Automated Probabilistic Address Standardisation and Verification

http://datamining.anu.edu.au/linkage.html

Peter Christen[⋆] and Daniel Belacic

Department of Computer Science,
Australian National University,
Canberra ACT 0200, Australia,
{peter.christen,daniel.belacic}@anu.edu.au

Abstract. Addresses are a key part of many records containing information about people and organisations, and it is therefore important that accurate address information is available before such data is mined or stored in data warehouses. Unfortunately, addresses are often captured in non-standard and free-text formats, usually with some degree of spelling and typographical errors. Additionally, addresses change over time, for example when people move, when streets are renamed, or when new suburbs are built. Cleaning and standardising addresses, as well as verifying if they really exist, are therefore important steps in data mining pre-processing. In this paper we present an automated probabilistic approach based on a hidden Markov model (HMM), which uses national address guidelines and a comprehensive national address database to clean, standardise and verify raw input addresses. Initial experiments show that our system can correctly standardise even complex and unusual addresses.

Keywords: Data mining pre-processing, address cleaning and standardisation, hidden Markov model, G-NAF, postal address guidelines.

1 Introduction

Most real world data collections contain noisy, incomplete, incorrectly formatted, or even out-of-date data. Cleaning and standardising such data are therefore important first steps in data pre-processing, and before such data can be stored in data warehouses or used for further data analysis or mining [11, 16]. In most settings it is desirable to be able to detect and remove duplicate records from a data set, in order to reduce costs for business mailings or to improve the accuracy of a data analysis task. The cleaning and standardisation of personal information (like addresses and names) is especially important for data linkage and integration, to make sure that no misleading or redundant information is introduced. Data linkage (also called record linkage) [10] is important in many

[⋆] Corresponding author

application areas, such as compilation of longitudinal epidemiological studies, census related statistics [19], or fraud and crime detection systems.

The main tasks of data cleaning [16] are the conversion of the raw input data into well defined, consistent forms, and the resolution of inconsistencies in the way information is represented or encoded. Personal information is often captured and stored with typographical and phonetical variations, parts can be missing or recorded in different (possibly obsolete) formats, or be out-of-order. Addresses and names can change over time, and are often reported differently by the same person depending upon the organisation they are in contact with. Moreover, while for many regular words there is only one correct spelling, there are often different written forms for proper names (which are commonly used as street, locality or institution names), for example *'Dickson'* and *'Dixon'*. For addresses to be useful and valuable, they need to be cleaned and standardised into a well defined format. For example, various abbreviations should be converted into standardised forms, nicknames should be expanded into their full names, and postcodes should be validated using official postcode lists.

In this paper we report on a project that aims to develop techniques for fully automated cleaning, standardisation, as well as verification, of raw input addresses. In Section 2 we introduce the task of address cleaning and standardisation in more detail and present other work that has been done in this area. While traditional approaches have been based on either rules that need to be customised by the user according to her or his data, or manually prepared training data, our system is based on a mainly unsupervised approach. The main contribution of our work is the automated training of a probabilistic address standardisation system using national address guidelines and a comprehensive national address database. We present our approach in Section 3, and discuss the methods used to automatically train our system in Section 4. First experimental results are then presented and discussed in Section 5, and an outlook to future work is given in Section 6.

2 Address Cleaning and Standardisation

The aim of the cleaning and standardisation process is to transform the raw input address records into a well defined and consistent form, as shown in Figure 1. Addresses can be separated into three components, corresponding to the *address site* (containing flat and street number details), *street* (containing street name and type), and *locality* (with locality, state and postcode information). As can be seen from Figure 1, these components are further split into several output fields, each containing a basic piece of information. The standardisation process also replaces different spellings and abbreviations with standard versions. Look-up tables of such standard spellings are often published by national postal services, together with guidelines of how addresses should be written properly on letters or parcels. This information can be used to build an automated address standardiser, as presented in more details in Sections 3 and 4.

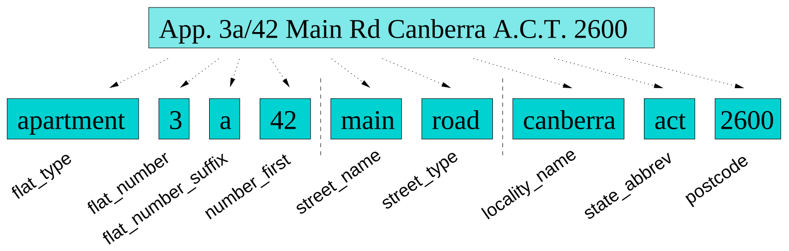

Fig. 1. Example address standardisation. The left four output fields relate to the *address site* level, the middle two to *street level*, and the right three fields to *locality level*

The terms data cleaning (or data cleansing), data standardisation, data scrubbing, data pre-processing, and ETL (extraction, transformation and loading) are used synonymously to refer to the general tasks of transforming source data into clean and consistent sets of records suitable for loading into a data warehouse, or for linking with other data sets. A number of commercial software products are available which address this task. A complete review is beyond the scope of this paper (an overview can be found in [16]). Address (and name) standardisation is also closely related to the more general problem of extracting structured data, such as bibliographic references or name entities, from unstructured or variably structured texts, such as scientific papers or Web pages.

The most common approach for address standardisation is the manual specification of parsing and transformation rules. A well-known example of this approach in biomedical research is *AutoStan* [12], the companion product to the widely-used *AutoMatch* probabilistic record linkage software. *AutoStan* first parses the input string into individual words, and using a re-entrant regular expression parser each word is then mapped to a token of a particular class (determined by the presence of that word in user-supplied look-up tables, or by the type of characters found in the word). This approach requires both an initial and ongoing investment in rule programming by skilled staff. More recent rule-based approaches, which aim at automatically induce rules for information extraction from unstructured text, include *Rapier* [5], which is based on inductive logic programming; *Whisk* [18], which can handle both free and highly structured text; and *Nodose* [1], which is an interactive graphical tool for determining the structure of text documents and for extracting their data.

An alternative to these rule-based, deterministic approaches are probabilistic methods. Statistical models, especially hidden Markov models (HMMs), have widely been used in the areas of speech recognition and natural language processing to help solve problems such as word-sense disambiguation and part-of-speech tagging [15]. More recently, HMMs and related models have been applied to the problem of extracting structured information from unstructured text. An approach using HMMs to find names and other non-recursive entities in free text is described in [3], where word features are used similar to the ones implemented in our system, and experimental results of high accuracy are presented using both English and Spanish test data. HMMs are also used for information extraction by [9], which addresses the problem of lack of training data by applying the statistical techniques of *shrinkage* to improve HMM parameter estimations (different hierarchies of expected similarities are built from a model). The issue of learning the structure of HMMs for information extraction is discussed in [17], where both labelled and un-labelled data is used, and good accuracy results are presented. A supervised approach for segmenting text (including US and Indian addresses) is presented by [4]. Their system *Datamold* uses hierarchical features and nested HMMs, and does allow the integration of external hierarchical databases for improved segmentation. Their results indicate that *Datamold* consistently performs better than the rule-base system *Rapier*. An automatic system that only uses external databases is presented in [2]. The authors describe *attribute recognition models* (ARMs), based on HMMs, which capture the characteristics of the values stored in large reference tables. The topology for an ARM consists of the three states *Beginning*, *Middle*, and *Trailing*. Feature hierarchies are then used to learn the HMM topology as well as transition and emission probabilities. Results presented on various data sets show an up to 50% reduction in segmentation errors compared to *Datamold*.

Earlier work [8] by one of the authors of this paper describes a supervised name and address standardisation approach that uses a lexicon-based tokenisation in combination with HMMs, work that was strongly influenced by [4]. Instead of directly using the elements of the input records for HMM segmentation, a tagging step allocates one or more tags (based on user definable look-up tables and some hard coded rules) to each input element, and sequences of tags are then given to a previously trained (using manually prepared tag sequences) HMM. Results on real world administrative health data showed better accuracy than the rule-based system *AutoStan* for addresses [8]. Training of this system is facilitated by a boot-strapping approach, allowing a reasonable amount of training data to be manually created within a couple of hours.

In this paper we present work which is mainly based on [2] and [8]. The main contribution of our work is the combination of techniques used in these two approaches, with specific application (but not limited) to Australian postal addresses. We use national address guidelines and a large national address database to automatically train a HMM, without the need of any manual preparation of training data. Our system is part of a free, open source data linkage system known as *Febrl* (Freely extensible biomedical record linkage) [6], which is written in the free, open source object-oriented programming language *Python*.

3 Probabilistic Address Standardisation

Our method is based on a probabilistic HMM which is automatically trained using information taken from national address guidelines (which are available in many countries) as well as a comprehensive national address database. The detailed approach on how this HMM is trained using these two sources is discussed in Section 4. Here we present the actual steps involved in the standardisation of raw input addresses, assuming such a trained HMM is available.

We assume that the raw input address records are stored as text files or database tables, and are made of one or more text strings. The task is then to allocate the words and numbers from the raw input into the appropriate output fields, to clean and standardise the values in these output fields, and to verify if an address (or parts of it) really exist (i.e. is available in the national address database). Our approach is based on the following four steps, which will be discussed in more detail in the four sections given below.

1. The raw input addresses are *cleaned*.
2. They are each split into a list of words, numbers and characters, which are then *tagged* using features and look-up tables that were generated using the national address database.
3. These tagged lists are then *segmented* into output fields using a probabilistic HMM.
4. Finally, the segmented addresses are *verified* using the national address database.

3.1 Cleaning

The cleaning step involves converting all letters into lower case, followed by various general corrections of sub-strings using correction lists. These lists are stored in text files that can be modified by the user. For example, variations of *nursing home*, such as `‘n-home’` or `‘n/home’` are all replaced with the string `‘nursing home’`. Various kinds of brackets and quoting characters are replaced with a vertical bar `‘|’`, which facilitates tagging and segmenting in the subsequent steps. Correction lists also allow the definition of strings that are to be removed from the input, for example `‘n/a’` or `‘locked’`. The output of this first step is a cleaned address string ready to be tagged in the next step.

3.2 Tagging

After an address string has been cleaned, it is split at white-space boundaries into a list of words, numbers, punctuation marks and other possible characters. Each of the list elements is assigned one or more *tags*. These tags are based on look-up tables generated using the values in a national address database, as well as more general *features*. For example, a list element `‘road’` is assigned the tag `‘ST’` (for street type, as `‘road’` was found in the street type attribute in the database), as well as the tag `‘L4’` (as it is a value of length four characters containing only letters). The tagging does not depend upon the position of a value in the list. The number `‘2371’`, for example, will be tagged with `‘PC’` (as it is a known postcode) and `‘N4’` (as it is also a four digit number), even if it appears at the beginning of an address (where it likely corresponds to a street number). The segmentation step (described below) then assigns this element to the appropriate output field.

Table 1. Example values from the national address database for features used for standardisation. Empty table entries indicate no such values are available in the database

Length	Numbers	Letters	Alpha-numeric	Others
1	3	a		.
2	42	se	b1	.,
3	127	lot	33a	1/7
4	1642	road	672a	3/1a
5	13576	place	lot12	1/23b
6 to 8	2230229	street	rmb1622	lot 1760
9 to 11		jindabyne	coleville2	anderson's
12 to 15		dondingalong	bundanoon305	house no: 2/41
16 or more		stonequarrycreek		armidale-kempsey

Look-up tags specify to the HMM in which attribute(s) of the national address database a list element appears. If it appears in several attributes, more than one look-up tag will be assigned to it. However, if a list element in an input address contains a typographical error, or does otherwise not exactly correspond to any look-up table value, no tag would be assigned to it. Therefore, the features are a more general way of representing the content of the different attributes in the national address database. Features characterise the lengths of an attribute value, as well as its content (if it is made of letters only, numbers only, if it is alpha-numeric, or if it also contains other characters). For example, an attribute value that only contains letters and has a length between 12 and 15 (feature tag `'L12_15'`) is in 73% a locality name, in 26% a street name, and in 1% a building name, as this is the distribution of values with letters only and a length between 12 and 15 in the national address database. A feature tag `'N6_8'`, as another example, corresponds to a number value with length between 6 and 8 digits. Table 1 gives example attribute values from the national address database.

In the tagging step, the look-up tables are searched using a *greedy* matching algorithm, which searches for the longest tuple of list elements that match an entry in the look-up tables. For example, the tuple `('macquarie','fields')` will be matched with an entry in a look-up table with the locality name `'macquarie fields'`, rather than with the single-word entry `'macquarie'` from the same look-up table.

The output of the tagging step is a list of words, numbers and separators, and a corresponding list of look-up and feature tags (as shown in the example given below). As more than one tag can be assigned to a list element (as in the street type example above), different combinations of tag sequences are possible, and the questions are which tag sequence is the most likely one, and how should the list elements be assigned to the appropriate output fields. This problem is solved using a probabilistic HMM in the segmentation step as discussed next.

3.3 Segmenting

Having a list of elements (words, numbers and separators) and one or more corresponding tag lists, the task is to assign these elements to the correct output fields. Traditional approaches have used rules (such as *"if an element has a tag 'ST' then the corresponding word is assigned to the '`street_type`' output field."*). Instead, we use a HMM [15], which has the advantages of being robust with respect to previously unseen input sequences, and that it can be automatically trained, as will be detailed in Section 4.

Hidden Markov models [15] (HMMs) were developed in the 1960s and 1970s and are widely used in speech and natural language processing. They are a powerful machine learning technique, able to handle new forms of data in a robust fashion. They are computationally efficient to develop and evaluate. Only recently have HMMs been used for address standardisation [4, 8, 17].

A HMM is a probabilistic finite state machine made of a set of states, transition edges between these states and a finite dictionary of discrete observation (output) symbols. Each edge is associated with a transition probability, and each state emits observation symbols from the dictionary with a certain probability distribution. Two special states are the *'Start'* and *'End'* state. Beginning from the *'Start'* state, a HMM generates a sequence of length k of observation 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. Observation symbol $o_i, 1 \leq i \leq k$ is generated in state i based on this state's probability distribution of the observation symbols. The same output sequence can be generated by many different paths through a HMM with different probabilities. Given an observation sequence, one is often interested in the most likely path through a given HMM that generated this sequence. This path can effectively be calculated for a given observation sequence using the *Viterbi* [15] algorithm, which is a dynamic programming approach. Figure 3 shows a HMM generated by our system for address standardisation.

Instead of using the original words, numbers and other elements from the address records directly, the tag sequences (as discussed in Section 3.2) are used as HMM observation symbols in order to make the derived HMM more general and more robust. Using tags also limits the size of the observation dictionary. Once a HMM is trained, sequences of tags (one tag per input element) as generated in the tagging step can be given as input to the *Viterbi* algorithm, which returns the most likely path (i.e. state sequence) of the given tag sequence through the HMM, plus the corresponding probability. The path with the highest probability is then taken and the corresponding state sequence will be used to assign the elements of the input list to the appropriate output fields.

Example: Let's assume we have the following (randomly created) input address `'42 meyer Rd COOMA 2371'`, which is cleaned and tagged (using both look-up and feature tags) into the following word list and tag sequence:

```
['42', 'meyer', 'road',  'cooma',    '2371' ]
['N2', 'SN/L5', 'ST/L4', 'LN/SN/L5', 'PC/N4' ]
```

with look-up tags `'SN'` for street name, `'ST'` for street type, `'LN'` for locality name, and `'PC'` for postcode; and feature tags for numbers (`'N2'` and `'N4'`) and letter values (`'L4'` and `'L5'`). The number of combinations of the tag sequences is $1 \times 2 \times 2 \times 3 \times 2 = 24$, for example `['N2', 'SN', 'ST', 'LN', 'PC']` or `['N2', 'L5', 'ST', 'SN', 'N4']`. These 24 tag sequences are given to the *Viterbi* algorithm, and using the HMM from Figure 3, the tag sequence with the highest probability that is returned is `['N2', 'SN', 'ST', 'LN', 'PC']`. It corresponds to the following path through the HMM (with the corresponding observation symbols – the output fields – in brackets).

```
Start → number_first (N2) → street_name (SN) → street_type (ST)
      → locality_name (LN) → postcode (PC) → End
```

The values of the input address will be assigned to the output fields as follows.

```
 number_first: '42'
  street_name: 'meyer'
  street_type: 'road'
locality_name: 'cooma'
     postcode: '2371'
```

3.4 Verification

Once segmented an input address can be easily compared to the existing addresses in the national address database. Different techniques can be used for this task, for example inverted indices as described in [7], which allow approximate matching (for example if parts of an address are missing or wrong). Alternatively, hash encodings (like *MD5* or *SHA*) can be used to create a unique *signature* for each address in the national database, allowing to efficiently compare a hash encoded input address with the full database. Similarly, hash encodings of the locality and street (and their combinations) allow the verification of only these parts of an address. This component of our system is currently under development, and more details will be published elsewhere.

4 Automated Hidden Markov Model Training

The automated HMM training approach is based on national address guidelines and a large national address database, and only needs minimal initial manual efforts. Guidelines for correctly addressing letters and parcels are increasingly becoming important as mail is being processed (sorted and distributed) automatically. Many national postal services therefore publish such guidelines[1]. Our system uses these guidelines to build the initial HMM structure, as shown in Figure 2. This is currently done manually, but in the future it is likely that electronic versions of such guidelines (for example as XML schemes) will become available, making the initial manual building of the HMM structure automated

[1] See for example: `http://www.auspost.com.au/correctaddress`

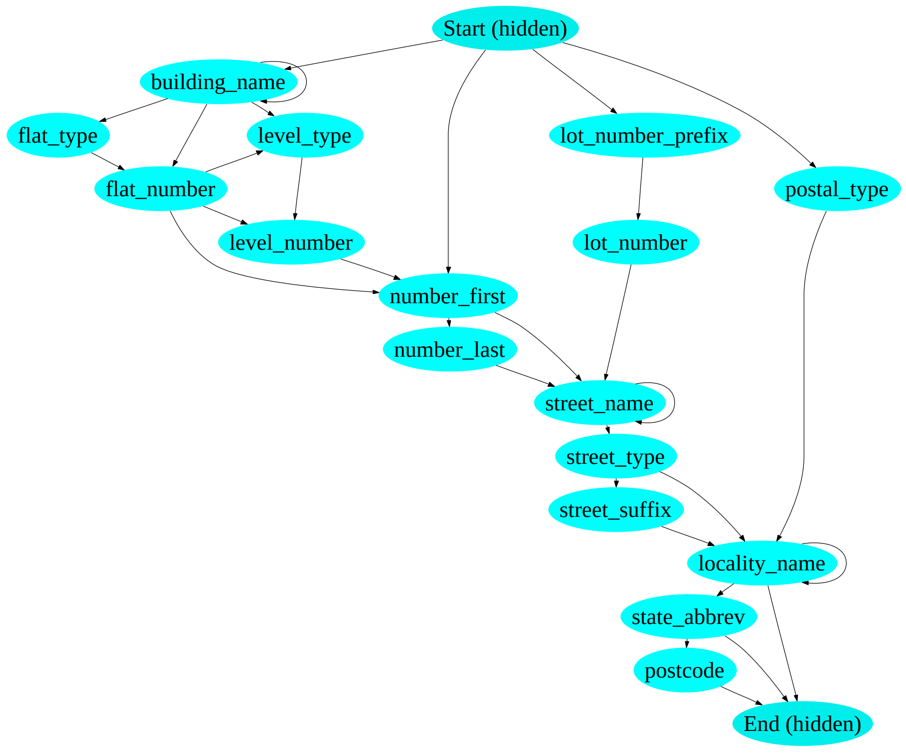

Fig. 2. Initial HMM topology manually constructed from postal address guidelines to support the automated HMM training

as well. The structure is built with the national address database in mind, i.e. the HMM states correspond to the database attributes, and aims to facilitate the automated training process which uses the clean and segmented records in such an address database.

A comprehensive, parcel based national address database has recently become available in Australia: *G-NAF* (the Geocoded National Address File) [13]. Developed mainly for geocoding applications in mind, approximately 32 million address records from several organisations were used in a five-phase cleaning and integration process, resulting in a database consisting of 22 normalised tables. G-NAF is based on a hierarchical model, which stores information about address sites (properties) separately from streets and locations [14]. For our purpose, we extracted 26 address attributes (or output fields) as listed in Table 2. The aim of the standardisation process is to assign each element of a raw user input address to one of these 26 output fields, as shown in the example in Figure 1. Only the G-NAF records covering the Australian state of New South Wales (NSW) were available to us, in total $4,585,707$ addresses. There are two main steps in the set-up and training phase of our address standardisation system as follows.

Table 2. G-NAF address attributes (or fields) used in the standardisation process

	G-NAF fields
Address site	flat_number_prefix, flat_number, flat_number_suffix, flat_type, level_number_prefix, level_number, level_number_suffix, level_type, building_name, location_description, private_road, number_first_prefix, number_first, number_first_suffix, number_last_prefix, number_last, number_last_suffix, lot_number_prefix, lot_number, lot_number_suffix
Street	street_name, street_type, street_suffix
Locality	locality_name, state_abbrev, postcode

4.1 Generation of Look-up Tables

The look-up tables are generated by extracting all the discrete (string) values for locality_name, street_name and building_name into tables and then combining those tables with manually generated tables containing typographical variations (like common misspellings of suburb names), as well as the complete listing of postcodes and locality names from the national postal services. Other look-up tables are generated using the official G-NAF data dictionary tables (for fields such as street_type, street_suffix, flat_type, or level_type). The resulting look-up tables are then cleaned using the same approach as described in Section 3.1, and used in the tagging step to assign look-up tags to address elements.

4.2 HMM Training

The required input data for the training are (1) the initial HMM structure as built using the postal address guidelines and as shown in Figure 2, and (2) the G-NAF database containing cleaned and segmented address records. The distribution of both transition and observation probabilities are learned based on frequency counts of the occurrences of attribute values in the G-NAF database. Each G-NAF record is an example path and observation sequence. Due to minor deficiencies in the data contained in G-NAF, such as the lack of postal addresses, postcodes, or the character slash '/' (which is often used to separate flat from street numbers), manually added tweaks must be automatically applied where appropriate to the model during training to account for the lack of observations and transitions, and to account for unusual but legitimate address types, such as corner addresses. A HMM trained using G-NAF is shown in Figure 3. Because training data often does not cover all possible combinations of transitions and observations, during application of a HMM unseen and unknown data is encountered. To be able to deal with such cases, *smoothing* techniques [4] (such as *Laplace* or *absolute discount* smoothing) need to be applied, which enable unseen data to be handled more efficiently. These techniques basically assign small probabilities to all unseen transitions and observations symbols in all states.

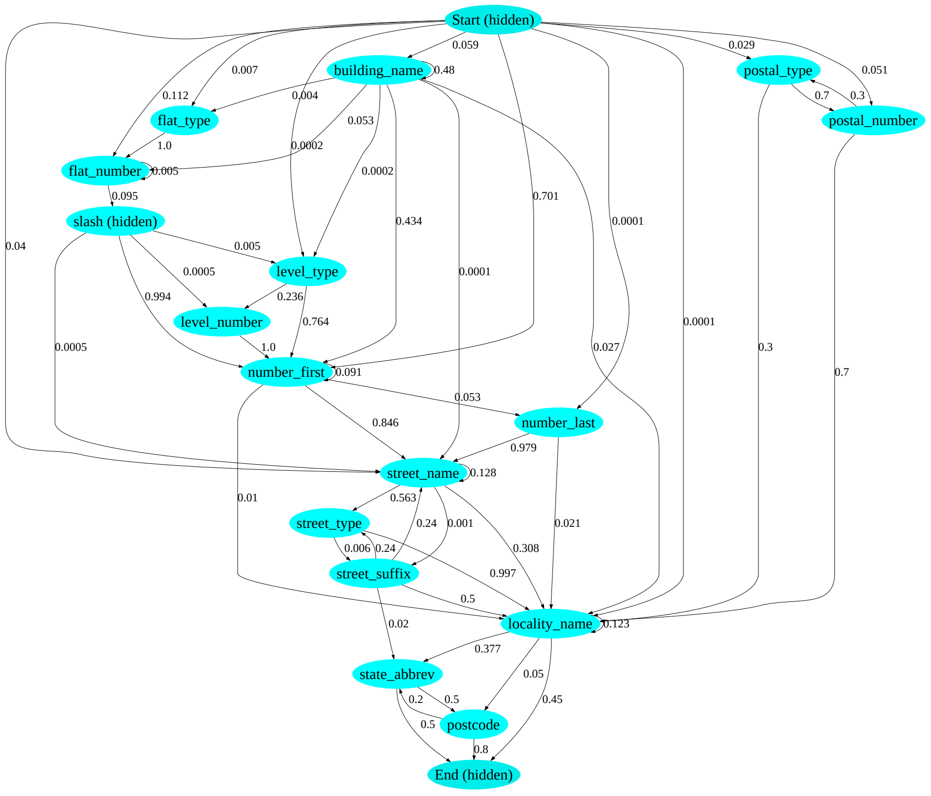

Fig. 3. HMM (simplified) after automated training using the G-NAF national address database (but before smoothing is applied)

5 Experimental Results and Discussion

Special care must be taken when evaluating HMM based systems. If the records used to train a HMM are from the same or similar data set as the records used to evaluate the performance of the same HMM, the model may become over-fitted to the training data and may not accurately reflect the real performance of the HMM. To test the accuracy of our probabilistic standardisation approach raw addresses from three data sets were used. The first contained 500 records with addresses taken from a midwives data collection, the second 600 nursing home addresses, and the third a 150 record sample of unusual and difficult addresses from a large administrative health data set. There are three major variations possible in our system for standardising addresses:

1. **Features and look-up tables (F&LT)**
   During the tagging step of standardisation, each element in the address is assigned one or more tags depending if it can be found in one or more look-

up tables. Once all tables have been checked, the element will also be given a feature tag as described in Section 3.2. However, elements of one character length are only given a feature tag and look-up tables are not searched.

2. **Look-up tables only (LT)**
   This is similar to the supervised system [8] as previously implemented in *Febrl* [6]. An address element is given one or more look-up tags, depending if it can be found in the look-up tables. If it is not assigned any tags, it is given a feature tag. Again, elements of one character length are only given feature tags.
3. **Features only (F)**
   Single address elements are only given feature tags and look-up tables are not used. Any sequence the greedy matching algorithm finds of length two or more elements is assigned a tag from the look-up tables as normal. Unlike the other two options, elements were not placed into their canonical form, since there is no look-up table used to check for original forms.

While HMM's were trained using all three options of smoothing (*no smoothing*, *absolute discount*, and *Laplace*), *no smoothing* was not tested as it is deemed to be highly inflexible and unable to cope with unseen input data. *Laplace* smoothing was tested, but not analysed extensively as initial tests showed a quite poor performance. All results, unless specified, are therefore assumed to be from a HMM with *absolute discount* smoothing applied. Comparison test were also performed using the supervised *Febrl* address standardiser [6, 8].

Records were judged to be accurately standardised if all elements of an input address string were placed into the correct output fields. It was not appropriate to check for correct canonical correction, since feature based tagging will not transform any words. Addresses not fully correct were judged on an individual basis for level of correctness, either *'close'* or *'not close'*, depending upon the criticality of the error. For example, numbers being classified as `number_last` instead of `number_first` were considered *'close'*, whereas street types being judged localities are considered *'not close'*. A second measure of accuracy, called *'could be accuracy'*, was used to show the level of accuracy of the HMM when including *'close'* (but incorrectly standardised) records as correct.

In many data sets the majority of input addresses are of fairly simple structure. We therefore counted the frequency of the following three sequences and included their numbers (labelled as *'Easy addresses'*) in the results Table 3.

```
(number_first,number_last,street_name,street_type,locality_name,postcode)
(number_first,street_name,street_type,locality_name,postcode)
(street_name,street_type,locality_name,postcode)
```

As expected, the data set with unusual addresses contained much less easy addresses, while for the other two data sets around 90% were easy addresses.

Performance was averaged over 10 runs of the system for each category of execution. All standardisation runs were performed on a moderately loaded Intel Pentium M Centrino 2.0 GHz with 512 MBytes of RAM.

Table 3. Experimental accuracy and standardisation timing results on three test data sets using *absolute discount* HMM smoothing. See text for discussion what *easy addresses* are

	Midwives	Nursing homes	Unusual
Total number of addresses	500	600	150
Easy addresses **(F&LT)**	446	542	31
Easy addresses **(LT)**	438	538	27
Easy addresses **(F)**	445	542	31
Easy addresses *Febrl*	410	529	22
Accuracy **(F&LT)**	97.40%	96.67%	92.67%
Accuracy **(LT)**	95.40%	98.50%	72.67%
Accuracy **(F)**	96.60%	92.67%	79.33%
Accuracy *Febrl*	96.80%	96.00%	96.00%
'Could be' accuracy **(F&LT)**	98.00%	97.80%	94.67%
'Could be' accuracy **(LT)**	97.40%	98.50%	80.00%
'Could be' accuracy **(F)**	97.00%	96.50%	80.67%
'Could be' accuracy *Febrl*	97.60%	98.30%	96.00%
Milli-seconds per record **(F&LT)**	92	445	720
Milli-seconds per record **(LT)**	11	18	37
Milli-seconds per record **(F)**	6	7	7
Milli-seconds per record *Febrl*	7	9	10

5.1 Discussion

As can be seen by the difference between actual accuracy and *'could be'* accuracy in Table 3, not only is the accuracy of the new system quite high, especially when using the **(F&LT)** variation, but quite a large number of the incorrect records were only marginally incorrect in non-critical parts of an address. Perhaps half of the remaining errors were caused by a known deficiency in the greedy tagging system, which has to do with the value `'st'` being a known abbreviation both for *'Saint'* and *'Street'*. Most remaining errors were examined in depth, but in general it was impossible even for a human to determine the exact correct output. Accuracy using our automatically trained system versus a manually trained *Febrl* HMM is equal to or better than in all cases tested. Quite surprisingly, accuracy using the **(F)** HMM was quite comparable to the **(LT)** based HMM.

Also, the *Febrl* address HMM failed on almost all non NSW addresses given, due to them generally being outside the scope of its look-up tables, thus the tagging was ineffective. However the **(F&LT)** and **(F)** HMM's both successfully standardised most non NSW addresses by using the feature information where the look-up tables came up blank. This has promising possibilities for using the HMM to standardise addresses outside the domain of G-NAF without any retraining necessary. There are also possible applications where licensing or other reasons are non permissive for distribution of the G-NAF national address database and corresponding look-up tables generated.

Timing performance using the **(F&LT)** HMM is relatively poor due to the large number of possible combinations of tag sequences, however still quite acceptable, especially since accuracy is generally more highly valued than time taken, and the fact that addresses can be easily standardised in parallel.

6 Outlook and Future Work

In this paper we have presented an automated approach to address cleaning and standardisation based on national postal address guidelines and a comprehensive national address database (G-NAF), and using a probabilistic hidden Markov model (HMM) which can be trained without manual interaction. Standardising addresses is not only an important first step before address data can be loaded into databases or data warehouses, or be used for data mining, but it is also necessary before address data can be linked or integrated with other data.

There are still various improvements possible to our system. Currently corner addresses are implicitly supported, but explicitly creating HMM states such as a second street name and type is a more complete solution. Characters such as dash, brackets, commas, etc. are currently processed in the cleaning step, but handling them in the HMM could improve accuracy. Other minor improvements include training the HMM using corrected G-NAF data, and ways to minimise the number and size of manual tweaks to the HMM. The look-up tables contain some common typographical error correction data, drawn from manually created lists. It should be possible to build far more comprehensive lists automatically by matching between the G-NAF address data and correctly standardised example addresses, in order to find typographical variations.

Each distinct tag sequence given to the HMM will always have the same output states and *Viterbi* probability. This can be used to advantage by *caching* the set of input tags and the resulting probability during execution. Since up to 90% of addresses in some data sets have the same output fields, it is highly likely that there will be a considerable number of addresses with the same tag sequence. These redundant calculations can be eliminated by checking the tag sequence against a cache of sequences. If found in the cache, directly return the probability, otherwise the sequence will be run through the HMM and the resulting probability and input tags will be added to the cache. Using the **(F&LT)** variation, addresses can have dozens of possible tag sequences, thus the caching of results should give considerable performance improvements.

While developed with and using Australian address data, our approach can easily be modified to other countries, or even other domains (for examples names, medical data, etc.) as long as standardisation guidelines and a comprehensive database with standardised records are available.

Acknowledgements

This work is supported by an Australian Research Council (ARC) Linkage Grant LP0453463 and partially funded by the NSW Department of Health.

References

1. Adelberg, B: Nodose: a tool for semi-automatically extracting structured and semistructured data from text documents. In proceedings of ACM SIGMOD International Conference on Management of Data, New York, pp. 283–294, 1998.
2. Agichtein, E. and Ganti, V.: Mining reference tables for automatic text segmentation. In proceedings of the ACM SIGKDD'04, Seattle, pp. 20–29, August 2004.
3. Bikel, D.M., Miller, S., Schwartz, R. and Weischedel, R.: Nymble: a high-performance learning name-finder. In proceedings of ANLP-97, Haverfordwest, Wales, UK, Association for Neuro-Linguistic Programming, pp. 194–201, 1997.
4. Borkar, V., Deshmukh, K. and Sarawagi, S.: Automatic segmentation of text into structured records. In proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, California, 2001.
5. Califf, M.E. and Mooney, R.J.: Relational learning of pattern-match rules for information extraction. In proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), Menlo Park, CA, pp. 328–334, 1999.
6. Christen, P., Churches, T. and Hegland, M.: A Parallel Open Source Data Linkage System. Proceedings of the 8th PAKDD'04 (Pacific-Asia Conference on Knowledge Discovery and Data Mining), Sydney. Springer LNAI-3056, pp. 638–647, May 2004.
7. Christen, P., Churches, T. and Willmore, A.: A Probabilistic Geocoding System based on a National Address File. Proceedings of the 3rd Australasian Data Mining Conference, Cairns, December 2004.
8. Churches, T., Christen, P., Lim, K. and Zhu, J.X.: Preparation of name and address data for record linkage using hidden Markov models. BioMed Central Medical Informatics and Decision Making 2002, 2:9, Dec. 2002. Available online at: `http://www.biomedcentral.com/1472-6947/2/9/`
9. Freitag, D. and McCallum, A.: Information extraction using HMMs and shrinkage. In papers from the AAAI-99 Workshop on Machine Learning for Information Extraction, Menlo Park, CA, pp. 31–36, 1999.
10. Gill, L: Methods for Automatic Record Matching and Linking and their use in National Statistics. National Statistics Methodology Series No. 25, London 2001.
11. Han, J. and Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
12. AutoStan and AutoMatch, User's Manuals. MatchWare Technologies, 1998.
13. Paull, D.L.: A geocoded National Address File for Australia: The G-NAF What, Why, Who and When? PSMA Australia Limited, Griffith, ACT, Australia, 2003. Available online at: `http://www.g-naf.com.au/`
14. Paull, D.L. and Marwick, B.: Understanding G-NAF. Proceedings of SSC'2005 (Spatial Intelligence, Innovation and Praxis), Spatial Sciences Institute, Melbourne, September 2005.
15. Rabiner, L.R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, vol. 77, no. 2, Feb. 1989.
16. Rahm, E. and Do, H.H.: Data Cleaning: Problems and Current Approaches. IEEE Data Engineering Bulletin, 2000.
17. Seymore, K., McCallum, A. and Rosenfeld, R.: Learning Hidden Markov Model Structure for Information Extraction. In proceedings of AAAI-99, workshop on Machine Learning for Information Extraction, 1999.
18. Soderland, S: Learning information extraction rules for semi-structured and free text. Machine Learning, vol. 34, no. 1–3, pp. 233–272, February 1999.
19. Winkler, W.E.: The State of Record Linkage and Current Research Problems. RR99/03, US Bureau of the Census, 1999.