Documentation for Address Matching project

Peter G. Hufton
Data Analytics and Statistics division, MHCLG

February 8, 2019

Ministry of Housing,
Communities &
Local Government

Addresses are fallible. Among a multitude of problems, they are subject to differences in formatting, vulnerable to typing and encoding errors, and may change over time. Still, it is a fact that addresses form a necessary reference when considering a property or building. At present, the data records in the data sources at MHCLG are almost always identified by an address. Using an address as a reference introduces many problems, especially when attempting to align data from different sources in order to perform complex analyses. The address-matching project attempts to overcome the limitations of addresses, by connecting a poorly-formatted address to a corresponding Unique Property Reference Number (UPRN). This is achieved through a combination of rules-based and machine-learning approaches. In this way, we are able to map addresses to UPRNs with a high degree of reliability, and we show how this can be used to align data from different sources. In the future, we expect the address-matching project to facilitate complex analyses and play an important role in the day-to-day operations of the department.

1 Introduction

In February 2017, the government issued a white paper entitled “Fixing our broken housing market” [1]. The white paper begins by acknowledging current shortcomings in the housing market. There is a shortage of housing, the cost of housing is at a record high of eight times the average national income, and the proportion of people living in the private rental sector since 2000 has doubled. Consequently, the first two objectives in the MHCLG Single Departmental Plan were listed in May 2018 as “1. Delivering the homes the country needs,“ and “2. Making a vision of a place you call home a reality” [2]. The former refers to increasing the housing supply, through improving the productivity, competition and transparency in the housing market, while the latter concerns user access and experience, for example by reforming the private rented sector so that it is fair and more affordable. How, though, do we fix this broken housing market without a real understanding of the operations today’s housing market?

The success of data to understand our world—its people, its social trends, and its markets—as well as predicting and shaping its future, is perhaps the defining characteristic of the present century. Indeed, according to many we are in the opening chapters of the “Data Revolution” [3]. With the right analysis and intentions, data can be converted into value, both economic and social [4]. It is thus with firm-footing that we at MHCLG hold that to provide the best services possible—to make the best policy decisions, to best address shortcomings in the housing market—we require the best data available.

In MHCLG we have data from a large number of different sources, from both within the civil service as well as private companies; some tof these dtat sources are shown in Fig. 1. For example, there is data of Energy

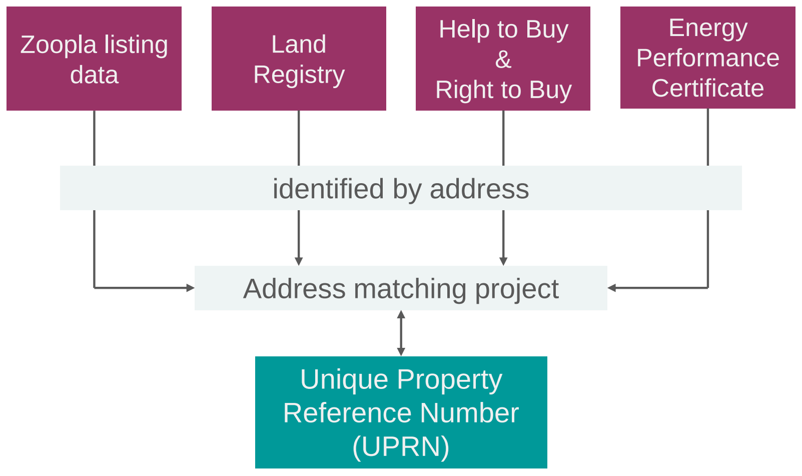

Zoopla listing data
Land Registry
Help to Buy & Right to Buy
Energy Performance Certificate
identified by address
Address matching project
Unique Property Reference Number (UPRN)

Figure 1: The address matching project aims to connect data sources, by first connecting them to a UPRN listed in AddressBase Premium.

Performance Certificates (EPCs), summarising the energy efficiency of buildings across the United Kingdom. We hold data on Help to Buy and Right to Buy schemes, the former being described as "the biggest government intervention in the housing market since [the latter]" [5]. We also have data from HM Land Registry, detailing the ownership of land and property in England and Wales. In terms of private companies, we currently receive data of listings from Zoopla–the UK's second most-visited property website [6]. Data may also be scraped from other locations on the internet. In short, we have a lot of data from a lot of different sources.

To analyse the housing market and support policy decisions, it is essential that these disparate data sources be aligned with each other. Having a lot of separate data is all well and good, but how can we find patterns in the data if these sources remain isolated? For example, data from Zoopla tells us when a property is listed as being sold, but how can we know if a sale was successful (or even if there are fraudulent shenanigans at work!) without consulting the data of HM Land Registry? Another example occurred during investigations into building safety following the Grenfell Tower fire: vital information about high-rise buildings was collected from Local Authorities, but other vital information about the buildings was elsewhere in the EPC data set.

Therein lies a problem. In all of these data sources, each property or dwelling is referenced by its address. Addresses, however, are fallible. Firstly, there is variation in the way an organisation (or indeed one particular civil servant) may choose to format an address. Who is to say that the way one an address from one source is formatted is the same as another? Consider these questions: Are the names of businesses included within an address? Is the postal county (gradually deprecated between 1996 and 2016 [7]) included in the address? Are the abbreviations "Road" as "Rd" and "Street" as "St" ever to be used? Do we include alternative Welsh-language street names? In which order? Uncertainty in the answers to these questions leads to a lot of variation in the formats of addresses. Secondly, addresses are subject to many other errors. Addresses are prone to spelling mistakes, errors or missing information in the data entry stage, or digital encoding errors. Addresses may also contain outdated information, for example where a property has been split into flats, or where a commercial unit has changed hands. Addresses with these problems are sometimes referred to as "fuzzy": they contain (hopefully) the necessary elements of an address, but are made fuzzy by sources of noise/variation.

It is therefore not a straightforward task to align the data in these sources. How can we see if two sources are talking about the same dwelling if their addresses are so different? The purpose of the Address Matching project is to connect these data sources through careful consideration of fuzzy addresses. The output of such a project is clear: a better, interlinked set of data. We expect such a development to allow us to move closer to an understanding of how the housing market operates in real-time. The project will support better predictive modelling of policy, as well as facilitating better monitoring of the direct consequences of changes in policy.

The Data Science team at MHCLG has developed an in-house solution to the problem of fuzzy addresses. In our solution, we identify AddressBase Premium as the gold standard of addresses in England, Wales and Scotland [8, 9]. AddressBase Premium, provided by Ordnance Survey, is an up to date and accurate database of the country's properties, using the same data used by Royal Mail. The details of over 40 million addresses are included therein. Each unique property is assigned a Unique Property Reference Number (UPRN), alongside its address. This UPRN provides a more robust reference than an address, since it is not subject to the same sources of noise. The aim of this project becomes to match a fuzzy address from one of our data sources to a "reference" address in AddressBase, and therefore find its UPRN. Once sources are connected to their UPRNs, they can be linked to other sources through this central reference.

The problem of matching addresses is by no means a new problem; the problem has been considered both outside the government [10, 11] and within [12–14]. Some methods include a step of "normalising" a given address: this step involves attempting to correct mistyped words (or "tokens") [10, 13]. In this way, the mistyped phrase "Barn Lne" may be corrected to "Barn Lane". One way of implementing such an approach involves computing the Levensthein distance between an unknown token (e.g., "Lne"), and a reference dictionary of known tokens. The Levenshtein distance is defined as the number of single character edits (insertions, deletions and substitutions) required to change one word into the other. For the example of "Lne" and "Lane", this is equal to one. While including such an approach allows mistyped tokens to be corrected and used for matching, it is not pursued in our present implementation of the address matching project. We find that such an approach is unnecessary for the kind of addresses we are receiving, and that a suitable accuracy of matching can be obtained without these steps. Furthermore, this step would increase the runtime of our process, perhaps by a very considerable amount.

Other methods involve the parsing of an unstructured input address strings into the components of an address [14]; such an approach is in development within the ONS [14]. In other words, components such as the business name, flat number, street name, and town will be identified as such. This might be achieved through a combination of rules-based approaches and natural-language processing. A simple form of these approaches is used in our current implementation of address-matching: namely, we extract the building numbers and postcode from a string, and these become our priority when searching for a match. More advanced natural language processing, however, is not used here. Again, our results indicate that suitable accuracy can be obtained without such an approach being taken, whereas including such methods would likely affect efficiency and take considerable time to implement.

In the context of MHCLG, there are a number of unique considerations concerning address matching. One such consideration is the volume of addresses which need to UPRNs. MHCLG has recently acquired access to data from the online real estate company Zoopla. Each month, the department receives an update to this data with a size around 120GB and typically averaging 18 million listings, each of which potentially needs matching to its corresponding UPRN. It is hence imperative that the departmental approach to address matching is scalable to such volumes of addresses. The size of the data involved, then, dictates that our solution must operate from within the department. In a different vein, in most cases there are limitations regarding the licensing/sensitivity of data which prevents us from being able to send the data outside of the department. For this reason, the solution to the problem of fuzzy addresses must be based in-house. It is also important that the solution is designed with the types of data we use in MHCLG in mind and is tested on our specific data sources. One consequence of this consideration will be the inclusion of addresses belonging to buildings, rather than individual properties. Lastly, we remark that the solution should be designed with the efficiency of our IT systems in mind.

In its present incarnation, the address-matching project developed by the data science team at MHCLG utilises a combination of rules-based and machine-learning approaches. Loosely speaking, rules-based approaches are used to select and narrow down a list of potential matches to (hopefully) a single match. This involves several layers of comparison of the building numbers, the words, and the postcodes in each potential match. Secondly, machine-learning approaches are used to assign a confidence rating to each supposed match: this is achieved by having trained a logistic regression model on a set of sample data. The confidence rating allows users to filter out or manually inspect poorly-matched addresses.

The implementation of address matching developed here was not started from scratch; rather, it was developed from an earlier solution provided to us by Home England. The reason we chose this solution was since, going forward, we are aiming to use the same data sources (shared through GDAP) and some of the services around this will also be shared.

This document details the approaches taken by the data science team at MHCLG in developing a solution to the problem of address matching. This project has gone through several iterations, each increasing in complexity and efficacy; in this document we will describe earlier approaches before ending at the most up-to-date procedure.

At the current stage of development, the address matching procedure is well-tooled to handle poorly-structured address strings which may have missing data, contain spelling mistakes, and abbreviations, and to do so on appropriate time scales to handle the volume of demand which exists within the department. We begin by presenting some summary statistics of the project in Sec. 2; in Sec. 4 we consider the earlier versions of the project. In Sec. 5 we present a brief discussion of Parent UPRNs. In Sec. 6 we consider the most recent version, while in Sec. 7 we compare the results of the new and old AddressMatching procedures. We review the ability of the most recent version of the address matching project to connect disparate data sources in Sec. 8, while our conclusion is Sec. 9. In the appendix we present the practical information for using the address matching stored procedure.

2 Summary statistics

The latest version of the address matching project was tested on samples of historic address string records from different sources. For this method, we found:

- a 92–99% rate of success for fetching a single UPRN for each input address string.
- 1–8% of input strings were mapped to multiple UPRNs; in these cases the input address strings were not suitably specific to refer to a single address.
- 0–1% of input strings were mapped to no UPRNs; in these cases no matching addresses were found in our AddressBase records.
- of the successful results, we found 0–4% were false positives.
- a confidence measure can be calculated by which questionable matches can be filtered or investigated.
- a typical runtime of 19–96 seconds per thousand addresses, depending on settings, input data, and the load on the server (ADD-Staging).

3 Address matching service

In this section we consider the practical information for matching batches of addresses. In order to match addresses, we require teams to send us (the Data Science team) an Excel sheet containing: (1) a unique identifier for each address, (2) the ‘fuzzy’ address string, and (3) a postcode for each address. In the former version (1.4), a postcode is needed to be supplied for each address for the Mode=“First Four” setting to function; in the upcoming version (1.5), this requirement will be removed. Thereafter, the process is as follows:

$$\begin{array}{c}\text{Excel sheet}\\\text{of ‘fuzzy’ addresses}\end{array} \rightarrow \text{MS SQL Server} \rightarrow \begin{array}{c}\text{AddressMatching}\\\text{stored procedure}\end{array} \rightarrow \begin{array}{c}\text{Excel sheet}\\\text{including UPRNs,}\\\text{Parent UPRNS, UDPRNs,}\\\text{and reference addresses}\end{array} \tag{3.1}$$

An Excel sheet is returned to the team containing the original information, along with the matched UPRNs, Parent UPRNs (if applicable), UDPRNs, and reference addresses. In some cases multiple matches may be returned for an individual input address.

4 Old Versions of AddressMatching

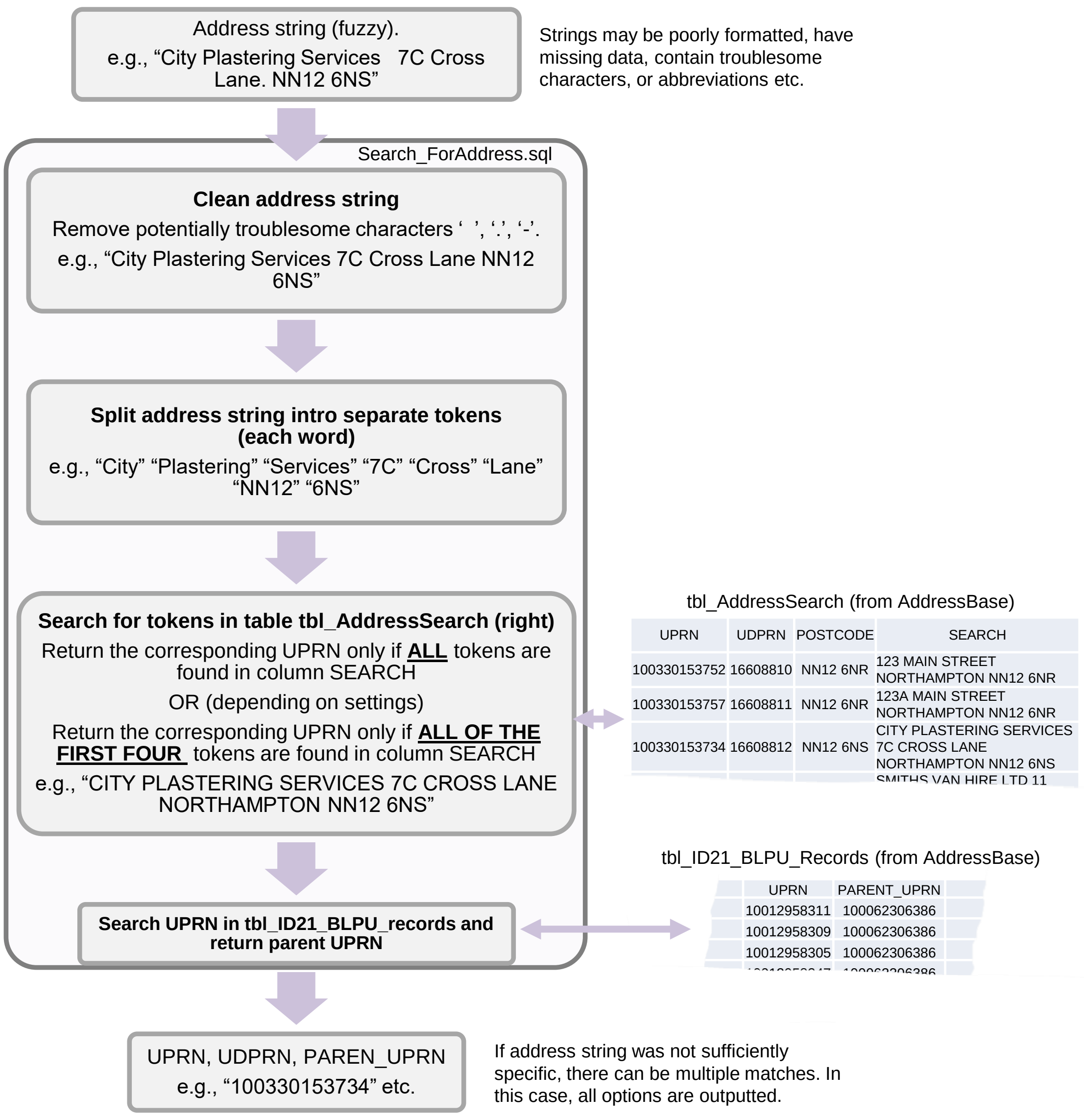
4.1 AddressMatching versions 1.0–1.4

To us, address matching describes the task of matching an unformatted text string—containing the necessary elements of an address—to the appropriate reference address recorded in AddressBase, and thereby its Unique Property Reference Number (UPRN). Once matched to a UPRN, the property may more easily be connected information such as geospatial data, or linked with information from other sources, such as energy certification and building safety.

The Data Science team has utilised an iterative approach to the problem of address matching. Each iteration (i.e., version) of the project builds on the previous by adding functionality and addressing new requirements as they become known to us. In developing out address matching process, we didn't start from scratch, but rather expanded on an earlier solution provided to us by Homes England. This chapter describes the version of the address-matching functionality which was previously shown to the English Housing Survey. This version has since been improved (but not yet fully quality controlled), and these improvements are discussed in Sec. 6.

As an input, our approach uses only a textual-data string provided as an address. In other words, it does not use any geospatial matching or other references. Our solution to the problem of address matching relies on AddressBase Premium from Ordnance Survery. The input string is matched to the closest match in AddressBase Premium. In some cases, the addresses in AddressBase Premium contain the name of businesses, but this is not always the case and it is not always current. The version of AddressBase records presently in use was updated on 4 April 2018; we recognise that this is not ideal as this may mean we do not have the latest versions of UPRNs and Addresses. This means that we have a higher number of wrong or no matches. Full details of the AddressBasePremium data supplied by OS can be found in Ref. [9].

The address-matching project was coded as an SQL stored procedure to be used through Microsoft SQL Server. The technical reason for this over other approaches is to best utilise the computational resources of ADD-staging, which have been designed with SQL Server in mind. This allows batches of address to be matched with few clicks through Microsoft SQL Server Studio.



Address string (fuzzy).
e.g., "City Plastering Services 7C Cross Lane. NN12 6NS"

Strings may be poorly formatted, have missing data, contain troublesome characters, or abbreviations etc.

Search_ForAddress.sql

Clean address string
Remove potentially troublesome characters ' ', '.', '-'.
e.g., "City Plastering Services 7C Cross Lane NN12 6NS"

Split address string intro separate tokens (each word)
e.g., "City" "Plastering" "Services" "7C" "Cross" "Lane" "NN12" "6NS"

Search for tokens in table tbl_AddressSearch (right)
Return the corresponding UPRN only if **ALL** tokens are found in column SEARCH
OR (depending on settings)
Return the corresponding UPRN only if **ALL OF THE FIRST FOUR** tokens are found in column SEARCH
e.g., "CITY PLASTERING SERVICES 7C CROSS LANE NORTHAMPTON NN12 6NS"

tbl_AddressSearch (from AddressBase)

UPRN	UDPRN	POSTCODE	SEARCH
100330153752	16608810	NN12 6NR	123 MAIN STREET NORTHAMPTON NN12 6NR
100330153757	16608811	NN12 6NR	123A MAIN STREET NORTHAMPTON NN12 6NR
100330153734	16608812	NN12 6NS	CITY PLASTERING SERVICES 7C CROSS LANE NORTHAMPTON NN12 6NS
			SMITHS VAN HIRE LTD 11

Search UPRN in tbl_ID21_BLPU_records and return parent UPRN

tbl_ID21_BLPU_Records (from AddressBase)

UPRN	PARENT_UPRN
10012958311	100062306386
10012958309	100062306386
10012958305	100062306386
[illegible]	[illegible]

UPRN, UDPRN, PAREN_UPRN
e.g., "100330153734" etc.

If address string was not sufficiently specific, there can be multiple matches. In this case, all options are outputted.

Figure 2: Flow chart describing versions 1.0–1.4.

AddressMatching versions 1.0–1.4 refer to separate stages of development for the same essential procedure.

The operation of the address-matching procedure is remarkably simple: the algorithm is little more than a "bag of words" search. Figure 2 shows the operation of the procedure schematically. Before the matching starts, addresses are cleaned of all potentially troublesome characters/phrases:

- Punctuation: `( ) \ / - & ‘ ’ , . _ ; : ? ! * # + @`
- SQL Keywords: `AND`, `OR`, `NOT`.

After cleaning, the address data is split into words (i.e., *tokens)*. If the input address contains numbers or a postcode, those will be similarly treated as words. The stored procedure then searches against a reference table `tbl_AddressSearch`, which contains a list of UPRNs and their corresponding reference addresses; this table was extracted from AddressBase Premium. The order in which the words appear is unimportant. We then search for all records in AddressBase Premium that contain either some or all of the words.

There are two modes of operation address-matching process, depending on the settings the user chooses upon execution: mode = "All" or "First Four".

- In the first case, the algorithm requires that **all** tokens in the input search string are found in the reference address from AddressBase.
- In the second case, the algorithm requires only the **first four words** and the **postcode** match the reference address. This is a less strict condition, which returns more matches (but also increases the chances of a false-positive match). This allows results to be obtained where the first case fails. The choice of the first four words was inspire by tests conducted in the past, which determined that the most useful address data (e.g., building numbers and names) was contained in this part of the address. This functionality was added in version 1.4.

Once an address is matched, the reference address is outputted along with its UPRN, and Unique Delivery Point Reference Number (UDPRN). If relevant, the property's Parent UPRN (i.e., the reference number associated with the building) will also be returned (see Sec. 5 for a discussion of Parent URPNs). In both cases, if the input string is not suitably specific, multiple matching UPRNs may be found; in this case, all matching UPRNs are outputted.

Limitations. These versions of the procedure have a few serious limitations:

1. For the first setting, mode="All", the requirement that **all** tokens need to be found in the reference table brings about a serious limitation in the efficacy of the procedure, in which additional information prevents a match from occurring. Consider the examples:

Input address string	Reference address	Matched?
`456 Fake Road, Town, County PS7 6DE`	`456 Fake Road, Town PS7 6DE`	✗
`Business Name LTD 123 Fake Street, Town PS7 6DE`	`123 Fake Street, Town PS7 6DE`	✗

(4.1)

In these cases, the algorithm will fail to match the input string to the reference address, since not all tokens are matched. Ideally, over-specification should not present a problem.

2. The setting Mode="First Four" attempts to overcome limitation 1 (above) by considering a string's first four tokens and its postcode only. This requires an input postcode to be supplied alongside the input address string for each record. This works well for many examples of residential properties, e.g., returning to the examples in table (4.1):

Input address string	Reference address	Matched?
`456 Fake Road, Town,` ~~`County`~~ `PS7 6DE`	`456 Fake Road, Town PS7 6DE`	✓
`Business Name LTD 123` ~~`Fake Street, Town`~~ `PS7 6DE`	`123 Fake Street, Town PS7 6DE`	✗

(4.2)

we see that for the first example, the correct address is now matched. This is typically the case for residential addresses. For other examples, however, this approached fails (see the second example above). This is especially the case with commercial addresses. For certain test sets, this approach performs worse (i.e., there are fewer single-result matches) than the "All" setting. It follows that a more robust approach is needed to overcome these limitations.

3. Another limitation concerns addresses with multiple identical tokens. Consider the example:

Input address string	Reference address
`York House, Fake Street, York YO1 6DE`	`Bradford House, Fake Street, York YO1 6DE`
	`Leeds House, Fake Street, York YO1 6DE`
	`York House, Fake Street, York YO1 6DE`

(4.3)

   In this example, notice that the token "York" is repeated. In this case, the algorithm would return all three results (even though there is one obvious best match), since all three addresses on the left-hand side contain all the input's tokens.

4. This approach does nothing to match address strings with typing errors and other mistakes.

5. This approach is not good with house or flat numbers as they are considered among the words.

6. This approach does not identify addresses which may be best matched to parent UPRNs. Many addresses of entire buildings return no results (see Sec. 5 for discussion).

7. This approach does not allow us to estimate the confidence that a given match is correct.

5 Parent UPRNs

Strictly speaking, each UPRN refers to a single dwelling. This may typically be a house which constitutes a single residency, or else a single apartment within a larger building. In the latter case, the building itself is typically described by a **Parent UPRN**. Parent UPRNs are linked to a list of children UPRNs contained therein. Parent UPRNs also occur when a single building has been split into flats, in which case the original buildings UPRN is promoted to a Parent UPRN. Further details on the lifecycle of the UPRN can be found in Ref. [15]. A typical parent UPRN may have the address "`CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`", while its children may have addresses "`FLAT 1 CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`", "`FLAT 2 CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`", etc.

The department has use cases that require both mapping to child and parent UPRNs independently. For example, analyses of building safety may require the user to consider a building as a whole (rather than individual apartments) by using a Parent UPRN. This requirement was in fact discovered as a direct result of discussion with the Building Safety Analysis team.

Unfortunately, while AddressBase contains a reference address for each UPRN, Parent UPRNs do not have a stored reference address. This makes searching for Parent UPRNs difficult with AddressBase. As a result of this, the above version of the address-matching solution (version 1.4) searches for child UPRNs only; attempts to search using the address of an entire building will often not return any results. This is improved in version 1.5.

6 Current version of AddressMatching (version 1.5)

The current version of the AddressMatching stored procedure—version 1.5—attempts to overcome the shortcomings of previous approaches by better accounting for fuzzy matches. In order to do this, we identify the **building number(s)** and **postcode** as the most important indicator of a residency's address. In brief, the algorithm begins by searching the reference table `tbl_AddressSearch` for addresses with matching building numbers and postcodes. Of these results, it compares tokens between the input and reference string in order to decide the best match. Depending on the number of matching tokens, the algorithm assigns a confidence rating to a given match. These steps are outlined schematically in Fig. 3; further details are given below.

Clean address string. As before, the address string is cleaned by removing troublesome punctuation. In addition, we now also remove former postal counties, e.g., 'Lancashire' and 'Lancs'. While formerly part of the address, these terms have been deprecated and do not appear in the AddressBase records; eighty such terms are removed (see Appendix C for a full list). In some cases, these terms may form a valuable part of an address, for example a property might be name 'Devon House'. To avoid complications arising from this, postal-county

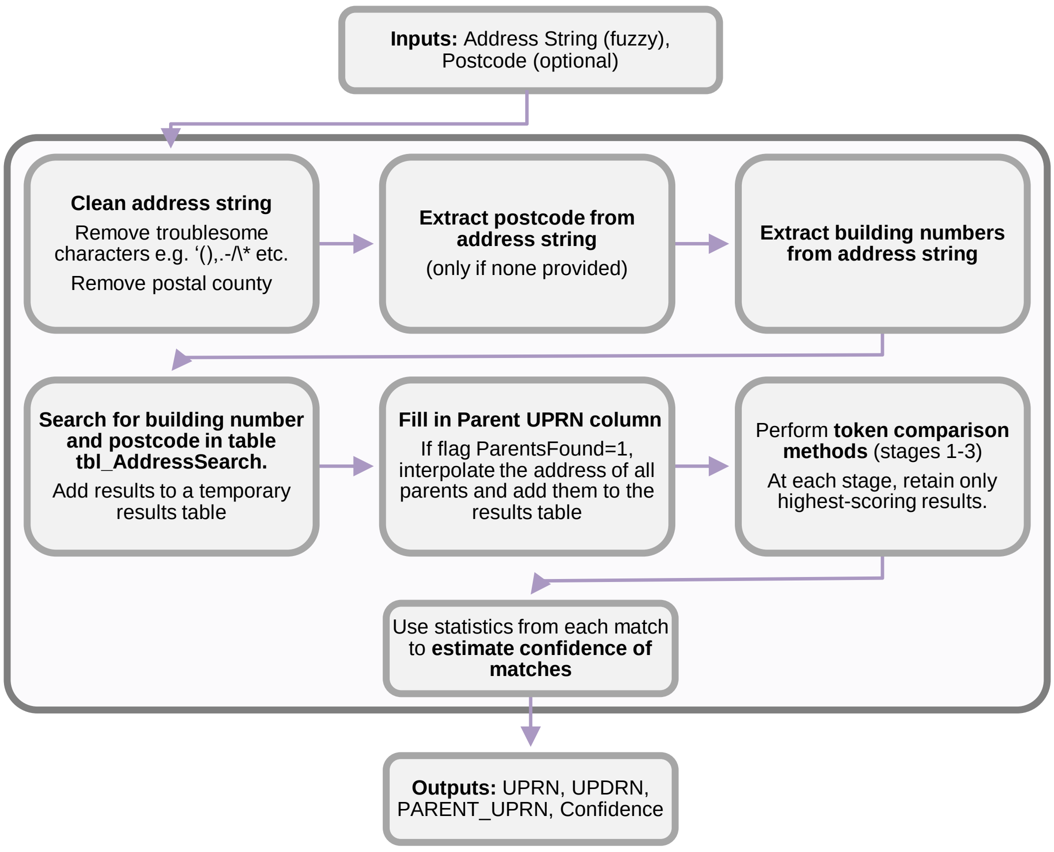

Inputs: Address String (fuzzy), Postcode (optional)
Clean address string
Remove troublesome characters e.g. '(),.-/* etc.
Remove postal county
Extract postcode from address string
(only if none provided)
Extract building numbers from address string
Search for building number and postcode in table tbl_AddressSearch.
Add results to a temporary results table
Fill in Parent UPRN column
If flag ParentsFound=1, interpolate the address of all parents and add them to the results table
Perform token comparison methods (stages 1-3)
At each stage, retain only highest-scoring results.
Use statistics from each match to estimate confidence of matches
Outputs: UPRN, UPDRN, PARENT_UPRN, Confidence

Figure 3: Flow chart describing versions 1.5.

terms are removed only if they appear after the first 16 characters. The address cleaning stage has also been moved to its own function.

Extract postcode and building numbers form address string This is done by pattern matching. For example, terms satisfying patterns "`XX99 9XX`" and similar are identified as postcodes, while terms like "`99`" and "`99X`" are identified as building numbers. For strings with multiple building numbers, all building numbers are collected in this way.

"Stage 0": Search for matching building numbers and postcode. In this stage, we begin to search the AddressBase records for suitable potential matches. If both a building number and a postcode are found in the input string, we narrow down the number of potential matches to reference addresses including these details. We thereby reduce the number of results to a handful of potential matches. This allows us to do more complicated token comparison without introducing computational burden.

If a property contains multiple building numbers, the procedure looks for a match where the numbers appear in the same order within the string. This allows the procedure to distinguish between two potential matches `Flat 1 Building 2` and `Flat 2 Building 1`.

If a reference with matching building number and post code cannot be found, the results are narrowed down by the postcode only. (Results without matching numbers are quite harshly punished later on when estimating confidence!).

The addresses of Parent UPRNs may be added at this stage, depending on the settings chosen. Details are found in Sec, 6.1.

"Stage 1": Token comparison 1. In the following three stages, we identify which of the potential matches is the best fit by performing token-comparison methods. We begin by determining the intersection of the input search string and the reference address of each potential match. This is the set of all the tokens which appear in both strings. Importantly, before comparing tokens, we also clean the **reference address** using the same procedure as for the input string. This introduces consistency and ensures that unwanted punctuation etc. does not affect our results. We assume the most likely match will have the largest intersection, so we discard all other results.

This count ignores tokens which are repeated in one string and not the other. This can be seen to be important for a number of cases. Consider the below, for example

Input address string	Reference address	#
`York House, Fake Street, York YO1 6DE`	`Bradford House, Fake Street, York YO1 6DE`	6
	`Leeds House, Fake Street, York YO1 6DE`	6
	`York House, Fake Street, York YO1 6DE`	7

(6.1)

we see that the correct result scores higher than the others. We therefore overcome limitation 2 discussed in Sec. 4.

"Stage 2": Token comparison 2. We ask how many of the tokens in the input search string are found in each potential match; the most likely match will have the most in common with the input string, so we discard all other results. This score differs from stage 1 in how it deals with repeated tokens. A lot of addresses (those scraped from the internet, for example) contain repeated tokens, for example "123 Fake Street, Fake Street...": The token comparison at this stage considers all of these tokens, and allows us to further filter down our list of potential matches. This score also provides a measure to evaluate the correctness of a match which we will use in estimating our confidence of a match (see Sec. 6.2)

"Stage 3": Token comparison 3. Vice versa, we ask how many of the tokens in the reference address are found in the input address. We retain only the match(es) with the highest **percentage** matching. By filtering the results in this way, we are narrowing down the strings to those where the reference address contains (1) the highest number of tokens found in the input address and (2) the fewest tokens in addition to this. Thus, reference address which are much longer (i.e., more specific) than the input address are discarded in favour of those more similar in length, resulting in a better quality match. As above, this score also provides another measure to evaluate the quality of the match.

6.1 Including Parent UPRNs

The address-matching procedure has also been expanded to include Parent UPRNs, in the event that the address string best matches a building rather than a single property. Although the address of a Parent UPRN is not provided in AddressBase, it may effectively be *inferred* from its children. A function was made to infer a Parent UPRN's address in this way. It essentially creates an ordered list of all tokens found in the children addresses, for each token which appears in the majority (i.e., over 50%) of the children. This function removes certain tokens which carry no information: `FLAT`, `APARTMENT`, `UNIT`, and `STUDIO`. Consider an example which contains children "`FLAT 1 CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`", "`FLAT 2 CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`", etc. The parent address returned is "`CHANCELLORS COURT, ORDER HALL ST, WC1N 3JP`".

The Parent UPRNs and their addresses are included into the above address-matching procedure as follows. At Stage 0, the Parent UPRNs of all the results from the 'BuildingNumber + Postcode" search are obtained. Using the above method, their addresses are then inferred, and they are then appended to the list of results at Stage 0. Thereafter, these parent address are subject to the same procedures as normal UPRNs. Hence, if the Parent UPRN is a better fit for the input address, it is returned as the output.

Including parent UPRNs in this way expands the use of the address-matching procedure to consider whole buildings. It is worth noting, however, that it effects the runtime of the program, typically requiring twice or three times as long to run.

6.2 Estimating confidence

After matching an address, we wish to estimate the probability that the match is correct. This estimation of confidence provides a way to filter out poorly-matched addresses, or to otherwise manually inspect such matches. To this end, we want a mathematical function which maps numerical information about a given match to a probability. Logistic regression is a commonly-used statistical technique for inferring such a function.

We proceed with a rigorous implementation of logistic regression to the problem of address matching, utilising machine-learning methods of model-selection and optimisation. Initially, a number of variables (i.e., *features*) were chosen that may effect the likelihood that a match is correct. A full list of features considered is given in Table 1.

Name	Description
TokensIn	The number of tokens in the cleaned input address
TokensOut	The number of tokens in the cleaned reference address
TokensIntersect	The number of tokens in both the input and reference address (no repeats)
TokenMatched1	The number of tokens in the input address which appear in the reference address
TokenMatched2	The number of tokens in the reference address which appear in the input address
IsExactMatch	Whether the cleaned input address is equal to the cleaned reference address
IsParent	Whether the matched UPRN is a parent UPRN
BuildingNumFound	Whether (1) numerical tokens are found in input address, and (2) all these numerical tokens appear in reference address
BuildingNumExact	Whether the same exact, ordered list of building numbers is found in bouth input and reference address
LenBuildingNum1	Length (number of numerical tokens) of the building number in input address
LenBuildingNum2	Length (number of numerical tokens) of the building number in reference address
LenBuildingNumIntersect	Length of intersection of building numbers from input and reference addresses
NumsNotInIntersect1	Number of numerical tokens in input string not in intersection
NumsNotInIntersect2	Number of numerical tokens in reference string not in intersection
NumMatchesStage0	The number of matches at stage 0 of the address matching algorithm
NumMatchesStage1	The number of matches at stage 1 of the address matching algorithm
NumMatchesStage2	The number of matches at stage 2 of the address matching algorithm
BinMatchesStage0	Binarised version of NumMatchesStage0: whether NumMatchesStage0 is above 1
BinMatchesStage1	Binarised version of NumMatchesStage1: whether NumMatchesStage1 is above 1
BinMatchesStage2	Binarised version of NumMatchesStage2: whether NumMatchesStage2 is above 1
PercentNum1	% of numbers in input matched: LenBuildingNumIntersect/LenBuildingNum1
PercentNum2	% of numbers in reference matched: LenBuildingNumIntersect/LenBuildingNum2
PercentIntersect	% of tokens in interesection: TokensIntersect/TokensIn
PercentMatched1	% of tokens in input matched: TokensMatched1/TokensIn
PercentMatched2	% of tokens in reference matched: TokensMatched2/TokensOut

Table 1: List of features considered in logistic regression

Figure 4 provides a visualisation of the basic principle of logistic regression, obtained after performing logistic regression to the address-matching procedure. In this figure, the high-dimensional space has been projected onto two components (through a linear change of basis); the details of this projection are not important (but are chosen to optimally project the differences in correct vs incorrect). The idea of logistic regression is to best divide correct and incorrect matches in a high-dimensional space. The solid dividing line shows the boundary, above which points (i.e., matches) are more likely to be correct (circles) and below which they are more likely to be incorrect (crosses).

Labelling. We begin by labelling a sample set of data. This involves running the address-matching algorithm as above over a batch of sample data, and outputting each of the variables in Table 1 for each match. Each match is then graded by hand (i.e., the input address string and reference string were compared) to say whether the match was correct (1) or incorrect (0). In the language of logistic regression, the variables of Table 1 form the *dependent variables*, while the grading constitutes a *response*. The labelling process itself is infamously time-consuming and laborious, but alas necessary.

It is important that the sample set of data is suitably representative of addresses which will be addressed when the address-matching procedure is in production. For this reason, samples were taken from various data sources: 5545 matches from EPC data, 4682 matches from Zoopla data, and 2119 matches from random historical searches were all labelled, totalling N=12346 matches. Of these matches, 315 were labelled as incorrect. This is consistent with the "rule of ten" principle, which states that we require a minimum number of ten occurrences (of each type) per dependent variable. Only matches in which a single address was returned were considered: those which returned no matches or multiple matches were discarded (and are not counted in N).

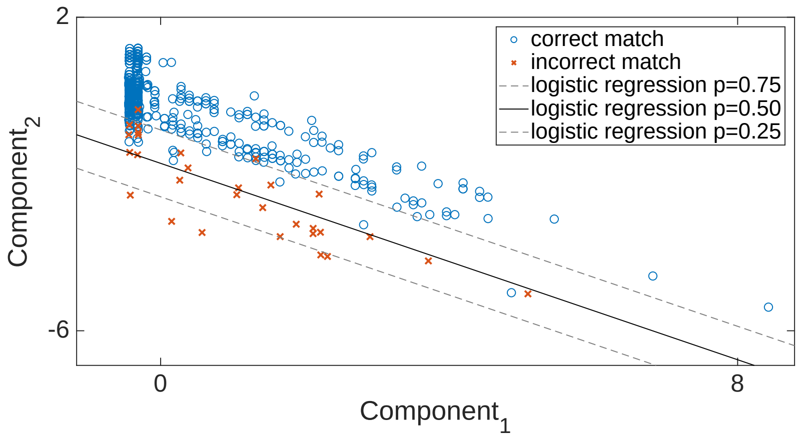


Figure 4: Visualisation of logistic regression as performed for the address-matching procedure, projected onto two dimensions. The aim is to best divide correct and incorrect matches in a high-dimensional space.

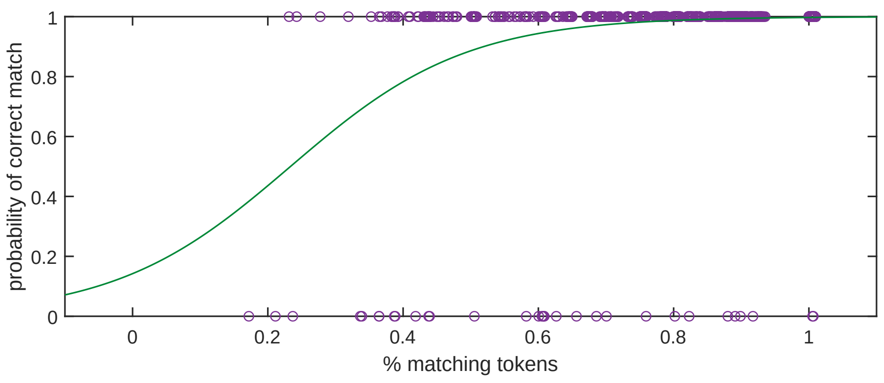


Figure 5: Logistic regression in one dimension for illustrative purposes. Circles represent individual address matches which were correct (1) or incorrect (0). The probability of a success or a failure is mapped to a sigmoid function. A better fit can be obtained by including more dependent variables.

Logistic regression in one dimension. The idea of logistic regression is to fit a sigmoid function to predict the probability of a given response (in our case whether a match is correct or incorrect) based on a set of dependent variables. This is easiest to visualise in one dimension; Fig. 5 shows the result in one-dimension for illustrative purposes, where we consider only the percentage of tokens matched. The functional form of the mapping is given by

$$p(\text{correct}) = \frac{1}{1+\exp[-(\beta_0+\beta_1 x)]} \tag{6.2}$$

where x is the dependent variable and β_i are parameters to be determined by fitting.

***m*-dimensional logistic regression.** Logistic regression is more commonly performed for a larger number of features. To parametrise an *m*-dimensional sigmoid function, various fitting methods can be used. These typically used gradient descent to minimise a distance measure. These fitting methods can be performed without much trouble using Python packages. Specifically, we use the `LogisticRegression` function from the `sklearn.linear_model` module. We used the commands:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='liblinear', penalty='l1').fit(X, y)
```

where `X` is a *N*×*m* numpy array of our dependent variables, and `y` is a *N*×1 numpy array of responses.

High dimensional logistic regression and model selection. Table 1 contains a total of m=25 features, so logistic regression in our case can amount to fitting a 25-dimensional sigmoid function. However, it is likely that many of these features (when used in combination) are not good predictors of the quality of a given match. In a similar vein, including too many of these features also increases the probability of overfitting to the sample data, so that the model becomes less predictive of unseen datasets. Finally, increasing the dimensionality of the sigmoid increases the probability that the fitting algorithm utilised by `sklearn` fails to converge.

Each combination of features in itself constitutes its own logistic model: since we have m=25 features, there is a total of 2^{25} possible logistic models to choose from. A rigorous approach of model selection and validation must be implemented to avoid the aforementioned problems and determine an effective model from these possibilities. To ensure that the model is optimised and the features are chosen to be statistically significant, we utilise (1) cross-validation to evaluate the goodness-of-fit of a given model, and (2) forward selection of features:

Evaluating the goodness-of-fit of a given model. Before implementing the `sklearn` logistic regression function, the dataset is split into two parts: a training set and a testing set. This is done at a ratio of 90% to 10%, as is typical. The logistic model is trained on the former set using the `sklearn` logistic regression function, and the parameters for logistic regression are obtained.

After the model has been trained on the training set, we need to evaluate the goodness-of-fit of the model. This is done by analysing the ability of the model to predict the responses of the as-yet-unseen testing set. Separating the data in this way removes the problem of overfitting to a single dataset. There are several measures of the goodness-of-fit of a logistic model. These measures are all closely related to the likelihood-ratio test, which broadly asks how likely the response is given the prescribed model. Many of these measures are referred to as pseudo-R^2, since they function in an analogous way to the squared multiple correlation R^2 in linear regression. Here, we use the Nagelkerke $R^2{}_\mathrm{N}$ measure, which provides a distance between 0 and 1, with 0 being no better than random and 1 indicating a perfect fit. This measure is evaluated for the testing data.

Cross validation refers to the repetition of the above, while the training data sets and testing data sets are permutated: the fraction which constitutes the testing data is rotated, so after all repetitions all data has constituted the training data exactly once. In our case, this amounts to ten repetitions. The pseudo-R^2 is computed at each repetition, and finally averaged over all repetitions. This provides a distance measure which is more robust to statistical variations.

Additionally, in our case we also reproduced the above 100 times, shuffling the order of datapoints between each iteration. The pseudo-R^2 was averaged over these 100 repetitions. This gives us further confidence on the statistical significance of our results.

Forward selection. The forward selection of features involves building the model up one feature at a time. We begin by asking which feature is singularly the most useful. To determine this, the above logistic regression technique is applied using one variable at a time, and the average pseudo-R^2 measure is determined using the cross-validation procedure described above. Whichever feature provides the highest pseudo-R^2 measure is added to the model. Forward selection then proceeds to ask which of the remaining features is best used in addition to this feature, again by calculating a pseudo-R^2 distance for each remaining feature. The process is iterated until all features are included.

The results of forward selection are shown in Table 2. The table provides an ordered-list of when each feature was added in the forward-selection process, along with the Pseudo-R^2 measure at each stage. Initially, we see that including additional features increases the goodness-of-fit of the model (i.e., we can better identify good and bad matches). The features best suited to this are somewhat unsurprising: buildings with incorrect numbers and few matching tokens are not likely to be good matches (similarly, the least useful features are also somewhat unsurprising: TokensIn and TokensOut tell us very little information about the match itself). The pseudo-R^2 reaches a maximum after 14 features are included: this point marks the optimum cut-off point. After this point, the model's efficacy decreases as it falls victim to overfitting etc.

Backward selection. The forward selection approach outlined above provides a methodical way of optimising our choice of model; a means of further optimisation may be obtained by also considering backward selection. In other words, we ask whether the model can be further improved by removing any of the features. Using these two techniques in tandem is known as mixed selection. Starting with the forward-optimised model, a test of backward selection on the above model showed that an improvement can be achieved by removing the predictor "PercentNum2" from the model. In this case, we achieve a Pseudo-R^2 measure of 0.64763. Additional testing showed no further improvement by additional or removal of any individual feature. Thus, it follows that we can be confident that our model constitutes at the very least a local optimum (where results cannot be improved by addition or removal of any single feature). We henceforth use the above 13 (those in Table 2) features in our logistic model. The parameters of the logistic function are provided in the rightmost column.

It is not an easy task to interpret the parameters of logistic regression. There is a high degree of correlation between the features considered in our case. For example, TokensIntersect and TokensMatched2 are quite

Name	Pseudo-R^2 measure		Included?	Coefficients β
PercentNum1	0.27209	More useful feature	✓	4.56617636
PercentMatched1	0.43363		✓	5.69875062
PercentNum2	0.53239		✗	
BinMatchesStage1	0.58809		✓	1.70818323
PercentMatched2	0.61636		✓	3.73172005
BuildingNumFound	0.63391		✓	2.18266788
IsExactMatch	0.63843		✓	2.08826112
TokensIntersect	0.64097		✓	0.46681653
LenBuildingNum2	0.64326		✓	-1.86369782
IsParent	0.64462		✓	-1.04432127
LenBuildingNum1	0.64567		✓	0.82199287
BuildingNumExact	0.64637		✓	0.73439667
TokensMatched2	0.64674		✓	-0.19557523
NumMatchesStage0	0.64685	$\leftarrow$ Optimal cut-off	✓	0.00697187
NumsNotInIntersect1	0.64680		✗	Intercept: -13.0343033
PercentIntersect	0.64655		✗	
BinMatchesStage0	0.64608		✗	
NumMatchesStage2	0.64546		✗	
NumMatchesStage1	0.64714		✗	
LenBuildingNumIntersect	0.64556		✗	
BinMatchesStage2	0.64486		✗	
NumsNotInIntersect2	0.64323		✗	
TokensMatched1	0.64195		✗	
TokensIn	0.63901		✗	
TokensOut	0.63633	Less useful feature	✗	

Table 2: Ordered list of features as they were added to the logistic regression model using the forward-selection method. Shown is the pseudo-R^2 measure of goodness of fit for the logistic regression after including all features up to and including the present (specifically, we have used the Nagelkerke $R^2{}_{\mathrm{N}}$ measure). The "Included" column indicates whether a feature is included in our final, optimised model. "PercentNum2" has been removed after considering backward selection, attaining a Pseudo-R^2 measure of 0.64763 (see text). The rightmost column provides the coefficients for the logistic regression after choosing the optimal features.

clearly strongly correlated. More subtle correlations between combinations of features are also very likely, and this is referred to as collinearity. This makes interpreting the coefficients in the right column difficult, since it's not clear how changing one variable influences another. This, however, does not have an effect of the usefulness of the model to evaluate the confidence of a given match.

Confusion matrix. While the confidence rate provides a method of scoring each particular match, it can also potentially be used to classify a given match as correct or incorrect. This essentially involves saying that all matches above a certain threshold we classify as a correct match, and all those below the threshold we classify as an incorrect match. This binary classifier—correct or incorrect—can lead to types of errors: those incorrect matches which are mistakenly classified as correct (aka Type I errors), and those correct matches which are mistakenly classified as incorrect (Type II errors).

In terms of optimising for the overall error rate, the optimum threshold to use is 50%. We calculate the logistic probabilities using the parameters in Table 2 for the 12346 records of training data used previously. The threshold 50% is then applied to classify each of these matches. While this test should strictly be applied to an unseen data set to avoid overfitting, for this data we don't expect this to be a problem since there are only 13 parameters for many more data points. The information describing the types of error are shown in the confusion matrix, Table 3.

Summary. We have used machine-learning methods and cross-validation to provide a rigorous estimation of the confidence of a given match. This measure of confidence allows us to either: (1) set a threshold, below which questionable matches are discarded, or (2) investigate questionable matches by hand in order to decide

	True 0	True 1	Total
Predicted 0	187	128	315
Predicted 1	34	11997	12031
Total	221	12125	12346

False positive rate (i.e.., 1−specificity)	40.6%
True positive rate (i.e., sensitivity)	99.7%
Positive predicted value (i.e. precision)	98.9%
Negative predicted value	84.6%

Table 3: (left) Confusion matrix using probability threshold 50%; (right) Important measures for classification and diagnostic testing derived from the confusion table.

whether to retain or discard them.

Limitations. The above logistic regression method is subject to a number of limitations:

- The confidence rating is subject to the training data being representative of the input data, which may not always be the case. Data varies wildly depending on its source, for example whether it is residential or commercial.
- The labelling of data is inherently subjective: when marking a match as correct or incorrect, there is no way of being absolutely certain that an assigned match is the one the user (e.g., the estate agent or surveyor) intended. This subjectivity will affect our confidence estimation.

7 Direct comparison of old and new versions

This new method produces a very serious improvement in the number and quality of matches which are obtained. Below we present a number of comparisons between the former version (version 1.4) of the address-matching procedure and the new version (version 1.5). The results were:

Statistics from EPC data. The techniques were each applied to the data Anna Carlsson-Hyslop provided to the team in September 2018 as data unmatched by NatCen (batch names: `EPC_ANNA_FILE1`, `EPC_ANNA_FILE2`, `EPC_ANNA_FILE3`, `EPC_ANNA_FILE4`, and `EPC_ANNA_FILE5`), which contain a total of 5616 address strings. A comparison of the three techniques described above are presented in the following table:

	New version (exclud. Parents)	New version (includ. Parents)	Old version (all tokens)	Old version (top 4 tokens)
time per 1000 addresses	19s	95s	25s	17s
% with 0 matches	<1%	<1%	41%	5%
% with 1 match	99%	99%	58%	44%
% with many matches	1%	<1%	1%	52%

(7.1)

We see the new version obtains a much larger proportion of single matches than the previous two versions. Of those with a single match in the latest version (including Parents), we found <1% were false positives.

Statistics from Zoopla data. The methods were tested on a random sample of 5000 Zoopla addresses belonging to residential properties.

	New version (exclud. Parents)	New version (includ. Parents)	Old version (all tokens)	Old version (top 4 tokens)
time per 1000 addresses	33s	96s	123s	36s
% with 0 matches	<1%	<1%	45%	13%
% with 1 match	92%	94%	51%	82%
% with many matches	8%	6%	3%	5%

(7.2)

Of those with a single match in the latest version (including Parents), we found 4% were false positives.

Building safety data. The method was tested on a sample of 283 records relating to building safety in high-rise building. Many of these addresses refer to entire buildings, and are thus best mapped to a Parent UPRN.

The results are as follows:

	New version (exclud. Parents)	New version (includ. Parents)	Old version (all tokens)	Old version (top 4 tokens)
time per 1000 addresses	84s	149s	53s	35s
% with 0 matches	7%	7%	58%	51%
% with 1 match	32%	86%	4%	4%
% with many matches	61%	7%	38%	45%

(7.3)

We see a large improvement when we include Parent UPRNs for this data set. This, however, comes with a hindrance to performance.

8 Connecting disparate data sources

An ultimate aim of the addressing matching method is to connect data from disconnected sources. In MHCLG, we have data from sources such as Zoopla listing data, Land Registry, Help to Buy and Buy to Rent policies, and Energy Performance Certificates. The traditional reference is the address; however we have seen addresses are fallible: they are subject to errors and ambiguity. The address matching method is intended to connect points from each source to one another, by connecting each data point to a UPRN.

In this section, we investigate the ability of the address matching method (latest version) to connect such data sources. As our test case, we attempt to connect a set of Zoopla listing data to the appropriate Energy Performance Certificates. We begin by selecting a sample of data: we choose all Zoopla listed properties in the postcode NG18. The results are shown in Fig. 6. The address matching method is applied separately to both

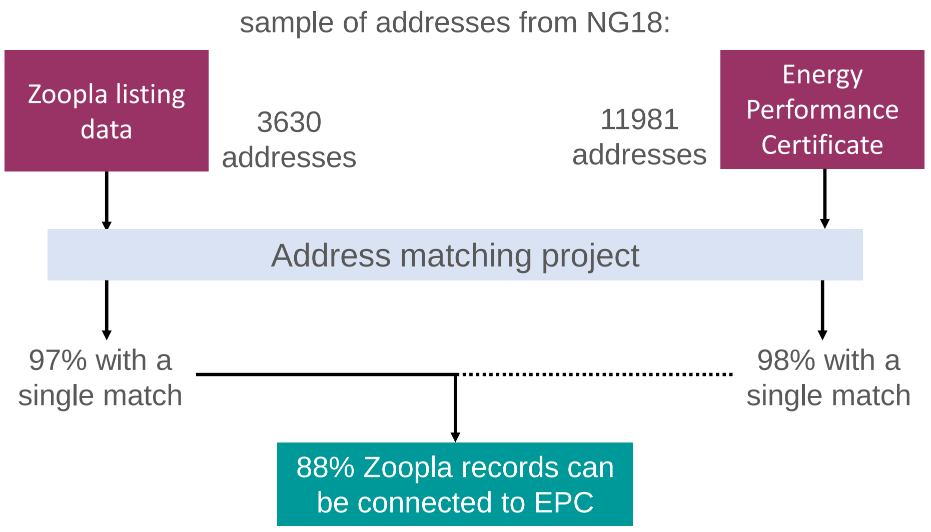


Figure 6: The address matching method can be used to connect records from disparate data sources.

data sources: this took a combined time of 4.9 minutes. Of the 3630 Zoopla-listed properties, 97% were mapped to a single UPRN. Similarly, of the 12052 EPC records, 98% were mapped to a single UPRN. Next, we compare the two tables of results. We find that 88% of all the Zoopla-listed properties have a UPRN which was mapped to an EPC record. In other words, we were able connect 88% of all Zoopla-listed properties to their Energy Performance Certificates.

While 88% is a very high match rate, we should also investigate what is the cause of the 9% of Zoopla data which were mapped to a single UPRN but not found in the EPC records. There doesn't appear to be any pattern in this data. Rather, it is the case that the EPC data does not cover every UPRN in NG18. There is a total of 21321 UPRNs with the postcode NG18, but only 11981 (56%) EPC data points. This is because (1) EPCs were introduced in 2008, and only exist for properties which were constructed, sold, or let since then, and (2) the EPC data we used only goes up to 2016: this is likely to have a large impact on the match rate of Zoopla records to EPC. It is also a possibility that the Zoopla records which were not matched correspond to fraudulent activities, in which someone is trying to let a property without a valid EPC. Future work will involve investigating the discrepancies between data sources beyond the scope of address matching.

9 Conclusions

The address matching method developed by the data science team at MHCLG provides a solution to matching fuzzy addresses to a specific UPRN which is reliable, scalable, and provides a quantified measure of confidence for each match. Unlike other solutions found outside the department, the solution developed here is not subject to the license restrictions on data sets when transferring to an external service, and the solution has been designed with the demand that exists within MHCLG in mind. Over the development of the address matching procedure, the algorithm has grown in terms of complexity and accuracy, so that it is now suitable for considering less structured sources which may contain spelling mistakes and abbreviations.

Limitations.

- The procedure will not produce any valid results where an incorrect postcode has been supplied.
- The procedure requires that the reference table from AddressBase `tbl_AddressSearch` is contemporary with the input address data.
- The procedure ultimately ignores mistyped tokens, since they do not match tokens in the reference string. More complicated token matching techniques exist (for example calculating the Levenshtein distance—the number of different characters) which attempt to match mistyped words, but these introduce their own problems: computational inefficiency, difficult implementation, and increased false positives. For our purpose, our results indicate counting the number of matching tokens to be sufficient.

References

[1] MHCLG. Fixing our broken housing market. https://www.gov.uk/government/publications/fixing-our-broken-housing-market, 2017.

[2] MHCLG. Ministry of Housing, Communities and Local Government single departmental plan. https://www.gov.uk/government/publications/department-for-communities-and-local-government-single-departmental-plan/ministry-of-housing-communities-and-local-government-single-departmental-plan, 2017.

[3] Harvard Business Review Andreas Weigend. The Social Data Revolution(s). https://hbr.org/2009/05/the-social-data-revolution.html, 2009.

[4] Martin Hilbert. Big data for development: From information-to knowledge societies. 2013.

[5] Telegraph Ian Cowie. Budget 2013: winners and losers of Osborne's Help to Buy pledge. https://www.telegraph.co.uk/finance/property/buying-selling-moving/9959021/Budget-2013-winners-and-losers-of-Osbornes-Help-to-Buy-pledge.html, 2013.

[6] Zoopla. Zoopla ranked again as UK's second most visited property website. https://www.zoopla.co.uk/press/releases/zoopla-ranked-again-as-uks-second-most-visited-property-website/, 2012.

[7] Telegraph Christopher Hope. Counties to be axed from postal addresses. https://www.telegraph.co.uk/news/uknews/royal-mail/7919253/Counties-to-be-axed-from-postal-addresses.html, 2013.

[8] Ordnance Survey. AddressBase products user guide. https://www.ordnancesurvey.co.uk/docs/product-guides/addressbase-product-guide.pdf, 2016.

[9] Ordnance Survey. AddressBase products Premium support. https://www.ordnancesurvey.co.uk/business-and-government/help-and-support/products/addressbase-premium.html, 2019.

[10] GBG. Fuzzy Address Matching. https://www.gbgplc.com/fuzzy-address-matching/, 2019.

Column Name	Comments	Example
Id_Identity		3946683
DateAdded		2018-09-21 11:59:12:345
Batch_Name	over which the stored procedure will be run	EPC_BATCH01
External_Identifier	should be unique	N3068395
External_Address	fuzzy address string	2 hart court, fake street, kings lynn pe9 6gh
External_Postcode	(optional)	pe9 6gh

Table 4: Table design and example record of input table `tbl_BatchSearch_PGH`

[11] Aligned Assets. iMatch - UK Address Matching Software. https://www.aligned-assets.co.uk/products/imatch/#, 2019.

[12] Government Digital Service Richard Vale. Tackling address matching together. https://data.blog.gov.uk/2016/08/09/tackling-address-matching-together/, 2016.

[13] Government Digital Service Jen Lambourne. Tackling address matching together. https://data.blog.gov.uk/2016/08/19/the-language-of-addresses/, 2016.

[14] ONS Alistair Calder. Building an address index for census and beyond . https://gss.civilservice.gov.uk/wp-content/uploads/2017/01/Sld-Addressing-for-Census-and-beyond-Calder-ONS.pdf, 2017.

[15] Ordnance Survey. The UPRN lifecycle from planning to demolition. https://www.geoplace.co.uk/-/the-uprn-lifecycle-from-planning-to-demolition, 2015.

A Usage notes – Searching By Address

A.1 Matching a single address

A single address can be matched within ADD-staging by following the example:

```
SELECT *
FROM DS_AddressMatching.dbo.tbl_Search_ByAddress_PGH('2 hart court, fake street, kings lynn, pe9
6gh','pe9 6gh','1','0')
```

This will return a table of the form shown in Table 5. The parameters refer to (1) the address string, (2) the post-code (optional), the (3) IncludeParents flag, and (4) ConfidenceThreshold. These are discussed in more detail below.

A.2 Batch address matching

The latest version of the batch-AddressMatching stored procedure can be found following the path: `ADD-Staging.DS_AddressMatching.dbo.usp_Seach_ByAddress_PGH`. Using Microsoft SQL Sever Management Studio, this can be found by navigating in the left hand (object explorer) pane ADD-Staging.DS > DS_AddressMatching > Progamability > Stored Procedures > dbo.usp_Seach_ForAddress_Develop_PGH. A user must have write privileges for the `DS_AddressMatching` database for the stored procedure to be viewable.

The address matching stored procedure is designed to work on batches of address strings. The procedure takes a table of input address strings, which can be found by navigating ADD-Staging.DS > DS_AddressMatching > Tables > dbo.tbl_BatchSearch_PGH. The design of the table and an example record are shown in Table 4. When adding a new batch to the table, each row should have the same unique Batch Name.

The stored procedure can be executed by right-clicking the stored procedure `dbo.usp_Seach_ByAddress_PGH` in the left hand (object explorer) pane and choosing Execute Stored Procedure. The user will be prompted to provide three parameters:

1. `Batch_Name`, which should be the same as appears in the table `dbo.tbl_BatchSearch_PGH`

Column Name	Comments	Example
Batch_Name		EPC_BATCH01
External_Identifier		N3068395
External_Address		2 hart court, fake street, kings lynn pe9 6gh
External_Postcode		pe9 6gh
FOUND	Address of match (i.e., reference address)	2 Hart Court King's Lynn PE9 6GH
POSTCODE	Postcode of match	PE9 6GH
UPRN	UPRN of match	100090946278
UDPRN	UDPRN of match	18805946
PARENT_UPRN	Parent UPRN of match	100090974936
IsParent	Whether a given UPRN is a parent or not	0
FoundCount	Number of matches returned	1
Confidence	Estimated % probability match is correct	99.21
Version	Version of address matching procedure	03/12/2018 - V1.5

Table 5: Table design and example record of output table `tbl_BatchReturn_PGH`

2. `IncludeParents` flag: a flag to set whether the addresses of Parent UPRNs (i.e., entire buildings) are to be inferred from their children, and included in the search. This increases the match rate when the data includes addresses for entire buildings, but decreases performance by a factor of 2-3. Parents are included when `IncludeParents=1` (default), and excluded when `IncludeParents=0`.
3. `ConfidenceThreshold`: sets a confidence threshold below which results are removed. This provides a way of automatically filtering low-confidence results. The value is a percentage. Examples values include '0' (no thresholding, default), '5' (remove very unlikely results), and '50' (a conservative threshold). By default, the thresholding is turn off, and the inspection of low-confidence matches is left as a task for the user.

The stored procedure may alternatively be executed by using a query: (2) using the query:

```
USE [DS_AddressMatching]
GO
DECLARE @return_value int

EXEC return_value = [dbo].[usp_Search_ByAddress_PGH]
@Batch_Name = N'<batch name here>'
@IncludeParents = N'<0 or 1 here>'
@ConfidenceThreshols = N'<percentage here>'

SELECT 'Return Value' = return_value

GO
```

These two methods will execute the AddressMatching function on all the rows of `dbo.tbl_BatchSearch_PGH` with the relevant Batch Name. The results are inserted into the table `dbo.tbl_BatchReturn_PGH`. If the procedure has already been run before for a given Batch Name, all records with that Batch Name will be removed from `dbo.tbl_BatchReturn_PGH` and the AddressMatching procedure will be re-ran for this batch. Table 5 summarises the form of the output table. The input data (External_Identifier, External_Address, External_postcode) is produced alongside the AddressBase data from the proposed match (FOUND, POSTCODE, UPRN, UDPRN, PARENT_UPRN), as well as data describing the matching procedure itself (IsParent, FoundCount, Confidence, Version). IsParent indicates whether the match corresponds to a ParentUPRN. FoundCount indicates the number of matching results which have been found: if this is zero it implies no matching results exist, while if it is two or more, it indicates the input address was not suitably specific to be mapped to a unique address. Confidence provides an estimate of the probability that the match is correct. Low confidence results can either be filtered out or manually inspected.

B Usage notes – Searching By UPRN/Parent UPRN

In certain cases, someone may wish us to return the addresses for a list of UPRNs. This is essentially the reverse of the above process, however it in practice it is much easier as it does not involve any 'fuzzy' matching. Such a request was made in the past by the Building Safety Team. In this instance, we found that Parent UPRNs were mixed in with the list of UPRNs. Thus, we have developed a tool to (1) return the address of a UPRN, or else (2) return an interpolated address of a Parent UPRN . The usage notes are presented below.

B.1 For a single UPRN/Parent UPRN

The address of a single UPRN/Parent UPRN can be matched within ADD-staging by following the example:

```
SELECT *
FROM DS_AddressMatching.dbo.tbl_Search_ByUPRN_PGH(12345678910, '')
```

This will return a table of the form shown in Table 7. The parameters refer to (1) the UPRN/Parent UPRN, (2) the mode of use (discussed below).

B.2 For a batch of UPRNs/Parent UPRNs

The stored procedure for returning a list of address from UPRNs can be found following the path: `ADD-Staging.DS_AddressMatching.dbo.usp_Seach_ByUPRN_PGH`. The procedure takes a table of input UPRNs/Parent UPRNs, which can be found by navigating ADD-Staging.DS > DS_AddressMatching > Tables > dbo.tbl_BatchUPRNSearch_PGH. The design of the table and an example record are shown in Table 6. When adding a new batch to the table, each row should have the same unique Batch Name.

Column Name	Comments	Example
Id_Identity		3946683
DateAdded		2018-09-21 11:59:12:345
Batch_Name	over which the stored procedure will be run	EPC_BATCH01
External_Identifier	should be unique	N3068395
UPRN	UPRN or Parent UPRN to be searched	100090946278

Table 6: Table design and example record of input table `tbl_BatchUPRNSearch_PGH`

The stored procedure can be executed by right-clicking the stored procedure `dbo.usp_Seach_ByUPRN_PGH` in the left hand (object explorer) pane and choosing Execute Stored Procedure. The user will be prompted to provide three parameters:

1. `Batch_Name`, which should be the same as appears in the table `dbo.tbl_BatchUPRNSearch_PGH`
2. `Mode`: There are three modes available for this procedure
   (a) `''` or `'DEFAULT'`: If the UPRN is not a parent, it returns the address. If the UPRN is a Parent, it infers the address from its children and returns this.
   (b) `'PARENT-INTERPOLATE'`: Infers the address of a Parent UPRN from its children only.
   (c) `'PARENT-ALL'`: Returns the UPRN and address of all the children underneath a Parent UPRN.

The results are inserted into the table `dbo.tbl_BatchUPRNReturn_PGH`. If the procedure has already been executed before for a given Batch Name, all records with that Batch Name will be removed from `dbo.tbl_BatchUPRNReturn_PGH` and the AddressMatching procedure will be re-ran for this batch. Table 7 summarises the form of the output table.

C List of removed terms

The following list contains all postal counties and their abbreviations which are removed when cleaning the address string.

Column Name	Comments	Example
Batch_Name		EPC_BATCH01
External_Identifier		N3068395
External_UPRN		100090946278
UPRN		100090946278
PARENT_UPRN		100090974936
UDPRN		18805946
FOUND	Address from AddressBase (or inferred if Parent)	2 Hart Court King's Lynn PE9 6GH
POSTCODE	Postcode from AddressBase	PE9 6GH
NumberOfChildren	The number of children a Parent UPRN has (0 if not a Parent UPRN)	0

Table 7: Table design and example record of output table `tbl_BatchUPRNReturn_PGH`

Avon	Bedfordshire	Beds	Berks
Berkshire	Buckinghamshire	Bucks	Cambridgeshire
Cambs	Cheshire	Cleveland	Co Durham
Cornwall	County Durham	Cumbria	Derbyshire
Devon	Dorset	E Sussex	East Sussex
Essex	Glos	Gloucestershire	Greater London
Greater Manchester	Hampshire	Hants	Herefordshire
Hertfordshire	Herts	Isle of Wight	Kent
Lancashire	Lancs	Leicestershire	Leics
Lincolnshire	Lincs	Merseyside	Middlesex
Middx	N Humberside	N Yorkshire	Norfolk
North Humberside	North Yorkshire	Northamptonshire	Northants
Northd	Northumberland	Nottinghamshire	Notts
Oxfordshire	Oxon	Powys	S Humberside
S Yorkshire	Shropshire	Somerset	South Humberside
South Yorkshire	Staffordshire	Staffs	Suffolk
Surrey	Tyne and Wear	W Midlands	W Sussex
W Yorkshire	Warks	Warwickshire	West Midlands
West Sussex	West Yorkshire	Wilts	Wiltshire
Worcestershire	Worcs		

These postal counties constitute elements of the address which have been formally deprecated; for this reason, they are not included in the references addresses. The terms are removed from an input address only if they appear after the first 16 characters, to avoid discarding potentially useful address information.