# Chapter 3
# Data Pre-Processing

## 3.1 Data Quality Issues Relevant to Data Matching

Most real-world databases contain noisy, inconsistent and missing data due to a variety of factors [19, 135, 218]. It is generally accepted that low data quality costs businesses and governments billions of lost revenue every year. It has been estimated that data quality problems can result in up to 12 % lost revenue for businesses [177]. For any type of data analysis, processing and management, the *garbage-in garbage-out* principle holds. If the quality of the input data is low, then the output generated is normally not of high quality or accuracy either.

A large body of work has covered the various issues involved with data quality in depth [19, 177, 218]. There are several dimensions to data quality. The ones relevant to data matching are:

- *Accuracy*. How accurate are the attribute values in the database(s) used for matching or deduplication? Is it known how the data have been entered or recorded? Have data entry checks been performed, and have the data been verified for correctness using external reference data (such as references of known and valid addresses)?
- *Completeness*. How complete are the data? How many attribute values are missing in the databases used? Is it known why certain attribute values are missing? Are attributes missing that would be of use for data matching?
- *Consistency*. How consistent are the values within a single database used for matching or deduplication, and how consistent are values across two or more databases used for matching? The format and coding of individual attributes even within a single database can change over time. Is it known if the databases contain duplicate records for the same entity (for example because a person moved to a different address and therefore was recorded as a new separate customer)?
- *Timeliness*. How old are the data available? For the matching of two databases, have the data been recorded at the same time or not? This can be a crucial factor to a successful matching because personal information, such as people's addresses, telephone numbers, and even names, change over time. If the data to be matched

P. Christen, *Data Matching*, Data-Centric Systems and Applications,
DOI: 10.1007/978-3-642-31164-2_3, © Springer-Verlag Berlin Heidelberg 2012

have been recorded at different points in time then this needs to be taken into account during the data matching process.
- *Accessibility*. Are all the data required available in the database to be deduplicated or the databases to matched? Is there enough information in the form of attributes that cover different aspects of the entities in the databases to allow detailed comparisons and accurate classification? If for example only names but no address information is available then accurate matching of two large databases will be impossible because many records might contain the names 'John Smith' or 'Mary Miller'.
- *Believability*. Can the values stored in the databases be regarded as credible or true? Or is it possible that values are wrong or impossible?

Arguably the most important data quality dimensions for data matching and deduplication are *accuracy* and *consistency*, because a large portion of efforts in the indexing, comparisons and classification steps (that will be covered in Chaps. 4–6) deal with inaccurate and inconsistent data. If data would be of perfect quality, then data matching could be accomplished through straightforward database join operations and no sophisticated indexing techniques or approximate comparison functions would be needed. As long as data are of imperfect quality, however, techniques are needed that can deal with inaccurate and inconsistent data while still achieving high matching quality.

Various root causes for data quality problems have been identified [177]. The ones that are relevant to data matching are:

- *Multiple data sources*. If data are recorded by different organisations or different systems, at different locations, at different points in time, or using different data entry modes [72], then such data will likely be inconsistent.
- *Subjective judgement of data production*. If certain aspects of the entities in the databases to be matched were not recorded because they were deemed not to be of importance, then this information will be missing. This can potentially hamper data matching, if not enough data are available to accurately compare and classify pairs or groups of records. For example, if dates of birth have not been recorded then the matching of two hospital patient databases might have to rely upon patient's name and address details only.
- *Limited computing resources*. As will be discussed in Chap. 4, data matching is a computationally expensive process. If the databases to be matched are large and not enough computing and storage power are available, then it might not be feasible to run a sophisticated and accurate data matching algorithm. The results achieved with a simpler matching algorithm might not be accurate enough for certain applications. The use of cloud computing resources might be difficult for data matching because of privacy and confidentiality concerns.
- *Security/accessibility trade-off*. This root condition is highly relevant when databases that contain personal information are to be matched across organisations. Privacy regulations or security concerns might prevent that data which contain personal information can be accessed, thereby preventing certain data matching

projects. The topic of privacy within the context of data matching will be covered in detail in Chap. 8.

- *Coded data across disciplines*. This condition will affect the consistency of data between different databases. If the databases to be matched originate in different organisations or different disciplines, then careful mapping between different formats and encodings is required before any matching can be attempted.
- *Complex data representations*. Many traditional data matching algorithms can only be applied on data that are made of strings (such as name and address values) or numerical values (such as dates or age values). Increasingly, however, entity information is stored using more complex representations, such as XML schemas [270], or it consists of different types of entities that are potentially linked with other entities. Multi-relational, normalised databases are commonly used to represent different types of entities in an organisation and their interactions or relationships. Data matching algorithms must be able to deal with such types of complex data. This topic will be further covered in Sect. 9.2.
- *Volume of data*. As the size of the databases held by many organisations are ever increasing, deduplicating or matching them becomes more challenging, because more computing resources and more time is required. Chapter 4 deals with the topic of indexing for data matching, which is aimed at making the data matching process more scalable to very large databases.
- *Input rules too restrictive or bypassed*. This root cause can result in data of low quality because data are entered into fields or attributes that originally had a different purpose. For example, assume an emergency department's patient database where the personal details of emergency patients are recorded. The design of the system does require a valid date of birth to be entered for each patient's record. Imagine some patients arriving semiconscious or unconscious and without any identification documents. In such cases, no detailed information about their date of birth will be available. The receptionist or nurse who enters the data will likely be under time pressure. A simple solution for them to enter a valid record is to guess a patient's age and to then enter a date of birth value with the day and month values set to '01'. As a result, an unexpected high percentage of records in such a database will have a date of birth of 1st January.
- *Changing data needs*. The information need of organisations often changes over time, as they adapt to new regulations, implement new information systems, restructure themselves, or as they merge with other organisations. Only data that are useful and relevant for the operation of an organisation are normally collected, and therefore what information is stored in databases changes over time. New fields or attributes might be added to a database, attributes no longer considered relevant might be removed, or formats and codes might change over time. If data that have been recorded over time (or at different points in time) are being matched or deduplicated, then these changes can make the matching challenging, because only the information in attributes commonly available across time can be employed in the matching process.
- *Distributed heterogeneous systems*. Data recorded and stored in different systems potentially have different formats, different types and different values. When such

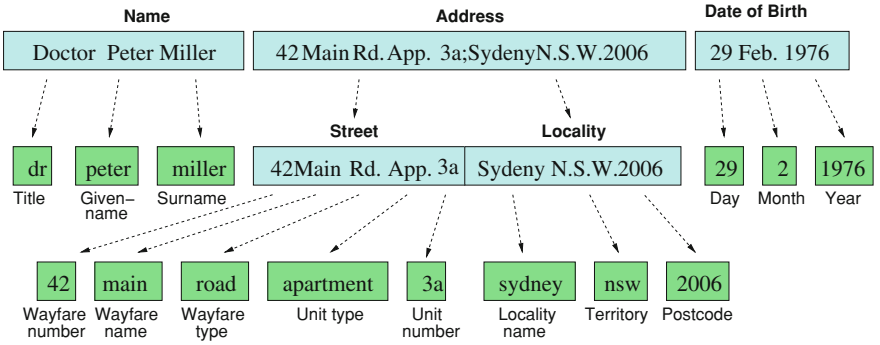



**Fig. 3.1** An example of data pre-processing applied to one record consisting of personal details. Cleaning includes removing unwanted characters and converting all letters into lower case. Standardisation consists of correcting typographical errors such as replacing 'sydeny' with 'sydney', and replacing abbreviations with standard forms. The third step is the segmentation of the input into well-defined output fields that are then used as the actual attribute values in the deduplication or data matching process

data are being matched, a careful analysis prior to matching is required to make sure that the same type of information (that will be compared in detail between records) is available in the same format and structure.

The remainder of this chapter covers in more detail how values in the input database(s) can be pre-processed to make them suitable for data matching and deduplication, as illustrated in Fig. 3.1. Data pre-processing for data matching consists of four major steps, as will be discussed in Sect. 3.5. First, however, discussions on the specific characteristics that names and other personal information pose to data quality, and where variations and errors in names come from, are needed.

## 3.2 Issues with Names and Other Personal Information

Names and other personal details play a crucial role in daily life because people are using them to identify individuals, ranging from family and friends, to work colleagues, and all the way to politicians and celebrities. For organisations both in the private and public sectors, names are often a primary source of identification of the individuals they are in contact with.

Personal names are a major component of the information used in many data matching or deduplication processes to identify records that refer to the same individuals. A large amount of the data collected by businesses and governments are about people. The identifying data collected about individuals generally include their names and addresses, dates of birth, social security or drivers license numbers, telephone numbers, and email addresses.

Much of the daily news fed to us through different channels is also about people, and therefore names commonly appear in news articles, on Web sites, and even most scientific and technical documents include their authors' names, affiliations, and other contact details.

Personal names are frequently used in Web searches to find information about individuals, in online stores to find movies, songs, albums or books by certain artists or writers, and when querying digital libraries to find articles or documents written by a specific author. The ten most popular query terms used with the Google Web search engine over the past decade include several personal names (of certain popular celebrities).[1]

Personal names and other identifying details have characteristics that make them different from general text [40, 208, 210]. These characteristics need to be considered when databases are matched or deduplicated, because they will influence how efficient and accurate the matching can be conducted. The following list highlights some of the issues with names, with an emphasis on the characteristics of names from English speaking and other Western countries.

- While in many languages for general words there is only one correct form, there are often several variations for what is seen as the same personal name. For example, there are more than forty variations of 'Amelia'[2]: 'Aemelia', 'Aimiliona', 'Amalea', 'Amalee', 'Amaleta', 'Amalia', 'Amalie', 'Amalija', 'Amalina', 'Amaline', 'Amalita', 'Amaliya', 'Amaly', 'Amalya', 'Amalyna', 'Amalyne', 'Amalyta', 'Amelie', 'Amelina', 'Ameline', 'Amelita', 'Ameliya', 'Amelya', 'Amelyna', 'Amelyne', 'Amelyta', 'Amilia', 'Amy', 'Delia', 'Em', 'Emelie', 'Emelina', 'Emeline', 'Emelita', 'Emi', 'Emma', 'Emmeline', 'Emmi', 'Emmie', 'Emmy', 'Mali', 'Malia', 'Malika', 'Meelia', 'Melia', 'Meline', 'Millie' and 'Milly'.
- In daily life, people often use or are given nicknames, rather than the name they were given by their parents at birth. Such nicknames can be short forms of their given name (such as 'Liz' for 'Elizabeth', 'Tina' for 'Christina', or 'Bob' for 'Robert'), they can be a variation of their surname of family name (such as 'Vesty' for 'Vest'), or their nickname is based on some life event, physical characteristic ('Ginger' for a red-haired person), or a character sketch of an individual [40]. Matching such nicknames can obviously be much more difficult than matching small name variations like the ones shown above. In certain cases, it will be impossible to find a match on a nickname at all.
- There are generally no legal regulations of what constitutes a name, with only some specific restrictions with regard to religious, political, or historical characters in certain countries.
- Names are language and culture specific [208]. In Anglo-Saxon countries (including the UK, USA, Canada, South Africa, Australia, Ireland, and New Zealand), names are made of a given or first name and a surname or family name, with an optional middle name (or initial) in-between, and possibly a name prefix or suffix

[1] See: http://www.google.com/press/zeitgeist.html.

[2] See: http://www.thinkbabynames.com/meaning/0/Amelia.

(such as 'Jr' or 'Snr'). In several European countries compound names are common, such as 'Hans-Peter' in Germany or 'Jean-Pierre' in France. Hispanic names often consist of two surnames.

- People can change their names over time, most commonly when they get married or divorced. While traditionally in many western countries a wife will take on the surname of her husband, this tradition is changing rapidly and today a husband might decide to take on his wife's surname. Alternatively, a couple might decide to compound their two surnames. For example, when 'Sally Smith' marries 'John Miller' she changes her name to 'Sally Smith-Miller', while her husband changes his name to 'John Miller-Smith'. If they have children, they need to decide which compound surname to give to their children.
- Outside of English speaking or Western cultures, each language has its own names and its own naming conventions, with cultures within the same language having their own ways of how names are selected for babies when they are born, and how they can change over an individual's life-time [40, 208].
- For languages that are based on characters different to the Roman alphabet, the way names are transliterated into the Roman alphabet is crucial. There might be several standards for transliterating for example Chinese, Japanese, Korean, Thai or Arabic names into the Roman alphabet, leading to variations of the same name. Individuals who are unfamiliar with standard transliteration systems might decide on a Roman version of their name in an ad hoc fashion, or alternatively choose or add a Western given name to their full name to better fit into a Western culture [208]. Arabic names commonly consist of several components and can contain various prefixes and suffixes that can be separated by hyphens or whitespaces, and that change over an individuals life-time depending upon his or her circumstances.

All these issues make data matching or deduplication using personal names a challenging undertaking, because the name values for the same individual might differ across two databases, or even within a single database. In our increasingly multicultural world where people are more mobile than ever before, where international travels and living in a country different to one's home country are common, and with the globalisation of businesses, the appropriate cleaning and standardisation of names in databases used for data matching are crucial components to achieve accurate matching results.

Besides names, addresses of where people live or where businesses are located, are a second major component of the information used in data matching [76]. While addresses are generally more standardised than names, there are several specific issues that need to be considered.

Addresses in most countries consists of a locality component and a street component, as illustrated in Fig. 3.1. The locality component generally contains a postcode or zipcode which allows mail to be efficiently directed to the destination locality. Postcodes and zipcodes are determined by a country's postal organisation. In some countries, such as Australia, each postcode covers an area of roughly the same number of households or businesses in order to allow a balanced handing of postal mail. However, as new suburbs are being built and existing areas change their characters,

postcode boundaries do change over time, and new postcodes are being generated. In other countries, the area of individual postcodes can be vastly different from others, and postcode boundaries do not change even when populations change.

The street component of an address usually consists of a street number, street name, and a street type. Additional street address elements can include flat or apartment numbers, floor numbers, and business or institution names. Alternatives to street addresses are post boxes and road-side mailboxes. While postal services in individual countries generally provide guidelines or standards of how a mailing address should be written, even if an address on a letter or parcel does not follow such guidelines, the item generally still arrives at its destination because the post man or woman or courier uses their local knowledge when delivering mail.

Because people know their mail arrives even if the address they provide is not totally accurate, a phenomena that has been reported is that individuals who reside close to an area that has a higher social status (for example if it is known that more rich people or celebrities live in that area) commonly use the name of the more prestigious area rather than the name of the area they live in, in order to impress friends and family. It is unlikely however that they would use such inaccurate address details when providing information to government agencies.

The third component of personal information that is commonly used for data matching are dates, such as dates of birth, dates of death, travel dates, or dates of admission to a hospital, to name a few. The major issue with dates is that if an individual does not know or does not remember a date when required, then commonly some approximation of the true date is being recorded. This might happen when dates are required from elderly people, or individuals need to report dates of other family members. If an accurate date is unknown, a common placeholder is to use the first day of the month (if the month of an event is known), or even the first day of January if only the year of when an event occurred is known.

Both personal names and people's addresses will likely change over time. Today, though unlikely, even the gender of a person can change. The only pieces of demographic information for an individual that do not change are their date and place of birth (that is why these two pieces of information are recorded on passports).

## 3.3 Types and Sources of Variations and Errors in Names

Given the many issues on name variations covered in the previous section, some discussion about studies that have investigated names variations and errors is required.

In an early study on spelling errors in general words, Damerau found that the majority of errors, over 80 %, were single character errors [89]. These were either a single letter that was deleted, an extra letter that was inserted, a letter that was substituted with another letter, or two adjacent letters that were transposed. The most common type of errors were character substitutions, followed by character deletions, then character insertions and finally the transposition of two characters. Multiple errors in a word were even less frequent than character transpositions. Damerau's

work lead to the development of edit distance based approximate string comparison function that aim to overcome such character-based variations, as will be described in Sect. 5.3.

Several other studies that followed from Damerau's work have reported similar results with regard to the types and distributions of variations or errors [133, 172, 214]. However, a more recent study that investigated patient names within hospital databases found different types and distributions of variations [113]. The most common type of variation, with 36 %, in these data were the insertion of an additional name word, initial or title word. The second most common type with 14 % were differences of several characters due to spelling mistakes or the use of nicknames. Other types of variation were differences in punctuation (like in 'O'Brian', 'OBrian' or 'O Brian') with 12 % of all variations, and changed surnames for female patients with 8 % of all variations. In this particular study, single character variations only accounted for 39 % of all variations compared to the over 80 % reported by Damerau [89]. This study highlights the differences between names and general text that was discussed in the previous section.

These differences need to be considered when data matching algorithms are being developed and employed on data that contain personal names. The most commonly occurring variations and errors can be categorised into [175, 243]:

- Spelling variations due to typographical errors that do not affect the phonetical structure of a name, such as 'Meier' and 'Meyer', or 'Christina' and 'Kristina'. These variations still pose a problem for data matching and need to be dealt with.
- Phonetic variations where the phonemes are modified for example through mishearing during data entry, and the structure of a name is changed substantially, such as from 'Sinclair' to 'St. Clair'.
- Double names that might be given in full, only the first name but not the middle name, or given as compound names (like 'Peter Paul Miller', 'Peter Miller', 'Peter Paul-Miller' or 'Peter-Paul Miller'. The variations here include potential different separators, missing name components, or even swapped name components.
- Name alternatives such as nicknames, married names or other deliberate name changes; and initials only (mainly for given and middle names).

A survey on spelling correction by Kukich has provided further details about character level misspellings that occur during data entry of general text [172]. She described three types of errors: (1) typographical errors, where the assumption is that the individual who was doing the data entry knew the correct spelling of a word but made a typing mistake (this author's favourite such mistake is to type 'Sydeny' instead of 'Sydney'); (2) cognitive errors, which are assumed to come from a lack of knowledge of the correct spelling or from misconceptions; and (3) phonetic errors, coming from the substitution of a correct spelling with a similar sounding one that is also correct.

The second and third type of errors will be a major cause for name variations, such as the many variations of the name 'Amelia' on p. 43, in databases where values are entered manually. The combination of spelling variations and phonetic and typographical errors further challenges data matching when using name data.

The major factor that causes different name variations and errors to occur, and that determines their likely types and their distribution, is the nature of how data are being entered [72]:

- With handwritten forms or texts that are scanned and where optical character recognition (OCR) techniques are applied [133, 214], the types of error most likely to occur will be substitutions between similar looking characters (such as between 'q' and 'g' or 'S' and '5'), or substitutions of a character sequence with a single character that looks similar (such as 'm' and 'r n', or 'b' and 'l i').
- When data are typed manually, then errors can occur that are specific to the layout of the keyboard used, with neighbouring keys being hit by mistake more likely (such as 'n' rather than 'm', or 'e' instead of 'r') than keys further apart. While in certain cases this can be quickly corrected (because the resulting name or word is clearly wrong), such errors can go unnoticed due to time pressure on or distraction of the person who is doing the data entry. Spell checkers are only of limited use for personal names. Data entered through mobile devices such as tablets or smartphone will also have different error characteristics specific to the device and its error prediction capabilities.
- If data are entered through dictation over the telephone, for example through a survey study, then the dictation process is a confounding factor to the manual keyboard based data entry. If there are ambiguities with a name, the person who is doing the data entry might not request a spelling clarification or correction but rather assume a default spelling which is based on their knowledge and cultural background. Studies have shown that errors occur more likely for names that come from a language or culture that is different to the one of the person who is doing the data entry, or if names are long or complicated, such as for example 'Kyzwieslowski' [113].
- A limitation in the maximum length of characters allowed in an input field can force the use of abbreviations, initials only, or even result in disregard of certain name parts (such as middle names).
- As a final source of variations, individuals from time to time report their names in different forms, depending upon the person or organisation they are in contact with, or they deliberately provide wrong or modified names. This is commonly the case in databases that are collecting crime and fraud related information, as was discussed in Sect. 1.4.4. And while an individual might report their details accurately and consistently and in good faith, somebody else might report a family member's or friend's details inconsistently or wrongly either for malicious reasons or simply because they do not know the correct details (for example only know a person by their nickname).

For all the reasons described so far, in many situations it is not straightforward to find the 'correct' variation of a name value that is misspelt or that contains mistakes. Within the domain of data matching, one therefore has to deal with legitimate name variations as well as errors introduced during data entry and recording. While the former need to be preserved to improve data matching quality, the latter should be

corrected if possible [40]. The challenge lies in distinguishing between the two. In the following four sections, different techniques for data pre-processing of names and other personal details are presented. The objective of these techniques is to convert the raw input data into a form that facilitates efficient and accurate data matching.

## 3.4 General Data Cleaning Tasks

Before discussing the specific steps of data pre-processing for data matching and deduplication in Sect. 3.5, in this section the three main tasks that are involved in data cleaning for any type of data analysis, mining, or processing, are presented. They are (1) handling missing values, (2) smoothing noisy values, and (3) identifying and correcting inconsistent values. Here, these three tasks are discussed with regard to their application to data matching and deduplication.

In applications such as data mining, where the aim is to detect novel and useful patterns in large databases [135], applying data cleaning can lead to much improved analysis results if the cleaning is conducted appropriately to the data mining techniques and algorithms employed. Missing values and noisy data such as outliers can have severe effects on both unsupervised and supervised learning tasks. Outliers can affect the results of data clustering, while missing values can lead to biased classification results or frequent patterns that include missing values and that therefore are not practically useful [135, 218].

When applied on data that are to be used for data matching or deduplication, different criteria need to be considered. Rather than detecting patterns, classes, rules or clusters in a database, data matching and deduplication are concerned with identifying individual records that refer to the same entities. Data cleaning must only modify the data in ways that support the application of data matching techniques. The following considerations for the three data cleaning tasks need to be taken into account:

- *Handling missing values*. Different options can be employed to handle missing values [135]:
  - Remove a record if it contains missing values. For data matching, this option will result in the removed records not being considered in the matching process at all, thereby potentially missing true matches. This option however might have to be taken if several crucial attribute values are missing in a certain record. For example, if all name and address values are missing then it is unlikely that there is enough information in other attributes to allow accurate matching.
  - Remove an attribute that contains missing values altogether from an input database, or do not use it for matching. For this option, if the attribute that contains missing values is crucial for the matching, then not considering it might be detrimental to matching quality. Even if many records have a missing value in such an attribute, then for those records that do have a value in this attribute the value should be used.

- Filling in a missing attribute value manually. This option might be possible for small databases or individual records, but this approach generally requires domain knowledge and potentially external reference data in order to identify the most likely value that should be inserted manually.
- Filling in a missing value automatically with a constant value. This option can only be applied on attributes that contain numerical values, which are rarely used for data matching or deduplication.
- Filling in a missing value with the attribute mean, median or mode. This option can only be applied on attributes that contain numerical values.
- Filling in a missing value with the mean, median or mode of a certain class of records for this attribute (for example, calculate and fill in the average salary separately for records that have a male gender from those that have a female gender). Again, this option can only be applied on attributes that contain numerical values.
- Determine the most likely value to be filled in using a rule or classification based approach. This approach is commonly used for data matching. The dependencies between certain groups of attributes allows this approach to be carried out with high efficiency and accuracy. For example, a missing gender value can be inferred based on a given name that is uniquely male or female, such as 'John' or 'Mary'. For other given names, such as 'Ashley', the gender might not be so easily determined. Another example where missing values can be inferred automatically are postcodes and suburb (or town) names. The postal services in many countries publish look-up tables of all combinations of postcodes and suburb names, and if one of these values is missing in a record and there is a one-to-one correspondence between a postcode and a suburb name then the correct value can be inferred from such look-up tables. These look-up tables can also be useful to detect and correct inconsistent values within a single record as will be discussed below.

Work on data editing and imputation has been pioneered by statisticians [107], and rule-based techniques to find the optimal value to be filled into a missing attribute value are commonly used by national census agencies to improve the quality of their survey data [143].

- *Smoothing noisy values*. Noisy data can consist of random errors or variance in values, or of outliers outside of an expected range of values (such as an age value of more than 120). They are often handled through binning, regression or clustering approaches that group similar values together and replace them by a central value such as a bin average or median, or a cluster centroid [135].
  Such approaches might not be suitable when data are cleaned for data matching, because such a smoothing could result in many records having the same values in a smoothed attribute. If the values in an age attribute, for example, are binned into decades (i.e. all records with an age value from 0 to 9, 10 to 19, 20 to 29 and so on are replaced with their corresponding bin averages of 4.5, 14.5, or 24.5, respectively), then the age attribute would lose much of the discriminating information that helps identify individuals that have the same age.

Even outliers can contain information that is relevant to data matching. Returning to the example of an age value of 120 given before, if this age is based on a recorded date of birth, for example 21/07/1891, then this could potentially be a data entry error where the actual date could be 21/07/1981. Such data entry mistakes can be handled by approximate comparison techniques as will be discussed in Chap. 5.
The standardisation of attribute values described in the following section can be seen as a form of smoothing data, but applied specifically to the values in attributes that contain names, addresses, or dates, for example.

- *Identifying and correcting inconsistent values*. Here, inconsistencies within a single record and between different records need to be distinguished. The former case can sometimes be dealt with through external look-up tables and rules that (similar to filling in missing attribute values) can be used to detect if the values in two attributes contradict each other (for example a record with given name 'Paul' and gender 'F'). If such inconsistencies should be corrected or not depends upon the data at hand, and any knowledge about the quality of the data and the way they were entered. Section 3.5.4 further discusses this issue in the context of verifying the consistency of addresses.
  If the attribute values in a single record are inconsistent, then at least one value needs to be changed (corrected). Unless there is certainty about which of two (or more) values is most likely the wrong one, any such change can result in further mistakes being introduced rather than corrected. In the above example, either it is assumed that the 'F' gender value is wrong and should be changed into 'M', or the given name value could be wrong and its correct value is actually 'Paula'.
  Because a major aspect of the steps involved in data matching is to be able to deal with inconsistencies between attribute values, appropriate advice is to only change inconsistent attribute values if there is certainty about which value is wrong and needs to be corrected. If it is not possible to ascertain this, then the inconsistent values should rather be kept, and appropriate approximate comparison and classification techniques need to be applied that can deal with such inconsistencies but still achieve high matching accuracy. Such techniques are discussed in Chaps. 5 and 6.
  Inconsistencies between different records should be corrected as much as possible before data matching or deduplication is conducted. Different codings for the same attribute, for example, either within a single database or across two databases, should be converted into the same values. For a gender attribute, for example, if one database uses the values 'F' and 'M' while the other database uses '1' and '0' then in a pre-processing step the values in either database need to be changed. Ideally, values should be changed such that they become more easily to understand and interpret [19].

Data exploration and profiling, supported through a variety of tools [19, 62, 278], are important steps that help to establish the quality of the data at hand, and to decide what types of data cleaning to employ on which parts of the data. Exploration and profiling involves collecting basic summary statistics for all attributes in a database, such as the minimum and the maximum values in an attribute, the most commonly

occurring values, the distribution of the occurrence of all values in an attribute, how many records have a missing value in an attribute, and so on.

## 3.5 Data Pre-Processing for Data Matching

Data pre-processing refers to the tasks of converting the raw input data from the databases to be matched or deduplicated into a format that allows efficient and accurate matching [76]. Figure 3.1 on p. 42 illustrated this process on a single example record. The example databases used in the previous chapter also illustrated the process. Figure 2.2 on p. 25 shows the raw input databases, and Fig. 2.3 on p. 27 their pre-processed versions.

It is assumed that the attributes (or fields) in the input database(s) contain values that are separated by whitespace characters. These values are known as tokens. They can be words, single characters (such as initials), numbers, or compound elements such as apartment and street numbers concatenated by a slash ('3/42'), or telephone numbers made of concatenated groups of digits ('045-768-2231'). How these tokens are pre-processed is described in the following subsections.

### *3.5.1 Removing Unwanted Characters and Tokens*

This first step of data pre-processing corresponds to a data cleaning step. The attribute values in the input database(s) might contain certain individual characters, and certain words, terms, or abbreviations, that do not contain information that is of use for data matching or deduplication, and that can and should be removed from the attribute values. Other characters or tokens need to be converted into a standardised form, for example different types of parenthesis or quotes should be replaced with one specific parenthesis or quote character, which will facilitate the standardisation and segmentation applied to the cleaned attribute values in the next steps.

Either hard-coded rules or look-up tables, such as the example shown in Fig. 3.2, are used to accomplish this first data pre-processing task. Look-up tables are generally easier to adjust to changing data needs compared to hard-coded rules, however employing hard-coded rules can be more efficient and faster than using look-up tables. For each record in the input database(s), its attribute values are scanned to see if they contain any of the tokens that are to be removed or converted. If such a token is found it is removed or converted. It is possible to have different look-up tables or rules for different types of input attributes, for example one look-up table for name attributes and one for address attributes.

Another component of this first pre-processing task is to convert all letters into either lowercase or uppercase characters, and to convert Unicode characters into ASCII characters or the other way around. Which format is chosen depends upon the characteristics of the data at hand and the limitations and requirements of the

```
# Remove characters and words from input
  ' ' := '.', '?', '~', ':', ';', '^', '=', ' na ', ' n/a '
  ' ' := ' n.a. ', ' c/o ', ' c/- ', ' also ', ' name ', '!'
  ' ' := ' only ', ' abbrev ', ' locked ', ' on ', ' of '
  ' ' := ' unk ', ' unkn ', ' missing ', '*'

# Correct words and symbols
     ' roman catholic ' := ' r/c ', ' r / c ', ' rc '
  ' church of england ' := ' c/e ', ' c / e ', ' c of e '
   ' no fixed address ' := ' nfa ', 'n/f/a ', ' n.f.a.'
      ' nursing home ' := ' n / home '
    ' other territory ' := ' o/t ', ' o.t.'
               ' and ' := '+', '&'
                 ' ( ' := '<', '(', '[', '{'
                 ' ) ' := '>', ')', ']', '}'
                 ' | ' := '"', '"', '`', '||', '|', "''"
                 ' - ' := '-', '_'

# Correct roman numbers
                ' 1 ' := ' i '
                ' 2 ' := ' ii '
                ' 3 ' := ' iii '
                ' 4 ' := ' iv '
                ' 5 ' := ' v '
                ' 6 ' := ' vi '
                ' 7 ' := ' vii '
                ' 8 ' := ' viii '
                ' 9 ' := ' ix '
               ' 10 ' := ' x '

# Correct ordinal numbers
            ' first ' := ' 1st '
           ' second ' := ' 2nd '
            ' third ' := ' 3rd '
           ' fourth ' := ' 4th '
            ' fifth ' := ' 5th '
            ' sixth ' := ' 6th '
          ' seventh ' := ' 7th '
           ' eighth ' := ' 8th '
            ' ninth ' := ' 9th '
            ' tenth ' := ' 10th '
```

**Fig. 3.2** An example correction look-up table as used by the FEBRL [62] system (described in more detail in Sect. 10.2.4). The correction works by replacing any character sequence (string in quotes) found in an attribute value of an input record that is listed on the right-hand side of a ':=' with the character sequence on the left-hand side of the ':='. As can be seen, a variety of characters and words are replaced by a single whitespace character (i.e. they are removed from an attribute value), while several variations of the same abbreviations or characters are replaced by an expanded or standardised version. Lines starting with a '#' character are comment lines

data matching or deduplication system used. A last component in this first task is to replace all multiple occurrences of whitespace characters with a single whitespace only, and to remove all leading and trailing whitespaces. For example, assuming a '␣'

symbolises a single whitespace character, the input string '␣␣Paul␣␣Peter␣␣␣␣Miller␣' would be converted into 'paul␣peter␣miller'.

### 3.5.2 Standardisation and Tokenisation

The second step of data pre-processing is the standardisation of the tokens in the attribute values by detecting and correcting values that contain known typographical errors or variations, expanding abbreviations and replacing them with standard forms, and replacing nicknames with their proper name forms.

In this data pre-processing task, individual or groups of tokens are compared with extensive look-up tables that contain values with variations and errors and their corresponding standardised and corrected values. For addresses, for example, separate such look-up tables are required for street names, locality names, state and territory names, and country names; while for personal names look-up tables are needed for title words, given names (ideally separate for female and male) and surnames. Figure 3.3 shows such a table for locality names (suburbs, towns and cities).

Such look-up tables can either be generated from databases within an organisation, or be acquired from commercial providers, or (in the cases of address data) can be available from national postal services. Look-up tables that contain common typographical variations and errors, such as the examples shown in Fig. 3.3, can be compiled from attribute values as they are entered into a database and flagged as being an unknown value. An approximate string comparison function (which will be discussed in Chap. 5) can for example be used to detect the correct value in a certain attribute that is most similar to an unknown value. Candidate variations for a look-up table can then be generated automatically to be validated by a domain expert before being added into a tagging look-up table. For example, using the look-up table in Fig. 3.3, if an input locality name 'bewerly hills' is entered by a client in a Web form, the most similar valid locality name would be 'beverly hills', and therefore 'bewerly hills' can be added as a possible variation of 'beverly hills' into the locality name look-up table.

In the tokenisation process, each token is commonly assigned one or more tags which designate the type of the token according to the look-up table(s) where this token was found, or based on some hard-coded rules. The outcomes of this process is illustrated in Fig. 3.4 assuming the look-up table from Fig. 3.3 is used. The tags are used in the third data pre-processing step to segment the sequence of tokens in an attribute value into their most appropriate output fields, as will be described in the next subsection.

The tokenisation process is normally started with the first set of tokens on the left of an attribute value. A sequence of one or several tokens is considered at any time. The tokenisation is conducted in a 'greedy' fashion [76], in that longer token sequences are considered first before shorter ones. If the longest token sequence in any of the look-up tables used in a tokenisation process contains $l$ tokens (for example

```
# Locality names

tag=<LN>  # Tag for locality name words
              alexandria := alezandria

           alfords point := alfonds point, alford point
           alfords point := alforts point, alfrods point

           beverley park := bevely park, bevelly park
           beverley park := beverley park, beverlly park

           beverly hills := beverley hills, beverly hill

          sydney airport := syd inter airport, syd airport

the university of sydney := sydney university, sydney uni
the university of sydney := uni sydney, university sydney
```

**Fig. 3.3** An example tagging look-up table as used by the FEBRL [62] system. The tag 'LN' is used to designate all following entries in this look-up table as locality names. A sequence of tokens that occurs in an attribute value that is listed on the right-hand side of the ':=' will be replaced by the sequence of tokens on the corresponding left-hand side, and the sequence of tokens is assigned the 'LN' tag, as illustrated in Fig. 3.4

'syd inter airport' contains $l = 3$ tokens), then at any step of the process the next $l$ tokens in an input attribute value are considered. If the tokenisation process starts from the left, then the first $l$ tokens, denoted with $t[1], t[2], \ldots, t[l]$, are considered to be the candidate set of tokens in the first step. If these $l$ tokens match a token sequence in any of the look-up tables, then they are replaced by the sequence of corrected tokens, and the tag of this corrected sequence is assigned to the set of tokens.

For example, using the look-up table from Fig. 3.3, if a token sequence starts with 'syd inter airport' then these three tokens are replaced by the standardised compound token 'sydney airport' which is assigned an 'LN' tag to designate that it corresponds to a known locality name. Note that even correct known token sequences that are found in an input field, such as 'sydney airport', are assigned the corresponding tag.

If at any step in the tokenisation process no token sequence of length $l$ is found in the input attribute value, then the length of the candidate token sequence is reduced from $l$ to $l-1$ (i.e. tokens $t[1]$ to $t[l-1]$ are considered), and again all look-up tables are searched for this token sequence. The length of the candidate token sequence is reduced until either a token sequence is found in a look-up table, or a single token is assigned an appropriate hard-coded tag (as for example listed in Table 3.1). It is also possible that a token is assigned several tags if the token is found in several look-up tables, as shown in Fig. 3.4.

It is important that longer candidate token sequences are considered first, such that for example the token sequence 'sydney uni' is correctly identified to correspond to the standardised locality name 'the university of sydney', rather than the single

| | | |
|---|---|---|
| Paul | Peter | Miller |
| paul | peter | miller |
| GM | GM,SN | SN |

| | | | | | |
|---|---|---|---|---|---|
| 17 | Epinng | Rd | Bevely Park | N.S.W. | 2011 |
| 17 | epping | road | beverly park | nsw | 2011 |
| NU | LN,SN | ST | LN | TR | PC |

**Fig. 3.4** Two examples of input values, the first being a name and the second an address. The first row in each example shows the raw uncleaned input, the second row shows the cleaned and standardised tokens, and the third row shows the tag(s) assigned to each token that indicate their type. A description of these tags is given in Table 3.1

token 'sydney' is assigned as locality name and then the second token 'uni' is left as a potentially unknown token.

The standardisation and tokenisation process continues as long as there are unprocessed tokens in an attribute value. At the end of the tokenisation process, all tokens in an attribute value will have been replaced by corrected and standardised forms, and they will have one or more tags assigned to them, as illustrated in Fig. 3.4.

### *3.5.3 Segmentation into Output Fields*

The third step of data pre-processing is the segmentation of the tokenised and tagged attribute values into well-defined output fields that are suitable for data matching or deduplication, as was illustrated in Fig. 3.1. This step is the most challenging step in data pre-processing, because often there are several possible assignments of tokens to output fields. The challenge is to identify the most likely assignment. This task is also known as *parsing* [143], and is related to the field of *information extraction* which is concerned with identifying structured information in semi-structured or free format text [230].

The objective of segmentation is to have each output field contain a single piece of information, made of one or a small number of tokens, rather than having several pieces of information in one field or attribute, as was illustrated in Fig. 3.1 on p. 42. The values in these output fields are then used in the detailed pairwise comparison of record pairs (as will be discussed in Chap. 5), which generally leads to much improved matching quality compared to when the unstandardised and unsegmented input attribute values would be used. The following lists show the output fields that are commonly used for data matching or deduplication:

- *Personal names*. Title, name prefix, given or first name, initials, middle name, family name or surname, alternative family name or surname, name suffix.

- *Street addresses*. Unit prefix, unit type, unit number, unit suffix, street or wayfare number, street or wayfare name, street or wayfare type, building name, postal address number, postal address type, institution name, institution type.
- *Address localities*. Locality or town name, territory or state name, postcode or zipcode, country.
- *Dates*. Day, month, year.
- *Telephone numbers*. Country code, area code, number, extension.

Not all of these output fields will be available in all databases, and for many records some of the fields will not contain a value. The actual output fields used and their names also depends upon the data at hand and of course will differ from country to country.

Challenges occur when there are ambiguities in a token sequence that is to be segmented. For example, the middle name 'Peter' in the three name words 'Paul Peter Miller' shown in Fig. 3.4 could either refer to this person's middle name or to his surname (with 'Miller' being a second surname from the original compound surname 'Peter-Miller').

Different segmentation techniques have been developed for different types of input data, such as personal names, business names, or addresses. Sections 3.6 and 3.7 will cover the two main types of techniques employed, rule-based and statistical, in more detail.

The standardisation and segmentation steps in data pre-processing are not necessarily independent of each other. They can be combined into one process, where the segmentation is conducted on the unstandardised tokens first, and the standardisation is applied based on the segmented tokens. For example, the abbreviation 'St' in an address attribute can either stand for the street type word 'Street', or be part of a town name such as 'Saint Mary', depending on the overall token sequence in the address.

### *3.5.4 Verification*

A possible fourth step of data pre-processing is the verification of the correctness of the values assigned to the different output fields, and the validation of value combinations in several attributes. For names, for example, such verification can include checking if the combination of values in the given name and gender attributes are valid (a given name 'John' and gender value 'F' is generally not a valid combination), or if a title word does contradict the given name value of a record (a title 'Ms' is not valid for a record with given name 'John'). Such tests can be based on look-up tables of known give names that are uniquely male or female, as was previously discussed in Sect. 3.4.

For addresses, testing the existence and correctness of address values can be carried out using external reference databases, that, for example, contain all validated addresses in a country. Such reference databases are commonly available from national postal services or commercial providers. They allow the verification of

**Table 3.1** List of tags used by the data standardisation module of the FEBRL system [62]

| Tag | Description | Component | Based on |
|---|---|---|---|
| LQ | Locality qualifier word | Address | Look-up table |
| LN | Locality (town, suburb) name | Address | Look-up table |
| TR | Territory (state, region) name | Address | Look-up table |
| CR | Country name | Address | Look-up table |
| IT | Institution type | Address | Look-up table |
| IN | Institution name | Address | Look-up table |
| PA | Postal address type | Address | Look-up table |
| PC | Postcode (zipcode) | Address | Look-up table |
| N4 | Numbers with four digits (not known postcodes) | Address | Hard-coded rule |
| UT | Unit type (e.g. 'flat' or 'apartment') | Address | Look-up table |
| WN | Wayfare (street) name | Address | Look-up table |
| WT | Wayfare (street) type (e.g. 'road' or 'place') | Address | Look-up table |
| TI | Title word (e.g. 'ms', 'mrs', 'mr', 'dr') | Name | Look-up table |
| SN | Surname | Name | Look-up table |
| GF | Female given name | Name | Look-up table |
| GM | Male given name | Name | Look-up table |
| PR | Name prefix | Name | Look-up table |
| SP | Name separators and qualifiers (e.g. 'aka' or 'and') | Name | Look-up table |
| BO | 'baby of' and similar values | Name | Look-up table |
| NE | 'nee', 'born as' or similar values | Name | Look-up table |
| II | Initials (one letter token) | Name | Hard-coded rule |
| ST | Saint names (e.g. 'saint george' or 'san angelo') | Address/name | Look-up table |
| CO | Comma, semi-colon, colon | Address/name | Hard-coded rule |
| SL | Slash '/' and back-slash '\' | Address/name | Hard-coded rule |
| NU | Other numbers | Address/name | Hard-coded rule |
| AN | Alphanumeric tokens | Address/name | Hard-coded rule |
| VB | Brackets, braces, quotes | Address/name | Hard-coded rule |
| HY | Hyphen '–' | Address/name | Hard-coded rule |
| RU | Rubbish (for tokens to be removed) | Address/name | Look-up table |
| UN | Unknown (none of the above) | Address/name | Hard-coded rule |

This table is adapted from Table 3 in [76]. A 'hard-coded rule' refers to the cases where a specific piece of program code is used to assign a tag to an input character or token. As can be seen, most tags are based on look-up tables and specific to either addresses or names. Some of the hard-coded tags take care of special characters or help to characterise tokens not found in any of the look-up tables

different parts of a segmented address, including the verification of locality name and postcode combinations, and if such a combination is known in a given territory or state. Other verification steps for addresses include the test if a street name and type combination occurs in the locality value given in a record, and even if a street number occurs in the given street or not.

If an invalid combination is found then it can either be flagged for manual inspection, or be corrected automatically (but being aware of the potential that a correction can introduce new errors, as was described on p. 51). In case no correction is being

made for an invalid combination, a flag can be added to the record indicating the attributes that contain inconsistent values. This information can then be used in the matching process to, for example, lower the similarity value between two records if the flag indicates that some of the address values in a record might be wrong.

## 3.6 Rule-Based Segmentation Approaches

Rule-based techniques for segmentation of names and addresses have been employed in the field of data matching for several decades. The basic idea of such techniques is to process the list of tokens and tags either from left to right or from right to left, and using hand-crafted or learned rules to assign the token or tokens covered by a rule to their appropriated output field.

Processing token sequences starting from the left is appropriate for most name values from Western countries, as well as the street component of addresses, while processing token sequences starting from the right can be appropriate when locality details (postcodes, suburb, state, and country names) are available in the attributes to be standardised.

Rule-based approaches are best suited for input fields that contain controlled and well-structured information, such as telephone numbers or names that are made of only a small number of tokens [230]. For addresses, developing efficient and accurate rule-based systems is much more difficult [76], because a much larger number of rules is needed that can deal with the much larger variability in token sequences that represent addresses.

A rule-based system is made of two parts. The first is a set of rules in the form of '**if** *condition* **then** *action*' [217]. The *condition* of a rule tests for the occurrence of a certain tag or tag sequence, and the *action* is the assignment of the tokens covered by a rule into the appropriate output fields. The *condition* of a rule is generally testing for tags rather than tokens, because tags are more general than tokens and therefore rules based on tags can cover more variability in the input. If the *condition* of a rule is true then the rule is 'triggered' or 'fired' and the *action* of the rule is executed.

The second part of a rule-based system is the ordering or the policies of which rules should be fired first when the *condition*'s of several rules are true for a certain sequence of tags. The ordering can either be based on the specificity of the rules, in that rules that cover more tags are fired first, or it can be based on which output fields are most important and should have values assigned to them (for example, the given name and surname output fields are more important than the middle name or name suffix fields), or the ordering can be based on a manual sorting of the rules using domain knowledge. Often various heuristics are applied, and special cases are handled with individuals rules [230].

Figure 3.5 shows an example subset of rules for segmenting name values. Rules are normally applied on the tags (denoted with $t[i]$) that have been assigned to the tokens (denoted with $o[i]$) in the tokenisation step as was described in Sect. 3.5.2. While the eight rules shown may cover most known simple names in a database, more complex

**if** t[i] = 'TI' **then** title ← o[i]
**if** t[i] = 'PR' **then** name_prefix ← o[i]

**if** t[i] = 'GM' and t[i+1] = 'SN' **then** given_name ← o[i], surname ← o[i+1]
**if** t[i] = 'GF' and t[i+1] = 'SN' **then** given_name ← o[i], surname ← o[i+1]
**if** t[i] = 'SN' and t[i+1] = 'GM' **then** given_name ← o[i+1], surname ← o[i]
**if** t[i] = 'SN' and t[i+1] = 'GF' **then** given_name ← o[i+1], surname ← o[i]

**if** t[i] = 'UN' and t[i+1] = 'SN' and $i+1 = L$ **then** given_name ← o[i], surname ← o[i+1]
**if** t[i] = 'SN' and t[i+1] = 'UN' and $i+1 = L$ **then** given_name ← o[i+1], surname ← o[i]

**Fig. 3.5** A small example of a subset of rules used to segment a name input value using the tags defined in Table 3.1. The sequence of tokens is denoted by $o[i]$ and the corresponding sequence of tags by $t[i]$, with $1 \leq i \leq L$ and $L$ being the number of tokens in the given name input value. The first rule assigns a known title token into the $title$ output field, while the second rule assigns a known name prefix into its appropriate output field. The next four rules assign known given name and surname values into the appropriate fields, while the last two rules only assign given name and surname values into the corresponding output fields if there is no other tag that follows. Only one rule is applied on a tag (or tag sequence), and once the tag (or tag sequence) is covered by a rule it is removed from the input tag sequence

names made of several components (name prefixes and suffixes, middle names, etc.) will require many more specific rules.

When a rule-based system is developed based on hand-crafted rules, initially the basic rules (such as the ones shown in Fig. 3.5) are implemented and applied on the data that are to be standardised. All records that are not covered by any rule are then used to develop additional rules. Such a *failure driven* iterative approach [217] over time results in a rule-base that covers most if not all variations of tag sequences that occur in an attribute. The manual investigation of the input values not covered by any rule, and generating appropriate new rules for them, is however a labour intensive task that needs to be repeated each time data with new characteristics are to be standardised.

There are various ways of how rules can be represented, including regular expressions, SQL statements, pattern items and lists, and even specific pattern languages or scripts written in programming languages such as Java or C++ [230]. The early AutoStan/AutoMatch [251] suite of data cleaning and matching software, developed in the 1990s by Matthew Jaro (formerly of the US Census Bureau and founder of MatchWare Technologies), for example, employed a look-up table based tokenisation phase followed by a re-entrant regular-expression rule-based parsing and segmentation phase. Regular expressions allow for rules where tokens are checked for certain string patterns (not just equality). For example, the test **if** o[i] = 'stre?t' **then**... will return true for the two words 'stret' and 'street'. The re-entrant approach of AutoStan means that once a token (or token sequence) is covered by a rule (and assigned into an output field), the token (or token sequence) is removed from the input token sequence and the rule-base is applied on the new shorter token sequence. AutoStan rule-bases employed for segmenting addresses could contain hundreds if not thousands of rules for production systems.

An alternative to the labour intensive manual development of a rule-base is to employ a rule-learning algorithm [230]. Rules can be learnt in an automatic fashion if training data in the form of correctly segmented input examples are available. Such training data can either be generated manually, or be the output of an earlier segmentation of similar data. In its most general form, a rule learning system learns individual rules that are disjunctions and that each cover a subset of the training data set.

Assuming a training data set $D$ is provided that contains $n$ training records, $d_1, \ldots, d_n$, each made of a sequence of tags and the output fields they are assigned to. Figure 3.6 shows three such training records. The objective of a rule learning system is to learn a set of $k$ rules $r_1, \ldots, r_k$ that cover all training records in $D$. The *condition* part of each rule $r$ covers a certain subset $s(r)$ of all training records in $D$, which is called the *coverage* of a rule. The *action* part of a rule will be correct for some training records in $s(r)$ and wrong for others. The subset of training records where the action is correct is denoted with $s'(r) \subseteq s(r)$. The *precision* of a rule is calculated as $p = |s'(r)|/|s(r)|$ [230].

The objective of a rule-learning system is to learn rules that provide good coverage and have high precision, because such rules will be well suited for the segmentation of new unsegmented input records. Finding the optimal set of rules for a given training data set is intractable, therefore practical rule learning algorithms are based on heuristic approaches [230]. The two broad categories of heuristics are bottom-up and top-down approaches. In the first category, a very specific rule (that only covers one training record) is made more general by removing a test from the *condition* of the rule such that the coverage of the rule is increased (at the cost of likely loosing some precision). In top-down approaches, general rules that cover many training records but have low precision are made more specific by adding further tests to the *condition* of the rule until some stopping criteria is reached. There are various ways of how these approaches can be implemented and different algorithms have been developed, including *Rapier*, $(LP)^2$, *FOIL* and *WHISK*. An excellent survey of rule-based approaches to information extraction is provided by Sarawagi [230].

## 3.7 Statistical Segmentation Approaches

A major drawback with rule-based approaches to data segmentation is that rules are hard, meaning that they either fire or do not fire (i.e. cover a set of tokens in an input sequence or not), depending upon if a certain condition is fulfilled [230]. In practice this means that for unseen variations in the input data that are not covered by any rule in a rule base, a new rule is required. Manually generating a comprehensive rule-base is labour intensive and time consuming, and requires adjustments each time the data to be segmented changes [217]. Techniques that learn rules from training data also require training examples for any variation in the data in order to be able to learn the rule or rules that cover these examples. Therefore, comprehensive training data sets are required, which again can be expensive to generate or collect.

| | | | | | |
|---|---|---|---|---|---|
| 17 | epping | road | beverly park | nsw | 2011 |
| NU | LN | ST | LN | TR | PC |
| Wayfare number | Wayfare name | Wayfare type | Locality name | Territory | Postcode |
| | main | street | sydney | nsw | 2000 |
| | WN | ST | LN | TR | PC |
| | Wayfare name | Wayfare type | Locality name | Territory | Postcode |
| 42 | george | ally | newtown | | 2067 |
| NU | WN | ST | LN | | N4 |
| Wayfare number | Wayfare name | Wayfare type | Locality name | | Postcode |

**Fig. 3.6** Three example training sequences of addresses consisting of tokens (top rows in each example), tags (middle rows) and output fields (bottom rows). The tags are based on the list given in Table 3.1, while the output fields correspond to those shown in Fig. 3.1. Each token is assigned one tag only, and only the tags and names of output fields are used for training a rule-based or statistical segmentation model, but not the actual tokens that make up an address

Statistical approaches to segmentation try to overcome the rigid decision making of rule-based systems. They are instead based on probability distributions that provide likelihoods of which token in an input should be assigned to which output field. Similar to rule-based systems, these probabilities are learned from training data that consist of segmented token and tag sequences where each token is assigned its appropriated tag, as illustrated in the examples shown in Fig. 3.6. Similar to rule-based systems, tag sequences rather than token sequences are used to train the statistical segmentation models and to segment new input values [76].

The process of assigning tokens to output fields can be seen as a classification process where each token is assigned to its most likely output field (more generally called *label*) according to the learned model [230]. However, this task is following an ordering, in that the classification of a token depends upon the classification of the previous token(s) (assuming an assignment of tokens starts from the left of a token sequence) and possibly also the following token(s).

Different statistical models have been developed that capture these dependencies within tag sequences. The most popular techniques have been hidden Markov models (HMMs) [223], maximum entropy Markov models (MEMMs), and more recently Conditional Random Fields (CRFs). CRFs are capable of modeling a single joint distribution over the sequence of the predicted output fields for a given token sequence. The dependency of the classified output field of a token is based on the adjacent previous and next output fields. For HMMs, on the other hand, the classification of a token only depends upon the classification of the previous token but not the following one.

The process of training a statistical model for segmentation is based on either calculating the maximum likelihood or maximum margins for all given tag sequences in the training data. The detailed mathematical descriptions of these techniques are

outside the scope of this book, the interested reader is refereed to the excellent survey given by Sarawagi [230]. In the remainder of this section, an example of an address segmentation approach based on HMMs is provided, which previously has been shown to outperform a manually developed rule-based approach [68, 76].

### *3.7.1 Hidden Markov Model Based Segmentation*

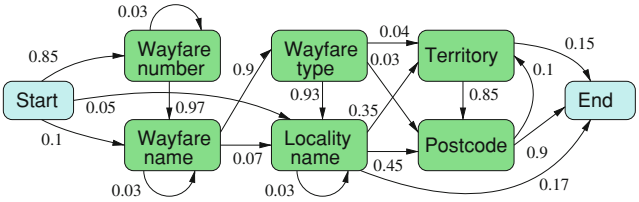
Hidden Markov models [223] were developed in the 1960s and 1970s. They are widely used in speech recognition and natural language processing. They are computationally efficient to train and are able to handle new unknown sequences in a robust fashion. They have been employed by several researchers for name and address segmentation [41, 56, 68, 76, 240].

A HMM can be viewed as a probabilistic finite state machine that consists of a set of (hidden) states, transition links between these states, and a set of output (or observation) symbols. Each link between two states has a nonzero probability assigned with it, and each state emits output symbols with a certain probability distribution. The transition and output probabilities are stored in two matrices. A simple example of a HMM for address segmentation, together with its transition and output probability matrices, is shown in Fig. 3.7.

Two special states of a HMM are the *Start* and *End* state. Beginning with the *Start* state, a trained HMM generates a sequence of output symbols $O = o_1, o_2, \ldots, o_k$ by making $k - 1$ transitions from one state to another until the *End* state is reached. The output symbol $o_i$, $1 \leq i \leq k$, generated in state $i$, is based on this state's probability distribution of the output symbols. The *Start* and *End* states are not actually stored in a HMM because no output symbols are emitted in these states. Instead of the *Start* state a list of initial state probabilities is used that provide the likelihoods that a sequence starts with a certain state.

For a given trained HMM, it is possible that the same sequence of output symbols can be generated by taking different paths through the HMM. Each path, however, will have a different probability according to the transition probabilities between the states in the path. Given a certain sequence of output symbols, for the task of segmentation one is interested in the most likely path through a given HMM that will generate this sequence. Using a dynamic programming approach, the *Viterbi* algorithm is an efficient way to compute this most likely path for a given sequence of output symbols [223].

Training data in the form of sequences of (state name, output symbol), possible manually prepared, are required to learn the transition and output probabilities. Each training record corresponds to a path through the HMM from the *Start* to the *End* state. While the set of output symbols can be created using the training data, the states of a HMM are generally fixed and are defined before training. When segmenting addresses, for example, the set of states will correspond to all possible output fields of an address, such as the ones listed on p. 56, while the output symbols correspond to all possible tags (as for example listed in Table 3.1) that can occur with addresses.




| From state | To state: Wayfare number | Wayfare name | Wayfare type | Locality name | Territory | Postcode | *End* |
|---|---|---|---|---|---|---|---|
| *Start* | 0.85 | 0.1 | 0.0 | 0.05 | 0.0 | 0.0 | 0.0 |
| Wayfare number | 0.03 | 0.97 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Wayfare name | 0.0 | 0.03 | 0.9 | 0.07 | 0.0 | 0.0 | 0.0 |
| Wayfare type | 0.0 | 0.0 | 0.0 | 0.93 | 0.04 | 0.03 | 0.0 |
| Locality name | 0.0 | 0.0 | 0.0 | 0.03 | 0.35 | 0.45 | 0.17 |
| Territory | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.85 | 0.15 |
| Postcode | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.9 |

| Output symbol | State: Wayfare number | Wayfare name | Wayfare type | Locality name | Territory | Postcode |
|---|---|---|---|---|---|---|
| NU | 0.9 | 0.01 | 0.01 | 0.01 | 0.01 | 0.05 |
| WN | 0.01 | 0.5 | 0.01 | 0.1 | 0.01 | 0.01 |
| WT | 0.01 | 0.01 | 0.92 | 0.01 | 0.01 | 0.01 |
| LN | 0.01 | 0.1 | 0.01 | 0.8 | 0.01 | 0.01 |
| TR | 0.01 | 0.06 | 0.01 | 0.01 | 0.93 | 0.01 |
| PC | 0.03 | 0.01 | 0.01 | 0.01 | 0.01 | 0.8 |
| N4 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.1 |
| UN | 0.01 | 0.31 | 0.02 | 0.05 | 0.01 | 0.01 |

**Fig. 3.7** A simplified Hidden Markov model (top) for addresses, based on the output fields shown in Fig. 3.1. The table in the middle shows the transition probabilities, while the table at the bottom shows the output probabilities. Adapted from [76]

The training process iterates over all training records and adjusts the transition and output probabilities according their output field and tag sequences. For example, the transition probability of 0.93 from state 'Wayfare type' to 'Locality name' in Fig. 3.7 results from 93 % of all training records containing these two output fields in sequence, while only 4 % of training records had a value in the 'Territory' output field directly after a value in the 'Wayfare type' output field, and only 3 % of training records had a 'Wayfare type' output field that was directly followed by a 'Postcode' output field.

Instead of using the actual tokens found in an input field, the tag or tags that were assigned to each token are used as the output symbols of the HMM [76]. This makes a HMM more general and more robust, and also computationally more efficient because the number of different tags is much smaller than the number of different tokens that will be encountered in the input data.

Once a HMM is trained on a set of example input values, sequences of tags from new records that are to be segmented into output fields can be segmented efficiently using the *Viterbi* algorithm [223], which returns the most likely state sequence of the given tag sequence through the HMM. This sequence of states, which corresponds to a sequence of output fields, is then used to assign each token of an input value to an output field. For example, consider the following token sequence from an address with its corresponding tag sequence which is based on the tags from Table 3.1 on p. 57:

| 32 | Garden | Place | Brisbane | 7014 | Queensland |
|---|---|---|---|---|---|
| NU | WN | WT | LN | PC | TR |

Applying the Viterbi algorithm for this tag sequence on the example HMM from Fig. 3.7 will lead to the following state sequence which has the highest likelihood:

$$Start \rightarrow \text{Wayfare number} \rightarrow \text{Wayfare name} \rightarrow \text{Wayfare type} \rightarrow \text{Locality name} \rightarrow \text{Postcode} \rightarrow \text{Territory} \rightarrow End.$$

The tokens in this address will therefore be assigned to the following output fields:

| | |
|---|---|
| Wayfare number: | 32 |
| Wayfare name: | Garden |
| Wayfare type: | Place |
| Locality name: | Brisbane |
| Territory: | Queensland |
| Postcode: | 7014 |

When generating models for segmentation of address and other types of data, a major issue in many application domains is how to collect appropriate training data. Such training data need to be of high quality and be broad enough to cover the diversity of input values that likely occur in the attributes that are to be segmented.

One possible approach is to bootstrap the training process by manually cleaning, tokenising and segmenting a small number of input values and assigning each tag to its most likely output field (such as the examples shown in Fig. 3.6), to then use this small set of training data to train a first segmentation model (such as a first rough HMM), and to then use this first model to segment a larger number of input values [76, 230]. This second set of segmented input values will likely contain too many wrongly segmented values, and careful manual inspection and correction of these input values is required. Once done, a second training set of segmented input values is available that can be used to train a second HMM. This second HMM will likely be more accurate than the first one. This process of segmenting input values using a HMM, correcting the wrongly segmented values, and using the new set of segmented input values to train a more accurate HMM can be repeated until a HMM

of satisfactory quality is available. This approach has shown to be much less time consuming compared to the manual generation of hand-crafted rules [76].

An alternative approach is to use cleaned and segmented input values that are available either in a reference databases or from earlier segmentation of the same types of data [5, 65]. The important aspect with this approach is that these data are of high quality and contain a large diversity of attribute values, such that the trained segmentation model is robust with regard to different unknown input addresses. For addresses, such reference databases can either be obtained from national postal service or they are already available in a database or data warehouse of an organisation. The structure and attributes of these segmented addresses needs to be the same as the desired structure of the addresses that are to be segmented. Having access to such a reference database allows an automatic learning process of segmentation models which can provide fully automated address standardisation [65].

## 3.8 Practical Considerations and Research Issues

An important initial activity in any data matching or deduplication project must be the assessment of the quality of the data that are to be matched. Known as data exploration or data profiling, this task can be achieved through a variety of tools that are either integrated in a data matching software, are external standalone programs, or are part of larger data processing, analysis or data warehousing systems.

At a minimum, for each attribute that will be used for matching, the number of different attribute values and their frequency distribution, the type of values in an attribute (such as string, number, date, etc.), as well as the number of records that have an empty value in an attribute should be known. This information is relevant when attributes are selected to be part of blocking keys during the indexing step (as will be discussed in the following chapter), and when appropriate comparison functions are chosen in the comparison step (as was covered in Chap. 5).

Many data cleaning and standardisation techniques rely heavily upon look-up tables. These tables contain, for example, personal names and their variations and common misspellings, or suburb, town, or state names, and postcodes from a certain country. To achieve cleaned and standardised data that are of high quality, it is important that these look-up tables are carefully customised according to where the data to be matched are sourced from. This does not just hold for names used in addresses, but also for given names and surnames which often have different spelling variations in different countries (even for example within English speaking countries). While such customisation of look-up tables will initially be a time consuming and labour intensive process, in the end the effort will be worthwhile because of the improved matching quality that can be achieved. Further on, the cleaned and standardised data will likely be useful for other applications within an organisation as well. Besides look-up tables, both the rules in rule-based segmentation systems and the training data for statistical segmentation systems also need to be customised to the data that are being matched or deduplicated.

What type of data cleaning and standardisation approach to use depends both upon the quality of the raw input data, and the amount of resources (with regard to labour, funding, and computing power) that is available for a given data matching or deduplication project. A further practical consideration is if a matching or deduplication exercise on a certain set of data is a one-off project or if it is likely that the data will be reused for future data matching projects. In the latter case it is worth to invest more efforts into data pre-processing than in the first case, especially if the matched data are used as an authoritative data repository (such as a master patient index database) for different applications within an organisation, and any new data will be matched with this authoritative data repository.

While data quality has been recognised as a massive problem that costs many organisations large amounts in lost revenue and wasted resources, the amount of research in the area of data pre-processing (cleaning and standardisation) is surprisingly low. One reason for this might be that data pre-processing is a very domain specific task that involves significant amounts of domain expertise and manual customisation and intervention. How to automate data pre-processing techniques with the aim to reduce manual efforts will be a valuable research undertaking.

Another interesting research direction will be to investigate how well different data pre-processing techniques are able to improve the outcomes of matching or deduplicating different types of data, and if there is a way to identify an optimal approach to how data pre-processing should be applied. This question can only be considered in combination with a specific data matching technique employed. Still, a large comparative investigation of different data cleaning and standardisation techniques applied on databases of different quality and with different characteristics would lead to a much improved understanding of how data pre-processing affects the outcomes of a data matching or deduplication exercise.

## 3.9 Further Reading

There is a large body of work available on the topic of data quality, addressing the many issues and challenges involved in this topic from different angles. Batini and Scannapieco [19] provide a detailed discussion of concepts, methodologies and techniques that can be employed to assess and improve data quality. Pyle [218] covers data quality and data preprocessing specifically for data mining applications. Lee et al. [177] on the other hand provide a road map to data quality that covers this topic at a less technical level more suitable for managers and practitioners that need to implement systems where data quality is important.

The many different issues that can arise when dealing with names have been discussed by various authors [40, 57, 72, 175, 208, 210, 243]. A large body of information about names is also available in online resources that cover names and their origins, names and their variations, and the changing popularity of baby names.

The interested reader is referred to Web sites such as: http://www.thinkbabynames.com, http://www.babynames.com, http://www.rogerdarlington.co.uk/useofnames.html, and http://en.wikipedia.org/wiki/Personal_name.

An excellent recent survey of information extraction techniques which is of relevance to name and address segmentation is provided by Sarawagi [230]. Techniques that specifically deal with data cleaning and standardisation for data matching are presented by Churches et al. [76] and by Herzog et al. [143]. The use of reference databases to automate the standardisation process of addresses has been described by Agichtein and Ganti [5] and Christen and Belagic [65].

Two novel approaches to data cleaning have recently proposed by Arasu and Kaushik [12] who used a grammar-based framework that can be used to reason about and manipulate data representations, and Guo et al. [130] who employed latent semantic association to conduct unsupervised address standardisation.