept. 1, 2023), pp. 9295–9306. issn: 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2022.3222538. url: https://www.turing.ac.uk/sites/default/files/2022-11/aida_ai_assistants_tkde_2022_0.pdf (visited on 02/11/2024).

[12] Rufus Pollock. Data Package (v1). CSV Dialect. Feb. 20, 2013. url: https://specs.frictionlessdata.io/csv-dialect/ (visited on 05/10/2023).

[13] Yakov Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. IETF. 2005. url: https://datatracker.ietf.org/doc/rfc4180/ (visited on 07/23/2021).

[14] Charles Sutton et al. “Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling”. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD ’18: The 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. London United Kingdom: ACM, July 19, 2018, pp. 2279–2288. isbn: 978-1-4503-5552-0. doi: 10.1145/3219819.3220057. url: https://dl.acm.org/doi/10.1145/3219819.3220057 (visited on 07/24/2021).

[15] G. J. J. van den Burg, A. Naz´abal, and C. Sutton. “Wrangling messy CSV files by detecting row and type patterns”. In: Data Mining and Knowledge Discovery 33.6 (Nov. 2019), pp. 1799–1820. issn: 1384-5810, 1573-756X. doi: 10.1007/s10618-019-00646-y. url: http://link.springer.com/10.1007/s10618-019-00646-y (visited on 07/23/2021).

[16] Gerardo Vitagliano et al. “Pollock: A Data Loading Benchmark”. In: Proceedings of the VLDB Endowment 16.8 (Apr. 2023), pp. 1870–1882. issn: 2150-8097. doi: 10.14778/3594512.3594518. url: https://www.vldb.org/pvldb/vol16/p1870-vitagliano.pdf (visited on 02/11/2024).