Squeezing More Out of Your Data: Business Record Linkage with Python

by

John Cuffe
U.S. Census Bureau

Nathan Goldschlag
U.S. Census Bureau

CES 18-46 **November, 2018**

Abstract

Integrating data from different sources has become a fundamental component of modern data analytics. Record linkage methods represent an important class of tools for accomplishing such integration. In the absence of common disambiguated identifiers, researchers often must resort to "fuzzy" matching, which allows imprecision in the characteristics used to identify common entities across dfferent datasets. While the record linkage literature has identified numerous individually useful fuzzy matching techniques, there is little consensus on a way to integrate those techniques within a single framework. To this end, we introduce the Multiple Algorithm Matching for Better Analytics (MAMBA), an easy-to-use, flexible, scalable, and transparent software platform for business record linkage applications using Census microdata. MAMBA leverages multiple string comparators to assess the similarity of records using a machine learning algorithm to disambiguate matches. This software represents a transparent tool for researchers seeking to link external business data to the Census Business Register files.

Keyword*:*

JEL Classification:

Any opinions and conclusions expressed herein are those of the authors and do not necessarily represent the views of the U.S. Census Bureau. All results have been reviewed to ensure that no confidential information is disclosed. We thank Wei Oyang, Elisabeth Perlman, Daniel Kim, Joseph Staudt, and participants at the CES brown bag seminar series.

Cuffe: Mojo Development Team, U.S. Census Bureau, (corresponding author) john.cuffe@census.gov
Goldschlag: Center for Economic Studies, U.S. Census Bureau, nathan.goldschlag@census.gov

1 Introduction

Integrating data from different sources has become a fundamental component of modern data analytics. Record linkage techniques play a central role in generating new research insights and producing data infrastructure that can be used to create new public use statistics. It is very rare that a researcher is blessed with datasets from different sources that share high quality disambiguated identifiers. In the absence of such identifiers, researchers often rely on "fuzzy" matching techniques, which allow imprecision in the characteristics that identify common entities across different datasets. There is no agreed upon way to operationalize the concept of "fuzzy" in a matching procedure. The record linkage literature has identified numerous individually useful fuzzy matching techniques, but has not, to the best of our knowledge, produced a consensus on how to integrate those techniques within a single framework. Despite this, there have been efforts to integrate different types of similarity measures. One such example is the Freely Extensible Biomedical Record Linkage (FEBRL) (Christen, 2008). We build upon that work by 1) focusing specifically on business record linkage problems in the context of Census microdata and 2) incorporating machine learning algorithms to improve match disambiguation. To this end, we introduce the Multiple Algorithm Matching for Better Analytics (MAMBA) software platform. MAMBA is an easy-to-use, flexible, scalable, and transparent software platform for performing business record linkage to the Census Bureau's Business Register (BR) databases.[1] This framework provides researchers with a optimized, off-the-shelf tool for record linkage to Census business microdata, with the capacity to be extended to additional Census business datasources.

A researcher needing to link datasets from different sources that lack common identifiers has a wealth of record linkage software options.[2]. However, many of the available methods suffer serious shortcomings either in the assumptions of the models used or a lack of transparency. Konda et al (2016) highlight four traits for an effective entity matching (EM) system: a how-to guide, minimal user burden, runtime, and scalability. We take the core of this list, arguing that any software should be easy to use, flexible, scalable, produce accurate matches, and be transparent. Unfortunately, most off-the-shelf matching software fail on one or more of these metrics. Solutions that overcome the need for scalability and accuracy are typically only usable by "power users", requiring relatively advanced programming skills in the selected language (Konda et al., 2016)[p. 1583], reducing the ease of use, flexibility, and transparency of the software as the researcher will need to make custom adjustments.

[1]The Business Register is the Census Bureau's establishment-level universe of all non-farm employer businesses in the US. Throughout we make reference to the BR, but the programs have been built using the Standard Statistical Establishment List files

[2]For example SAS DQ Match, Matchit and RECLINK (Blasnik et al., 2010) in STATA.

Particularly troubling is the lack of transparency in decision rules. In some cases, matching algorithms are black boxes where information on the process used to generate the match is proprietary. This leaves the researcher to either utilize sub-optimal matches or use ad-hoc decision rules that, while improving accuracy, reduce the transparency and replicability of matches (Winkler, 2014).

Most matching approaches take one of two forms. Supervised models utilize a set of "true" matches between two datasets to produce match probabilities for all match combinations based on a statistical model. This "classical" record linkage is most relevant in cases where one data set represents a superset of another, often (theoretically) the case when matching external records to internal Census data on businesses and individuals. Unsupervized methods, generally based on graphical models, seek to match entities where the truth is not known by creating latent entities and then matching objects in the data to these latent entity representations. These methods are able to link multiple data sets (rather than a single pair), and are ideal in circumstances where no set of data is a superset of another. Both approaches, however, typically rely on "fuzzy" string comparators–even unsupervised methods will evaluate the similarity of strings to generate match candidates.

String comparator algorithms are designed to account for clerical mistakes by mathematically assessing the closeness of two strings. For example, a string comparator will identify '123 Main Street' and '123 Mian Street' as a likely match. Many different types of string comparator algorithms have been introduced in the record linkage literature.[3] Each comparator has unique strengths and weaknesses and will be more or less effective depending on the type of linkage effort under consideration and the characteristics of the data being matched. In particular, while providing researchers with the relative strength of these measures, string comparisons of this nature often make assumptions on the linearity of the likelihood of the probability of a relationship between two strings, a common pitfall in classification model selection (Schild et al., 2017).

The MAMBA software platform is designed to improve upon existing fuzzy matching procedures used at the Census Bureau for business record linkage. The algorithm leverages string comparators derived from FEBRL (Christen, 2008), an open-source matching software suite that contains many different types of string comparators. Our algorithm uses 13 of these string comparators to measure the similarity between business name and address fields in the BR and an external dataset.[4] The resulting similarity scores are fed into a Random Forest classifier, yielding a match designation (false or positive) and its associated probability

[3]For example, see Jaro (1989), Winkler (1990), and Levenshtein (1966).

[4]For the purposes of developing our algorithm we use business name and address information derived from university-vendor transactions associated with university-based grant funded research projects. See Goldschlag et al. for details about these data.

that captures the uncertainty of the classification.

MAMBA offers several improvements over other existing matching techniques. First, it is much more transparent and flexible. Methods such as SAS DQ, in contrast, rely on proprietary algorithms, which make it difficult for researchers to understand why certain matches are made or tune the underlying algorithms to suit specific matching needs. MAMBA, on the other hand, provides the relative importance of individual string comparators algorithms and match probabilities that allow researchers to better understand the uncertainty associated with matches. These match probabilities can also be used to adjust statistical inference for the uncertainty in the underlying matched data. Matching procedures based on SAS DQ algorithms provide only coarse measures of quality based on the blocks used (e.g. street, city, and state) along with the level of "fuzziness" chosen (DQ match sensitivity). Having accurate measures of match quality is crucial because failing to account for this type of variation can bias regression coefficients (Tancredi and Liseo, 2015). Third, MAMBA leverages the rich heterogeneity of different string comparators and accounts for the complex interactions between them in predicting a match. While each string comparator on its own is useful, our results suggest that taking many different comparators together can be substantially more effective. Finally, our matching software is scalable with computational resources, which allows for faster processing and larger matching exercises. A fundamental problem in record linkage is the combinatorial nature of the search space. Typically, blocking strategies are used to limit the number of comparisons made. While our matcher also integrates blocking, we also accommodate matching against the entire Business Register by decentralizing the comparisons.

In the following sections we outline several different types of comparators and provide an example of the benefits of using multiple comparators to identify matches. The remainder of the paper is organized as follows. In Section 2 we discuss the benefits of the use of multiple comparators. Section 3 describes the MAMBA procedure and illustrating our model selection criteria. Section 4 describes how the random forest model was developed. Section 5 concludes, providing directions for future research and development.

2 String Comparators

Our approach relies on harnessing the heterogeneity between different types of string comparators. Each string comparator (e.g. Jaro, Jaro-Winlker, Levensetein, AD-distance) has strengths and weaknesses. We are by no means the first to intuit the benefits of using multiple comparators or that different comparators may perform better or worse on different types of data (Sun et al., 2015; Cohen et al., 2003; Van der Loo, 2014; Zarembo et al., 2015).

Our primary contribution is our specific implementation in MAMBA, combining a suite of string comparators within a generalized framework that (1) provides a research ready tool for linking external business data to Census microdata, (2) leverages the underlying heterogeneity in similarity measures, and (3) implements a machine learning model that allows the relative importance of each measure to vary depending on the types of data being matched. Before describing MAMBA in greater detail, we first describe each similarity measure used and illustrate the substantial variation between "families" of similarity measures.

2.1 Jaro

The Jaro distance between two strings is the number of single-character transpositions required to transform one string into another (Jaro, 1989). For strings A and B, with some number of matching characters m and transpositions t, the Jaro distance is defined as follows.

$$d_j = \frac{1}{3}\Big(\frac{m}{|A|} + \frac{m}{|B|} + \frac{m - t/2}{m}\Big) \tag{1}$$

Where $|A|$ and $|B|$ is the length of string A and B respectively. Two characters in the strings are considered a match only when they are closer to one another in terms of order in their respective strings than one less the floor of the half of the maximum string length. A matching character is defined as follows.

$$m = \left\lfloor \frac{max(|A|, |B|)}{2} \right\rfloor - 1 \tag{2}$$

2.2 Winkler Modification (Jaro-Winkler)

Winkler (1990) modified the Jaro distance by giving a heavier weight to pairs of strings where the first l characters match, with a constant scaling factor (usually .01) p. Formally, for a Jaro score d_j, the Jaro-Winkler score, d_w, is defined as follows.

$$d_w = d_j + \big(lp(1 - d_j)\big) \tag{3}$$

2.3 Qgrams

A q-gram (or n-gram) breaks a set of strings into the same number of q length substrings, then divides the number of shared sub-strings by the average length of the two strings. Designed to find the syntactic similarity of strings, q-grams have been used for machine translation and spelling correction, especially by Google (Franz and Brants, 2006). Formally, for two strings A and B, the q-gram score is as follows.

$$q = \frac{\sum_{i=1}^{n} \begin{cases} 1 & \text{if } A_i = B_i \\ 0 & \text{otherwise} \end{cases}}{(|A| + |B|)/2} \tag{4}$$

Where i is each q-length substring.

2.4 Longest Common Substring

Longest common substring (LCS) algorithms seek to identify, for strings A and B, the length of the longest common substring. A substring is a set of characters that are in the same relative order and contiguous in both strings.

2.5 Cosine Similarity

In some cases, we may not be interested in the *order* of two strings, but rather the degree to which they contain the same information. For example, we may wish to compare "123 Main Street" and "StrMain 123". For these cases, the Cosine similarity measure proves useful since it does not rely on the order characters in two pairs of strings. Rather, this measure counts all of the unique characters and white spaces in two strings A and B, and calculating the cosine difference between them. Formally:

$$c = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5}$$

Where i indicates the particular element of vector A or B.

2.6 Levenshtein (Edit) Distance

The Levenshtein (1966) family of string comparators seeks to assess the difference between two strings by calculating the number of single-character edits required to make the strings match. Some versions (e.g. Bartolini et al., 2002; Smith and Waterman, 1981) of this comparator punish insertions, edits, and deletions differently. Conceptually, the score is the running sum, across strings A and B, of the number of deletions, insertions, or character swaps required to convert string A into string B. For each character i, we find the smallest value in the running sum depending on if we are deleting character i from string A inserting character j into string A, or substituting a character. Formally, the score for character two strings A and B, of length i and j respectively, is:

$$R_{A,B}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} R_{A,B}(i-1,j) + 1 \\ R_{A,B}(i,j-1) + 1 \\ R_{A,B}(i-1,j-1) + 1_{A_i \neq B_j} \end{cases} & \text{otherwise} \end{cases} \tag{6}$$

where $1_{A_i \neq B_j}$ is an indicator equal to 1 if $A_i \neq B_j$ and zero otherwise.

2.7 An Illustrative Example

An ideal string similarity measure for our purposes would maximize the difference in scores it assigns to matches and non-matches. The significant variation in the structure and quality of business name and address strings in administrative data provides scope for different types of similarity measures to do better or worse under certain conditions. For example, some algorithms may perform well with basic spelling errors while others do better in accounting for abbreviations. In order to provide some intuition behind the types of variation we have in mind, Table 1 shows several hypothetical business name and addresses combinations. Clearly, Pair 1 is a match with a simple typographical error. Most, if not all, comparators would assign relatively high similarity scores to these cases. Pair 2, on the other hand, may be more of challenge for some string comparators with a possible typographical error on the address (0 substituted with 9) and an unorthodox abbreviation in the input business name data. Pair 3 might be a match, but there are substantial differences in the name and address fields. Finally, while pairs 4 and 5 are clearly non-matches, the addresses for Pair 4 are extremely similar, and both names have a similar common substring ("cars"), which may lead some algorithms to produce false positives.

Table 1: Hypothetical Match Pairs

Pair	Input Name	Input Address	BR Name	BR Address
1	MADUSAS OIL	123 MAIN	MEDUSAS OIL	123 MAIN
2	SOLO TRAV	80 DAGOBAH	SOLO TRAVEL	89 DAGOBAH
3	MOES TAVERN	4 EVRGRN	FLAMIN MOES	4 EVERGREEN
4	KIRK CARS	1701 A	PICARD CARS	1701 D
5	FREDS BAKERY	13 ELM	JASONS FARM	31 MEL

Note: Hypothetical examples only, following standardization procedures to remove punctuation, street identifiers (e.g. street, ave.).

Table 2 shows the similarity measures for each pair of example business names shown in Table 1. All similarity measures are scaled between 0 (meaning no matching traits) and 1 (perfect match) and are weighted by the length of both strings. There are several points to note. First, even for the most similar pair of names, Pair 1, there is variation among the string comparators. The two-character qgram (Qgram) and Longest Common Substring (LCS) methods scoring these very similar strings substantially lower, punishing the potential typo more severely.[5] Pair 4 shows some potential weaknesses of the Winkler and Cosine similarity measures, with both scoring the strings above 0.7, contrasted with the qgram giving a score of 0.45. Comparing Pair 2 and Pair 3 we see that all of the comparators were much less certain about Pair 3 except for the cosine algorithm, which give a much higher score. It is also important the heterogeneity in the range of scores each algorithm assigns. The qgram and Levenshtein, for example, yield a much broader range of scores compared to the Winkler and Cosine comparators. This pattern is interesting as it points to the fact that a "good" classifier would not necessarily give high scores to good matches, but instead maximize the distance between the scores assigned to "good" and "bad" match candidates. A larger range, then, gives the matching model a greater chance of accurately distinguishing between matches and non-matches.

[5]For these examples $q = 2$, so the measures shown for qgram are bigrams.

Table 2: Hypothetical Match Pairs: Name Scores

Pair	Input Name	BR Name	Wink.	Qgram	LCS	Lev.	Cosine
1	MADUSAS OIL	MEDUSAS OIL	.89	.83	.82	.91	.93
2	SOLO TRAV	SOLO TRAVEL	.96	.82	.90	.82	.93
3	MOES TAVERN	FLAMIN MOES	.45	.25	.36	0	.69
4	KIRK CARS	PICARD CARS	.71	.45	.50	.64	.74
5	FREDS BAKERY	JASONS FARM	.57	.08	.17	.25	.52

Note: Hypothetical examples only, following standardization procedures to remove punctuation, street identifiers (e.g. street, ave.). All scores scaled between 0 and 1, controlling for string length.

For addresses, we see a similar pattern in Table 3. The qgram gives a lower score to Pair 2 than the others, identifying correctly the small difference in the address, however it also punishes Pair 1 for a similar potentially typographical error. Pair 4 shows the advantage of the qgram, which gives a low score to what are very different strings. In contrast, the Winkler and Cosine algorithms give relatively high scores to Pair 4. However, Pair 2 shows the potential advantage for the Winkler and Cosine approaches, which retain relatively high scores for a likely shorthand address in our hypothetical business data. The address scores again highlight the limitation of the Winkler and Cosine algorithms, which may struggle to correctly differentiate matches effectively since the range of scores they produce is much more compact.

Table 3: Hypothetical Match Pairs: Address Scores

Pair	Input Name	BR Name	Wink.	Qgram	LCS	Lev.	Cosine
1	80 DAGOBAH	89 DAGOBAH	.94	.82	.80	.90	1
2	4 EVRGRN	4 EVERGREEN	.95	.67	.74	.73	.82
3	1701 A	1701 D	.94	.75	.71	.86	.80
4	13 ELM	31 MEL	.88	.25	.71	.43	1

Note: Hypothetical examples only, following standardization procedure to remove punctuation, street identifiers (e.g. street, ave.). All scores scaled between 0 and 1, controlling for string length.

There are four main lessons we learn from these examples. First, even small changes in strings can alter the scores of different comparators drastically. Second, the ideal method for classifying matches and non matches would not necessarily rely on true matches achieving high scores, but instead focus on maximizing the difference in scores assigned to matches and non-matches. Third, none of these similarity measures are without weaknesses. The Winkler and Cosine similarity measures tend to award similar scores to both matches and

non-matches, which could lead to false positives. The two-character qgram, on the other hand, may punish minor mistakes too heavily, leading to false negatives. Finally, in the case of business record linkage, it is important to consider simultaneously both the business name and address scores, and take the information provided from both where possible.

These examples also hint at the advantages of using the heterogeneity between similarity measures in a classification model. In the next section, we introduce the Multiple Algorithm Matching for Better Analysis (MAMBA) platform, software designed to incorporate the heterogeneity contained between string comparators to produce more, higher quality matches. Table 4 summarizes the string comparators implemented in MAMBA.

Table 4: Implemented String Comparators

Name	Description
Jaro	The minimum number of single-character alternations required to convert one string to another.
Jaro-Winkler	Jaro comparator with heavier weight assigned to the first four characters of the string.
Sorted Winkler	Winkler algorithm where each word is sorted into alphabetical order before application.
Levenshtein Distance	Measures how many substitutions/deletions/insertions are required to turn an input string into a BR string.
Bag Distance (C)	An approximation of the Levenshtein distance
SW Distance	Smith-Waterman algorithm for string distance, checking local sequence alignment (Smith and Waterman, 1981).
Longest Common Substring (2)	String comparison on the longest common substring of at least two characters between the strings.
Longest Common Substring (3)	String comparison on the longest common substring of at least three characters between the strings.
Sequence Match (C)	Generating matches based on length matching sequences within string.
Qgram (2)	The proportion of all 2-character substrings in the strings that match.
Qgram (3)	The proportion of all 3-character substrings in the strings that match.
Positional Qgram (3)	The proportion of all 3-character substrings in the strings that match *in the same order*.
Cosine Similarity	Cosine similarity of histograms of the unique characters in each string.

Note: Note: (C) indicates the measure is the average of comparing string 1 to string 2 and string 2 to string 1, which yield different results for the algorithm. (#) indicates the length of the substring used in each algorithm.

3 The MAMBA Software Suite

The MAMBA software platform takes an input dataset of business names and addresses and matches them to the BR using multiple string comparators and a machine learning classification algorithm. The software also meets each of the five criteria mentioned by Konda et al (2016):

1. Easy to use: although implemented in Python, SAS, and SQL, the software can be used without any familiarity with these languages, and is customized to be operated

on existing Census research infrastructure.

2. Transparent: the process, code, and the procedure for generating match probabilities are clearly documented and provided to researchers.

3. Flexible: the software can accommodate records with missing geographic or name and address information, as well as new training data and additional matching criteria without violating core model assumptions. It can also match on firm and establishment level for name and address pairs as well as name or address only matching within the same framework.

4. Scalable: by leveraging Python's multiprocessing capabilities, matching is parallelized, reducing the time-consuming match process but also ensuring the procedure is scalable.

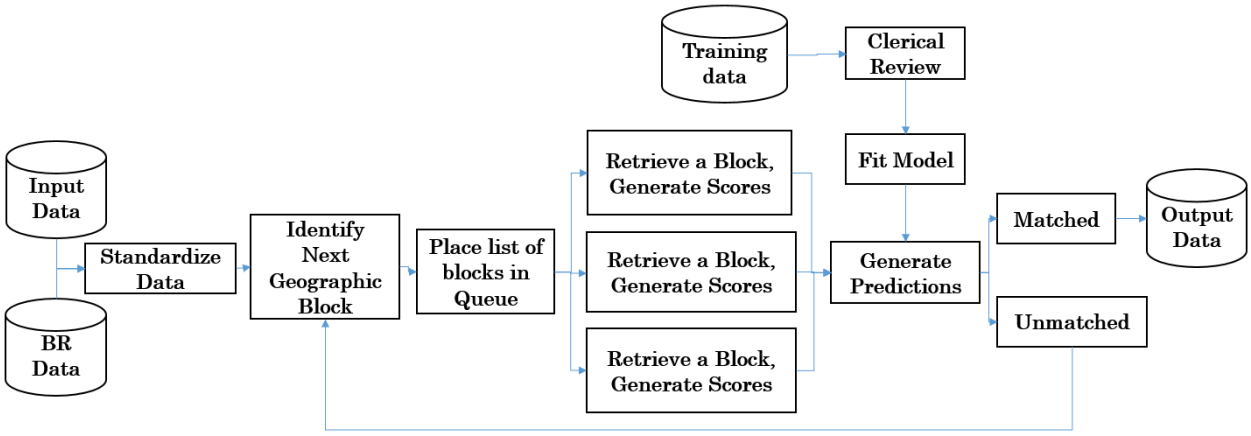
5. High-quality matches: the model out-performs existing methods used for business record linkage at the Census Bureau, such as SAS DQ, by improving match Precision (the proportion of matches the model identifies as "true" that are true matches).

Figure 1 provides a visual representation of the various components of the MAMBA algorithm. The procedure begins by standardizing names and addresses in the BR and the input data by replacing common abbreviations and suffixes. Next, the process uses a blocking strategy based on geographic criteria to reduce the search space. The blocks include 5-digit zip code, city, 3-digit zip code, state, and an overall match comparing any residual inputs against the entire BR.[6] In the case of the overall match, to again reduce the search space and computation burden of the comparisons a soundex block is used to compare only records with at least some baseline level of similarity. Specifically, for state-level and non-geographic matches, candidate pairs must have at least one word with a matching soundex code. For example, if an input record contains the word "main" in the address, we would seek to compare that record to all BR matches also containing a word that sounds like "main". MAMBA is implemented as a residual match, meaning that after each match block only un-matched records are at risk of matching in subsequent blocks. Then, using training data either provided with the software package or by the user, fits a random forest on a random grid of paramters, identifying the best possible model for the training data.

To generate a list of potential matches, the program leverages Python's multiprocessing capabilities to execute the string comparators in parallel. A set of blocking values to be considered is generated, which contains the intersection of blocking values contained in both the input data and BR. For example, for the state block, if the input data contains businesses

[6]Future iterations of the software may add additional geographic criteria present in the BR (e.g. county, MSA) contained in the input data.

Figure 1: Graphical Description of MAMBA Procedure.



in only two states, both of which appear in the BR, then the set of blocking values would include only those two states. A set of worker objects select a blocking value and calculate similarity scores between all possible pairs of names and addresses within that blocking value. All matches with an average score greater than .5 are further processed as potential matches.[7] Potential matches are then fed into a random forest classifier trained on a set of clerically reviewed matches. Records classified as matches are then stored in the final output data and unmatched cases are fed back into set of records at risk for a match in the next block.

3.1 Standardization

The first step in our process is to apply a standardization procedure to both the BR and input data similar to that used in (Dinlersoz et al., 2017) and (Graham et al., 2015). We do this because while a human may recognize "Street" and "st" as providing the same meaning, we cannot rely on string comparators to draw the same conclusion. Our procedure applies approximately 500 corrections to eliminate issues such as common abbreviations (e.g. standardizing derivations of 'Corporation' to 'Corp'), street names (1st becomes First), and directions (Northeast becomes NE). We admit these standardizations do not account for all possible variations, and future research is needed to apply modern Natural Language Processing methods such as stemming to improve standardization procedures.

[7]This value was selected to mirror the procedure of the SWELL (Kutzbach et al., 2016) rules. Additionally, the vast majority of candidate pairs with an average score below .5 are very low quality matches, thus by not carrying these cases forward to the matching model we are able to save computation time.

3.2 Blocking and Filtering

Given the combinatorial nature of the search space, we apply several filters to limit the number of calculations made.[8] First, we implement a geographic blocking strategy, limiting our similarity calculations to pairs of records within each geographic or blocking group. For example, a group of records from the input data in zip code 12345 will be compared to BR records from the same zip code. The procedure matches to (in order of retrieval) Employment Identification Number (EIN) (if available on the input dataset), combination of 5-digit zip code and 2 digit NAICS code, 5-digit zip code, city, 3-digit zip code, state, and then finally to the entire BR.[9] MAMBA is implemented as a residual match, meaning only unmatched records from a given block are placed in a queue to be matched in the next block type. We also restrict to comparisons of records that are in operation within a specified window of time. For example, for a BR record observed between 2009 and 2011, the input record must have a start or end date between 2009 and 2011. This window of overlapping activity is flexible. Researchers may select to increase the size of the activity window. For our example above, if a researcher allows for a 2-year gap, then the BR record must either appear between 2007 and 2013 to match an input record first appearing in 2009 and disappearing after 2011.

In addition to blocking and year restrictions, we also execute two additional filters before calculating similarity measures. First, we remove all cases where the 3-character qgram on the business name is less than a certain value. For EIN, 5-digit zip code, city, and 3-digit zip code that value is 0.3. For state and the non-geographic block that value is 0.4. This comparator has proven effective in parsing the difference between matches and non-matches (see Figure 2 below). This restriction allows us to remove 99.5% of potential matches in any given block, significantly reducing computation requirements. We selected the filter value of 0.3 because it is the average score for a non-match for the 3-character qgram in our training data–in short, this figure ensures we do not bias our data to find false negatives while also ensuring we do not unnecessarily evaluate candidates that are unlikely to be matches. We select a higher value, 0.4, for the state and non-geographic blocks because of the sheer number of comparisons that would be made otherwise.

In addition to the qgram filter, we block on soundex. For cases in the state and non-geographic block, we only compare records where at least one word in the name and address share a common soundex code. Again, we do this to reduce the number of potential comparisons while not introducing risk of false-negatives. For a "true" match to violate this

[8] For a sense of scale, in the absence blocking an input dataset of 100,000 observations, and given that each year the BR contains over 6.5 million observations, would require 650 billion comparisons for a single year.

[9] The particular set of blocks used in any given matching exercise is chosen by the researcher implementing the match. See Appendix A for details.

condition it would have to be a match despite none of the word spellings in either the name or address starting with the same letter and none of the words having a similar phonetic spelling pattern. In these cases, we doubt our model would be capable of labeling the pair a match even if it were included in downstream processing. Once we have a sufficiently small number of potential comparisons, we calculate each similarity measure outlined in Section 2. These scores are then used in a classification model to predict which of the pairs that survive our filters is a true match. That classification model relies on training data, which we turn to next.

3.3 Training Data Generation

In order to train our classifier we performed a clerical review of approximately 2,000 records to identify matches. We use a simple weighting mechanism to select records for comparison based on the average score of all of the comparisons for that record.[10] To ensure a sufficient number of (rare) matches, we select an equal number of records with an average score on both name and address matches between 0.5, and 0.7, 0.7 and 0.9, and 0.9 to 1 (exclusive), which results in a heavy over-sample of high-scoring record pairs. This oversampling of high scoring matches ensures our model is exposed to sufficient variability in both scores and match classifications. We exclude both perfect matches, with a score of 1, and matches with a score of less than 0.5 for either name or address. The MAMBA procedures allow researchers to develop their own additional training data drawn from their input data, and we encourage them to do so. The MAMBA suite is provided with capabilities to produce training data based on new inputs, and by developing their own data researchers will be able to tune MAMBA to their own specific case.

Once a sample of clerical review cases are identified, we classify each as either a true positive or false positive. We classified an input-BR pair of records as a true positive if there is no contradictory information to show they were not a match. For example, if a pair shared the same name, but the input address was "123 Main Street" and the BR address "123 Main Street 234", indicating a suite or apartment, we would classify a match. However, if the addresses were "123 Main Street 234" and "123 Main Street 789", we would deem this as contradictory information and classify the record pair as not matching. This approach is advantageous as it allows the clerical reviewer to ignore simple typographical errors (e.g. if records have a close name match but the address has a typo). This method also allows the reviewer to account for the fact that the level of detail in the business name may vary between the input data and the BR. For example, if a firm in the BR has multiple establishments for a

[10] See Kutzbach et al. (2016) for a description of such a sampling scheme.

single address for each department (e.g. Accounting, HR), and the input data only contained the name and address and did not specify a department, we would generate multiple matches for this record.[11]

3.4 Match Classification Model

For each set of training data MAMBA generates the Random Forest model with the best fit on criteria such as precision, recall, or accuracy, as selected by the researcher from the default list for Python's sci-kit learn.[12] The model is calculated by performing a grid search over the number of trees and the number of features per tree selected for each model. The results discussed below were developed from a Random Forest with 32 trees using a maximum of 15 features per tree, with a maximum tree depth of 10.[13] The model achieves a cross-validated precision score of over 95%, and substantially out-performs alternative approaches. MAMBA is specifically designed to be flexible, allowing the feature selection to reflect the characteristics of the matching problem at hand. In other applications of the MAMBA programs, researchers can expect to have a different feature selection criteria and fit. This again speaks to the importance of cross-validation and sufficient training data to ensure an efficient grid search. To enhance the flexibility of the software, each instance of the model is generated through a randomized grid search, fit to maximize the selected scoring criteria (Precision, Recall etc) by the user. This feature requires no additional input from the user, and greatly enhances the accuracy of the model.

3.5 MAMBA Lite

Another selectable option for the researcher is MAMBA Lite. With this option, the match classification model above is calculated as normal, but the program then selects the best 8 (5 for name or address only matching) features in terms of an impurity metric for that model. It then fits a new model using the same methods as before, but only on these limited features. Our initial results suggest a negligible different in model performance, for example, a reduction in .5% drop in precision for our training data. The trade off for this slight loss in performance of the model is vastly reduced memory and time requirements for the program,

[11] As described in Appendix A, our model also the match with the highest predicted probability for each input record, removing many potential false-positives.

[12] For details see `http://scikit-learn.org/stable/modules/model_evaluation.html`

[13] These parameters were selected from a cross-validated grid search, however we do note the unusually high number of maximum features per tree relative to total features (15/26) may be a cause for concern that we have over-fit our data. However the findings from this model are consistent with others with lower numbers of features per tree.

by in effect reducing the computation time by two-thirds. Further research will perform the selection of model criteria programmatically, instead of the current arbitrary criteria.

3.6 MAMBA Outputs

MAMBA outputs three files. First, it selects a small subset of pairs with an average score on both name and address of greater than 0.5 to serve as clerical review candidates for the model. This enables the researcher to generate their own candidate pairs relatively easily. While we provide the MAMBA software with training data, using your own may guard against sub-optimal performance. Second, it generates a file that contains all pairs of records that the selected model identifies as matches, regardless of the number of matches per input record. The output file contains the "best" match for each input record, which is the one with the highest predicted probability. We break ties by highest average score amongst all comparators used, then by length of the BR entry, and finally, if pairs are still tied, we randomly choose a single match. We provide both files so that researchers may sample from the distribution of predicted probabilities in the case of multiple matches if they wish.

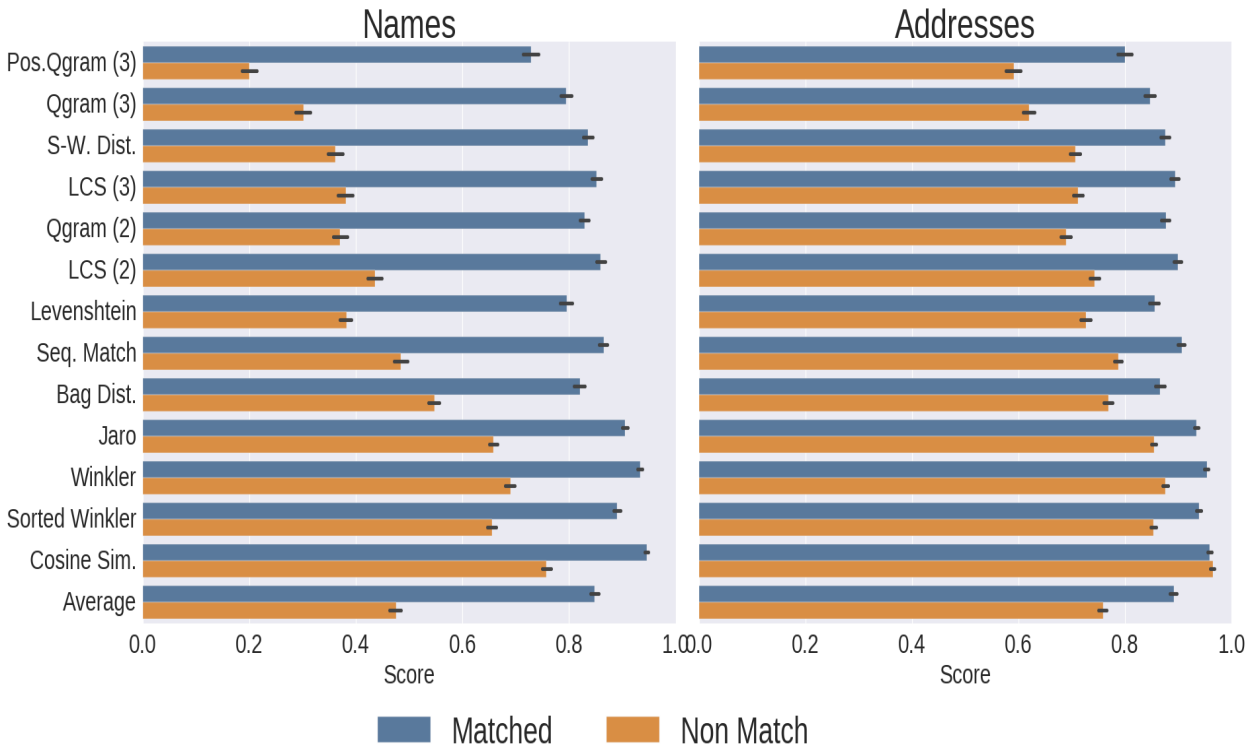
4 Classification Model Selection

In the following sections we describe in greater detail the model selection process that led to the preferred model described in Section 3. These models are meant to sift through the potentially numerous matches that meet our basic filters. In the following section we provide a description of the relationship between our similarity measures and the match classification as observed in our training data. We then describe both regression-based classification models and our preferred random forest model in greater detail. We must stress that one of the advantages of MAMBA is for the user to create their own training data, and thus results in individual situations may differ. The data used are sourced from the Innovation Measurement Initiative (IMI) at the Census Bureau in collaboration with the Institute for Research and Innovation in Science (IRIS) at the university of Michigan. The training data consist of names and address of university vendors on federal research grants (Goldschlag et al.; Ikudo et al.).

4.1 Patterns in the Training Data

Figure 2 below shows the average scores for observations classified as matches (blue), and non-matches (orange) for each string comparator. Name scores are generally higher than

Figure 2: Mean Scores for Name and Address Matches in Training Data.



Note: Figure shows mean and standard deviation (black) for 1900 hand-reviewed candidate pairs.

address scores, regardless of whether or not the observation was a match–the average score for name matches is approximately 0.89, whereas for address the average score for matching records is approximately 0.82. There also appears to be more of a gap between match and non-match scores for names than addresses. The positional qgram shows an average difference of 0.4 between match and non-match scores, compared to the cosine similarity measure for addresses assigns almost identical scores to matching and non-matching records. The y-axis of the figure is sorted by the difference between the average score for matches and non-matches between the names, and reveals which methods may be most useful: qgrams, LCS, and some of the distance classifiers show significant differentiation between matches and non-matches. The cosine similarity measure and the Jaro family, on the other hand, show the least differentiation.

4.2 Regression-Based Classification

Using our training data, where the match classifications is known, we can begin to establish the relationship between the string comparator scores and whether a pair of name and addresses is a match. In order to establish a baseline, we consider several logistic regression

specifications using a simple average of the string comparator scores. Table 5 shows the estimation results using as our independent variables the average name and average address scores along with an interaction term between the two. The models show conflicting evidence for the presence of dependence between the scores: the model including the interaction term has a higher true-positive rate, but a *worse* false-positive rate and AUC. It is also likely that this type of specification would suffer from significant over fitting if not cross validated.

Table 5: Logistic Regression Results: Averaged Scores

	Model	
	Additive	Interaction
Name	8.125*	6.044*
	(0.382)	(1.21)
Address	4.463*	1.396
	(0.258)	(1.418)
Interaction		4.096*
		(1.651)
Intercept	-8.915*	-7.251*
	(0.397)	(1.082)
N	1,900	1,900
AUC	91.87	90.13
Precision	87.70	90.37
Recall	84.61	83.83
F1	85.81	86.68

Source: MAMBA, Business Register, UMETRICs, authors' calculations.
Note: Observations are pairs of candidate matches. Standard Errors in Parentheses. $*$ indicates $p < .01$.

The next candidate model is a simple additive model in which all of our similarity measures enter with no interaction terms. The model provides similar performance to the previous two regressions. The results, shown in Table 6, also show that only a handful of comparators provide statistically significant predictive power. This lack of significance is likely due to the highly inflated standard errors. Only the positional qgram for name increases the likelihood of a match, while the cosine similarity for address decreases the likelihood of the record being considered a match.

Collinearity is a severe issue for these regression models. The average variance inflation factor for all of the similarity measures is approximately 72, large enough to not only severely inflate the standard errors but also potentially bias predictions on out-of-sample data. Such high variance inflation will also generate unstable estimates for the training model. The

Table 6: Logistic Regression Results: Individual Scores

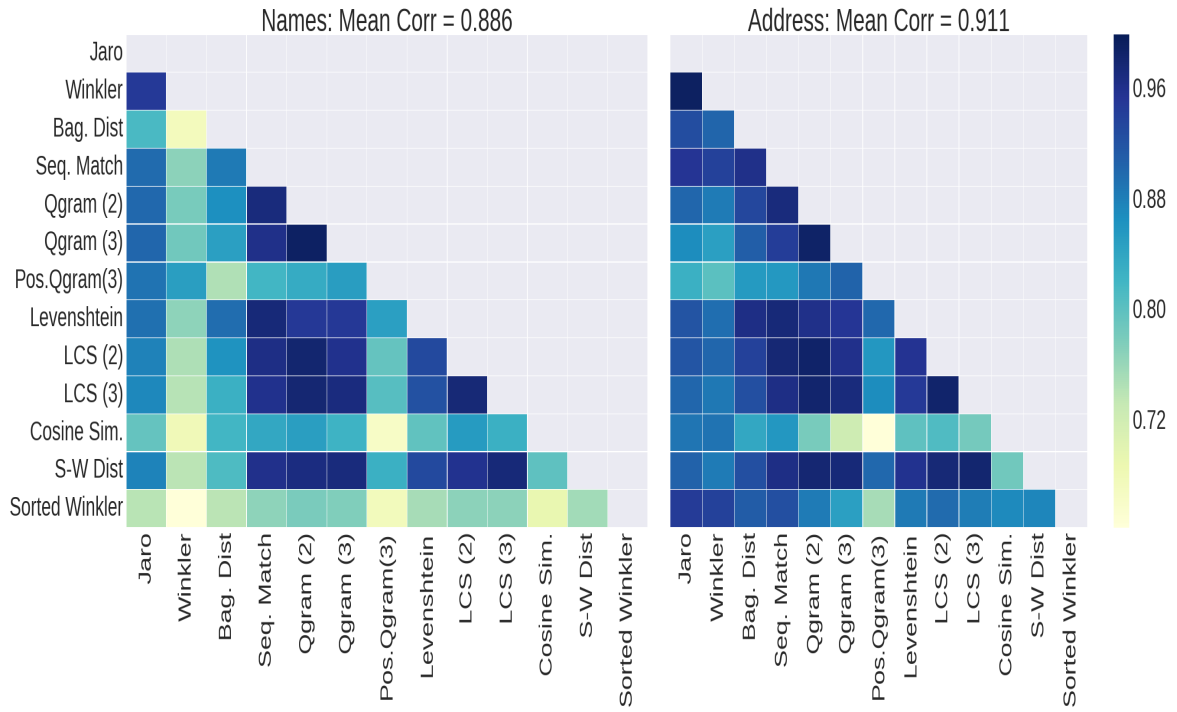
Variable	Estimate	Std. Error
Intercept	0.832*	(0.097)
Jaro (Name)	1.303	(4.207)
Winkler (Name)	2.435	(3.368)
Bag Dist. (Name)	-1.988	(1.169)
Sequence (Name)	-0.26	(2.825)
Qgram(2) (Name)	2.049	(6.48)
Qgram(3) (Name)	2.628	(4.383)
Pos. Qgram(3) (Name)	1.632*	(0.56)
Levenshtein (Name)	-1.85	(1.775)
LCS(2) (Name)	0.986	(3.77)
LCS(3) (Name)	0.332	(2.33)
Cosine (Name)	2.302	(1.668)
Smith-Waterson (Name)	0.238	(2.452)
Sorted Winkler (Name)	-0.107	(0.777)
Jaro (Address)	0.828	(4.645)
Winkler (Address)	1.181	(3.757)
Bag Distance (Address)	-3.176	(1.954)
Sequence (Address)	-0.223	(3.621)
Qgram(2) (Address)	2.765	(5.166)
Qgram(3) (Address)	4.015	(3.389)
Pos. Qgram(3) (Address)	0.534	(0.915)
Levenshtein (Address)	-2.681	(2.293)
LCS(2) (Address)	1.599	(3.131)
LCS(3) (Address)	1.71	(2.648)
Cosine (Address)	-3.566*	(1.621)
Smith-Waterson (Address)	0.968	(2.957)
Sorted Winkler (Address)	0.693	(1.58)
N	1,900	
AUC	0.970	
Precision	0.915	
Recall	0.926	
F1	0.920	

Source: MAMBA, Business Register, UMETRICs, authors' calculations.
Note: Observations are pairs of candidate matches. Standard Errors in Parentheses. $*$ indicates $p < .01$.

cause of this collinearity is clear when we examine the correlation between the measures. Figure 3 below shows the correlations between the comparators for names and addresses. Even mean-centered, the correlations are often near perfect, with the average correlation for names being 0.87. In addition, the additive model is unable to capture interactions between

similarity measures which are present in the data, as seen with the dependence between average name and address scores.

Figure 3: Correlation Amongst String Comparators

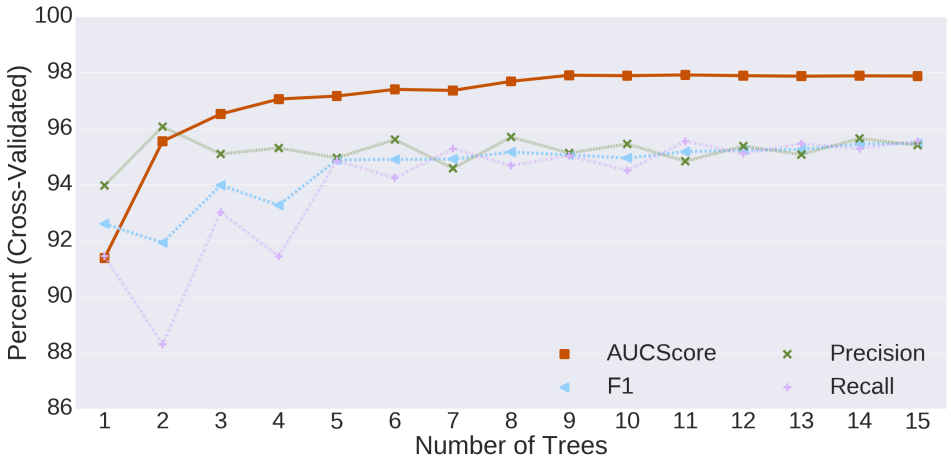


Note: Figure shows correlation amongst variables (mean-centered) for name and address matches.

4.3 Random Forest Classification

For our purposes, a Random Forest (Breiman, 2001) classifier represents the ideal approach for three reasons. First, a Random Forest decreases the variance of the modeled outcome without increasing bias or falling victim to over-fitting, a distinct possibility with our relatively small and specialized input data. Second, given its underlying decision-tree structure, Random Forest classification allows for interactions between variables not otherwise available to methods such as logistic regression. For two variables m and k, interactions between variables occur in a random forest model when a split on variable m in a tree makes a split on k more or less likely, and these interactions can also be tested for as additional features in the model. Finally, random forest classifiers do not suffer from the consequences of collinearity in the same way regression models do, which is severe amongst the string comparators we have selected. Briefly, a random forest is a series of decision tree classifiers using random subsets of both the data and features (covariates). Taking the majority vote to identify positives

Figure 4: Model Performance and Number of Trees.



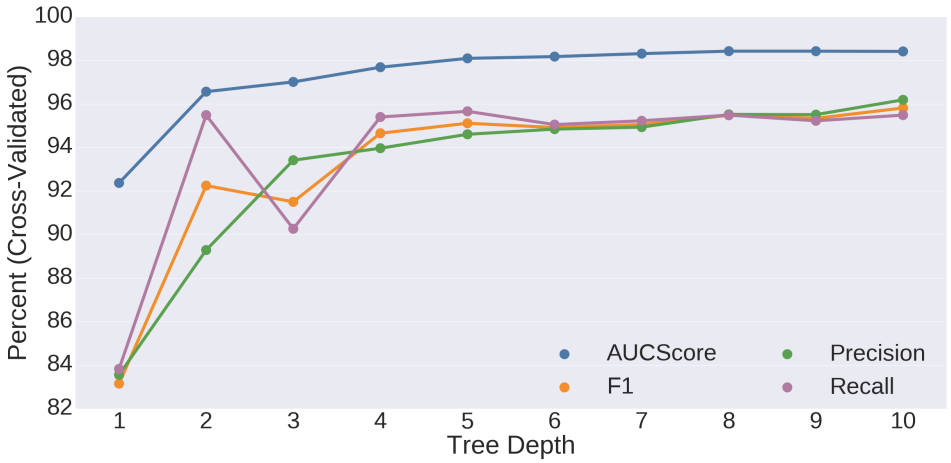
Note: Figure shows k-fold cross-validation scores for four scoring criteria, given the number of trees entered into each model.

and negatives for each decision, the algorithm reduces over-fitting and increases accuracy. Utilizing Python's *scikit learn* module, we implement a random forest model tuned to select the number of trees, and number of features per tree to maximize cross-validated precision (Pedregosa et al., 2011). By default, MAMBA *only* returns results from a 10-fold cross-validation of the training data, to ameliorate the possibility of over-fitting to small training samples.

Random forest classifiers have several important parameters, including the number of trees, the number of features used in each tree, and the depth of the tree. Predictions can be sensitive to the choice parameters. Figure 4 below shows how model performance changes with the number of trees (the x-axis). All statistics (AUC, precision, recall, and F1) vary substantially with 5 or fewer trees, after which all measures stabilize. Note also that with more than 5 trees all of our measures of quality are quite high, with precision, recall, and F1 about 95 and an AUC of about 98.

Another key parameter of random forest models is the depth, or number of branches in each decision tree. Deeper forests allow for greater non-linearities and interactions between variables by leveraging more information from each tree. The drawback of deeper trees, however, is the risk of over-fitting the training data and reducing the validity of out-of-sample predictions. Figure 5 shows how prediction quality changes with different depths. It shows substantial variation across the number of trees, with overall fit *decreasing* with deeper trees, an indicator of over-fitting for the data. The inconsistent results here, combined with the variation seen with the number of trees selected, led us to design MAMBA in such a way as to maximize the researcher's quantity of interest (e.g. Precision) *without having the*

Figure 5: Model Performance and Maximum Depth of Trees.



Note: Figure shows k-fold cross-validation scores for four scoring criteria, given the number of trees entered into each model.

researcher make any decisions on the precise model to chose. We argue this is beneficial, as the researcher can be more transparent in her discussion of methodological choices without fear of accusation of over-fitting.

4.4 Comparing Model Performance

Ultimately, the random forest model is our preferred method because it outperforms the regression methods along a number of dimensions. Figure 6 shows the comparison of quality measures between the three Logistic regression models above and our Random Forest classifier. It shows that the Random Forest out-performs the Logistic Regressions on all four of our metrics, with a 4-7% increase in Precision over the Logistic models, and a larger differences (between approximately 8% and 4%) for Recall.

In addition to out performing the regression methods, our random forest model appears to better leverage the heterogeneity across the similarity measures. To see this, we execute a single decision tree using each string comparator separately and compare that to our tuned random forest model using all of the comparators. Figure 7 shows the AUC, precision, recall, and F1 scores of the single-comparator decision trees and the fully saturated random forest model. The results demonstrate the advantages of leveraging all of the heterogeneity across the similarity measures. On average, the random forest model shows an 8.2% improvement on AUC, 6% improvement on F1, 6.5% improvement on Recall, and 3% improvement on Precision. Even compared to the best single comparator, 3-character LCS, our model increases all of the quality measures by about 3%. Moreover, while some measures tend to

Figure 6: Comparing Model Performance

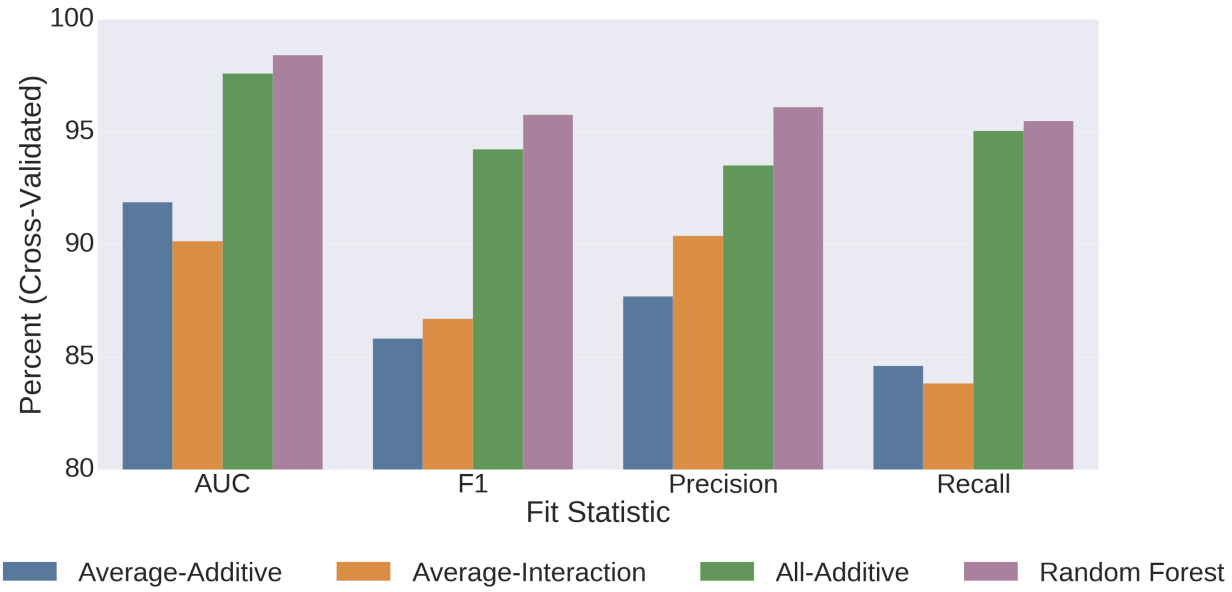


Figure shows k-fold cross-validation scores for four scoring criteria, given the number of trees entered into each model.

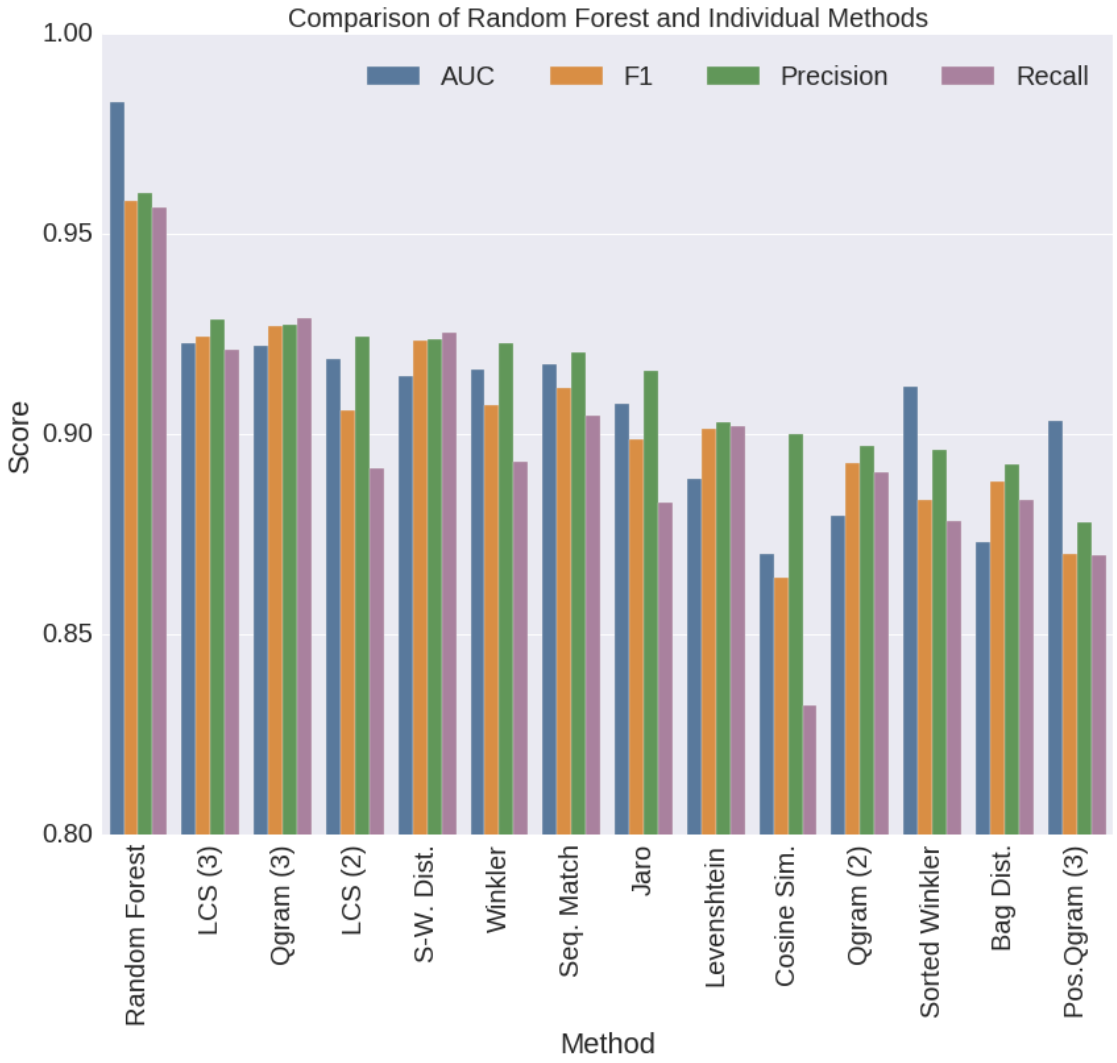
produce higher Precision scores, there is no measure that stands out as much more powerful than the rest. Using each string comparator in isolation does not improve the match quality nearly as much as when the measures are taken together.

4.5 Comparision with SAS DQ

In order to get a sense of how well our matching method performs we compare it to a matching technology often used with Census business microdata–SAS DQ. To compare against the SAS DQ Match procedure, we randomly sampled 700 unique pairs of records, and applied our Random Forest classifier and a SAS DQ algorithm that matches on names and addresses *and* additional passes on only name.[14] We then hand-coded the 700 pairs as matching or non-matching, and compared the codings to the two models. Table 7 below shows the percentage of the 700 placed into four categories, with our hand-coded data serving as the "truth". The table shows that the MAMBA algorithm generates 4 times as many false-positives as the

[14] We include both passes on name and address as well as only name in our SAS models because replicating our exact blocking and matching strategy would disadvantage SAS substantially. SAS DQ procedure relies on merging two datasets by match codes, and when the match codes to not appear (for example, when address is blank on our input records) SAS cannot create that record. Our model, however, still assigns a probability in these cases. The consequence for researchers is the need to add additional "passes" at matches for SAS in order to sufficiently approximate our algorithm. As such, we also match for name only within geographic blocks at the exact, 95, and 85 level. We note here that this additional step puts greater emphasis on SAS programming and adds complexity to the model, whereas our approach retains its simplicity, a key factor to consider for researchers without in-depth SAS knowledge.

Figure 7: Random Forest Performance Compared to Individual Comparators.



Note: Figure shows k-fold cross-validation scores for four scoring criteria for Decision Tree models using name and address information for each method, as well as results from best fitting Random Forest (criteria: Precision).

SAS algorithm, however generates 2.5 times the number of *true* positives. This is due to SAS' high false-negative rate.

Table 7: Comparing SAS DQ and MAMBA.

Type	SAS	Random Forest
True Negative	45.51	35.70
False Negative	31.44	1.42
False Positive	2.70	12.51
True Positive	20.34	50.36

Note: Percentages of total sample created from 700 hand-coded pairs.

The divide between false negatives and false positives is a key element of any classification exercise. The fact that MAMBA's produces more false positives in our application could be addressed by adjusting the match cutoff. Figure 8 shows graphically the location of each type of outcome by average across all string comparison and predicted probability for the MAMBA routine. Clearly, our default cutoff of .5 for a match results in several false-positive matches with a predicted probability below .6–increase the cutoff to .6 would sharply reduce the number of false-positives with only a small number of true positive outcomes removed. Rather than in the SAS environment, a researcher utilizing MAMBA could add additional criteria to her data selection process to account for this possibility. One may also argue this comparison represents an unfair bias towards MAMBA relative to SAS as the testing data used is materially similar to the training data, whereas the methods used to develop the SAS DQ algorithm were developed using other data sources, the implication being that our training data is over-fitted to particular kinds of errors common to our training data. However we argue this is an illustrative case–we encourage researchers to develop their own training data, which can be easily implemented into the MAMBA suite, providing the flexibility to train models on particular data without the concern of over-fitting to their particular data source.

5 Conclusion

In this paper we introduce the MAMBA software, a flexible, easy-to-use, scalable matching software that out-performs similar off-the-shelf products. MAMBA builds upon prior work, namely the Freely Extensible Biomedical Record Linkage (FEBRL) (Christen, 2008), to leverage the heterogeneity across string comparators using a machine learning algorithm. The Random Forest algorithm used to classify matches is able to achieve higher quality matches while reducing the impact of false positives and false negatives. In a comparison to

Figure 8: Classification Types by Comparator Score and Predicted Probability.

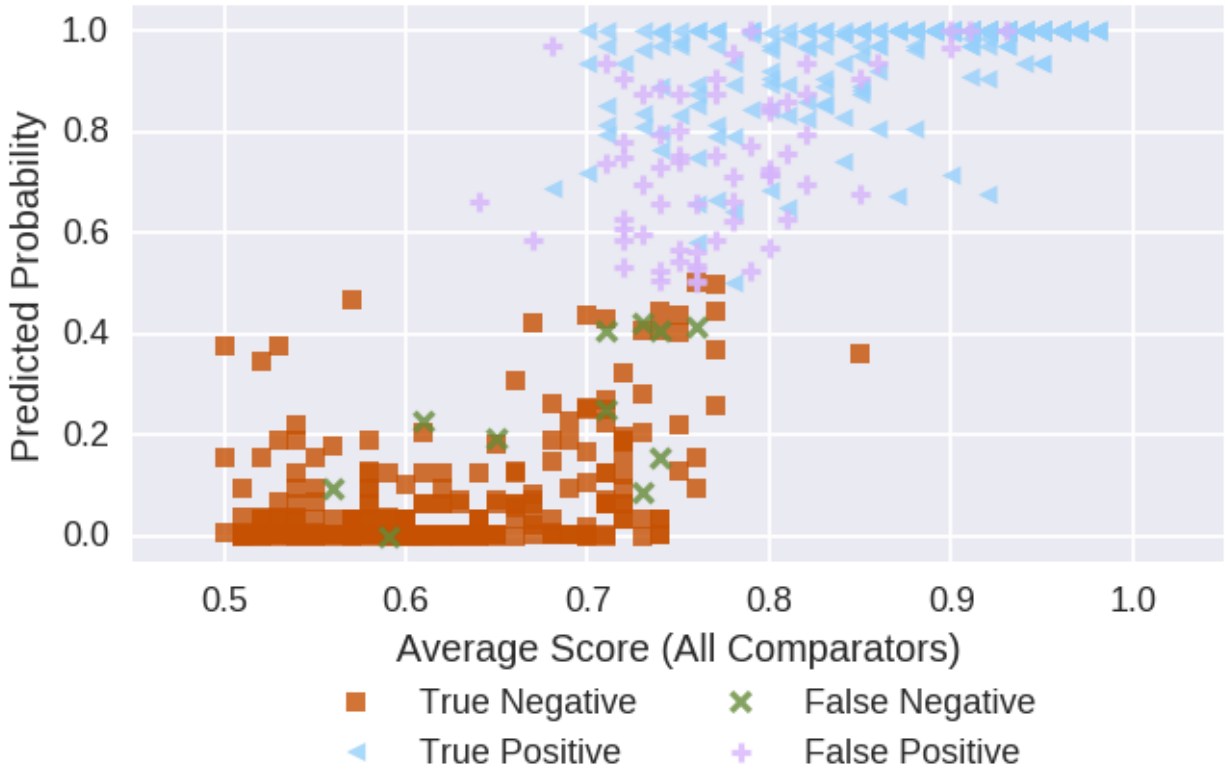


Figure shows classification type from Table 7 by average score across all name and address comparators and predicted probability for Random Forest model.

one of the more frequently used matching methods for Census business microdata, SAS DQ, MAMBA produces more than twice as many true positives, much fewer false negatives, with the cost of a modestly higher false positive rate. In addition to the high quality matches, our software also provides researchers with match weights, which researchers can use in their analyses to adjust inferences based on the uncertainty of the match. The software is currently available for beta-testing use by researchers matching business records to the Census Bureaus BR.

References

Bartolini, I., Ciaccia, P., and Patella, M. (2002). String matching with metric trees using an approximate distance. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval*, SPIRE 2002, pages 271–283, London, UK, UK. Springer-Verlag.

Blasnik, M. et al. (2010). Reclink: Stata module to probabilistically match records. *Statistical Software Components*.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Christen, P. (2008). Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management-Volume 80*, pages 17–25. Australian Computer Society, Inc.

Cohen, W., Ravikumar, P., and Fienberg, S. (2003). A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78.

Dinlersoz, E., Goldschlag, N., Myers, A., and Zolas, N. (2017). An anatomy of u.s. firms seeking trademark registration. In Corrado, C., Miranda, J., Haskel, J., and Sichel, D., editors, *CRIW: Measuring and Accounting for Innovation in the 21st Century*. NBER.

Franz, A. and Brants, T. (2006). All our n-gram are belong to you. `https://research.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html`.

Goldschlag, N., Lane, J., Weinberg, B. A., and Zolas, N. Proximity and economic activity: An analysis of vendor-university transactions. *Journal of Regional Science*, 0(0).

Graham, S. J., Grim, C., Islam, T., Marco, A. C., and Miranda, J. (2015). Business dynamics of innovating firms: Linking us patents with administrative data on workers and firms.

Ikudo, A., Lane, J., Staudt, J., and Weinberg, B. Occupational classifications: A machine learning approach. *NBER Working Paper Series*, August 2018.

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420.

Konda, P., Das, S., Suganthan GC, P., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J., et al. (2016). Magellan: toward building entity matching management systems over data science stacks. *Proceedings of the VLDB Endowment*, 9(13):1581–1584.

Kutzbach, M., Gathright, G., Green, A., McCue, K., Moti, H., Rodgers, A., Vilhuber, L., Wasi, N., and Wignall, C. (2016). Robustness of employer list linking to methodological variation. Joint Statistical Meetings.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and

reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Schild, C.-J., Schultz, S., and Wieser, F. (2017). Linking deutsche bundesbank company data using machine-learning-based classification. In *Deutsche Bundesbank Research Data and Service Centre*, volume 2017.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197.

Sun, Y., Ma, L., and Wang, S. (2015). A comparative evaluation of string similarity metrics for ontology alignment. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, 12(3):957–964.

Tancredi, A. and Liseo, B. (2015). Regression analysis with linked data: problems and possible solutions. *Statistica*, 75(1).

Van der Loo, M. P. (2014). The stringdist package for approximate string matching. *The R*, page 2.

Winkler, W. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.

Winkler, W. E. (2014). Matching and record linkage. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(5):313–325.

Zarembo, I., Teilans, A., Rausis, A., and Buls, J. (2015). Assessment of name based algorithms for land administration ontology matching. *Procedia Computer Science*, 43(Supplement C):53 – 61. ICTE in Regional Development, December 2014, Valmiera, Latvia.

Appendices

A MAMBA User Guide

A.1 run_Match.bash

This is the .bash file that executes the match. It contains the variables below that researchers can alter to suit their needs.

A.2 /programs

This folder contains the three core files described below as well as the supplementary files.

B Preparing Your Data

The data the user seeks to match to the Business Register is referred to as the 'input' data. The data needs to contain the following columns, however you can leave a column entirely blank if you do not have it in your data. At this time, the column headings must match exactly[15]:

1. inputid: a unique identifier (can either be numeric or string, but will be read as string) for each input object (e.g. business establishment) the user wishes to match.
2. ein: the Employer Identification Number (numeric characters only) for the establishment.
3. name: the name of the input business to be matched.
4. address: the address of the input object to be matched.
5. city: the city where the address is located.
6. state: the two digit postal code for the state (e.g. CA, NY)
7. zipcode: the 5 or 9 digit zip code for the address. This is converted into a 3- and 5-digit zip code for matching. NOTE: remove any '-' or non-numeric characters from this field.

[15]Future versions of the program could include 'soft coding' of variable names

8. input_minyear: the first year that you wish to attempt to match for the particular inputid at the particular name/address combination.

9. input_maxyear: the last year that you wish to attempt to match for the particular inputid at the particular name/address combination.

C Running the MAMBA Program

The user only needs to run the `run_match.bash` file located in the main MAMBA directory. The file is a .bash shell that executes a sequence of programs that generate matches. To run the program, just *cd* into the drive where your MAMBA suite resides, then enter `qsub run_match.bash`. The user can edit the following variables:

- **cd**: the location of the MAMBA folder.
- **outputdir**: the location of your data. This should contain your input .csv files, and also be the location you would like the log and .csv output files delivered to.
- **brstartyear**: the first year that you want to start looking for matches in the BR.
- **brendyear**: the last year that you want to look for matches in the BR.
- **numWorkers**: the number of workers utilized by the runner. Number of workers must be at least 2 less than the **numcpus**. Maximum on Research 1 cluster is 22 workers+monitors, as we have an additional CPU reserved for the main program.
- **numcpus**: the number of cpus required to operate. Must be at least 1 larger than the number of workers.
- **memusage**: the memory usage (in megabytes) required for match. Match function requires approximately 10000mb per worker, however this may vary depending on input data.
- **MAMBALite**: Do you want to use all matching criteria (slower) or a smaller subset (faster). If `True`, then the MAMBA algorithm selects the 8 (5 for name/address only) matching criteria for the best-fitting model on your training data. Experimentation shows a miniscule (.5%) change in the full versus lite models, with substantial time savings.

- **matchtype**: Are you matching by name and address, name only, or address only (address only not currently implemented) Values include ‘‘NameAddress’’ and ‘‘Name-Only’’.
- **both**: If matching by name and address, do you want to match records where both name and address are present on the input data? Values include ‘‘True’’ and ‘‘False’’.
- **name**: If matching by name and address, do you want to match records where only names are present on the input data? Values include ‘‘True’’ and ‘False’. Not used for address only data..
- **address**: If matching by name and address, do you want to match records where only addresses are present on the input data? Values include ‘‘True’’ and ‘‘False’’. Not used for name only data..
- **continuematchflag**: Is this match a continuation of a previously disconnected session? Values include ‘‘True’’ and ‘‘False’’.
- **matchlevel**: Do you want to match on the firm or establishment elvel? Values include ‘‘firm’’ and ‘‘estab’’.
- **searchvar**: This variable tells sas how to aggregate the ssl data. Values include ‘‘alpha’’ for firm level matching and ‘‘estab’’ for establishment level matching.
- **ein**: Do you want to block by EIN? Values include ‘‘True’’ and ‘‘False’’.
- **industry**: Do you want to block by the combination of 2-digit NAICS industry and zip code? Values include ‘‘True’’ and ‘‘False’’.
- **zip5**: Do you want to block by 5 digit zip code? Values include ‘‘True’’ and ‘‘False’’.
- **city**: Do you want to block by city? Values include ‘‘True’’ and ‘‘False’’.
- **zip3**: Do you want to block by 3 digit zip code? Values include ‘‘True’’ and ‘‘False’’.
- **state**: Do you want to block by state FIPS code? Values include ‘‘True’’ and ‘‘False’’. Note: For ‘name and address’ matching, state-level matches require at least one word, or the entire string, in both name and address to match on soundex code to the BR establishment in order to be matched using the MAMBA procedure. See our working paper for further information.

- **all**: Do you want to match all remaining records on the input data against all records in the BR? Values include ‘‘True’’ and ‘‘False’’.Note: For ‘name and address’ matching, no-geography matches require at least one word, or the entire string, in both name and address to match on soundex code to the BR establishment in order to be matched using the MAMBA procedure. See our working paper for further information.

- **brchunks**: The number of chunks you want to split the BR into for all match (default =1000).

- **yeargap**: How many years before and after your input year are you willing to declare as a match. For example, a value of ‘2’ would result in input data from 2012 being compared to the BR between 2010 and 2014.

- **predict**: Does the user want the program to generate predictions based on training data? Values include ‘‘True’’ and ‘‘False’’. Outputs **outputdir**/allmatch_DATE.csv and **outputdir**/bestmatch_DATE.csv files.

- **clericalreview**:Does the user want the program to return a sample of scores with an average score on both name and address of above .5 to perform their own clerical review? Values include ‘‘True’’ and ‘‘False’’. Outputs **outputdir**/clerical_review_candidates_DATE.csv file.

- **trainingdatafile** :The file name, in the name MAMBA directory, of the training data you wish to use.

- **scoringcriteria** : Which scoring criteria does the user wish to select as the model selection strategy? Possible values include (but not limited to): ‘neg_log_loss’, ‘precision’, ‘recall’, ‘f1’. See the`http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter` for further descriptions of each of the methods, as well as other scoring options.

All of these variables are loaded into the .bash sessions’ environment, which the SAS and Python programs inherit. The .bash shell is designed so the programs run in sequence, holding until the previous script has completed. The next section will discuss each of these scripts in turn.

D Programs run by MAMBA

D.1 standardizedata.sas

This program performs three main functions. Firstly, it generates a file based on the Business Register SSL files. Each row of this output file contains a record for the unique combination of the variables `name1`, `name2`, and the concatenation of the two for both mail and physical address. This results in approximately 69 million records between 2002 and 2015, however the start and end years are programmable by the researcher with the `brstartyear` and `brendyear` environment variables defined in `MAMBA` program. The program then calculates the first and last year that each `empunit_id_char` appears at any one particular address. Secondly, it performs a series of standardizations to remove street identifiers (e.g. street, avenue) and correct for common abbreviations. Finally, it performs the same standardizations on the input data.

D.1.1 Called Programs

- fixstreet.sas: corrects and standardizes street addresses.
- dropsuffix.sas: drops various company suffixes (e.g. CO and INC)
- cleandata.sas: drop and clean common abbreviations and spelling errors from input names.

D.2 create_db.py

This program creates an `SQLITE` database for the main program. Given the size of the data, and the requirement to load chunks independently of one another, the SQL interface in Python provided an easy, flexible way to have multiple workers access a single datasource simultaneously. The program itself creates a database with three tables, one for the Business Register 'wide' version, one for the 'long' version of the BR which accounts for word tokens used as filters for the state and all blocks, and another for the input data. The file also creates the blocking indexes for state and non-geographic match blocks on the BR file.

D.3 MatchFunction.py

This file is the main heart of the MAMBA procedure. These programs executes the match function and creates the predicted matches, outputting the `allmatchDATE.csv`, `bestmatchDATE.csv`,

`clerical_review_candidatesDATE.csv` files in the main folder, discussed below. The major functions executed are:

D.3.1 Runner:

This creates an object to serve as a hub for workers, so we can share common parameters across our workers that are operating on separate nodes. A runner is created for each 'block', allowing for full flexibility on the parameters fed to the workers. Runner objects execute the following functions.

D.3.2 worker

Each worker object performs one of two tasks: it either retrieves a chunk of input and BR data from the database and calculates scores for all of the candidate match pairs, then feeds any candidate pair with an average score, across all string comparators, of greater than .5, or if the names scored an average of .8 or greater. Alternatively, it retrieves a list of candidate pairs and generates predictions based on the random forest.

D.3.3 writer

Due to limitations of the sqlite database, only one write command at a time is permitted. We compensate for this by dedicated a single cpu to perform all write commands to the database, including both predicted matches and clerical review candidates. If there are no chunks of data to write, it generates a series of match candidates similar to a regular worker.

D.3.4 run_geog_block, run_state_block, run_all_block

These functions create the runner for either geographic or to match against the entire BR. It creates the blocking indexes for each of the blocks, before creating a Runner object to execute the matches.

D.3.5 Called Programs

- febrl_methods_cython.py: This file creates the 'cythonized' version of the febrl methods. Cython is a C-like interface for Python, which substantially reduces runtime. Creates several additional files with suffixes .pyx or .c. Also included in the repository are the setup files.

- addressonly_functions.pyx: This file contains the cythonized matched functions for address only matching.

- nameonly_functions.pyx: This file contains the cythonized matched functions for address only matching.
- nameaddress_functions.pyx: This file contains the cythonized matched functions for address only matching.

E Other Programs

In the /programs directory, we also include two additional programs.

E.1 create_clericalrev_sample.py

This program creates a clerical review sample based on the candidates generated from the `clericalreview` option above. Note this does not occur automatically. This program takes the candidates and generates a weighted sample, based on average score, on which the researcher can produce clerically-reviewed matches.

E.2 review_tool_nameaddress.py

This program is a GUI for clerical review, allowing the researcher to easily identify clerical review matches for NAME and ADDRESS matching. The program runs by allowing the researcher to directly label matches and non-matches, creating an output .csv file of reviewed matches. To use, simply open Python in a terminal interactive session (/apps/Anaconda/bin/python) and follow the instructions.

E.3 review_tool_nameonly.py

This program is a GUI for clerical review, allowing the researcher to easily identify clerical review matches for NAME ONLY matching. The program runs by allowing the researcher to directly label matches and non-matches, creating an output .csv file of reviewed matches.To use, simply open Python in a terminal interactive session (/apps/Anaconda/bin/python) and follow the instructions.

F Generating Your Own Training Data

In some (many) cases, you will want to develop your own training data. In order to do this, you will want to turn the `clericalreview` variable to 'True' and the `prediction` variable

to 'False'. Then, the program will run, generating scores on a 5% sample of your input data. You can then run the review_tool.py file to generate your predictions, and replace the `trainingdata` command with the file name (do not include the .csv ending) of your training data. If you do create your own training data, the program automatically fits a Random Forest to maximize your **scoringcriteria** through a parametrized grid search.

G The Output Files

The process creates three possible output files, for the DATE that you put into the .bash file. For the allmatch and bestmatch files below, you are returned an empunitid-inputid pair, with the first and last years that that match is valid in the Business Register. The user can then take this information, however they wish, to match to the BR.

G.1 allmatchDATE.csv

This file contains ALL input-BR establishment pairs predicted to be a match by the random forest. Included are the predicted probabilities and the number of matches (MatchCount) per inputid.

G.2 bestmatchDATE.csv

This file contains the 'best' matches for each input id according to the random forest. Included are the counts of matches (MatchCount) and the predicted probability for the match. Note that in the unlikely event of a tie, the record with the longer name on the Business Register is selected.

G.3 clerical_review_candidatesDATE.csv

This file is a collection of matches that have not been run through the model. They have a average score of above .5 on whatever criteria you select. For name and address matching, BOTH scores have to be above .5. Use this to feed into the create_clericalrev_sample.py program to generate your training data.