*International Journal of Geographical Information Science*
Vol. 00, No. 00, Month 2009, 1–25

## RESEARCH ARTICLE

# Toponym Matching Through Deep Neural Networks

Rui Santos[a*] , Patricia Murrieta-Flores[b] , Pável Calado[a] and Bruno Martins[a]

[a]*Instituto Superior Técnico - INESC-ID, University of Lisbon,Portugal*;
[b]*Digital Humanities Research Centre, University of Chester, United Kingdom*

(*Received 00 Month 200x; final version received 00 Month 200x*)

Toponym matching, i.e. pairing character strings that represent the same real-world location, is a fundamental problem in the context of several applications related to geographical information retrieval and to the geographical information sciences. Handling unofficial and/or historical variants of place names, often quite dissimilar between them, or handling transliterations between languages, for instance involving different scripts, are some of the challenges when matching toponyms. The current state-of-the-art relies on string similarity metrics, either specifically developed for matching place names, or integrated within methods that combine multiple metrics. Still, these methods all share a performance plateau, in the sense that they rely on common sub-strings in order to establish similarity relationships. Existing methods do not effectively capture the character replacements involved in toponym changes due to transliterations or to changes in language and culture over time. In this article, we present a novel toponym matching approach, leveraging a deep neural network to classify pairs of toponyms as either matching or non-matching. The proposed approach uses Gated Recurrent Units (GRUs) — a type of recurrent neural network architecture that can be used for modeling sequential data — to build representations from the sequences of bytes that correspond to the strings that are to be matched. These representations are then combined and passed to feed-forward nodes, finally leading to a classification decision. The entire model can be trained end-to-end with a set of labeled toponym pairs. We present the results of a wide-ranging evaluation on the performance of the proposed method, using a large dataset collected from the GeoNames gazetteer. These results show that the proposed method can significantly outperform individual string similarity metrics used in previous studies, as well as previous methods based on supervised machine learning for combining multiple similarity metrics.

**Keywords:** toponym matching; duplicate detection; approximate string matching; deep neural networks; recurrent neural networks; geographic information retrieval

[*] Corresponding author. Email: ruipdsantos@tecnico.ulisboa.pt

ISSN: 1365-8816 print/ISSN 1362-3087 online
© 2009 Taylor & Francis
DOI: 10.1080/1365881YYxxxxxxxx
http://www.informaworld.com

## 1. Introduction

Toponym matching refers to the task of pairing character strings that represent the same real-world location. For example, the names *Bahliz* and *Paris* should be paired, since both refer to Paris, the capital city of France. Toponym matching is, thus, a fundamental problem in the context of several applications related to geographical information retrieval and the geographical information sciences, such as conflation of digital gazetteers or point-of-interest datasets (Hastings 2008, Zheng *et al.* 2010, Martins 2011, McKenzie *et al.* 2014, Moura *et al.* 2017), address parsing within geocoding and map search services (Joshi *et al.* 2008, Berkhin *et al.* 2015), or toponym resolution over textual contents (Anastácio *et al.* 2009, Santos *et al.* 2015, Monteiro *et al.* 2016), digitized maps (Weinman 2013, Simon *et al.* 2014), and digital library contents (Freire *et al.* 2011).

A standard approach to this problem involves computing a string similarity metric between the toponyms that are to be matched. Usually, an edit distance (Levenshtein 1966, Damerau 1964) or an heuristic such as the Jaro-Winkler metric (Winkler 1990) is used to compute the similarity, and then a decision is taken based on a threshold over the similarity value. While relatively effective, these methods require a laborious tuning of the similarity threshold. Previous research, either focusing on toponym matching (Recchia and Louwerse 2013) or on the related problem of person name matching (Cohen *et al.* 2003, Christen 2006, Moreau *et al.* 2008, Varol and Bayrak 2012), has in fact suggested that the performance of different string similarity algorithms is task-dependent, and that there is no single overall best technique.

More recent research in this area has proposed the usage of supervised machine learning as an effective approach for combining multiple string similarity metrics (Santos *et al.* 2017). Still, even the combination of several string similarity metrics will fail to properly address some of the hard challenges involved in toponym matching, such as handling unofficial and/or historical variants of the same place names, often quite dissimilar between them, or handling transliterations between languages, for instance involving different scripts. This occurs because existing string similarity metrics rely on common character sub-strings in order to establish the semantic similarity relationships, thus not effectively capturing the character replacements that are often involved in transliterations, or that are related to changes in language usage over time.

In this article, we present a novel toponym matching approach that can better deal with the aforementioned challenges. Our solution leverages a deep neural network architecture, with parameters learned from training data (Goodfellow *et al.* 2016), to classify pairs of toponyms as either matching or non-matching. The network uses Gated Recurrent Units (GRUs), a type of recurrent neural network architecture for modeling sequential data that was originally proposed by Chung *et al.* (2014), to build representations from the sequences of bytes that correspond to the strings that are to be matched. These representations are then combined and passed to a sequence of feed-forward nodes, finally leading to a classification decision. The entire model can be trained end-to-end through the back-propagation algorithm in conjunction with the Adam optimization method (Kingma and Ba 2015), provided access to a training set of labeled toponym pairs.

Besides advancing this novel method, we also present the results of a wide-ranging comparative evaluation on the performance of toponym matching methods, leveraging a very large dataset collected from the GeoNames[1] gazetteer. Our experimental data consists of 5 million toponym pairs, obtained from the lists of alternative place names

[1] http://www.geonames.org

that are associated to each record in the gazetteer. Results show that the proposed method can significantly outperform thirteen different individual string similarity metrics used in previous studies, as well as previous methods based on supervised machine learning for combining the results of multiple string similarity metrics.

The remainder of this article is organized as follows: Section 2 presents fundamental concepts (e.g., string similarity metrics that are frequently used for toponym matching) and previous research in the area. Section 3 describes the proposed deep learning method, detailing the neural network architecture that was used for classifying toponym pairs, and discussing the model training procedure. Section 4 presents the experimental evaluation, detailing the general protocol, dataset and evaluation metrics, and also presenting and discussing the obtained results. Finally, Section 5 concludes the article with a summary of the most interesting findings and with possible paths for future work.

## 2. Related Work

Most previous studies on toponym matching have relied on methods that involve computing a string similarity metric between the toponyms that are to be matched, and then taking a decision based on a threshold over the similarity value. The literature on approximate string matching and on string comparison metrics is, in fact, quite extensive — see Cohen *et al.* (2003) for a comprehensive review — but traditional methods can roughly be separated into three classes: character-based, vector-space based, and hybrid approaches. Character-based methods rely on character edition operations, such as deletions, insertions, substitutions and sub-sequence comparisons. Vector-space methods transform strings into vector representations, over which similarity computations are performed. Hybrid approaches combine both ideas, in order to improve effectiveness when matching names composed of multiple tokens. Section 2.1 overviews these traditional methods, while Section 2.2 presents previous work specifically focusing on toponyms.

### 2.1. *Traditional Methods for Approximate String Matching*

The best know string matching method is, perhaps, the Levenshtein (1966) edit distance metric. It is a character-based approach that represents the minimum number of insertions, deletions or substitutions needed to transform a string $t_1$ into another string $t_2$. For example, the edit distance between the toponyms *Lisboa* and *Lisbonne* is three, corresponding to one substitution and two insertions, namely $a \mapsto n$, $\epsilon \mapsto n$ and $\epsilon \mapsto e$. The distance metric can be computed through a dynamic programming algorithm, and a corresponding normalized string similarity measure can defined as $s(t_1, t_2) = 1 - \frac{d(t_1,t_2)}{\max(|t_1|,|t_2|)}$, where $d(t_1, t_2)$ corresponds to the Levenshtein edit distance between strings $t_1$ and $t_2$, with lengths $|t_1|$ and $|t_2|$, respectively. Many variants and/or improvements have been proposed (Navarro 2001), including the approach by Damerau (1964), in which one basic edit operation is added (i.e., the transposition of two characters). Other approaches allow for contiguous sequences of mismatched characters (i.e., affine gaps) in the alignment of two strings (Needleman and Wunsch 1970), or they instead correspond to adaptable approaches that learn appropriate weights for the different edit operations (Ristad and Yianilos 1998, Brill and Moore 2000, Bilenko and Mooney 2003b).

The Jaro metric is another example of a character-based string similarity metric, based on the number and order of common characters. This metric is specifically designed for matching short strings, such as person names (Jaro 1989). The heuristic procedure associated to the computation of the Jaro metric is based on the number of characters in common between the strings being compared that are located in similar positions, i.e. the difference between their positions should be no more than half the length of the longer string. In addition, the Jaro metric takes character transpositions into consideration. A refined version, advanced by Winkler (1990), extends the Jaro metric with a prefix scale that gives higher scores to strings that match from the beginning up to a given prefix length. More recently, noting that the Jaro-Winkler approach can be problematic if the strings to be matched contain multiple words that are differently ordered (e.g., when matching the toponyms *Madeira Island* and *Island of Madeira*), Christen (2006) proposed two variants named (i) sorted Winkler and (ii) permuted Winkler. The former algorithm sorts the tokens that compose both strings before calculating their Jaro-Winkler similarity, while the latter calculates the similarity over all possible token permutations and returns the maximum value.

On what concerns vector-space approaches, computing the cosine similarity metric between representations based on character $n$-grams, (i.e., based on sequences of $n$ consecutive characters, typically with $n = 2$ or $n = 3$) is a common approach. Different variants of the cosine metric have been used in practice (Cohen *et al.* 2003, Moreau *et al.* 2008), for instance leveraging binary representations for the occurrence of $n$-grams, versus using the common information retrieval term weighting heuristic known as Term Frequency times Inverse Document Frequency (TF-IDF). The Jaccard or the Dice similarity coefficients (Jaccard 1912, Dice 1945), when computed between sets of character $n$-grams occurring in the strings to be compared, are also commonly employed. Instead of regular $n$-grams, some previous studies have suggested that skip-grams, i.e. bi-grams of non-adjacent letters, considering gaps of zero, one, or two characters, are also an effective choice for representing the strings to be matched (Keskustalo *et al.* 2003).

Hybrid metrics combine the advantages of character-based and vector-spaced approaches, being flexible about word order and position, while still allowing for small differences in word tokens. Most of these methods are based on applying a sub-measure to all pairs of word tokens between the two strings (i.e., they rely on second-level measures), and then computing a final score based on these values. For instance, the scheme proposed by Monge and Elkan (1996) involves computing the average similarity between the most similar pairs of word tokens, according to a sub-measure such as the Jaro-Winkler similarity. Procedures such as the cosine similarity or the Jaccard coefficient can also be used as hybrid approaches, considering sets of tokens instead of sets of $n$-grams, and softening the metrics by allowing for small mismatches (e.g., by applying a threshold over the results of an inner similarity metric) when aligning the individual word tokens (Cohen *et al.* 2003, Moreau *et al.* 2008).

It is interesting to note that, in addition to the aforementioned metrics based on character operations, vector-space representations, or hybrids, some previous studies have also proposed to leverage phonetic encoding techniques (Christen 2006, Varol and Bayrak 2012). Phonetic techniques attempt to convert a string into a code that captures the way the words are pronounced. Strings with words encoded into the same representation can then be said to match. However, the conversion process is language-dependent, and most of the designed techniques, including classic approaches like Soundex or more recent methods such as Double Metaphone (Philips 2000), have been developed based exclusively on the Latin alphabet and the English phonetic structure.

### 2.2. *Previous Work Focusing on Toponym Matching*

Recchia and Louwerse (2013) reported on a comparison of different string similarity measures, such as those previously described, over a toponym matching task. For the comparison, romanized toponyms from different countries, taken from the GEOnet Names Server, had to be matched against alternative names for the same toponyms. The authors found that, in general, methods that rely on the number of shared short sub-strings (i.e., bigrams, tri-grams, and skip-grams in particular) tend to perform well, although they also observed a substantial variation in the methods that worked best over datasets from different countries. For instance, the best-performing algorithms on toponyms from China and Japan were Jaro variants, whereas edit distance performed best on toponyms from Taiwan, and $n$-gram based measures worked best on countries with toponyms in Romance and Germanic languages. As a general recommendation, the authors argue that practitioners should test several algorithms on a country-specific or language-specific dataset, and use the best-performing algorithm for future developments involving similar data.

Hastings and Hill (2002) have noted that the standard similarity metrics are not particularly well suited to toponym matching because, in everyday usage, the stylistic variability of places names is simply too great. Authors like Davis and De Salles (2007), Hastings (2008), or Kilinç (2016) have specifically designed methods for matching toponyms, in most cases corresponding to variations of the aforementioned classic procedures, and often leveraging some form of canonical representation for toponyms.

For instance, taking inspiration from previous work by Fu *et al.* (2005), Hastings (2008) proposed an algorithm that relies on token matching, using a language-dependent stemming procedure that attempts to reduce word tokens to canonical base forms. This algorithm essentially matches the occurrence of lower-cased tokens from one toponym, excluding place-type terms and common stop-words, against the lower-cased and possibly stemmed tokens of the other toponym, and vice-versa. Common abbreviations are expanded, and common misspellings are also corrected while reducing word tokens to canonical forms. For each token pairing between the two strings, a match score of zero, one-quarter, one-half, or one is assigned, giving quarter-weight to infix matches, half-weight to prefix/stemmed matches, and full-weight to exact matches. The accumulated bi-directional score is finally normalized by the total number of tokens in the two strings, to account for unmatched tokens.

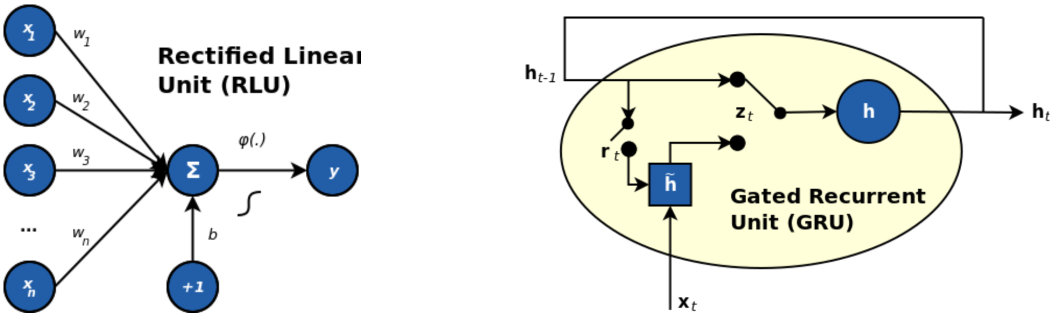
The methods proposed by Davis and De Salles (2007) and by Kilinç (2016) instead correspond to hybrid approaches. For example, in the method by Davis and De Salles (2007), the first step involves dividing the toponyms into tokens, using blanks, hyphens, and other similar symbols as delimiters. For matching individual tokens, Davis and De Salles proposed a variation of the Levenshtein edit distance that incorporates a practical scheme for matching accented and special characters — characters are organized into equivalence groups, so that characters belonging to the same group are considered to match. Two tokens are considered to match if their similarity is above a pre-defined threshold of $\alpha = 0.75$. The complete matching strategy involves four distinct phases, namely (1) replacing tokens that are known abbreviations by their full spelling (e.g., *avn.* would be replaced by *avenue*), (2) capturing non-standard abbreviations in one of the strings, namely single-character capitalized tokens and tokens that end with a dot character, by checking them against tokens in the other string that share a common prefix, (3) token alignment, using a procedure similar to that of the Levenshtein edit distance in order to align the tokens from one string against those from the other, and (4) computing a final similarity score through a linear combination of three different metrics between sets of tokens, one of which accounting for possible token inversions.

Several previous studies have also proposed heuristic combinations of different similarity metrics, computed over attributes such as place names, place types, and/or geospatial footprints, in order to match gazetteer records (Fu *et al.* 2005, Smart *et al.* 2010, Li *et al.* 2016). Taking inspiration from some of these studies, other authors have proposed to use supervised learning for gazetteer record conflation, using classifiers as a principled approach to combine multiple similarity metrics, computed over different types of attributes (Sehgal *et al.* 2006, Zheng *et al.* 2010, Martins 2011, McKenzie *et al.* 2014). For instance Martins (2011) found that string similarity was the most informative type of feature when detecting duplicate gazetteer records with a method based on the supervised training of a classifier leveraging the formalism of Support Vector Machines (SVMs). The same study also reported that $n$-gram overlap metrics, Jaro-Winkler, and variations of edit distance were particularly useful. Sehgal *et al.* (2006) found that edit distance outperformed both a Jaccard $n$-gram coefficient and Jaro-Winkler, when mapping between two sets of romanized place names from Afghanistan, also with a method based on SVMs. In a problem of matching points-of-interest described in two different location-based social networks, McKenzie *et al.* (2014) showed that the Levenshtein edit distance between names performed the best, within a set of independent methods that also included geographic distance and matches in categories and in long textual descriptions.

A recent study, leveraging the same dataset that is used in the experiments reported on this article, presented a comparison of 13 different string similarity metrics over a toponym matching task (Santos *et al.* 2017). This article has also reported on experiments with the usage of supervised machine learning for combining the multiple similarity metrics, this way avoiding the manual tuning of similarity thresholds. The results showed that the performance differences between the individual similarity metrics were relatively small, but that carefully tuning the similarity threshold is crucial for achieving good results. Methods based on supervised machine learning for combining the similarity metrics, particularly when considering ensembles of decision trees, achieved good results, significantly outperforming the individual similarity metrics. However, given that the individual metrics were always based on common sub-sequences of characters, several pairs of matching toponyms involving complex transformations (e.g., transliterations between different alphabets) could not be correctly detected, thus motivating the development of approaches such as the one reported on this article.

## 3. The Proposed Approach

In contrast to most previous work on toponym matching, which leveraged individual string similarity metrics or combinations of multiple metrics with parameters inferred through supervised learning (Santos *et al.* 2017), in this article we propose a novel approach, which leverages the supervised training of a deep neural network that can directly model the pair of character strings under comparison. A large set of toponym pairs, known to refer to the same real world place or not, is used to infer the parameters of the neural network. This network takes toponym pairs (i.e., two sequences of bytes) as input, and outputs a binary classification decision. After training, the classifier can be used to decide if any new pair of strings refers to the same location or not. This section provides a brief introduction to neural networks and deep learning, afterwards describing the specific architecture that we propose for toponym matching.



(a) (b)

Figure 1.: Diagrams illustrating (a) a Rectified Linear Unit (i.e., a simple perceptron with a particular activation function), and (b) a Gated Recurrent Unit.

## 3.1. *Artificial Neural Networks for Data Classification*

Artificial neural networks are computational artifacts that channel information through a series of mathematical operations, performed at the nodes of the network (Goodfellow *et al.* 2016). Mathematically, neural networks can be seen as nested composite functions. Their general purpose is to accurately classify inputs, and we have that all the parameters in the nested composite functions can be trained directly to minimize a given loss function computed over the outputs and the expected results. This is achieved through a training procedure known as back-propagation, in combination with gradient descent optimization of the parameters (Goodfellow *et al.* 2016).

### *3.1.1. Feed-Forward Neural Networks*

The individual nodes of an artificial neural network are often referred to as *perceptrons*, and the basic concept of using a single perceptron for classifying data patterns was introduced by Rosenblatt (1958). A perceptron (i.e., a single node neural network) computes a single output from multiple real-valued inputs by forming a linear combination according to input weights and then putting the output through some activation function. Mathematically, this can be written shown as in Equation 1, and a graphical depiction of a perceptron is shown in Figure 1(a).

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi\left(\mathbf{w}^T\mathbf{x} + b\right) \tag{1}$$

In the equation, $y$ is the returned prediction, $\mathbf{x} =< x_1, \ldots, x_n >$ is the vector of inputs, $\mathbf{w}$ denotes the vector of weights, $b$ is a bias term, and $\varphi(.)$ is the activation function.

The original perceptron, as proposed by Rosenblatt (1958), used a threshold step function as the activation function $\varphi(.)$, although nowadays, and especially in multilayer networks, the activation function is often chosen to be the logistic sigmoid, the hyperbolic tangent, or a rectified linear function (i.e., the ramp function $\varphi(x) = \max(0, x)$). Perceptrons leveraging a rectified linear activation function are often referred to as Rectified Linear Units (RLUs) and they are nowadays frequently used within more complex deep neural network architectures (Goodfellow *et al.* 2016).

Although a single perceptron has a limited mapping ability (i.e., a standard perceptron can only represent linear decision functions), perceptrons can be used as building blocks of more complex models. For instance, a MultiLayer Perceptron (MLP) consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer, until it reaches the output node(s). In the case of one such feed-forward network with a single hidden layer, the corresponding computations can be written as shown in Equation 2, and the generalisation to more hidden layers would be simple.

$$\mathbf{y} = \varphi\left(\mathbf{B}\boldsymbol{\varphi'}(\mathbf{A}\mathbf{x} + \mathbf{a}) + \mathbf{b}\right) \tag{2}$$

In the previous equation, $\mathbf{x}$ is a vector of inputs and $\mathbf{y}$ a vector of outputs. The matrix $\mathbf{A}$ represents the weights of the first layer and $\mathbf{a}$ is the bias vector of the first layer, while $\mathbf{B}$ and $\mathbf{b}$ are, respectively, the weight matrix and the bias vector of the second layer. The functions $\boldsymbol{\varphi'}$ and $\varphi$ both denote an element-wise non-linearity, i.e. the activation functions respectively associated to nodes in the hidden layer, and in the output layer.

Training the neural network corresponds to adapting all the weights and biases (e.g., the parameters $\mathbf{A}, \mathbf{B}, \mathbf{a}$ and $\mathbf{b}$, in the case of the feed-forward network expressed in the previous equation) to their optimal values, given a training set of inputs $\mathbf{x}$ together with the corresponding outputs $\mathbf{y}$. This problem can be solved with the back-propagation algorithm, which consists of two steps. In a forward pass, the predicted outputs corresponding to the given inputs are evaluated. In a backward pass, partial derivatives (i.e., the relationships between rates of change) of a given loss function with respect to the different parameters are propagated back through the network. In other words, back-propagation in neural networks moves backward from the final error through the outputs, weights and inputs of each layer, assigning those weights responsibility for a portion of the error, by calculating their partial derivatives.

The chain rule of differentiation can be used to compute the derivatives associated to nested composite functions. Those derivatives are used by a gradient-based optimisation algorithm to adjust the weights and biases up or down, whichever direction decreases error over the training instances, as measured through a loss function. An optimisation procedure that has been frequently used to train deep neural networks is the Adaptive Moment Estimation (Adam) algorithm from Kingma and Ba (2015). Adam computes parameter updates leveraging an exponentially decaying average of past gradients, together with adaptive learning rates for each parameter. In practice, it performs larger updates for infrequent parameters, and smaller updates for frequent parameters.

### *3.1.2. Recurrent Neural Networks*

Recurrent neural networks (RNNs) constitute an extension of conventional feed-forward networks, with the objective of handling variable-length input sequences (i.e., they were designed to recognize patterns in sequences of data such as character strings). An RNN handles a variable-length sequence by having a recurrent hidden state whose activation at each time step is dependent on that of the previous time step. Whereas in classic feed-forward networks the examples are fed to an input layer and straightly transformed into an output layer, never performing computations over a given node twice, in RNNs we take not just the current input instance (e.g., the representation for a given character within a string) but also what was perceived one step back in time (i.e., the previous character in the sequence).

Recurrent neural networks thus consider two sources of input, i.e. the present and the recent past, which are combined to determine how the network should respond to new data. RNNs have a chain-like structure corresponding to a feedback loop, as they ingest their own outputs moment after moment as part of the input, and preserve sequential information in a hidden state, which manages to span many time steps as it cascades forward to affect the processing of each new example. More formally, given a sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, an RNN updates its recurrent hidden state $\mathbf{h_t}$ by sequentially processing the input sequence and computing:

$$\mathbf{h_t} = \varphi\left(\mathbf{W}\mathbf{x_t} + \mathbf{U}\mathbf{h_{t-1}}\right) \tag{3}$$

In brief, we have that the hidden state $\mathbf{h_t}$ at time step $t$ is a function of the input at the same time step $\mathbf{x_t}$, modified by a weight matrix $\mathbf{W}$. This result is added to the hidden state of the previous time step $\mathbf{h_{t-1}}$, multiplied by its own hidden-state-to-hidden-state matrix $\mathbf{U}$, otherwise known as a transition matrix. The weight matrices are essentially filters that determine how much importance should be given to both the present input and the past hidden state. In a way, RNNs can be seen as a generalization of Markov chains, where the hidden states are supposedly encoding all previous history in the sequence.

Previous research has noted that standard RNNs have difficulties in modeling long sequences, and extensions have been proposed to handle this problem. Two well-known examples are Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber 1997) and Gated Recurrent Units (GRUs), originally proposed by Chung *et al.* (2014). GRUs involve different components, i.e. gating mechanisms, which interact in a particular way and according to Equation 4. A graphical depiction of a GRU is shown in Figure 1(b).

$$\begin{aligned}
\mathbf{z_t} &= \varphi_\mathbf{g}\left(\mathbf{W_z}\mathbf{x_t} + \mathbf{U_z}\mathbf{h_{t-1}} + \mathbf{b_z}\right)\\
\mathbf{r_t} &= \varphi_\mathbf{g}\left(\mathbf{W_r}\mathbf{x_t} + \mathbf{U_r}\mathbf{h_{t-1}} + \mathbf{b_r}\right)\\
\tilde{\mathbf{h}}_\mathbf{t} &= \varphi_\mathbf{h}\left(\mathbf{W_h}\mathbf{x_t} + \mathbf{U_h}(\mathbf{r_t} \circ \mathbf{h_{t-1}}) + \mathbf{b_h}\right)\\
\mathbf{h_t} &= \mathbf{z_t} \circ \mathbf{h_{t-1}} + (1 - \mathbf{z_t}) \circ \tilde{\mathbf{h}}_\mathbf{t}
\end{aligned} \tag{4}$$

In Equation 4, the operator $\circ$ denotes the Hadamard product (i.e., the entry-wise product of two matrices), while $\mathbf{x_t}$ denotes the input vector at time step $t$, and $\mathbf{h_t}$ denotes the hidden state at time step $t$. The parameters $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ denote the different weight matrices and bias vectors, adjusted when training the model.

Notice that a GRU involves two gates, namely a reset gate $\mathbf{r}$, that determines how to combine the new input with the previous memory, and an update gate $\mathbf{z}$, that defines how much of the previous memory to keep around. If we set the reset gate to all ones, and the update gate to all zeros, we again arrive at the plain RNN model that was discussed previously. The gating mechanism allows GRUs to better handle long-term dependencies. By learning the parameters for its gates, the network learns how its internal memory should behave, given that the gates define how much of the input and previous state vectors should be considered.

Recurrent neural network units such as GRUs can be used to model sequential data (i.e., for encoding a given input sequence into a vector representation, given by the hidden state after processing the last position in the sequence, which can then be processed by other network nodes), or for generating new sequences (e.g., by processing the hidden state generated at each position of the input sequence). In our case, we use GRUs within a deep model that considers multiple layers of recurrent and feed-forward processing.

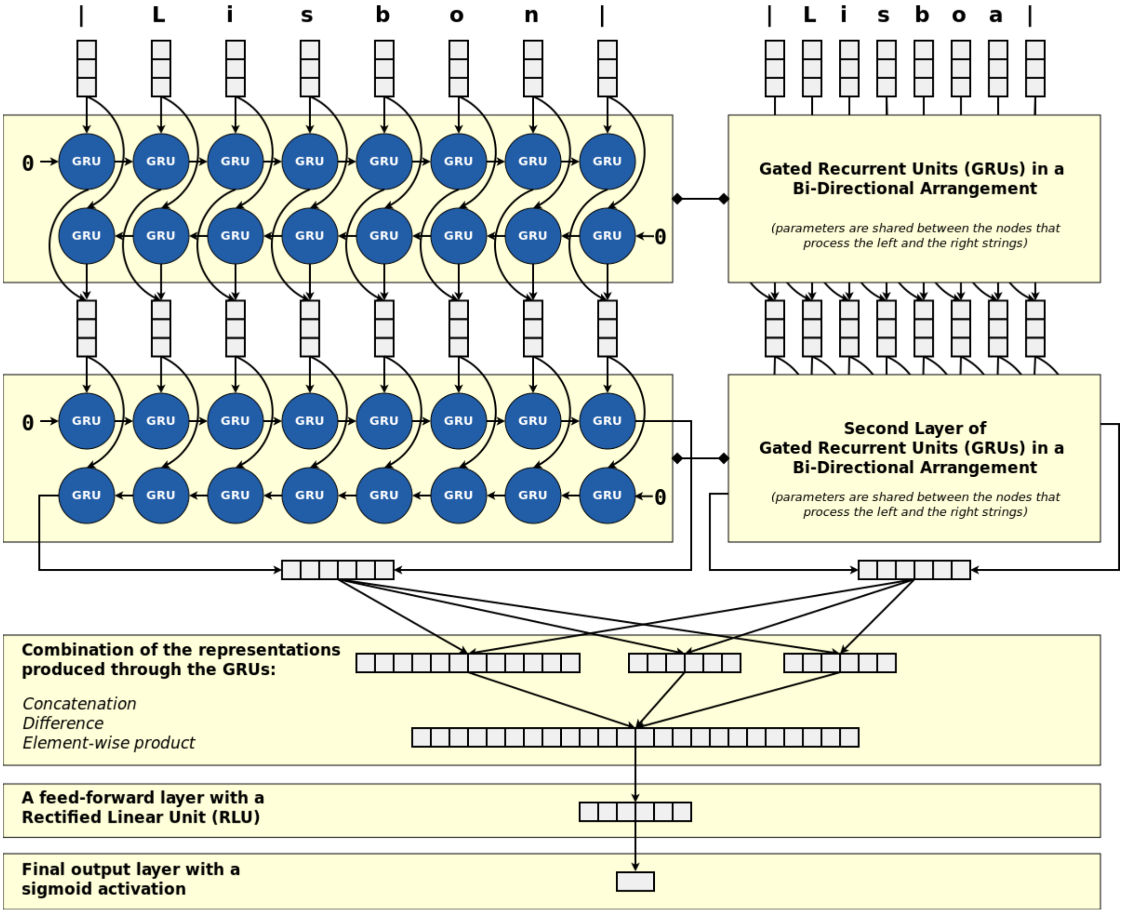

| L i s b o n |
| L i s b o a |
0
GRU
Gated Recurrent Units (GRUs) in a Bi-Directional Arrangement
(parameters are shared between the nodes that process the left and the right strings)
Second Layer of Gated Recurrent Units (GRUs) in a Bi-Directional Arrangement
(parameters are shared between the nodes that process the left and the right strings)
Combination of the representations produced through the GRUs:
Concatenation
Difference
Element-wise product
A feed-forward layer with a Rectified Linear Unit (RLU)
Final output layer with a sigmoid activation

Figure 2.: The neural network architecture proposed to address toponym matching.

## 3.2. *Toponym Matching Using a Deep Neural Network*

The neural network architecture proposed in this article for addressing the toponym matching problem, where recurrent nodes are perhaps the most important components, is illustrated in Figure 2. This architecture takes its inspiration on previously proposed models for natural language inference and for computing sentence similarities (Bowman *et al.* 2015, Yin *et al.* 2015, Liu *et al.* 2016, Mueller and Thyagarajan 2016).

The input to the network are two sequences of binary vectors that represent the strings to be compared. The strings are first converted to a unicode canonical normalized format (i.e., we use a fully decomposed UTF-8 representation, in which all combining character marks are placed in a pre-specified order) and they are also padded with a special symbol that denotes the beginning and termination of the toponym. The normalized strings are then represented as a sequence of one-hot binary vectors, in which a single bit is set to one for each byte in the sequence that corresponds to the unicode normalized string.

The binary vectors are provided as input to bi-directional GRUs, which produce a vector of real values, also referred to as an *embedding*, for each of the toponyms being compared. Bi-directional GRUs work by concatenating the outputs of two GRUs, one processing the sequence from left to right and the other from right to left (Schuster and Paliwal 1997). Our neural network architecture actually uses two different layers of bi-directional recurrent units. The first bi-directional GRU layer generates a sequence of real-valued vectors, that is then passed as input to the second bi-directional GRU layer. The second bi-directional GRU layer outputs a single embedding for the input, resulting from the concatenation of the last outputs that are produced by the GRUs that process the input sequences in each direction.

Notice that each of the input toponyms is conceptually processed by individual recurrent layers, in order to produce a compact representation for its contents, although the parameters of these GRUs are shared across the parts of the network that process each input toponym. In other words, the two layers of bi-directional GRUs, which are used to process each of the input strings, have the same set of parameters. In the literature, these network architectures, where different parts have their parameters tied, are typically referred to as siamese networks (Mueller and Thyagarajan 2016).

The two embeddings produced by the bi-directional GRU layers are then combined and compared through a set of different operations. Taking inspiration on the network architecture from Mou *et al.* (2016), proposed to address the problem of natural language inference, we produce a combined representation for the pair of toponyms resulting from (i) the concatenation of the embedding vectors, (ii) the element-wise product of the embedding vectors, and (iii) the difference between the embedding vectors. This combined representation is then passed as input to feed-forward network layers: a first layer that uses a simple combination of the inputs together with a non-linear activation function (i.e., a rectified linear unit), followed by another simple layer that produces the final output and that uses a sigmoid activation function.

The entire network is trained end-to-end through back-propagation in combination with the Adam optimization algorithm, using binary cross-entropy as the loss function to be optimized. In order to control overfitting and improve the generalization capabilities of the classification model, we use dropout regularization with a probability of 0.1 between each layer of the proposed neural network architecture. Dropout regularization is a simple procedure based on randomly dropping units, along with their connections, from the neural network during training. Each unit is dropped with a fixed probability $p$ independent of other units, effectively preventing the network units from co-adapting too much (Srivastava *et al.* 2014). An initial set of experiments showed that using dropout regularization with the small probability of 0.1 indeed improved the results.

## 4. Experimental Evaluation

This section describes the dataset used for supporting the experiments, presents the experimental setup that was applied, and discusses the obtained results. Previous experiments with standard string similarity metrics show that cross-lingual toponym matching remains a significant challenge, given that the task requires encoding the intrinsic semantic value of sequences of characters, in order to support comparisons. On the other hand, deep neural network architectures, such as the one proposed in this article, have successfully been applied to difficult challenges involving modeling sequences of text, such as language modeling, machine translation, or even measuring semantic similarity. Through experiments, we aimed to assess if indeed one such model could improve performance over state-of-the-art toponym matching techniques, such as those report by Davis and De Salles (2007), Recchia and Louwerse (2013), or Santos *et al.* (2017). We used a Python

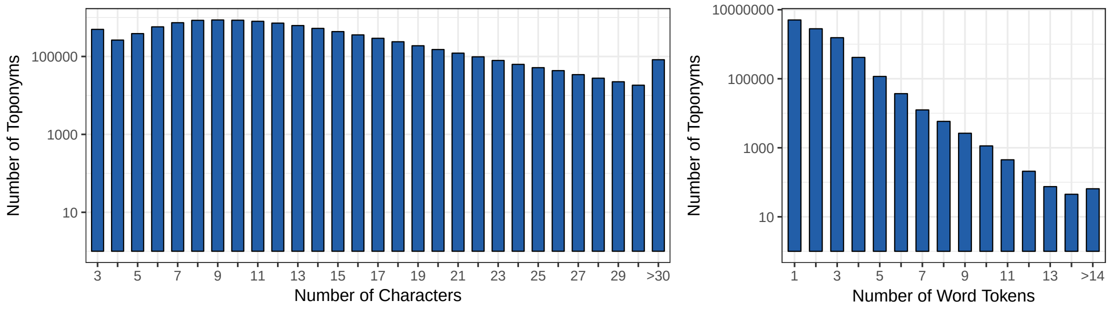


Figure 3.: Distribution for the length of the toponyms present in the dataset.

library named Keras[1] for implementing the neural network architecture introduced in Section 3, and then leveraged a large dataset collected from GeoNames to evaluate its performance. Keras was choosen over other deep learning libraries due to its high-level API, that already offers implementations for many of the components involved in the proposed architecture (e.g. GRUs and RLUs).

## 4.1. *Dataset And Experimental Methodology*

Our experiments relied on a dataset of five million pairs of toponyms, half of which corresponding to alternative names for a same place. This dataset, used already in previous work (Santos *et al.* 2017), was generated from lists of alternative place names associated to records in the publicly available GeoNames gazetteer. Each place that is described in GeoNames is associated to multiple names, often corresponding to historical denominations or to transliterations in multiple alphabets/languages. We can, thus, leverage this information to build a large dataset covering toponyms from all around the globe.

In our dataset, the matching pairs of toponyms correspond to alternative names for the same place, with more than two characters, that after converting all characters into their lower-cased equivalents do not match in every character. The non-matching pairs of toponyms correspond to names for different places, not necessarily within the same country, that also have more than two characters. In order to build a dataset that is both representative and challenging for automated classification, a significant portion of the non-matching pairs should not be completely dissimilar. According to this intuition, we preferred toponym pairs having a Jaccard similarity coefficient above zero — if the similarity between a non-matching pair of toponyms is equal to zero, we discard the pair with a probability of 0.75, when building the dataset.

Table 1 presents elementary characterization statistics for the dataset, which are further detailed in this section and illustrated in the accompanying figures. In Figure 3, we show the distribution for the length of the toponyms, both in the number of characters and the number of words. Figure 4 shows the distribution for the difference in the number of characters per matching and non-matching toponym pair, a quantity that is related to the difficulty in matching toponyms (i.e., matching pairs with a larger difference should be harder to classify). Figure 5 presents the geographical distribution of the dataset, showing the number of toponyms associated to particular regions (i.e., countries), and

[1] `http://keras.io`

Table 1.: Statistical characterization of the dataset used in our experiments.

| | |
|---|---|
| Number of toponym pairs | 5000000 |
| Number of matching toponym pairs | 2500000 |
| Number of pairs with toponyms from the same country | 4999260 |
| Number of pairs with equal toponyms after lowercasing and removing diacritics | 180453 |
| Number of pairs with matching equal toponyms after lowercasing and removing diacritics | 167933 |
| Number of pairs with matching toponyms that are completely dissimilar | 625754 |
| Number of pairs with non-matching toponyms that are completely dissimilar | 993409 |
| Average difference in number of characters per toponym pair | 3.74 |
| Average number of characters per toponym | 22.72 |
| Average number of word tokens per toponym | 3.59 |
| Number of characters in largest toponym | 188 |
| Number of pairs with both toponyms involving only Latin characters | 3320403 |
| Number of pairs with at least one toponym involving CJK characters | 625007 |
| Number of pairs with at least one toponym involving Cyrillic characters | 400860 |
| Number of pairs with at least one toponym involving Arabic characters | 390145 |
| Number of pairs with at least one toponym involving Thai characters | 221007 |
| Number of pairs with at least one toponym involving Greek characters | 25941 |
| Number of pairs with at least one toponym involving Armenian characters | 16361 |
| Number of pairs with at least one toponym involving Hebrew characters | 8503 |
| Number of pairs with at least one toponym involving Georgian characters | 4979 |
| Number of pairs with at least one toponym involving Devanagari characters | 2087 |

showing also the top 10 countries with the most toponyms. Finally, in Figure 6, we present the distribution for the number of toponym pairs per country (i.e., the number of pairs where both toponyms belong to the same country), focusing on the 10 countries with the most toponym pairs in our dataset. The same figure also shows the distribution for the number of toponym pairs per alphabet (i.e., the number of pairs where at least one of the toponyms only uses characters from a given alphabet). Authors like Recchia and Louwerse (2013) have noted that the performance of different string similarity algorithms varies according to the country, and that there is no single overall best technique.

Notice that most of the instances in our dataset correspond to toponyms in the Latin alphabet, in most cases composed of more than 3 words. Although we could have applied the same procedure to generate a bigger dataset from GeoNames records, handling 5 million instances is already quite challenging from a computational point of view. Our completely balanced dataset with 5 million instances is already showing some of the less

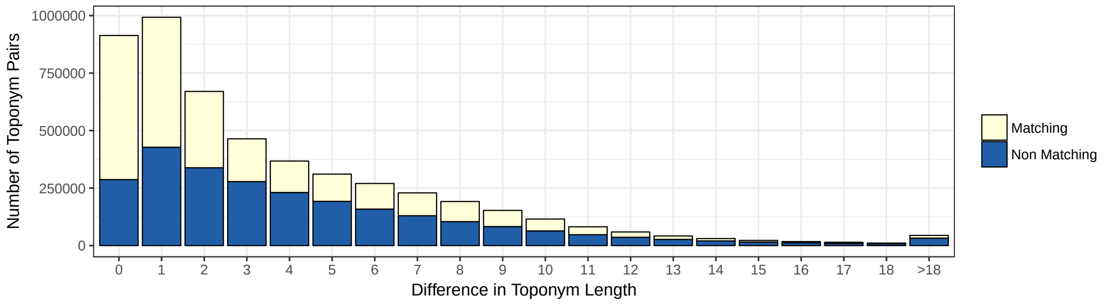


Figure 4.: Differences in the number of characters per matching and non-matching pair.

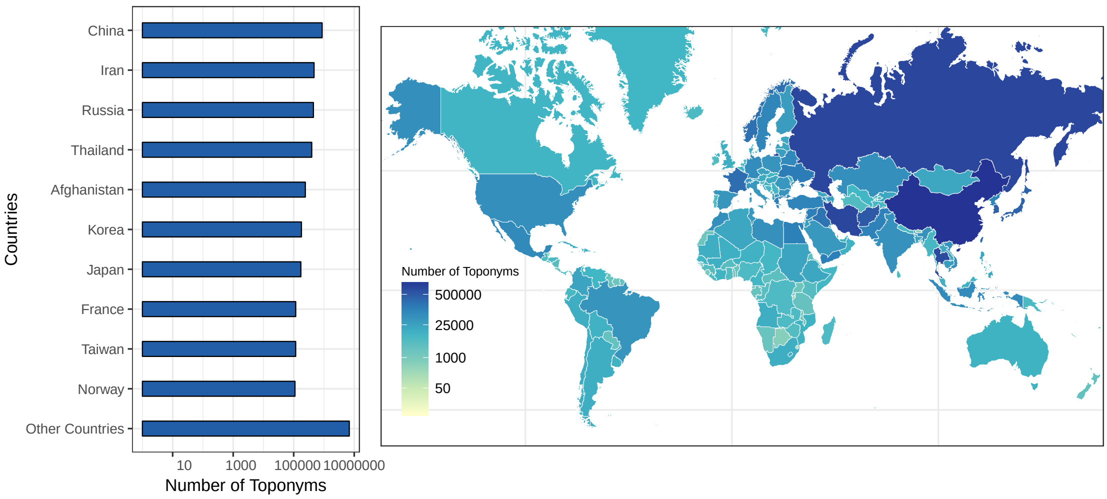


Figure 5.: Distribution for the number of toponyms per geographic region.

frequent combinations of alphabets and/or character transliterations.

A variety of experimental methodologies have been used to evaluate the effectiveness of methods for matching potential duplicates. Authors like Bilenko and Mooney (2003a) have advocated that standard information retrieval metrics, such as precision and recall, provide an informative evaluation methodology within duplicate detection studies, and previous work in the area has also used these metrics for evaluating toponym matching (Santos *et al.* 2017). In this work, we also use precision and recall to evaluate the quality of the proposed method, and complement these values with a detailed analysis of the cases for which the method produced correct and incorrect results.

Precision and recall are per-class metrics that focus on different aspects of quality for a classification method. Precision is the ratio formed by the number of items correctly assigned to a class divided by the total number of items assigned to that class. In a toponym matching problem with two possible classes, i.e. *match* versus *non-match*, precision for the *match* class corresponds to the number of matches that were correctly identified by the classifier, over the total number of matches (i.e., correctly plus incorrectly) identified by the classifier. Recall, on the other hand, is the ratio between the number of items correctly assigned to a class, over the total number of items in the class. Using the previous example, recall for the *match* class corresponds to the number of matches that were correctly identified, over the total number of correct matches in the dataset.

Since precision can be increased at the expense of recall, we also compute the F1-measure, which equally weights precision and recall through their harmonic mean. Finally, we also measured the quality of the results through the accuracy metric, which corresponds to the proportion of correct decisions (i.e., matches or non-matches) that were returned by the method under evaluation.

Leveraging the aforementioned dataset, we used a two-fold cross-validation methodology. This means that the available pairs of toponyms were split into two distinct subsets, with an equal number of matching and non-matching pairs. Different classification models, including the method detailed in Section 3, were trained on one of the subsets, and these models were then evaluated on the other subset of the data. For each evaluation metric, we report averaged values over the two subsets of the data.

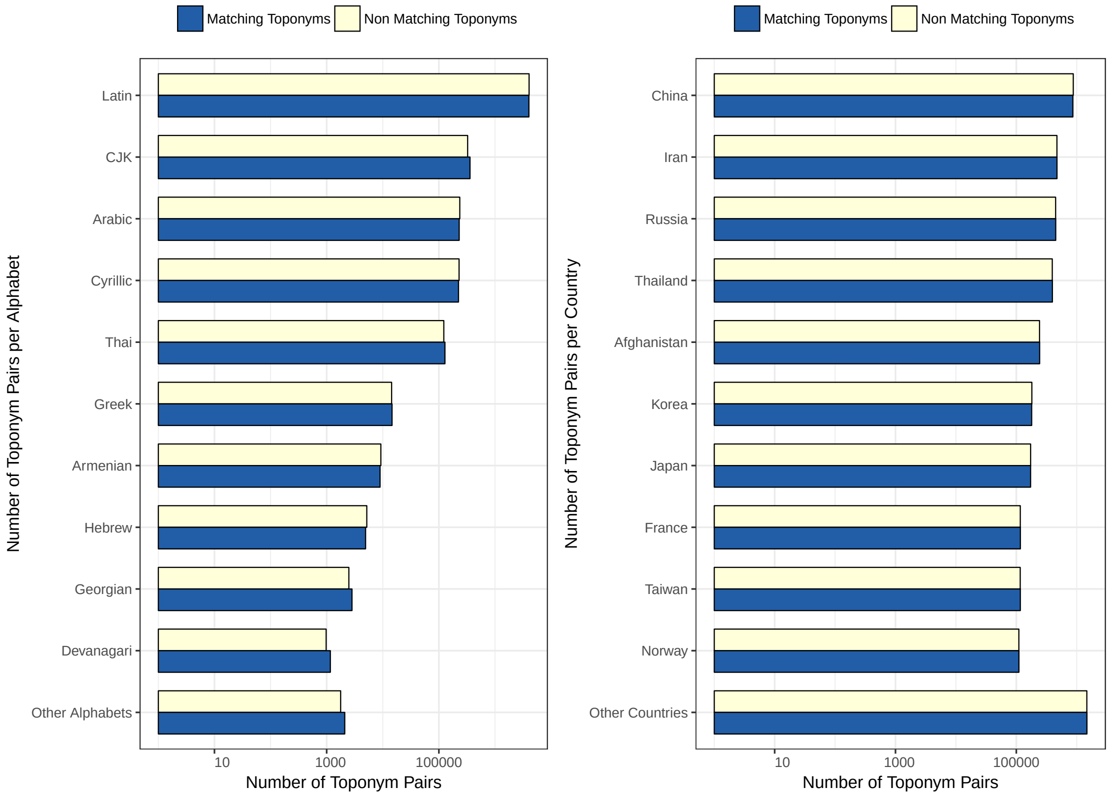


Figure 6.: Distribution for the number of matching and non-matching pairs of toponyms, over different alphabets and geographic regions.

It is important to notice that although the deep learning procedure involves many parameters that can influence its performance, no significant effort was put into fine-tuning our model for optimal performance. The dimensionality of the outputs for the GRUs was set to 60, except in the case of two tests where we tried to assess the influence of this parameter. Dropout was considered as a regularization procedure after each internal layer of the model, with a value of 0.01. The Adam optimization method (Kingma and Ba 2015) was used with the default value of 0.001 for the learning rate, a fuzz factor of 1e-08, and the $\beta_1$ and $\beta_2$ parameters respectively set to 0.9 and to 0.999 (i.e., the default values in the Keras library). Although consistent hyper-parameter tuning was not addressed, we do, nonetheless, report on experiments with variations of the deep neural architecture given in Section 3, for example by not considering some of the layers in the full model, or changing the dimensionality of the outputs for the GRU layers.

An initial set of experiments showed that slight variations in the results indeed occurred as a function of the hyper-parameters. The value of 60 for the dimensionality of the GRUs achieved the best accuracy on our experiments, with the full model given in Section 3.

## 4.2. *The Obtained Results*

Table 2 presents the obtained experimental results, effectively comparing the performance of (i) individual string similarity metrics, (ii) supervised learning methods that combine

Table 2.: Experimental results obtained by the proposed method, by individual string similarity metrics, and by combining such metrics through supervised machine learning.

| Method | Accuracy | Precision | Recall | F1-Score | Time (50K Pairs) |
|---|---|---|---|---|---|
| Damerau-Levenstein ($\alpha = 0.55$) | 65.07 | **78.65** | 41.36 | 54.21 | 0.27 sec. |
| Jaro ($\alpha = 0.75$) | 63.78 | 76.95 | 39.34 | 52.06 | 0.25 sec. |
| Jaro-Winkler ($\alpha = 0.70$) | 63.59 | 71.74 | 44.84 | 55.19 | 0.25 sec. |
| Jaro-Winkler Reversed ($\alpha = 0.75$) | **65.17** | 78.00 | 42.26 | 54.82 | 0.27 sec. |
| Sorted Jaro-Winkler ($\alpha = 0.70$) | 61.89 | 71.44 | 39.62 | 50.97 | 0.34 sec. |
| Permuted Jaro-Winkler ($\alpha = 0.70$) | 63.42 | 68.90 | **48.91** | **57.21** | 87.18 sec. |
| Cosine $N$-Grams ($\alpha = 0.40$) | 61.50 | 70.37 | 39.75 | 50.80 | 3.03 sec. |
| Jaccard $N$-Grams ($\alpha = 0.25$) | 61.72 | 71.50 | 38.97 | 50.44 | 0.81 sec. |
| Dice Bi-Grams ($\alpha = 0.50$) | 62.18 | 75.36 | 36.19 | 48.90 | 0.61 sec. |
| Jaccard Skipgrams ($\alpha = 0.45$) | 62.69 | 73.44 | 39.76 | 51.59 | 2.02 sec. |
| Monge-Elkan ($\alpha = 0.70$) | 59.57 | 65.83 | 39.79 | 49.60 | 0.54 sec. |
| Soft-Jaccard ($\alpha = 0.60$) | 59.43 | 69.65 | 33.43 | 45.18 | 0.56 sec. |
| Davis and De Salles (2007) ($\alpha = 0.65$) | 62.10 | 71.03 | 40.86 | 51.88 | 1.27 sec. |
| Support Vector Machines | 72.38 | 69.17 | **80.76** | 74.52 | 101.52 sec. |
| Random Forests | **78.67** | **78.03** | 79.80 | 78.91 | 131.60 sec. |
| Extremely Randomized Trees | 78.37 | 78.00 | 79.04 | 78.52 | 169.83 sec. |
| Gradient Boosted Trees | 78.54 | 77.51 | 80.42 | **78.94** | 99.05 sec. |
| Deep Learning Method | **88.71** | **88.43** | **89.07** | **88.75** | 40.60 sec. |

these multiple string similarity metrics, and (iii) the deep learning method proposed in Section 3. Besides precision, recall, F1, and accuracy, we also report the average processing time associated to the application of each method, for sets of fifty thousand records. All tests were performed on a standard PC with an Intel Core I7 6700 CPU running at 3.4 GHZ and an NVIDIA GeForce GTX 980 GPU, and with 16GB of RAM. In the case of the experiments leveraging supervised machine learning, we report the time involved in computing all the similarity metrics that are used as features (i.e., only in the case of the methods that combine multiple similarity metrics), plus the time involved in applying the classification algorithm to the toponym pair.

In Table 2, we do not show the time spent on model training, since we argue that training can be performed only once, on an offline stage, thus not impacting the performance of these methods when matching previously unseen toponyms. However, it is important to state that model training is computationally much more demanding in the case of methods based on deep learning. Using our hardware (i.e., a single GPU), training and evaluating each of the deep learning models included in Table 3, when leveraging the 2-fold cross-validation procedure, took approximately 5 days to complete.

The results presented on Table 2 regarding the 13 individual string similarity metrics and the supervised machine learning methods for combining multiple metrics have already been reported in the previous work by Santos *et al.* (2017), which showed that combining multiple metrics outperformed simpler approaches (Recchia and Louwerse 2013). Details about the computation of each similarity metric are given in the previous article, and the threshold value $\alpha$ considered for making the matching decisions was tuned to the best results in terms of accuracy. The tests with feature-based supervised machine learning methods relied on implementations provided by the scikit-learn[1] and XGBoost[2] Python packages. In particular, we experimented with models based on the formalism of lin-

[1] `http://scikit-learn.org/`

[2] `http://xgboost.readthedocs.io`

Table 3.: Results with different variations of our deep neural network architecture.

| Method | Accuracy | Precision | Recall | F1-Score | Time (50K pairs) |
|---|---|---|---|---|---|
| Proposed Neural Architecture | **88.71** | 88.43 | 89.07 | **88.75** | 40.60 sec. |
| Dimensionality of 30 in the GRUs | 87.79 | 87.72 | 88.35 | 88.04 | 49.50 sec. |
| Dimensionality of 90 in the GRUs | 88.40 | 88.10 | 88.79 | 88.44 | 53.21 sec. |
| Without Bi-Directional GRUs | 88.26 | **89.12** | 87.17 | 88.13 | 22.36 sec. |
| Single Bi-Directional GRU Layer | 88.22 | 88.40 | 87.97 | 88.19 | 23.09 sec. |
| Only Concatenating Representations | 87.78 | 87.68 | 87.90 | 87.80 | 39.82 sec. |
| Multiple RLU Layers | 88.57 | 87.79 | **89.60** | 88.68 | 42.11 sec. |

ear support vector machines and based on different types of state-of-the-art approaches leveraging ensembles of decision trees (i.e., random forests (Breiman 2001), extremely randomized trees (Geurts *et al.* 2006), and gradient boosted decision trees (Chen and Guestrin 2016)). All these models leveraged the 13 string similarity metrics as features.

The results on Table 2 show that, when properly tuned, the different similarity metrics achieve very similar results in terms of accuracy (i.e., results range from 61.50 to 65.17). The supervised machine learning methods that combine the multiple metrics significantly outperform the individual similarity metrics in terms of matching quality. However, the novel approach introduced in this article performs even better, with the best results corresponding to an increase in accuracy, precision, recall, and F1 of 9.9, 9.76, 8.84, and 9.74 points, respectively. As expected, the methods based on supervised machine learning are computationally much more demanding, particularly those that involved the computation of multiple string similarity metrics, given that each metric is already expensive when computed individually. Still, with modern hardware, it is fairly easy to process very large datasets using any of the methods that are listed on Table 2. In all cases that are not leveraging a deep neural network, most of the computational effort is indeed associated to computing similarity features. It should also be noted that the deep learning model leverages parallel processing over the GPU, whereas the remaining methods use only CPU threads.

Table 3 presents results for several variations of the deep learning method introduced in Section 3. These values shown that the different components included in our neural network architecture all contribute to improving the quality of the results. Specifically, the different rows in Table 3 correspond to the following model architectures:

- The full architecture that was described in Section 3;
- The architecture described in Section 3, when considering a lower dimensionality (i.e., vectors of 30 dimensions instead of 60) for the GRU outputs;
- The architecture described in Section 3, when considering a higher dimensionality (i.e., vectors of 90 dimensions instead of 60) for the GRU outputs;
- Keeping all components of the full model architecture except for the bi-directional GRU units, i.e., including the two layers of GRUs, for encoding each toponym, but each layer now only processing the strings from left to right;
- Using a single layer of bi-directional GRU units, together with the remaining components of the complete model architecture;
- Combining the representations for each string that are given as output by the GRU units through a simpler procedure, that only considers concatenating the vectors instead of using concatenation, vector difference, and element-wise vector product;
- Using three layers of Rectified Linear Units (RLUs) prior to the sigmoid node respon-

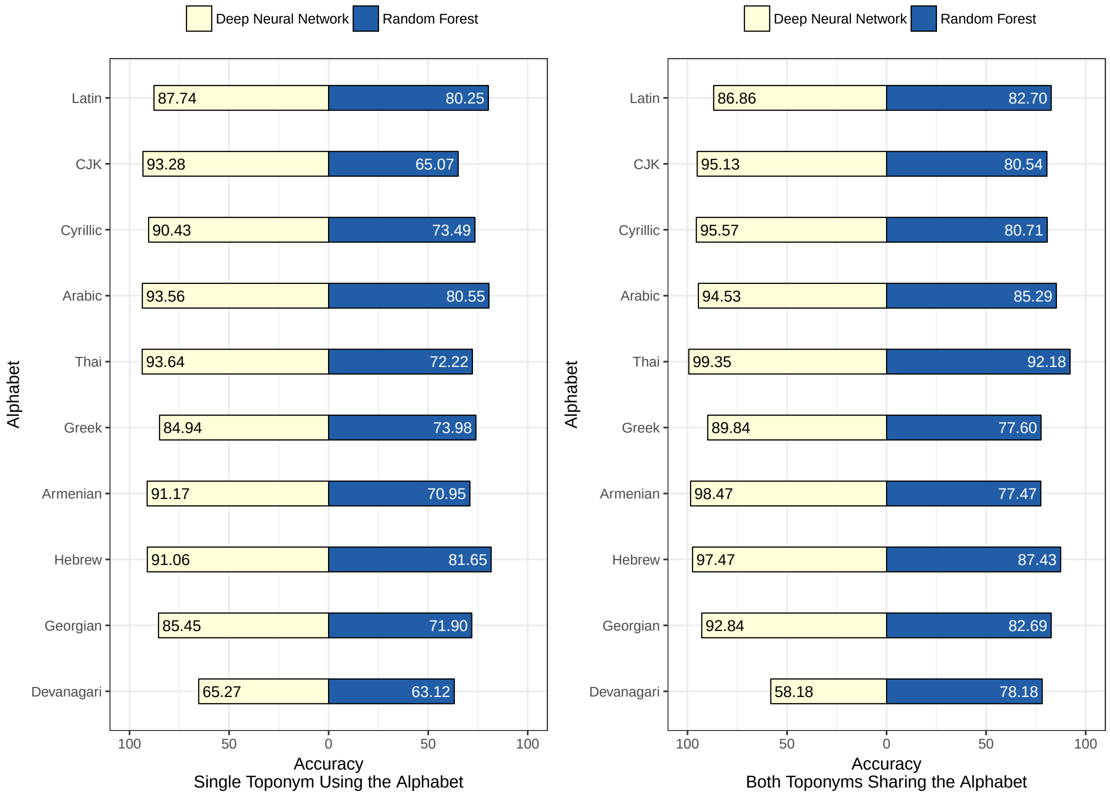


Figure 7.: Accuracy in the results for toponym pairs from different alphabets.

sible for the predictions, instead of considering a single RLU.

Notice that all the variations of the proposed deep learning method outperformed the simpler approaches that are shown in Table 2, although there are slight differences in the results from the different methods. The first two variations in Table 3 correspond to models that use a lower or higher dimensionality in the internal representations produced by the GRUs. The results show that this dimensionality has an impact on the computation time, although the quality of the results remains approximately the same for the values that were tested. Notice that the alternative model shown in Table 3 that corresponds to the use of a higher dimensionality for the GRUs is more demanding in terms of computational resources, at the same time leading to slightly inferior results. The next three alternative models that are shown on Table 3 correspond to experiments in which specific parts of the proposed neural architecture have been removed, leading to a slight decrease in accuracy although also, in the first two cases, to a significant improvement in terms of computational complexity. Given the large gains in terms of execution time, associated to the small losses in terms of accuracy, models with a single GRU layer or without considering bi-directional GRUs can, in fact, be preferred in some cases. The last model corresponds to an alternative architecture leveraging additional layers, but that nonetheless also leads to slightly inferior results in terms of accuracy.

We also analyzed the obtained results according to the alphabets being used in the toponyms (Figure 7), and according to the countries associated to the toponyms (Figure 8).

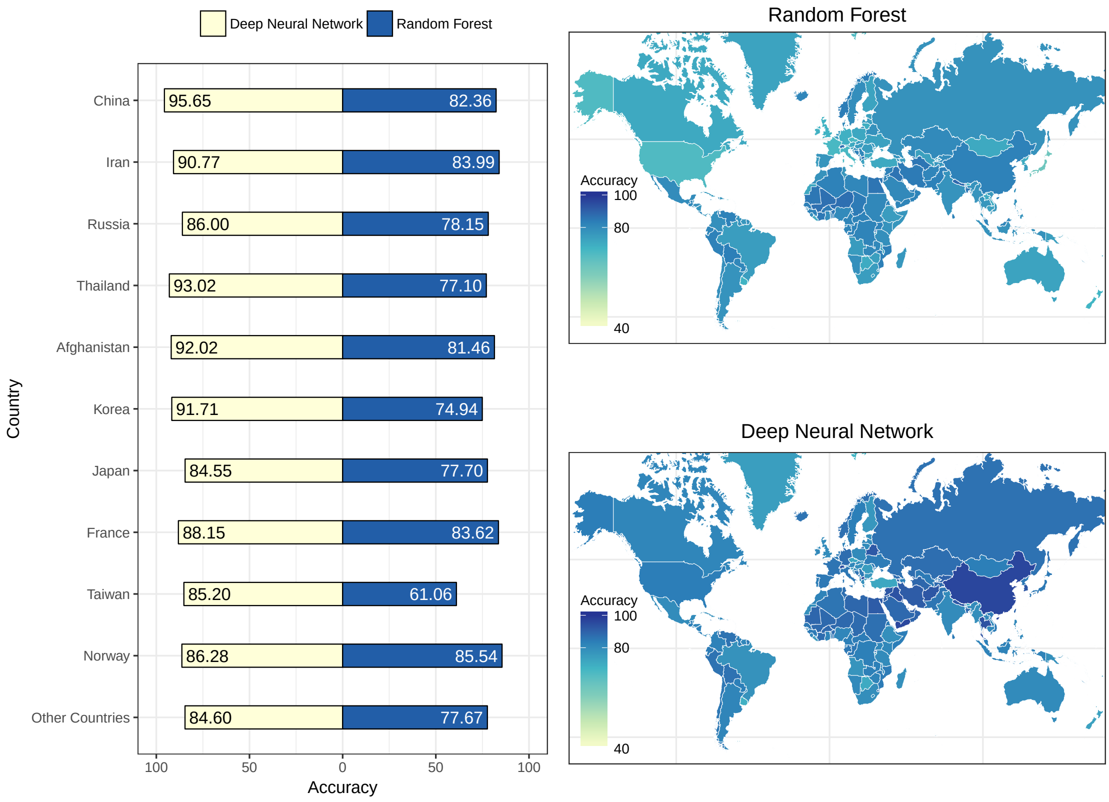


Figure 8.: Accuracy in the results for toponym pairs from different countries.

In both these cases, we report on the accuracy for the random forest model (i.e., the best model in terms of accuracy, from the set of baselines corresponding to the supervised training of classifiers leveraging similarity scores as features) versus the proposed neural network architecture. Regarding Figure 7, results are shown for the subset of toponym pairs where (i) at least one of the toponyms uses characters from a given alphabet (i.e., the toponyms in these pairs can have different alphabets), or (ii) both toponyms are consistent in their alphabet. Regarding Figure 8, results are shown for the subset of toponym pairs where both toponyms belong to the same country.

In both Figures 7 and 8, we can see that worse results are indeed being achieved by the random forest model, particularly for the case of toponym pairs where different alphabets are being used in each of the involved toponyms (i.e., the average difference between the accuracy results of both models is of 8.35 in the case of pairs where the toponyms share the alphabet, versus an average difference of 14.34 in the other case). These results support the hypothesis that the deep neural network architecture can indeed be more effective in cases involving complex character transliterations, which indeed were challenging for the methods surveyed by Recchia and Louwerse (2013) and by Santos *et al.* (2017).

Finally, Table 4 illustrates the results obtained with the proposed neural network architecture, showing examples of both matching and non-matching pairs of records, that were either correctly or incorrectly classified. Although the table contains only a small set of examples, a wider manual analysis of the results confirmed that, although the methods combining multiple similarity metrics can already detect some of the difficult matching

Table 4.: Illustrative examples for the results with the method based on deep learning.

| | Correctly Classified | Incorrectly Classified |
|---|---|---|
| Matching | ***Lingzhithang*** ; ***Lingzi Tāng*** | *Dinas a Sir Abertawe* ; *Swansea* |
| | ***Klejmont*** ; ***Claymont*** | *Cawdor Castle* ; *Chateau de Cawdor* |
| | ***St. Simons*** ; ***Saint Simons Island*** | 長間瀬 ; *Nagamase* |
| | ***Pontypool*** ; ***Pont-y-pŵl*** | *Ballachuhsh* ; *Baile a Chaolais* |
| | *Siedenbrünzow* ; *Zidenbrincov* | *North Frisia* ; *Noordfreesland* |
| | *Sodelhas* ; *Soudeilles* | 格伊森 ; *Greußen* |
| | ***Buironfosse*** ; ***Биронфос*** | *Bodenleve* ; *Badeleben* |
| | ***minaminohara*** ; ミナミノハラ | *Tehelieg* ; *Théhillac* |
| | 吉野町 ; よしのちょう | *Вільмуазан* ; 莱穆瓦桑 |
| | *Sunny Isles Beach* ; *Sani Ajls Bich* | *la Tallada d'Empordà* ; *la Tallada* |
| | ジョスラン ; ***Жосселен*** | *Virgala Mayor* ; *Birgaragoien* |
| | ***Coity Castle*** ; ***Chateau de Coity*** | *Scalabis* ; *Сантарен* |
| | ベッケドルフ ; ***Беккедорф*** | *Wonjangan* ; *Changal-li* |
| | *Saldaña* ; *Sal'dan'ja* | *Roxburgh* ; *Roxburghshire* |
| | 粗卡娘村 ; ***Cukaniangcun*** | *Southend-on-Sea* ; *Sautend-on-Si* |
| Non-Matching | ***Jaypāsa*** ; ***Jaypara*** | *Zlatinitsa* ; *Zlatina* |
| | ***Bliskastel'*** ; ***Bliesdorf*** | *Observatorio Griffith* ; *Griffith Park* |
| | *Hayy ad Duhayni* ; *Hayy ad Dihliz* | *Carnarvonshire* ; *Kernarvonas* |
| | *Frankfort* ; *Frederick* | *Cavalier* ; *Okrug Kavalir* |
| | *Cleveland* ; *Clermont* | *Ben Bhuidhe Mhor* ; *Beinn Bhuidhe* |
| | *Génissieux* ; *Geissan* | *Unterdorf* ; *Унтердітфурт* |
| | *Lichtenstein* ; *Likhtenrade* | *San Joaninho* ; *Sao Joanico* |
| | *Burry Port* ; *Burry Holm* | *Zandersleben* ; *Zandbostel* |
| | ***Laararja*** ; ***Laanabra*** | *Longess* ; *Longness* |
| | *Shepun* ; *Shepshed* | *Sordalsvatnet,nedre* ; *Norddalsvatnet ovre* |
| | *Rigin Idi* ; *Rigachikum* | *La Ferreira* ; *Ferreiros* |
| | 下晴山 ; やながさわ | *Pontino* ; ポンツァ |
| | ***Las Flores*** ; ***Flores de Leán*** | *Cabo Girao* ; *Pico do Galo* |
| | *yao shang* ; *gao yuan* | *Kiriake Gawa* ; *Ogogawa Sawa* |
| | *Saint Combs* ; *St. Columb Minor* | *Grovane* ; *Grøtvann* |

cases (e.g., cases with strings that have a high Levenshtein distance between them, such as *Orange County* and *Comté d'Orange*), problems remained in terms of detecting some of the more complex transliterations (e.g., pairs like *Buironfosse* and *Биронфос*, which the method based on random forests classified incorrectly as non-matching). The proposed neural network architecture, on the other hand, is much more effective in such cases. The correctly classified toponyms that are shown in bold on Table 4 correspond to cases where the random forest model failed to produce a correct classification.

The datasets used in our experiments, as well as a pre-trained neural-network following the architecture outlined in Section 3, have been made available online[1]. This can not only support the realization of comparative experiments with other datasets and/or with model variations over the same dataset, but also the integration of the ideas advanced over this article in other downstream tasks that involve toponym matching and toponym-based geographical search. Currently ongoing work leverages the model presented in this paper in the context of digital humanities applications, specifically as a way to address challenges related to gazetteer data conflation (Berman *et al.* 2016) and toponym resolution in historical itineraries (Blank and Henrich 2016).

[1] `http://github.com/ruipds/Toponym-Matching`

## 5. Conclusions and Future Work

This article presented a novel method, based on a deep neural network architecture, for addressing the task of matching toponyms. Taking inspiration on previous research focused on natural language inference, the proposed neural network leverages recurrent units for encoding two input sequences (i.e., the strings that are to be matched), latter processing these representations in order to classify the input toponyms as either matching or non-matching. Using a very large dataset, with five million pairs of toponyms collected from lists of alternative place names taken from the GeoNames gazetteer, we showed that the proposed method can significantly outperform (i) approaches based on thresholding the results of individual similarity metrics, and (ii) approaches based on supervised machine learning for combining the results of multiple similarity metrics. The proposed method is particularly effective in capturing the complex character transliterations involved in some pairs of toponyms, for example when matching Asian or Arabic toponyms against their Western transliterations, which would be almost impossible for the similarity metrics considered in previous research.

Despite the interesting results that are reported in the article, there are also many ideas for future work. We can, for instance, consider the usage or more advanced optimization methods for training deep neural networks, or consider systematic approaches for tuning hyper-parameters. More interestingly, we can also experiment with different model architectures, such those based on applying convolutions over the input sequences, highway network architectures, or considering the inclusion of attention mechanisms, again taking inspiration on recent proposals for textual entailment and other related natural language processing problems (Yin *et al.* 2015, Liu *et al.* 2016, Parikh *et al.* 2016).

For future work, we also plan to test our deep neural models, learned from GeoNames data, on the task of matching toponyms collected from other datasets. These can include datasets of historical place names collected from resources such as the Gazetteer of British Place Names[1], the Gazetteer of Scottish Places[2], or the Historical Gazetteer of England's Place Names[3]. Experiments with these data would allow us to assess the degree to which the inferred models can properly generalize to different contexts. We also plan to evaluate the proposed method on problems related to matching other types of names, such as variations on names for companies or individuals (Steinberger *et al.* 2013).

Given the overall good results, it is our belief that the method advanced in this article can indeed have important implications for tasks such as gazetteer conflation (Hastings 2008, Berman *et al.* 2016, Moura *et al.* 2017) or application areas such as geographic information retrieval, which involve matching toponyms against reference data (e.g., associating gazetteer entries to place names in textual documents, so as to disambiguate their meaning (Monteiro *et al.* 2016), or matching toponyms in user queries against databases of geographical information (Berkhin *et al.* 2015, McKenzie *et al.* 2014)). For future work, we also plan to evaluate extensions/adaptations of the proposed neural architecture, in some of the aforementioned downstream tasks.

[1] `http://www.gazetteer.co.uk/`

[2] `http://www.scotlandsplaces.gov.uk`

[3] `http://placenames.org.uk`

## *Acknowledgements*

This research was supported by the Trans-Atlantic Platform for the Social Sciences and Humanities, through the Digging into Data project with reference HJ-253525, and also through the Reassembling the Republic of Letters networking programme (EU COST Action IS1310). The researchers from INESC-ID also had financial support from Fundação para a Ciência e Tecnologia (FCT), through project grants with references PTDC/EEI-SCR/1743/2014 (Saturn) and CMUPERI/TIC/0046/2014 (GoLocal), as well as through the INESC-ID multi-annual funding from the PIDDAC programme, which has the reference UID/CEC/50021/2013.

## References

Anastácio, I., Martins, B., and Calado, P., 2009. Classifying documents according to locational relevance. *In*: *Proceedings of the Portuguese Conference on Artificial Intelligence* Springer.

Berkhin, P., *et al.*, 2015. A New Approach to Geocoding: BingGC. *In*: *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* ACM.

Berman, M.L., Mostern, R., and Southall, H., eds. , 2016. *Placing Names : Enriching and Integrating Gazetteers*. Indiana University Press.

Bilenko, M. and Mooney, R.J., 2003a. On Evaluation and Training-Set Construction for Duplicate Detection. *In*: *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation* ACM.

Bilenko, M. and Mooney, R.J., 2003b. Adaptive duplicate detection using learnable string similarity measures. *In*: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM.

Blank, D. and Henrich, A., 2016. A depth-first branch-and-bound algorithm for geocoding historic itinerary tables. *In*: *Proceedings of the ACM Workshop on Geographic Information Retrieval* ACM.

Bowman, S.R., *et al.*, 2015. A large annotated corpus for learning natural language inference. *In*: *Proceedings of the Conference on Empirical Methods in Natural Language Processing* ACL.

Breiman, L., 2001. Random Forests. *Machine learning*, 45 (1).

Brill, E. and Moore, R.C., 2000. An Improved Error Model for Noisy Channel Spelling Correction. *In*: *Proceedings of the Annual Meeting on Association for Computational Linguistics* ACL.

Chen, T. and Guestrin, C., 2016. XGBoost: A scalable tree boosting system. *In*: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM.

Christen, P., 2006. A Comparison of Personal Name Matching: Techniques and Practical Issues. *In*: *Proceedings of the Workshops of the IEEE International Conference on Data Mining* IEEE.

Chung, J., *et al.*, 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *In*: *Proceedings of the NIPS Workshop on Deep Learning.*

Cohen, W., Ravikumar, P., and Fienberg, S., 2003. A Comparison of String Distance Metrics for Name-Matching Tasks. *In*: *Proceedings of KDD Workshop on Data Cleaning and Object Consolidation* AAAI.

Damerau, F.J., 1964. A technique for computer detection and correction of spelling errors.

*Communications of the ACM*, 7 (3).

Davis, C.A. and De Salles, E., 2007. Approximate String Matching for Geographic Names and Personal Names. *In*: *Proceedings of the Brazilian Symposium on GeoInformatics* INPE.

Dice, L.R., 1945. Measures of the amount of ecologic association between species. *Ecology*, 26 (3).

Freire, N., *et al.*, 2011. A metadata geoparsing system for place name recognition and resolution in metadata records. *In*: *Proceedings of the Annual International ACM/IEEE Joint Conference on Digital Libraries* ACM.

Fu, G., Jones, C.B., and Abdelmoty, A.I., 2005. Building a Geographical Ontology for Intelligent Spatial Search on the Web. *In*: *Proceedings of the IASTED International Conference on Databases and Applications* ACTA Press.

Geurts, P., Ernst, D., and Wehenkel, L., 2006. Extremely randomized trees. *Machine Learning*, 63 (1).

Goodfellow, I., Bengio, Y., and Courville, A., 2016. *Deep Learning*. MIT Press.

Hastings, J.T., 2008. Automated conflation of digital gazetteer data. *International Journal of Geographical Information Science*, 22 (10).

Hastings, J. and Hill, L., 2002. Treatment of duplicates in the Alexandria digital library gazetteer. *In*: *Proceedings of the International Conference on Geographic Information Science* Springer.

Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Computation*, 9 (8).

Jaccard, P., 1912. The distribution of the flora in the alpine zone.. *New phytologist*, 11 (2).

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).

Joshi, T., *et al.*, 2008. Crosslingual Location Search. *In*: *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval* ACM.

Keskustalo, H., *et al.*, 2003. Non-adjacent digrams improve matching of cross-lingual spelling variants. *In*: *Proceedings of the International Symposium on String Processing and Information Retrieval* Springer.

Kilinç, D., 2016. An accurate toponym-matching measure based on approximate string matching. *Journal of Information Science*, 42 (2).

Kingma, D. and Ba, J., 2015. Adam: A method for stochastic optimization. *In*: *Proceedings of the International Conference for Learning Representations.*

Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10.

Li, L., *et al.*, 2016. Entropy-Weighted Instance Matching Between Different Sourcing Points of Interest. *Entropy*, 18 (2), 45.

Liu, Y., *et al.*, 2016. Learning Natural Language Inference using Bidirectional LSTM Model and Inner-Attention. *arXiv preprint arXiv:1605.09090.*

Martins, B., 2011. A supervised machine learning approach for duplicate detection over gazetteer records. *In*: *Proceedings of the International Conference on GeoSpatial Semantics* Springer.

McKenzie, G., Janowicz, K., and Adams, B., 2014. A weighted multi-attribute method for matching user-generated points of interest. *Cartography and Geographic Information Science*, 41 (2).

Monge, A.E. and Elkan, C.P., 1996. The Field Matching Problem: Algorithms and Applications. *In*: *Proceedings of the AAAI International Conference on Knowledge Discovery and Data Mining* AAAI.

Monteiro, B.R., Davis, C.A., and Fonseca, F., 2016. A survey on the geographic scope of textual documents. *Computers & Geosciences*, 96 (C).

Moreau, E., Yvon, F., and Cappé, O., 2008. Robust similarity measures for named entities matching. *In*: *Proceedings of the International Conference on Computational Linguistics* ACL.

Mou, L., *et al.*, 2016. Natural language inference by tree-based convolution and heuristic matching. *In*: *Proceedings of the Annual Meeting of the Association for Computational Linguistics* ACL.

Moura, T., Davis, C., and Fonseca, F., 2017. Reference data enhancement for geographic information retrieval using linked data. *Transactions in GIS*, 21 (4).

Mueller, J. and Thyagarajan, A., 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. *In*: *Proceedings of the AAAI Conference on Artificial Intelligence* AAAI.

Navarro, G., 2001. A Guided Tour to Approximate String Matching. *ACM Computing Surveys*, 33 (1).

Needleman, S.B. and Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48 (3).

Parikh, A.P., *et al.*, 2016. A Decomposable Attention Model for Natural Language Inference. *In*: *Proceedings of the Conference on Empirical Methods on Natural Language Processing* ACL.

Philips, L., 2000. The double metaphone search algorithm. *C/C++ Users Journal*, 18 (6).

Recchia, G. and Louwerse, M., 2013. A Comparison of String Similarity Measures for Toponym Matching. *In*: *Proceedings of the ACM SIGSPATIAL International Workshop on Computational Models of Place* ACM.

Ristad, E.S. and Yianilos, P.N., 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (5).

Rosenblatt, F., 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 (6).

Santos, J., Anastácio, I., and Martins, B., 2015. Using machine learning methods for disambiguating place references in textual documents. *GeoJournal*, 80 (3).

Santos, R., Murrieta-Flores, P., and Martins, B., 2017. Learning to Combine Multiple String Similarity Metrics for Effective Toponym Matching. *International Journal of Digital Earth*.

Schuster, M. and Paliwal, K.K., 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45 (11).

Sehgal, V., Getoor, L., and Viechnicki, P.D., 2006. Entity resolution in geospatial data integration. *In*: *Proceedings of the Annual ACM International Symposium on Advances in Geographic Information Systems* ACM.

Simon, R., *et al.*, 2014. Towards semi-automatic annotation of toponyms on old maps. *e-Perimetron*, 9 (3).

Smart, P.D., Jones, C.B., and Twaroch, F.A., 2010. Multi-source toponym data integration and mediation for a meta-gazetteer service. *In*: *Proceedings of the International Conference on Geographic Information Science* Springer.

Srivastava, N., *et al.*, 2014. Dropout: A Simple Way to Prevent Neural Networks from

Overfitting. *Journal of Machine Learning Research*, 15 (1).

Steinberger, R., *et al.*, 2013. JRC-Names: A freely available, highly multilingual named entity resource. *In*: *Proceedings of the International Conference on Recent Advances in Natural Language Processing* ACL.

Varol, C. and Bayrak, C., 2012. Hybrid Matching Algorithm for Personal Names. *Journal of Data and Information Quality*, 3 (4).

Weinman, J., 2013. Toponym Recognition in Historical Maps by Gazetteer Alignment. *In*: *Proceedings of the International Conference on Document Analysis and Recognition* IEEE.

Winkler, W.E., 1990. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.. *Proceedings of the Section on Survey Research Methods of the American Statistical Association*.

Yin, W., *et al.*, 2015. ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*.

Zheng, Y., *et al.*, 2010. Detecting nearly duplicated records in location datasets. *In*: *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* ACM.