Article

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

Wilfredo García [*]

Posted Date: 24 February 2024

doi: 10.20944/preprints202402.0858.v5

Keywords: comma separated values; CSV dialect detection; data mining

Article

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

W. García

ECP Solutions; wilfredo_garcia@outlook.es; ORCID: 0000-0002-9620-1119

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 during data loading by 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

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) [1]. 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" [2]. Furter more, 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 [3,4]. The automation problem is to determine the delimiters (also called dialect sniffing) of a given file. Dialect sniffing requires that the field delimiter, record delimiter and escape character be determined [5].

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 [6]. 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. [4].

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

Figure 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

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

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 [3]. 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 [7,8]. These trends suggest that the traditional practice of considering CSV files outside of database systems is tending to change [9].

2. Related work

The detection of dialects in CSV files is a little studied 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 [10]. A similar treatment, based on the discovery of the table structures once the information is loaded into the RAM, is

[1] An analysis of a 413 GB data body found CSV files available for download on 232 portals [4].

addressed by C. Christodoulakis et. al. [11]. In the latter, the methodology is based on the classification of the records present in the CSV files, to which a specific heuristic is 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 his methodology significantly improved the accuracy with which dialects were determined compared to tools such as Python's *csv* module, or the intrinsic functions of the *Pandas* package, also in the Python programming language. The methodology implemented in CleverCSV is based on the detection of patterns in the structure of the CSV records, in addition to the inference of the data types of the fields that compose each record. In this way, the utility applies the necessary heuristics to infer the potential dialect for a given CSV file through mathematical and logical operations devised to discern between possible dialects [5].

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 obtain the dialect that produces the best table [12].

3. Problem formulation

In order to properly formulate the dialect detection problem, it is necessary to establish some basic definitions.

Definition 3.1 *(CSV content). Given a CSV file* $\boldsymbol{\Upsilon}$ *its content is defined as* $\boldsymbol{\xi}\{\xi_1, \xi_2, \dots, \xi_n\}$*, where* $\boldsymbol{\xi_i} \in \boldsymbol{\Omega}$*, and* $\boldsymbol{\Omega}$ *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 may contain characters encoded in more than one encoder. For the purpose of this document, it is assumed that all characters share the same encoder.

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

Definition 3.2 *(CSV table). The table* $\boldsymbol{\Gamma_\delta}$ *is defined as a set of records composed of a given set of fields, which share the data typology 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* $\boldsymbol{\Phi}\{\varphi_1, \varphi_2, \dots, \varphi_k\}$*; i.e., a set of fields* φ_i *;* $i \in [1,2, \dots, k]$*. Then, the table can be expressed as* $\boldsymbol{\Gamma_\delta}\{\Phi_1, \Phi_2, \dots, \Phi_n\}$*; i.e., a set of records* Φ_i *;* $i \in [1,2, \dots, n]$**.**

The function $\mathrm{R}(\boldsymbol{\Upsilon})$ is in charge of reading the content of the file $\boldsymbol{\Upsilon}$, while the function $\boldsymbol{\Psi^{-1}}(\boldsymbol{\xi_\delta}, \boldsymbol{\rho_\delta})$ is in charge of parsing and transforming the CSV content $\boldsymbol{\xi_\delta}$ into a table $\boldsymbol{\Gamma_\delta}$. The process of parsing and transformation is clearly out of the scope of this study, 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 3.3 (CSV dialect). Let $\boldsymbol{\Gamma}$ be the data table from which the content $\boldsymbol{\xi}$ of file $\boldsymbol{\Upsilon}$ is generated, the dialect $\boldsymbol{\rho}$ is defined as the formatting rule to be applied to produce the output data stream.

So that, by the dialect definition, it is verified that

$$\boldsymbol{\Upsilon} \leftarrow W(\boldsymbol{\xi} \leftarrow \boldsymbol{\Psi}(\boldsymbol{\Gamma}, \boldsymbol{\rho})); \boldsymbol{\rho}\{\upsilon_d, \upsilon_\mathrm{q}, \upsilon_\mathrm{e}, \upsilon_\mathrm{r}\} \in \boldsymbol{\Omega}$$

Definition 3.4 *(CSV dialect determination). Given a CSV file* $\boldsymbol{\Upsilon}$*, determining the dialect involves identifying the dialect* $\boldsymbol{\rho_\delta}$ *that satisfies the following statement* $\boldsymbol{\Gamma} \cong \boldsymbol{\Gamma_\delta} \leftarrow \boldsymbol{\Psi^{-1}}(\boldsymbol{\xi_\delta} \leftarrow R(\boldsymbol{\Upsilon}), \boldsymbol{\rho_\delta})$.

Thus, it can be concluded that for a CSV file $\boldsymbol{\Upsilon}$, created using a dialect $\boldsymbol{\rho}$, there exists a dialect $\boldsymbol{\rho_\delta}$ that verifies the condition $\boldsymbol{\Gamma} \cong \boldsymbol{\Gamma_\delta}$. Therefore, it is verifiable that the content of a CSV file is a function of its dialect.

1.1.1. Potential dialect boundaries

It should be noted that multiples potential dialects can produce similar tables outputs that are equal or approximately equal to the source table $\boldsymbol{\Gamma}$. Furthermore, $\boldsymbol{\rho_\delta}$ shares the same character set as the contents $\boldsymbol{\xi}$ for the CSV file $\boldsymbol{\Upsilon}$. That is, $\boldsymbol{\rho_\delta}$ can be practically any character within $\boldsymbol{\Omega}$ domain.

Thus, it is necessary to narrow down 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

$$\boldsymbol{\rho_\delta}\{\boldsymbol{\upsilon_d}[\text{“,” “;” TAB “|” “:” SPACE}], \boldsymbol{\upsilon}_{\mathbf{q}}[\text{“\textbackslash”“ “'” “\textasciitilde”}], \boldsymbol{\upsilon}_{\mathbf{e}}[\boldsymbol{\upsilon}_{\mathbf{q}}\ \text{“\textbackslash”}], \boldsymbol{\upsilon}_{\mathbf{r}}[\text{CRLF CR LF}]^{2}\}$$

4. Table uniformity

The table uniformity approach is proposed to solve the problem of dialect determination. The method is based on the measurement of the consistency of the table, which has been obtained through a dialect $\boldsymbol{\rho_\delta}$, and the dispersion of records along with the inference of the raw data types from the records.

Definition 4.1 *(Table consistency). Let $\boldsymbol{\Gamma_\delta}$ be a table $\boldsymbol{\Gamma_\delta}$, generated when reading a CSV file $\boldsymbol{\Upsilon}$ using a dialect $\boldsymbol{\rho_\delta}$, the table consistency, denoted by τ_0, is a ratio that describes how uniform a table is across its $\boldsymbol{k}$ fields and its $\boldsymbol{n}$ records.*

Definition 4.2 *(Records dispersion). Let $\boldsymbol{\Phi}$ be the sets of records from table $\boldsymbol{\Gamma_\delta}$, generated when reading a CSV file $\boldsymbol{\Upsilon}$ using a dialect $\boldsymbol{\rho_\delta}$, the records dispersion, denoted by τ_1, is a measure describing the magnitude of the change in the records composition throughout the table.*

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

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

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

where, for a given table $\boldsymbol{\Gamma_\delta}$, σ is the standard deviation of the number of fields across records; α represents the count of times the number of fields changes between records; R is the statistical range of 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 $\beta = M/\boldsymbol{n}$ 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 of the records in the 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 records and invariant records with little dispersion in their structure. The parameter τ_0 ranges from $0 \leq \tau_0 \leq 1$, being 1 for those tables with consistent records; while $\tau_1$ranges from $0 \leq \tau_1 < \infty$, being 0 for those tables with invariant record structure and without dispersion.

5. Type detection

Data type detection is the core basis of the methodology implemented. Recognition of the types of data fields from each record allows us to collect information about the contents of a given table. In this context, the records scoring, $\boldsymbol{\lambda}$, is defined as

$$\boldsymbol{\lambda} = \frac{\left(\sum_{i=1}^{k} S_i\right)^2}{100 * \boldsymbol{k}^2}$$

where S_i is the score for ith field φ in $\boldsymbol{\Phi}\{\varphi_1, \varphi_2, \dots, \varphi_k\}$ from the table $\boldsymbol{\Gamma_\delta}$. If the type of the ith field φ is known $S_i = 100$, $S_i = 0.1$ otherwise.

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

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

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 the table uniformity $\boldsymbol{\tau}\{\tau_0, \tau_1\}$ for the records $\boldsymbol{\Phi}\{\varphi_1, \varphi_2, \dots, \varphi_k\}$ contained in table $\boldsymbol{\Gamma}_\delta\{\Phi_1, \Phi_2, \dots, \Phi_n\}$, which has been generated by reading a CSV file $\boldsymbol{\Upsilon}$ using a dialect $\boldsymbol{\rho}_\delta$, and the score $\boldsymbol{\lambda}$ are computed, the table score $\boldsymbol{\varpi}$ is defined as

$$\boldsymbol{\varpi} = \left(\frac{\tau_0}{\Delta} + \frac{1}{\tau_1 + \boldsymbol{n}}\right) * \sum_{i=1}^{n} \boldsymbol{\lambda}_i \,;\ \forall\, \boldsymbol{n} > 1$$

where Δ is a threshold used for indicate the expected number of records to be imported from the CSV file $\boldsymbol{\Upsilon}$ which contains a number of records $\boldsymbol{m}$. For $\boldsymbol{m} > \boldsymbol{n}$, and an appropriate selection of $\boldsymbol{\rho}_\delta$, $\boldsymbol{\Psi}^{-1}(\boldsymbol{\xi}_\delta \leftarrow \mathrm{R}(\boldsymbol{\Upsilon}), \boldsymbol{\rho}_\delta)$ will generate a table in with $\Delta = \boldsymbol{n}$; therefore, by the definition stated, the table score is in the range $0 < \boldsymbol{\varpi} \leq 200$.

In the case $\boldsymbol{n} = 1$ we have

$$\boldsymbol{\varpi} = \boldsymbol{\lambda} * \frac{\eta + \left(\frac{1}{\boldsymbol{k}}\right)}{\boldsymbol{k} - floor(\eta * \boldsymbol{k}) + 1}$$

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 ξ, expected number of records to import Δ

Output: the dialect ρ_δ the that produces the more accurate table

```
1.  function determine ($\xi$, $\Delta$):
2.      P ← 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 ($\aleph$)
7.  end function
```

In order to develop a reliable methodology for determining the dialects of CSV files, the cases that produced failures in famous dialect detectors were used as a basis. This is because the dialect in well-structured files is easily determined with any simple heuristic.

The main pseudocode for dialect determination is described in *Algorithm 1*. At line 2 the set of predefined dialects are initialized; then, in line 4, a table Γ_δ is created by parsing the CSV content ξ with each 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 Γ_δ is scored and this result is saved within the current dialect in the collection $\aleph$. At line 6, the dialect that produce the highest scored table is selected.

The table uniformity pseudocode is outlined in *Algorithm 4*. The method uses a set of sentinels to measure table inconsistency through monitoring table changes through the 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 [13]. 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 following illustration shows the preview from the modified content of one of the files 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 "|" to include in the detection a

```
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,…
…
```

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 the file is compliant with RFC-4180 specifications. When running dialect detection, CleverCSV gets the vertical bar "|" as the delimiter as this field pattern gets a $P = 93.6395$ score vs a $P = 37.647059$ from the patterns with the "," character as delimiter. This behavior is due to the fact that the logic used strongly weights the count of delimiters over the detected data types. This behavior is due to the fact that the logic used strongly weights the delimiter count over the types of data detected, 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, the following is obtained 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 the results are $\lambda = 897.3315$, $\tau_0 = 1$, $\tau_1 = 0$, and $\varpi = 179.4663$. Then the comma "," character is selected as delimiter.

Algorithm 4: Table Uniformity

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

```
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 $i = 0$ 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.         $\sigma \leftarrow \sqrt{\mu/(n-1)}$
20.     else
21.         $\sigma \leftarrow \sqrt{\mu/n}$
22.     $\tau_0 \leftarrow 1/(1 + 2\sigma)$
23.     $R \leftarrow k_{max} - k_{min}$                    ⊳Range
24.     if $\alpha > 0$ then
25.         $\beta \leftarrow M/n$
26.     $\tau_1 \leftarrow 2 \cdot R\,(\alpha^2 + 1)\,(1 - \beta)/M$
27.     return $\tau_0, \tau_1$
28. end function
```

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 overinvesting efforts.

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 used for testing, all the 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 [14]). In addition, the dataset was enriched with data from the OpenRefine[6] testing, CleverCSV

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

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

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 in which CleverCSV was not able to accurately determine the dialect at the time of the research that originate the tool [15]. At the time of this research, 244 of these files were available online. A filter was applied to the files to exclude from the dataset all 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[7].

To set up the tests, all files were manually annotated in a file in order to verify, by comparison, the validity of the 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.

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 the quote character, or both.

Method	Success %	Erroneous %
Actual (10R)	99.32	0.68
Actual (25R)	99.32	0.68
Actual (50R)	***100***	***0.00***
CleverCSV	94.59	5.41

When using tables of ten or twenty-five records (10R, 25R) for dialect determination, the proposed method was not able to determine the 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 size of the table considered in the computations increases. As can be seen in the Figure 3, when the proposed heuristic is applied, it is settled that the 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. By increasing the table size to twenty-five (25R) 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 the changes in the structure of the tables used in dialect inference.

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

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2024 doi:10.20944/preprints202402.0858.v5

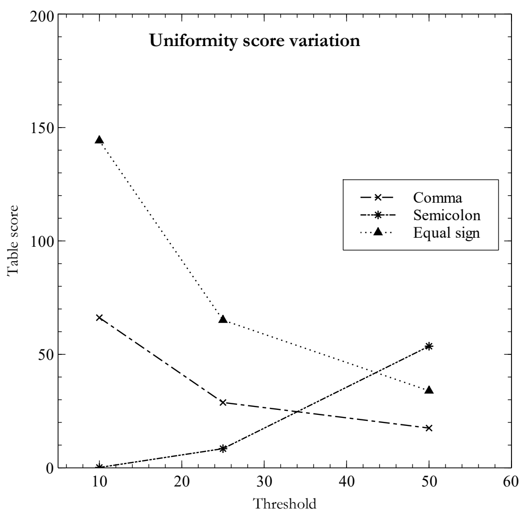

Figure 3. scoring variation of three different delimiters and their dialects when applying the uniformity heuristic over tables from the dd_Wickenburg_nobmp_623.csv file.

The results obtained after running the tests over the 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, the dialect detection improves by 9.81% compared to CleverCSV.

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 the quote character, or both.

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

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 the files listed as “normal forms”, were excluded from the dataset, producing the really messy subset of CSV files. Running 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.

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 the quote character, or both.

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

9. Discussion

The results obtained by the table uniformity method proposed in this research have their genesis in two aspects: the type of heuristics used, the behavior of the CSV file analyzer while producing tables using a certain dialect.

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 narrowed using the table consistency and data dispersion parameters.

Since the detection of data types is the foundation of the method, a wide range of typologies is required to be recognized. According to Mitlohner's research [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 $([a-zA-Z]+[\backslash([a-zA-Z]+[\backslash[\{][\hat{}\backslash]]*[\backslash]\}])[\{][\hat{}\backslash]]*[\backslash]\}])$, 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 in which the record threshold is decisive; however, in the tests carried out it was detected that the dialect of some files is determined incorrectly as the threshold is increased.

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 4, 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 almost zero for the dialect containing the semicolon; it remains almost unchanged among the dialects containing the other 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. The above serves as a basis for reaffirming that the table uniformity method helps to properly adjust the metrics obtained by inferring the data types.

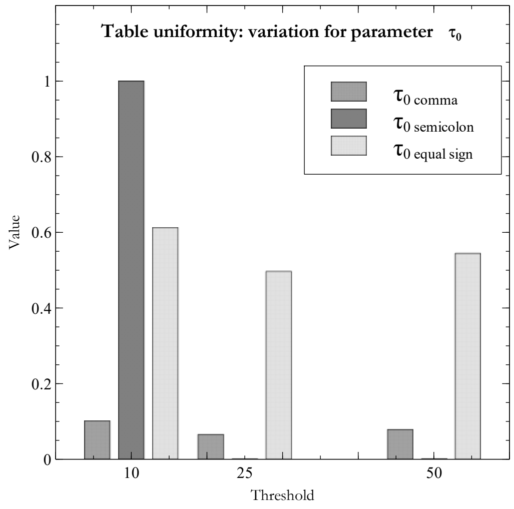

Figure 4. uncertainty caused by analyzing tables with a single field across all their records.

9.1. 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 for the import of CSV files where there is no balance between quotation marks. This situation breaks the RFC-4180 specifications and causes an import error in 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 [16]. This results in the correct interpretation of escape sequences that use the backslash "\" in those files in which a quote character has been detected as part of their dialect.

10. Appendix: algorithms pseudocode

Algorithm 2: Table Score

```
Input: CSV table $\Gamma_\delta$ with $n$ records, expected number of records to import $\Delta$
Output: the score $\varpi$ for the 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 $\lambda \cdot (\tau_0/\Delta + 1/(\tau_1 + n))$
6.      else
7.          $\eta \leftarrow \sqrt{\lambda}/10$
8.          return $\lambda \cdot \frac{\eta + (1/k)}{k - floor(\eta * k) + 1}$
9.  end function
```

Algorithm 3: 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 $i = 0$ to $n - 1$ do
3.          for $j = 0$ to $k_i - 1$ do
4.              if KnownDataType ($\Gamma_\delta[i, j]$) then
5.                  $\sigma \leftarrow \sigma + 100$
6.              else
7.                  $\sigma \leftarrow \sigma + 0.1$
8.          $\ell \leftarrow \ell + (\sigma^2 / (100 \cdot k_i^2))$
9.      return $\ell$
10. end function
```

11. References

1. Y. Shafranovich, “Common Format and MIME Type for Comma-Separated Values (CSV) Files,” IETF. Accessed: Jul. 23, 2021. [Online]. Available: https://datatracker.ietf.org/doc/rfc4180/
2. Library of Congress, “CSV, Comma Separated Values (RFC 4180),” LOC. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000323.shtml
3. C. Sutton, T. Hobson, J. Geddes, and R. Caruana, “Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling,” presented at the Knowledge Discovery and Data Mining Conference, London, United Kingdom, Aug. 2018. Accessed: Jul. 23, 2021. [Online]. Available: https://www.turing.ac.uk/research/publications/data-diff-interpretable-executable-summaries-changes-distributions-data
4. J. Mitlohner, S. Neumaier, J. Umbrich, and A. Polleres, “Characteristics of Open Data CSV Files,” in *2016 2nd International Conference on Open and Big Data (OBD)*, Vienna: IEEE, Aug. 2016, pp. 72–79. doi: 10.1109/OBD.2016.18.
5. G. J. J. van den Burg, A. Nazábal, and C. Sutton, “Wrangling messy CSV files by detecting row and type patterns,” *Data Min. Knowl. Discov.*, vol. 33, no. 6, pp. 1799–1820, Nov. 2019, doi: 10.1007/s10618-019-00646-y.
6. T. Döhmen, H. Mühleisen, and P. Boncz, “Multi-Hypothesis CSV Parsing,” in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, Chicago IL USA: ACM, Jun. 2017, pp. 1–12. doi: 10.1145/3085504.3085520.
7. I. Alagiannis, R. Borovica-Gajic, M. Branco, S. Idreos, and A. Ailamaki, “NoDB: efficient query execution on raw data files,” *Commun. ACM*, vol. 58, no. 12, pp. 112–121, Nov. 2015, doi: 10.1145/2830508.
8. M. Karpathiotakis, M. Branco, I. Alagiannis, and A. Ailamaki, “Adaptive query processing on RAW data,” *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1119–1130, Aug. 2014, doi: 10.14778/2732977.2732986.
9. S. Idreos, I. Alagiannis, R. Johnson, and A. Ailamaki, “Here are my data files. Here are my queries. Where are my results?,” in *Proceedings of 5th Biennial Conference on Innovative Data Systems Research*, Asilomar, California, USA, Jan. 2011, pp. 57–68. Accessed: Jul. 24, 2021. [Online]. Available: https://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper7.pdf
10. Dutch Stichting DuckDB Foundation, “DUCKDB.” Dutch Stichting DuckDB Foundation, Amsterdam NL, 13 2023. Accessed: Feb. 04, 2024. [Online]. Available: https://duckdb.org/docs/archive/0.9.2/
11. C. Christodoulakis, E. B. Munson, M. Gabel, A. D. Brown, and R. J. Miller, “Pytheas: pattern-based table discovery in CSV files,” *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2075–2089, Aug. 2020, doi: 10.14778/3407790.3407810.
12. L. Hübscher, L. Jiang, and F. Naumann, “ExtracTable: Extracting Tables from Raw Data Files,” 2023, doi: 10.18420/BTW2023-20.
13. M. F. Al-Saleh and A. E. Yousif, “Properties of the Standard Deviation that are Rarely Mentioned in Classrooms,” *Austrian J. Stat.*, vol. 38, no. 3, Apr. 2016, doi: 10.17713/ajs.v38i3.272.
14. G. Vitagliano, M. Hameed, L. Jiang, L. Reisener, E. Wu, and F. Naumann, “Pollock: A Data Loading Benchmark,” *Proc. VLDB Endow.*, vol. 16, no. 8, pp. 1870–1882, Apr. 2023, doi: 10.14778/3594512.3594518.
15. T. Petricek, G. J. J. V. D. Burg, A. Nazábal, T. Ceritli, E. Jiménez-Ruiz, and C. K. I. Williams, “AI Assistants: A Framework for Semi-Automated Data Wrangling,” *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 9295–9306, Sep. 2023, doi: 10.1109/TKDE.2022.3222538.
16. Rufus Pollock, “Data Package (v1),” CSV Dialect. Accessed: May 10, 2023. [Online]. Available: https://specs.frictionlessdata.io/csv-dialect/