Scientometrics (2024) 129:719–743
https://doi.org/10.1007/s11192-023-04712-7

A novel text representation which enables image classifiers to also simultaneously classify text, applied to name disambiguation

Stephen M. Petrie[1] · T'Mir D. Julius[1]

Received: 1 May 2020 / Accepted: 19 April 2023 / Published online: 5 June 2023

Abstract
We introduce a novel method for converting text data into abstract image representations, which allows image-based processing techniques (e.g. image classification networks) to be applied to text-based comparison problems. We apply the technique to entity disambiguation of inventor names in US patents, obtaining a list of IDs which identify individual inventors with high accuracy. The method involves converting text from each pairwise comparison between two inventor name records into a 2D RGB (stacked) image representation. We then train an image classification neural network to discriminate between such pairwise comparison images. The trained neural network then labels each pair of records as either matched (same inventor) or non-matched (different inventors), producing highly accurate results. Our new text-to-image representation method could also be used more broadly for other text comparison problems, such as entity disambiguation of academic publications, or for problems that require simultaneous classification of both text and image datasets.

Keywords Entity disambiguation · Text classification · Convolutional neural networks · Simultaneous text and image processing

Introduction

Databases of patent applications and academic publications can be used to investigate the process of research and innovation. For example, patent data can be used to identify prolific inventors (Gay et al., 2008) or to investigate whether mobility increases inventor productivity (Hoisl, 2009). However, the names of individuals in large bibliographic databases are rarely distinct, hence individuals in such databases are not uniquely identifiable. For example, an individual named "Chris Jean Smith" may have patents under slightly different names such as "Chris Jean Smith", "Chris J. Smith", "C J Smith", etc... There may also be one or more other inventors with patents under the same or similar names, such as "Chris J. Smith", "Chris

✉ Stephen M. Petrie
spetrie@swin.edu.au

[1] Swinburne University of Technology, Hawthorn, Australia

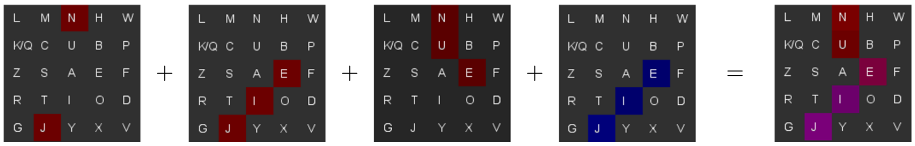


Fig. 1 Constructing a string-map image. The first four images show each sub-map for the example word "JEN", which are summed to construct the final string-map image (right-most image)

Smith", etc... Thus it is ambiguous which names (and hence patents) should be assigned to which individuals. Resolving this ambiguity and assigning unique identifiers to individuals—a process often referred to as named entity disambiguation—is important for research that relies on such databases.

Machine learning algorithms have been used increasingly in recent years to perform automated disambiguation of inventor names in large bibliographic databases (e.g. (Li et al., 2014; Ventura et al., 2015; Kim et al., 2016)). See Ventura et al. (2015) for a review of supervised, semi-supervised, and unsupervised machine learning approaches to disambiguation. These more recent machine learning approaches have often out-performed more traditional rule- and threshold-based methods, but they have generally used feature vectors containing several pre-selected measures of string similarity as input for their machine learning algorithms. That is, the researcher generally pre-selects a number of string similarity measures which they believe may be useful as input for the machine learning algorithm to make discrimination decisions.

Here we introduce a novel approach of representing text-based data, which enables image classifiers to also simultaneously perform text classification. This new representation enables a supervised machine learning algorithm to learn its own features from the data, rather than selecting from a number of pre-defined string similarity measures chosen by the researcher. To do this, we treat the name disambiguation problem primarily as a classification problem—i.e. we assess pairwise comparisons between records as either matched (same inventor) or non-matched (different inventors) (Trajtenberg et al., 2006; Miguélez & Gómez-Miguélez, 2011; Li et al., 2014; Ventura et al., 2015; Kim et al., 2016). Then, for a given pairwise comparison between two inventor records, our text-to-image representation method converts the associated text strings into a stacked 2D colour image (or, equivalently, a 3D tensor) which represents the underlying text data.

We describe our text-to-image representation method in "Comparison-map images" section (see Fig. 1 for an example of text-to-image conversion). We also test a number of alternative representations in "Testing alternative string-maps" section. Our novel method of representing text-based records as abstract images enables image processing algorithms (e.g. image classification networks), to be applied to text-based natural language processing (NLP) problems involving pairwise comparisons (e.g. named entity disambiguation). We demonstrate this by combining our text-to-image conversion method with a commonly used convolutional neural network (CNN) (Krizhevsky et al., 2012), obtaining highly accurate results ($F1$ 99.09%, precision 99.41%, recall 98.76%).

Related work

Inventor name disambiguation studies have often used measures of string similarity in order to make automated discrimination decisions. For example, counts of n-grams (sequences of n words or characters) can be used to vectorise text, with the cosine distance between vectors providing a measure of string similarity (Raffo & Lhuillery, 2009; Pezzoni et al., 2014). Measures of edit distance consider the number of changes required to transform one string to another, e.g. the number of additions, subtractions, or substitutions used in the calculation of Levenshtein distance (1966), or of other operations such as transpositions (the switching of 2 letters) used to calculate Jaro–Winkler distance (Jaro, 1989; Winkler, 1990). Phonetic algorithms, such as Soundex, recode strings according to pronunciation, providing a phonetic measure of string similarity (Raffo & Lhuillery, 2009).

Measures of string similarity such as these have been used to guide rule- and threshold-based name disambiguation algorithms (e.g. (Miguélez & Gómez-Miguélez, 2011) and (Morrison et al., 2017)). They can also be used within feature vectors inputted into machine learning algorithms. For example, Kim et al. (2016) use such string similarity feature vectors to train a random forest to perform pairwise classification. Ventura et al. (2015) reviewed several supervised, semi-supervised, and unsupervised machine learning approaches to inventor name disambiguation, as well as implementing their own supervised approach utilising selected string similarity features as input to a random forest model.

Two-dimensional CNNs have been used extensively in recent image processing applications (e.g. (Krizhevsky et al., 2012)), and one-dimensional (temporal) CNNs have been used recently as character-level CNNs for text classification (e.g. (Zhang et al., 2015)). Also, neural networks (usually CNNs) have been used previously to assess pairwise comparison decisions—e.g. in the case of pairs of: images (Koch et al., 2015), image patches (Zbontar & LeCun, 2016; Zagoruyko & Komodakis, 2015), sentences (Yin et al., 2016), images of signatures (Bromley et al., 1993), and images of faces (Hu et al., 2014). These networks are generally constructed for multiple images to be provided simultaneously as input, such as in the case of Siamese neural networks where two identical sub-networks are connected at their output (Bromley et al., 1993; Koch et al., 2015).

In this work we generate a *single* 2-dimensional RGB (stacked) image for a given pairwise record comparison. Thus any image classification network that processes single images can be used (with minimal modification) to process our pairwise comparison images, therefore enabling such neural networks to also simultaneously classify associated text records. We demonstrate this using the seminal “AlexNet” image classification network (Krizhevsky et al., 2012).

Data

We use a combination of two labelled datasets in this work to train the neural network and assess its performance. Each dataset was derived by separate authors, from the US National Bureau of Economics Research (NBER) Patent Citation Data File (Hall et al., 2001); i.e. a labelled dataset of Israeli inventors (Trajtenberg et al., 2006) (the “IS” dataset), and a dataset of patents filed by engineers and scientists (Ge et al., 2016) (the “E &S” dataset). These datasets were combined with US Patent and Trademark Office (USPTO) patent data as part of the PatentsView Inventor Disambiguation Workshop[1] hosted by the American Institutes for Research (AIR) in September 2015.

Each labelled dataset contains unique IDs (UIDs) that identify all inventor-name records from different patents belonging to each unique inventor. We also extracted several other variables from inventor-name records in the bulk USPTO patent data to use in our disambiguation algorithm: first name, middle name, last name, city listed in address,

[1] http://www.patentsview.org/community/workshop-2015.

international patent classification (IPC) codes (i.e. subjects/fields covered by the patent), assignees (i.e. associated companies/institutes), and co-inventor names on the same patent.

Disambiguation algorithm

Our novel inventor disambiguation algorithm involves the following main steps:

(1) **Duplicate removal:** remove duplicate inventor records.
(2) **Blocking:** block (or “bin”) all names by last name, and also by first name in some cases.
(3) **Generate pairwise comparison-map images:** convert text from each within-block pairwise record comparison into a 2D RGB (stacked) image representation.
(4) **Train neural network:** use 2D comparison-map images generated from manually labelled data to train a neural network to classify whether a given pairwise record comparison is a match (same inventor) or non-match (different inventors).
(5) **Classify pairwise comparison-map images:** deploy the trained neural network to classify pairwise comparison images generated from the bulk patent data, producing a match probability for each record pair.
(6) **Convert pairwise match probabilities into clusters:** convert the pairwise match/non-match probabilities generated by the neural net into inventor clusters—i.e. groups of inventor-name records that each belong to a distinct individual inventor. Assigning a UID to each of these groups then leads to a single set of disambiguated inventor names.

Note that the main purpose of the first two steps is to improve computational efficiency. That is, rather than process all possible pairs of patent–inventor records (which has time complexity $\mathcal{O}(n^2)$ for n records), the records are first grouped into similar clusters, or “blocks”, and pairwise comparisons are only made within those blocks. For further detail regarding steps 1 and 2, see “Appendices 1 and 2”. Steps 3–6 are described in detail below.

Comparison-map images

Our intent is to assess all possible within-block pairwise comparisons between patent–inventor records, classifying each comparison as either a match or non-match. To do this, we introduce a new method of converting any string of text into an abstract image representation of that text, which we refer to as a “comparison-map” image. Any image classification neural network can then be used to process these images and hence effectively perform text classification.

To generate a comparison-map image, we firstly define a specific 2D character layout—i.e. a grid of pixels specifying the positions of each letter. The layout of this “string-map” is shown in Fig. 1 (identical in each of the five images).[2] For a given word (e.g. “JEN”), we then add a particular colour (e.g. red) to the pixels of each letter in the word, as well as to any pixels in straight lines connecting those letters. In particular, we add colour to the pixels of the first and last letters (Fig. 1, left-most image), and to all connecting pixels in

[2] Note that any accented characters and other non-ASCII characters in the data are first converted to their corresponding ASCII equivalent before being applied to the string-map (e.g. *è* is converted to *e*).

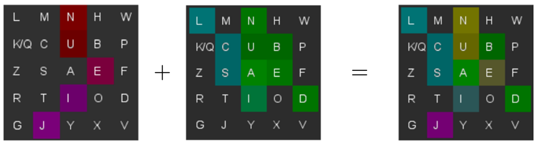


Fig. 2 Comparison of two strings. To compare the names "JEN" and "LINDA", we add the string-map for "JEN" (left image) to the string-map for "LINDA" (middle image) to generate the final comparison image (right image)

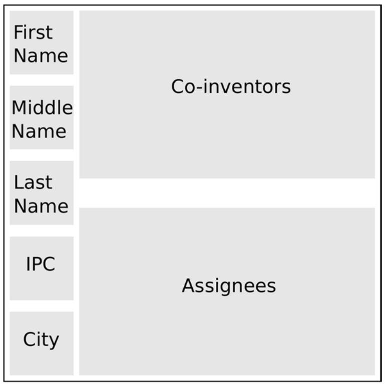


Fig. 3 Record-map layout. Shows the positioning of each string-map within a given record-map

a line connecting each two-letter bi-gram[3] (Fig. 1, second and third images, which correspond to the two bi-grams in "JEN"; i.e. "JE" and "EN"). For repeated letters, the bi-gram contains two of the same letter, so we add colour only to the pixel corresponding to that letter (e.g., for the name "JENNY", we would add colour for four different bi-grams: "JE", "EN", "NN", and "NY").

To highlight the beginning of each string-map, we also repeat the process for the first bi-gram only ("JE") in blue, rather than red (Fig. 1, fourth image). The final string-map for the word "JEN" is shown in Fig. 1 (right-most image). If we then add the string-map of any other word to the green channel of the same RGB image (with the first bi-gram again highlighted in blue), the resulting image represents the pairwise comparison of the two words (e.g. Fig. 2, right-most image).

For a given inventor name record, we generate string-maps for each variable in the record—i.e. first name, middle name, last name, city, IPC codes, co-inventors, and assignees.[4] These string-maps are combined into a single image, arranged as shown in Fig. 3, which we refer to as a "record-map".

Since a given patent–inventor record can have multiple assignees and/or co-inventors, we use a larger string-map for those variables (see Fig. 4, left image). This reduces

[3] Note that connecting pixels are selected using the Python Imaging Library (PIL) *ImageDraw.Draw.line()* function, which produces a consistent selection for a given bi-gram.

[4] Note that if any string-map contains more than one word, we add colour for each bi-gram composing each word, including adding blue to the first bi-gram of each word. For example, if the middle name contained the text "JEN LILY", then colour would be added to pixels corresponding to the bi-grams "JE", "EN", "LI", "IL", and "LY", and blue would be added to the pixels corresponding to the first bi-gram of each word; i.e. "JE" and "LI".

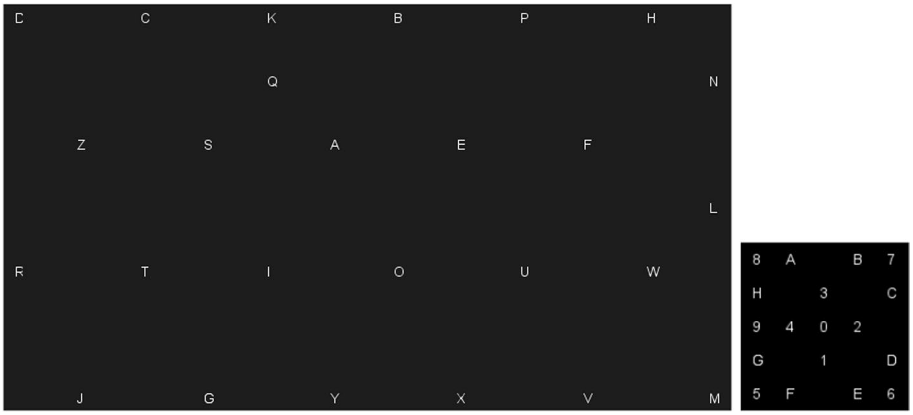


Fig. 4 Larger string-map for assignees and co-inventors, and IPC-map. The larger string-map used to convert a given list of assignees or co-inventors into an abstract image representation (left), and the IPC-map used to convert a given list of IPC classes into an abstract image representation (right)

the possibility that pixels will become saturated in cases where many assignees (or co-inventors) are overlayed onto the same string-map. We also add less colour to each pixel in these larger string-maps, again to reduce the possibility of saturation. For co-inventors, we include only the last names of each co-inventor on the patent (rather than including first, middle and last names, which would increase the saturation of the co-inventor string-maps). Co-inventor and assignee text often includes more than one word so, as was the case with inventor names, we add the pixel colours for each word to the same string-map, colouring pixels corresponding to all within-word bi-grams. Blue is used to colour the pixels of the first bi-gram of each word.

For IPC codes, which contain numbers as well as letters, we use a different string-map shown in Fig. 4 (right image).

We compare any two inventor name records by stacking the two associated 2D record-maps into the same RGB image, one as the red channel and the other as green (with the beginning two-letter bi-gram of each record sharing the blue channel). We refer to the resulting RGB image (or 3D tensor) representation as a "comparison-map" (Fig. 5).

Since red and green combined produce yellow in the RGB colour model, a comparison-map image generated from two similar records should contain more yellow (e.g. Fig. 5, left image), whereas a comparison-map image from two dissimilar records should contain more red and green (e.g. Fig. 5, right image) due to less overlap between the two record-maps. When training on labelled comparison-maps, we expect that the neural network will learn to identify features such as these, which are useful for discriminating between matched/non-matched records. That is, the neural network's learned pattern recognition on comparison-map images will essentially recognise underlying text patterns which are present in the associated patent–inventor name records.

Note that we chose the particular layout of the letters in the string-map shown in Fig. 1 heuristically, such that vowels (which are less important than consonants when assessing string similarity) are positioned towards the centre of the grid, where pixels are more likely to saturate. We also grouped letters with similar phonetic interpretations, such as "S" and "Z", close to each other. We anticipated that this heuristic layout might make it more straightforward for the network to learn which features are associated with matches/

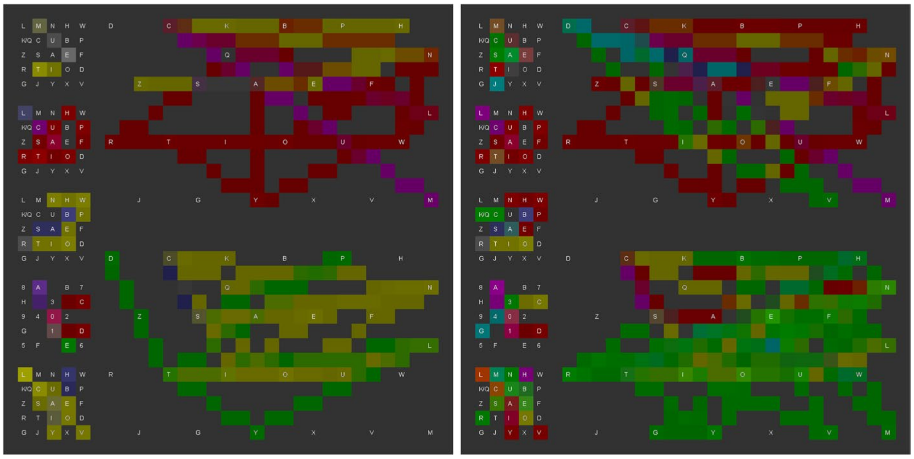


Fig. 5 Comparison-map examples. Two examples of comparison-map images. The left comparison-map image was generated using two matched records (Table 1, rows 1 and 2), and the right image from two non-matched records (Table 1, rows 1 and 3)

non-matches. However, we test how the heuristic layouts shown in Figs. 1, 2, 3, and 4 perform compared with alternative random layouts later in "Testing alternative string-maps" section, and find similar performance regardless of the chosen layout.

Benefits of the comparison-map image representation

Our method of converting text into a stacked 2D RGB bitmap for neural net-based image classification has several benefits:

- The powerful classification capabilities of previous image classification networks can be utilised for text-based record matching, with minimal modification.
- The neural network learns its own features from the data, rather than learning from a feature vector of pre-defined string similarity measures chosen by the researcher.
- Minor spelling variations and errors do not alter the resulting string-map very much, and the neural network can potentially learn that such minor features are unimportant for discriminating between matches and non-matches.
- Matched records with differing word ordering (e.g. re-ordered co-inventor names on different patents) are likely to be identified as matched, due to overlapping pixels.
- The neural net can potentially learn to ignore certain shapes of common words (e.g. "Ltd", "LLC", "Inc", etc...) which are not useful for discrimination decisions.
- Our novel disambiguation algorithm performs well under multiple different choices of alternative string-maps other than those shown in Figs. 1, 2, 3, and 4 (see "Testing alternative string-maps" section), suggesting that multiple alternatives of our comparison-map representations allow for robust pattern recognition and feature extraction.

Note that the above benefits of our text-to-image conversion method would also apply to other text-based comparison problems (e.g. data linkage, or disambiguation of academic

Table 1 Mock records of three patent–inventor name instances

Name	IPC codes	City	Co-inventors (last names)	Assignees
Emmett Lathrop Brown	A10C, A10D	Hill Valley	McFly, Clayton-Brown, Sanchez	Science Solutions
Emmett L. Brown	A11E	Hill Valley	Sanchez	Science Solutions Pty. Ltd.
James T. Brock	G03C	Melbourne	Edison, Da Vinci	Swinburne University of Technology, The University of Melbourne

Rows 1 and 2 are the same mock inventor, while row 3 is a different inventor

papers), or to problems that require simultaneous classification of both text and image datasets.

Modifications to neural network architecture

To demonstrate that our text-to-image conversion method can be combined with an image classifier to perform text-based classification, we apply the method to a commonly used image classification neural network; i.e. the seminal "AlexNet" CNN (Krizhevsky et al., 2012). AlexNet was originally designed to classify colour images ($224 \times 224 \times$ 3-pixel bitmaps) amongst 1000 classes. We slightly modify the network architecture to enable classification of pairwise comparison-map images ($31 \times 31 \times$ 3-pixel bitmaps) into two classes (match/non-match), by altering four hyperparameters as shown in Table 2. We use the NVIDIA Deep Learning GPU Training System[5] (DIGITS) v2.0.0 implementation of AlexNet, and use the Caffe backend (Jia et al., 2014). We use the default settings for the DIGITS solver (stochastic gradient descent), batch size (100), and number of training epochs (30). Rather than use the default learning rate (0.01), we use a sigmoid decay function to progressively decrease the learning rate from 0.01 to 0.001 over the course of the 30 training epochs, as testing indicated that this produced slightly higher accuracies. Instead of the 1000-neuron softmax output layer in AlexNet, we use a 2-neuron softmax output layer, which outputs a probability distribution across our two possible classes (match/non-match).

Note that the default settings of the DIGITS v2.0.0 implementation of AlexNet transform the input data by: (1) altering input images to show the deviation from the mean of all input images (by subtracting the mean image from each input image); (2) randomly mirroring input images; and (3) taking a random square crop from the input image. The main purpose of performing such transformations is to introduce variability into the training images that are expected to be present in the unlabelled data, however we do not use any of those transformations in this work because our images are much more self-consistent than those in the ImageNet database.

Converting pairwise probabilities into inventor groups, and assigning UIDs

After running the trained neural network on bulk patent data, each within-block pairwise comparison has an associated match probability. To assign UIDs to the bulk data, we

[5] https://developer.nvidia.com/digits.

Table 2 Hyperparameters that differ between the two neural network architectures

Hyperparameter	AlexNet	This work	Rationale for modification
Number of neurons in input layer	$224 \times 224 \times 3 = 150,528$	$31 \times 31 \times 3 = 2883$	Smaller size of input images
Kernel size in first convolutional layer	$11 \times 11 \times 3$	$3 \times 3 \times 3$	Smaller-scale features to learn
Stride length of kernels in 1st conv layer	4	1	Smaller kernel size
Number of neurons in output layer	1000	2	Fewer classes

See Krizhevsky et al. (2012) for more details on the network architecture

convert these pairwise probabilities into linked (matched) "inventor groups" using a clustering algorithm. Each inventor group is a linked cluster of inventor name records which all refer to the same individual. Briefly, the clustering algorithm involves converting each pairwise probability value to a binary value (match/non-match) using a pre-selected probability threshold ($\bar{p}$) as a cut-off. Each matched record is then clustered into a larger inventor group if the number of links (l) it has to the that group is $\geqslant$ the number of nodes in the group (n) times some threshold proportion value ($\bar{l}$); i.e. if $l \geqslant n\bar{l}$. This removes weakly-linked records from each group. For further detail on the clustering algorithm, see "Appendix 3". Note that choosing different $\bar{p}$ and $\bar{l}$ values generates different trade-offs between precision and recall.

Once the clustering algorithm has been applied to each block, every patent–inventor name instance has an associated unique inventor ID, and the disambiguation process is complete.

Results

Here we firstly describe our procedure for dividing our labelled datasets into training and test data. We then evaluate our inventor disambiguation algorithm, compare those results to previous studies, and test alternative string-map layouts.

Labelled and bulk datasets

We use the IS and E &S labelled datasets to train the neural network to discriminate between matched and non-matched pairwise comparisons. Each of the labelled datasets are randomly separated into 80% training data (used to train the neural network) and 20% test data (used to assess algorithm performance). We use 75% of the training data to train the network, and the remaining 25% to perform validation assessments during training in order to monitor potential overfitting.

Duplicate removal and blocking is then performed on the labelled data, and comparison-map images are generated for all possible pairwise record comparisons within each block (723,178 comparison-maps for training and 144,552 comparison-maps for testing).

We also perform duplicate removal and blocking on the bulk data, generating comparison-maps for all possible pairwise within-block comparisons (stored as 3D numerical arrays). The trained neural network is then deployed on the bulk patent data, generating match/non-match probabilities for all pairwise within-block comparisons (112,068,838 comparison-maps). Prior to processing the bulk data, we experimented with multiple different values for the pairwise comparison probability threshold ($\bar{p}$) and linking proportion threshold ($\bar{l}$), based on evaluating the trained neural network on the labelled test data. Different $\bar{p}$ and $\bar{l}$ values produce different trade-offs between precision and recall, and we use values that produce an optimal trade-off (highest $F1$ score). We state each $\bar{p}$ and $\bar{l}$ value whenever quoting results from a given run of our disambiguation algorithm.

Evaluation

To evaluate the performance of the disambiguation algorithm, we use the manually labelled IS and E &S test data to estimate pairwise precision, recall, splitting, and lumping based on numbers of true positive (tp), false positive (fp), true negative (tn), and false negative (fn) pairwise links within the labelled test data, as follows (e.g. (Ventura et al., 2015; Kim et al., 2016)):

$$\text{Precision} = \frac{\text{true pos. matches}}{\text{all pos. matches}} = \frac{\text{tp}}{\text{tp} + \text{fp}}, \tag{1}$$

$$\text{Recall} = \frac{\text{true pos. matches}}{\text{total true matches}} = \frac{\text{tp}}{\text{tp} + \text{fn}}, \tag{2}$$

$$\text{Splitting} = \frac{\text{false neg. non-matches}}{\text{total true matches}} = \frac{\text{fn}}{\text{tp} + \text{fn}}, \tag{3}$$

$$\text{Lumping} = \frac{\text{false pos. matches}}{\text{total true matches}} = \frac{\text{fp}}{\text{tp} + \text{fn}}. \tag{4}$$

Higher values are better for precision and recall, while lower values are better for lumping and splitting errors. We also use the pairwise $F1$ score:

$$\frac{F1 = 2 \times \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}.} \tag{5}$$

Since the $F1$ score accounts for the trade-off between precision and recall, it is the primary measure we use to compare the performance of different disambiguation algorithms.

Disambiguation algorithm performance

The precision, recall, and $F1$ estimates for two example runs of our disambiguation algorithm are shown in the bottom two rows of Table 3—first is the highest $F1$ result obtained using the heuristic string-map character order (Figs. 1, 2, 3, 4), and second is the highest $F1$ result obtained using a randomly-generated string-map character order

Table 3 Performance of two example runs of our disambiguation algorithm (bottom rows), compared with other studies evaluated on the IS or E &S labelled datasets

Method	$[\bar{p}; \bar{l}]$	Recall	Precision	$F1$
Kim et al. (2016); IS		98.13	99.89	99.00
Kim et al. (2016); E &S		98.10	99.95	99.02
Kim et al. (2016); Both[†]		98.12	99.92	99.01
Yang et al. (2017); IS		83.79	99.57	91.00
Yang et al. (2017); E &S		90.31	99.87	94.85
Yang et al. (2017); Both[†]		87.05	99.72	92.93
Ours; Both	[0.02; 0.1]	98.67	99.48	99.07
Ours*; Both	[0.03; 0.05]	98.76	99.41	**99.09**

All values in %

[†]Calculated by averaging the IS and E &S results

*Note that this result was obtained using a randomly-generated string-map character order (see "Testing alternative string-maps" section)

(see "Testing alternative string-maps" section for details). Table 3 also shows the best results (highest $F1$) obtained by previous studies which (1) disambiguate bulk USPTO patent data, and (2) evaluate their results using the same labelled datasets we use in this work (i.e. the IS and E &S datasets). Our inventor disambiguation algorithm performs well compared with these other disambiguation studies (Table 3, bottom row), marginally out-performing the previous state-of-the-art study of Kim et al. (2016) and obtaining a much higher $F1$ score than Yang et al. (2017) when measured via the IS and E &S datasets.

For completeness, we also compare our results to those of other studies which use alternative labelled datasets to the IS and E &S datasets used in this work—i.e. Table 4 shows the best results obtained by each study, regardless of the evaluation dataset. Note that Table 4 provides a less equitable comparison than Table 3, as there is generally a small amount of variation in an algorithm's $F1$ score when evaluated on different labelled datasets. Nonetheless, we include Table 4 here for completeness and consistency with previous inventor name disambiguation studies, which often include comparison to other studies with different evaluation datasets. Our disambiguation algorithm is again competitive with the other state-of-the-art inventor name disambiguation algorithms in Table 4, obtaining the highest $F1$ score compared with the other three studies which quote $F1$ results (top four rows, highest F1 score in bold), and obtaining the lowest splitting and lumping errors compared with the two studies which do not quote $F1$ results (bottom three rows, lowest splitting and lumping errors in bold).

Testing alternative string-maps

Here we compare the performance of our heuristic string-map layouts (Figs. 1, 2, 3, 4) to several alternative string-maps. The first alternative string-map we test has random character order; i.e. we keep the pixel co-ordinates identical to the co-ordinates of the associated heuristic layout, but randomise the order of each character (these randomised string-maps are shown in "Appendix 4", Fig. 7). We also test two alternative string-maps

Table 4 Performance of our disambiguation algorithm relative to other studies, regardless of evaluation dataset

Method	$[\bar{p}; \bar{l}]$	Splitting	Lumping	Recall	Precision	$F1$
Kim et al. (2016)				98.48	99.60	99.04
Morrison et al. (2017)				92	98	95
Yang et al. (2017)				96.15	99.61	97.85
Ours	[0.03; 0.05]			98.76	99.41	**99.09**
Li et al. (2014)†		3.26	2.34			
Ventura et al. (2015)		2.31	1.64			
Ours	[0.03; 0.05]	**1.24**	**0.58**			

All values in %

†Ventura et al. (2015) also use an "optoelectronics" (OE) labelled dataset to evaluate (Li et al., 2014), obtaining lower errors on the full OE dataset (splitting: 2.49%, lumping: 0.39%), but higher errors on a random sample of OE data (splitting: 10.54%, lumping: 1.21%)

in which we randomise both the pixel co-ordinate layout and character order ("Appendix 4", Fig. 8). One alternative uses the large string-map for co-inventors and assignees (Fig. 8, right image). The other alternative uses the smaller 5×5 pixel string-map for co-inventors and assignees (Fig. 8, left image), leading to a smaller comparison-map (see "Appendix 4", Fig. 9). We also investigate a string-map with random character order in which we exclude the blue channel for leading bi-grams (Fig. 1, fourth image).

Estimates of precision, recall, and $F1$ for each of these alternative string-maps are shown in Table 5. For each alternative string-map, we ran the algorithm multiple times using different settings of the comparison probability threshold ($\bar{p}$) and linking proportion threshold ($\bar{l}$), and only show results from the run which produced the highest $F1$ score. Results obtained from each of the alternative string-maps are quite similar to those obtained using the heuristically-determined layout ($F1$ scores range from 98.99 to 99.09%). This suggests that our method of converting text into abstract image representations facilitates robust feature learning for several alternative choices of string-map structure, such as randomised string-map character order and/or layout, heuristic order and/or layout, different string-map sizes, and the inclusion/exclusion of a blue channel for leading bi-grams.

Examining lumping errors in large inventor groups

Labelled datasets such as the IS and E &S datasets contain far fewer records than the bulk data. While such subsets of labelled data are useful for measuring several facets of algorithm accuracy, they are not very useful for measuring lumping errors from very common names that become relevant only when processing much larger amounts of data (such as the full bulk dataset). This is because, although very common names are likely to be present in relatively small subsets of labelled data, there are far fewer of them compared with the full bulk dataset. When processing the bulk data, large numbers of common first names

Table 5 Comparison of alternate string-map layouts

String-map layout	$[\bar{p}; \bar{l}]$	Recall	Precision	$F1$
Heuristic character order and layout	[0.02; 0.1]	98.67	99.48	99.07
Random order, heuristic layout	[0.03; 0.05]	98.76	99.41	**99.09**
Random order and layout	[0.05; 0.05]	98.77	99.29	99.03
Random order and layout, with small string-maps	[0.05; 0.2]	98.46	99.52	98.99
Random order, heuristic layout, no blue channel	[0.02; 0.05]	98.71	99.32	99.01

Each row shows the highest $F1$ result obtained for that string-map layout

can lead to a high degree of connectivity in large blocks of common last names, introducing lumping errors in large inventor name groups.

We can investigate the presence of these types of lumping errors by examining the largest inventor groups. In “Appendix 5”, Tables 7 and 8 , we show the name variation for the 10 largest inventor groups obtained using string-maps with *heuristic character order and layout* (i.e. the version of the disambiguation algorithm shown in the top row of Table 5). In many of the groups, there are several variations of first name within the same inventor group. Many of these look to be lumping errors, rather than different variations of the one first name used by the same inventor. The lumping error issue also seems to be more prevalent for very common Japanese last names such as Takahashi, Nakamura, and Kobayashi.

We also represent the first name variation information in heatmap form in Fig. 6a, which shows, for each of the top 50 largest inventor groups, the proportion that each *n*th variation of the first name contributes to the group. Inventor groups with only one variation of the first name will be plotted as a dark red (proportion = 1) square at the 0th position. Note that we see quite a bit of variation in first names across the top 50 largest inventor groups (Fig. 6a). We should also note that name variations are only indicative of potential lumping errors—i.e. while variations in first names may represent inventor name records that belong to different individuals, in some cases they may represent inventor name records that belong to a single individual which has used different variations of their name on different patents.

To reduce the amount of lumping errors in large inventor groups when processing the full bulk dataset, we can apply extra disambiguation steps of:

- separating pairwise matches with mismatched first names (as measured via a Damerau–Levenshtein Distance of $\geqslant 2$)[6] or mismatched middle initials, if those records do not share any assignees,
- for large inventor groups (> 100 records), which are more likely to contain large-group lumping errors due to very common names, using a more stringent criterion to identify mismatched first names (i.e. a Damerau–Levenshtein Distance of $\geqslant 1$, rather than $\geqslant 2$), unless one first name is a sub-string of the other (i.e. to avoid splitting nickname variations such as Chris and Christopher).

Incorporating these extra changes into the disambiguation algorithm leads to a substantial reduction in the number of lumping errors in large inventor groups (see Fig. 6b, c and

[6] Note that we use Python’s Jellyfish module to calculate the Damerau–Levenshtein Distance.

Fig. 6 Variation of first names within largest inventor groups. Shows the degree of variation of first names ▶ within each of the top 50 largest inventor groups. Results from three different versions of the disambiguation algorithm (with heuristic string-map character order and layout) are shown: standard (**a**), augmented *without* sub-string splitting applied to large groups (**b**), and augmented *with* sub-string splitting applied to large groups (**c**). Colours show the proportion that each *n*th variation of the first name contributes to the inventor group. Inventor groups with only one variation of the first name will be plotted as a dark red (proportion = 1) square at the 0th position. Note that **a** has the most first name variations, while **c** has the least. (Color figure online)

"Appendix" Tables 9 and 10) compared with in the absence of the changes (see Figure 6a and "Appendix" Tables 7 and 8). For the two different augmented versions of the disambiguation algorithm—one *without* sub-string splitting applied to large inventor groups (Fig. 6b), and one *with* sub-string splitting applied to large inventor groups (Fig. 6c)—we see the greatest reduction in lumping errors when large inventor groups are separated (Fig. 6c).

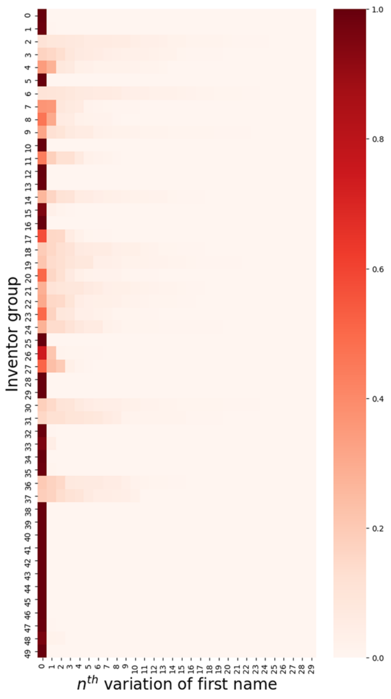
We also examine in Table 6 the degree to which the augmentation of the disambiguation algorithm, with and without sub-string splitting applied to large groups, affects the precision, recall, and $F1$ scores compared with the standard version of the algorithm. We do this for the two string-map methods that produced the highest $F1$ scores in Table 5; i.e. *heuristic character order with heuristic layout*, and *random character order with heuristic layout*. Table 6 shows that the augmented versions of the disambiguation algorithm have higher precision but lower recall and $F1$ scores, with sub-string splitting of large groups enhancing the differences from baseline.

Given all of the above considerations, we suggest that if large inventor groups are to be studied, then utilising the version of the disambiguation algorithm with sub-string splitting applied to large inventor groups would produce the most useful disambiguated inventor groups.

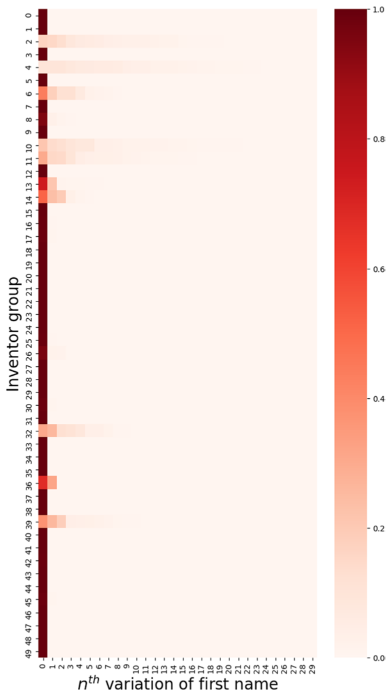
Conclusion

We introduced a new entity disambiguation algorithm and applied it to inventor names in USPTO patent applications. The text-to-image representations in our entity disambiguation algorithm provide a novel way of combining image processing with NLP, allowing image classifiers to perform text classification. We demonstrated this with the seminal AlexNet CNN, obtaining highly accurate results. We also analysed several variants of alternative string-maps, and found that the accuracy of the disambiguation algorithm was highly robust to such variation.

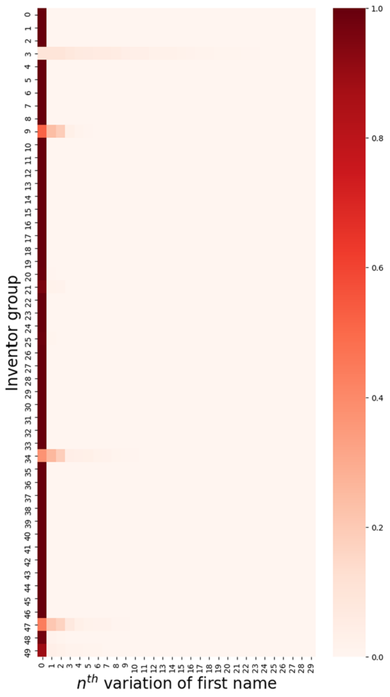
Since the core of our disambiguation algorithm is a classification method to determine how similar two text records are, it should be adaptable to other NLP problems which involve text matching of multiple strings, such as academic author name disambiguation, assignee disambiguation, or record linkage problems. For example, for assignee disambiguation, comparison-maps could be generated for pairs of assignee mentions in different patents, which would include string-maps for assignees and associated inventors. The challenges of adapting the algorithm for assignee disambiguation would include identifying a suitable labelled dataset of disambiguated assignees, identifying which fields to include as string-maps in each pairwise comparison-map, and adapting the blocking procedure for assignee data, however we believe these challenges would be solvable. The algorithm could also be modified for less common applications, such as processing records that


Inventor group
nth variation of first name

(a)


Inventor group
nth variation of first name

(b)


Inventor group
nth variation of first name

(c)

Table 6 Comparison of results with and without removal of lumping errors from large inventor groups

String-map layout	$[\bar{p}; \bar{l}]$	Disambiguation algorithm	Recall	Precision	$F1$
Heuristic character order and layout	[0.02; 0.1]	Standard	98.67	99.48	99.07
		Augmented, *without* sub-string splitting	97.42	99.88	98.64
		Augmented, *with* sub-string splitting	96.89	99.88	98.36
Random order, heuristic layout	[0.03; 0.05]	Standard	98.76	99.41	99.09
		Augmented, *without* sub-string splitting	97.73	99.46	98.59
		Augmented, *with* sub-string splitting	96.89	99.89	98.37

contain both text and image data. This could be done by combining each record's associated image with the abstract image representation of the record's text, in a single combined comparison-map.

Appendix 1: Removal of duplicate records

It is sometimes obvious that two inventor name records likely belong to the same individual, because the two records contain several fields that are identical. For example, if the last name, first name, city, and IPCs of two different records are all exactly identical, it is highly likely that the two records belong to the same individual. We remove such duplicate records based on the following duplication keys:

```
duplicnkey_ipc = lastname + firstname + city + '_'.join(ipcs)

duplicnkey_assignee = lastname + firstname + city + '|'.join(assignees)
```

For a given group of duplicate records sharing the same duplication key, all records except for the first record to be processed are removed from the bulk data. The first record then remains within the bulk data to be processed by the disambiguation algorithm, receiving a unique inventor ID once the algorithm has completed its run. That same ID is then assigned to each removed record in the corresponding group of duplicate records.

Appendix 2: Blocking

The blocking procedure broadly involves grouping together inventor name records into "blocks" (or "bins") using each inventor's last name, and sometimes also their first name. Latter parts of the algorithm will only assess pairwise comparisons within these blocks, never across different blocks.

We firstly group patent–inventor name records together by the first three letters of the last name (this first step is identical to the initial stage of the blocking procedure used by Ventura et al. (2015)). However, some of the resulting blocks contain very large numbers of records, and hence large numbers of pairwise comparisons. To improve efficiency, we further divide such large blocks into smaller blocks by progressively increasing the number of letters used for blocking. That is, if the number of records within a given block (n_b) is above some threshold number ($\bar{n}_\mathrm{b}$), then the records within that block are separated into smaller blocks according to the first *four* letters of the last name. We then continue sub-dividing any blocks that still have $n_\mathrm{b} > \bar{n}_\mathrm{b}$, according to the first five letters of the last name, then six letters, and so on. If all letters of the last name have been used and any blocks still have $n_\mathrm{b} > \bar{n}_\mathrm{b}$, then we append a comma to the string and begin progressively appending letters from the first name as well.

We use $\bar{n}_\mathrm{b} = 100$ throughout this work, as initial testing indicated that it produced a good balance between the following:

- *computational efficiency* i.e. smaller $\bar{n}_b$ leads to more numerous, smaller bins (hence fewer comparisons—which are $\mathcal{O}(n_\mathrm{b}^2)$ for each bin—and less computation time),
- *accuracy* i.e. smaller $\bar{n}_\mathrm{b}$ reduces the number of unnecessary comparisons between records (often non-matched records), which should reduce false positives,
- *recall* i.e. larger $\bar{n}_\mathrm{b}$ leads to fewer, larger bins, which decreases splitting errors (decreasing false negatives).

Together with the deduplication procedure, this reduces the number of pairwise comparisons from $\approx$ 77 trillion before the blocking procedure to $\approx$ 112 million.

Note that since latter parts of the algorithm only assess within-block pairwise comparisons and some inventors' sets of records may have been separated across two or more different blocks, there is a maximum limit to the possible recall attainable by the disambiguation algorithm. After running the blocking procedure on the labelled dataset, we use known pairwise matches in the labelled data to estimate this maximum limit to recall, obtaining the following values: 99.47% (E &S training data), 99.98% (E &S test data), 99.83% (IS training data), and 99.86% (IS test data).

Appendix 3: Clustering algorithm to assign inventor groups

Here we describe the clustering algorithm we use to convert pairwise match probabilities into groups of records each belonging to a single unique inventor. We firstly convert each pairwise probability between the ith and jth record (p_{ij}) into one of the binary classes (c_{ij}; either "match" or "non-match") based on a threshold probability value ($\bar{p}$) as follows:

$$c_{ij} = \begin{cases} \text{match}, & \text{if } p_{ij} \geqslant \bar{p}, \\ \text{non-match}, & \text{otherwise.} \end{cases} \tag{6}$$

The inventor group linking algorithm then primarily involves combining different subgroups together into the one group if they share enough links (pairwise matches). Within a given block, the algorithm involves the following steps:

(1) Order all patent–inventor name records by the number of links they have to other records (i.e. the number of asserted matches to other records), highest first.

(2) Assign a UID to each isolated (non-matched) patent–inventor name.
(3) Assign records to inventor groups. That is, for a given record, the corresponding inventor group initially comprises just the record itself and all records it is linked (matched) to. Each of these linked records (nodes) are kept in the current inventor group only if the number of links (l) it has to the current group is $\geqslant$ the number of nodes in the group (n) times some threshold proportion ($\bar{l}$); i.e. if $l \geqslant n\bar{l}$. This removes the most weakly-linked records from each group (i.e. the nodes with fewest links to their group), which are more likely to be false positive matches. Any outside-group links—i.e. links to nodes that are not within the current group—are also recorded during this step.
(4) Repeat Step 2, because some records may have become isolated (non-matched) following Step 3.
(5) Combine inventor groups together if the number of links they share is greater than a specified threshold. In particular, for an inventor group with n_{self} records (nodes), we combine it with any other group with n_{other} nodes if the number of links to that other group (l) satisfies both: $l \geqslant \bar{l}\, n_{\text{self}}$, and: $l \geqslant \bar{l}\, n_{\text{other}}$.
(6) For each resulting inventor group, assign an identical UID to all patent–inventor name records in the group.

Appendix 4: Random string-map layouts

Here we show the random string layouts analysed in "Testing alternative string-maps" section. Figure 7 shows the string-maps we use for runs where characters are positioned using an identical pixel co-ordinate layout to the heuristic layouts shown in Figs. 1 and 4 (main text), but where the order of each character has been randomised.

Figure 8 shows the string-maps used for runs where both pixel co-ordinate layout and character order are randomised.

The left image in Fig. 9 shows the comparison-map with random layout and character order in which we use the smaller 5 × 5 pixel string-map (Fig. 8, left image) for co-inventors and assignees, rather than the larger string-map (Fig. 8, right image). The right image in Fig. 9 shows the associated record-map layout.

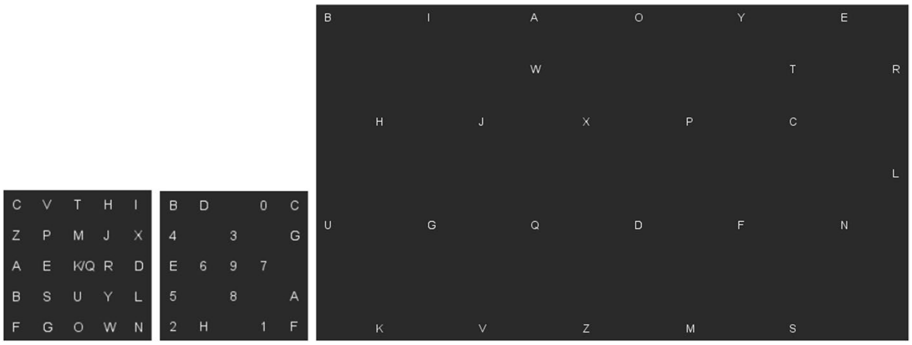


Fig. 7 Random character order. Here we show the smaller string-map (left), IPC-map (middle), and larger string-map (right) we use for runs in which the character order has been randomised

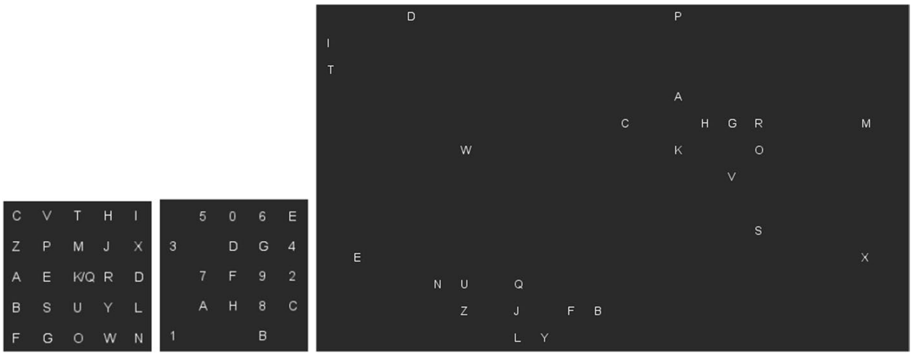


Fig. 8 Random character order and layout. Here we show the smaller string-map (left; identical to the left string-map in Fig. 7), IPC-map (middle), and larger string-map (right) with both random character order and random pixel co-ordinate layout

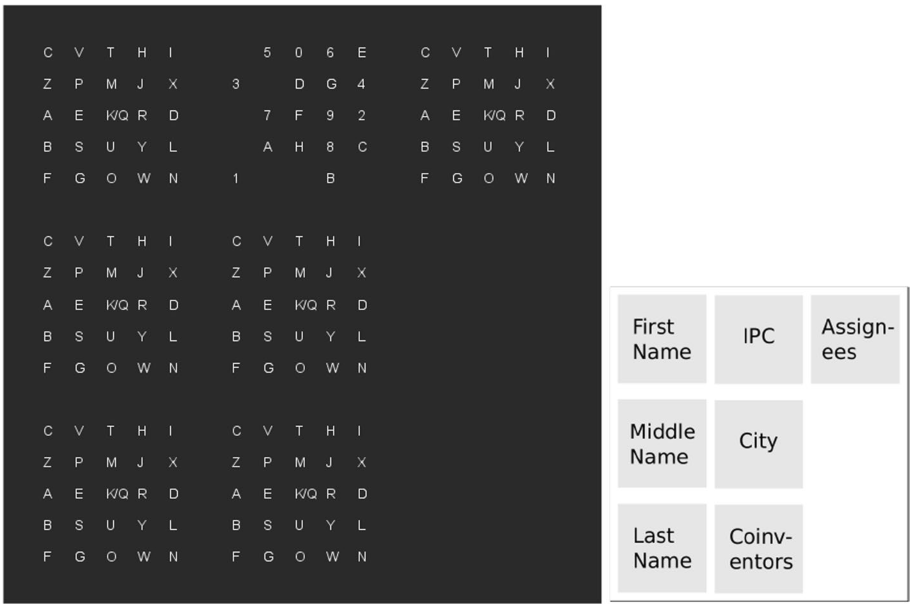


Fig. 9 Random character order and layout, with small string-maps. The left image shows the comparison-map used for runs with smaller string-maps for co-inventors and assignees, as well as random character order and random pixel co-ordinate layout. The right image shows the associated record-map layout

Appendix 5: Name variations for largest inventor groups

Here we show name variations present in the top 10 largest inventor groups, for the standard disambiguation algorithm with heuristic string-map character order and layout (Tables 7, 8), as well as for two different augmented versions of the disambiguation

Table 7 Name variations for largest inventor groups using the standard disambiguation algorithm (for inventor groups 0–4)

Group #	Size (records)	Last names	%	First names	%
0	4692	Silverbrook	99.98	Kia	100.0
		Silverbook	0.02		
1	3813	Yamazaki	100.0	Shunpei	99.21
				Shumpei	0.52
				Shunepi	0.21
				Shunnei	0.03
				Shupei	0.03
2	1737	Kobayashi	100.0	Masaaki	10.88
				Masato	8.92
				Masaki	8.52
				Masahiko	7.43
				Masahiro	7.08
				Masaru	6.16
				Masayuki	5.93
				Masakazu	5.93
				Masanori	5.64
				Masao	5.53
				⋮	⋮
3	1555	Yamamoto	100.0	Masayuki	17.04
				Masahiro	15.43
				Masaki	13.18
				Masaya	8.36
				Masanobu	6.50
				Masashi	6.24
				Masao	5.08
				Masaaki	4.18
				Masakazu	3.67
				Masato	2.96
				⋮	⋮
4	1457	Kobayashi	99.79	Hiroshi	34.52
		Kobayashi, legal representative	0.14	Hiroyuki	28.00
		Kobayashi, deceased	0.07	Hiroaki	8.10
				Hirokazu	6.79
				Hiromichi	3.16
				Hiroki	2.88
				Hiroyoshi	2.26
				Hirotada	2.26
				Hiroo	2.13
				Hiromi	1.99
				⋮	⋮

Shows name variations for the first half of the top 10 largest inventor groups; i.e. groups 0–4. See Table 8 for groups 5–9. Note that while variations in first name may represent inventor name records that belong to different individuals, in some cases they may represent inventor name records that belong to the same inventor that has used different variations of their name on different patents. For a given inventor name group, only the top 10 most frequest first names are shown (vertical ellipses denote absent records)

Table 8 Name variations for largest inventor groups using the standard disambiguation algorithm (for inventor groups 5–9)

Group #	Size (records)	Last names	%	First names	%
5	1407	Weder	100.0	Donald	100.0
6	1406	Nakamura	100.0	Masayuki	10.53
				Masahiro	9.46
				Masaru	9.32
				Masao	7.82
				Masato	6.90
				Masaki	6.26
				Masanori	6.12
				Masahiko	5.97
				Masashi	5.48
				Masakazu	4.13
				⋮	⋮
7	1397	Takahashi	100.0	Hiroyuki	36.44
				Hiroshi	35.79
				Hiroaki	8.52
				Hirokazu	6.80
				Hiroki	6.16
				Hiroyasu	1.65
				Hiromi	1.43
				Hirohisa	1.15
				Hiroharu	0.72
				Hiroko	0.50
				⋮	⋮
8	1334	Watanabe	100.0	Hiroshi	46.55
				Hiroyuki	30.43
				Hirofumi	5.92
				Hiroaki	5.47
				Hiroki	2.47
				Hiroyoshi	2.10
				Hiromi	1.95
				Hiromu	1.57
				Hirotaka	0.60
				Hirosi	0.37
				⋮	⋮
9	1332	Kobayashi	100.0	Takashi	31.83
				Takeshi	11.86
				Takao	9.98
				Takayuki	6.68
				Takahiro	5.78
				Takeo	5.78
				Takehiro	2.78
				Takumi	2.40
				Takaichi	2.33
				Takako	2.03
				⋮	⋮

Shows name variations for the second half of the top 10 largest inventor groups; i.e. groups 5–9. See Table 7 for groups 0–4

Table 9 Name variations for largest inventor groups using the augmented disambiguation algorithm *without* sub-string splitting applied to large groups

Group #	Size (records)	Last names	%	First names	%
0	4692	Silverbrook	99.98	Kia	100.0
		Silverbook	0.02		
1	3783	Yamazaki	100.0	Shunpei	100.0
2	1555	Yamamoto	100.0	Masayuki	17.04
				Masahiro	15.43
				Masaki	13.18
				Masaya	8.36
				Masanobu	6.50
				Masashi	6.24
				Masao	5.08
				Masaaki	4.18
				Masakazu	3.67
				Masato	2.96
				⋮	⋮
3	1407	Weder	100.0	Donald	100.0
4	1389	Nakamura	100.0	Masahiro	9.58
				Masaru	9.43
				Masayuki	9.43
				Masao	7.92
				Masato	6.98
				Masaki	6.34
				Masanori	6.19
				Masahiko	6.05
				Masashi	5.54
				Masakazu	4.18
				⋮	⋮
5	1265	Lapstun	100.0	Paul	100.0
6	1179	Nakamura	100.0	Hiroshi	43.85
				Hiroyuki	18.32
				Hiroki	12.38
				Hiroaki	12.13
				Hirotake	6.79
				Hirotaka	2.46
				Hiromi	1.87
				Hiroya	1.27
				Hirochika	0.76
				Hiro	0.17
7	1145	Sandhu	100.0	Gurtej	99.65
				Gurtel	0.26
				Gurtei	0.09
8	1079	Koyama	100.0	Jun	95.09
				Junichi	2.87
				Junji	1.48
				Junichiro	0.56
9	1075	Forbes	100.0	Leonard	100.0

Same as Table 7, except for the top 10 largest inventor groups obtained using the augmented version of the disambiguation algorithm *without* sub-string splitting applied to large groups

Table 10 Name variations for largest inventor groups using the augmented disambiguation algorithm *with* sub-string splitting applied to large groups

Group #	Size (records)	Last names	%	First names	%
0	4692	Silverbrook	99.98	Kia	100.0
		Silverbook	0.02		
1	3783	Yamazaki	100.0	Shunpei	100.0
2	1407	Weder	100.0	Donald	100.0
3	1389	Nakamura	100.0	Masahiro	9.58
				Masaru	9.43
				Masayuki	9.43
				Masao	7.92
				Masato	6.98
				Masaki	6.34
				Masanori	6.19
				Masahiko	6.05
				Masashi	5.54
				Masakazu	4.18
				⋮	⋮
4	1265	Lapstun	100.0	Paul	100.0
5	1145	Sandhu	100.0	Gurtej	99.65
				Gurtel	0.26
				Gurtei	0.09
6	1075	Forbes	100.0	Leonard	100.0
7	1026	Koyama	100.0	Jun	100.0
8	930	Wood, Jr	98.49	Lowell	100.0
		Wood	1.51		
9	905	Takahashi	100.0	Kenji	51.82
				Kenichi	23.87
				Ken	19.01
				Kenichiro	2.76
				Kensuke	1.33
				Kenichirou	0.44
				Kenichiroh	0.44
				Kenkichi	0.22
				Kennichi	0.11

Same as Table 9, except for the top 10 largest inventor groups obtained using the augmented version of the disambiguation algorithm *with* sub-string splitting applied to large groups

algorithm; one *without* sub-string splitting applied to large inventor groups (Table 9) and another *with* sub-string splitting applied to large inventor groups (Table 10).

Acknowledgements We thank Associate Professor Kai Qin and Professor Elizabeth Webster for their helpful comments on the manuscript. We also thank the reviewers of this manuscript for taking the time and effort to provide comments and suggestions that helped improve the quality of this work.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

References

Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., Lecun, Y., Moore, C., Säckinger, E., & Shah, R. (1993). Signature verification using a "Siamese'' time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence, 07*(04), 669–688. https://doi.org/10.1142/S0218001493000339

Gay, C., Latham, W., & Le Bas, C. (2008). Collective knowledge, prolific inventors and the value of inventions: An empirical study of French, German and British patents in the US, 1975–1999. *Economics of Innovation and New Technology, 17*(1–2), 5–22. https://doi.org/10.1080/10438590701279193

Ge, C., Huang, K., & Png, I. P. L. (2016). Engineer/scientist careers: Patents, online profiles, and misclassification bias. *Strategic Management Journal, 37*, 232–253. https://doi.org/10.1002/smj

Hall, B. H., Jaffe, A. B., & Trajtenberg, M. (2001). *The NBER patent citation data file: Lessons, insights and methodological tools*. National Bureau of Economic Research Working Paper 8498. https://doi.org/10.1186/1471-2164-12-148.

Hoisl, K. (2009). Does mobility increase the productivity of inventors? *Journal of Technology Transfer, 34*(2), 212–225. https://doi.org/10.1007/s10961-007-9068-5

Hu, J., Lu, J., & Tan, Y. P. (2014). Discriminative deep metric learning for face verification in the wild. In *Proceedings of the IEEE Computer Society conference on computer vision and pattern recognition*, 2014 (pp. 1875–1882). https://doi.org/10.1109/CVPR.2014.242.

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. https://doi.org/10.1080/01621459.1989.10478785

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, 2014 (pp. 675–678). https://doi.org/10.1145/2647868.2654889.

Kim, K., Khabsa, M., & Giles, C. L. (2016). Random Forest DBSCAN for USPTO inventor name disambiguation. https://doi.org/10.1145/2910896.2925465.

Koch, G., Zemel, R., & Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *Proceedings of the 32nd international conference on machine learning*, 2015 (Vol. 37). https://doi.org/10.1017/CBO9781107415324.004.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097–1105.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady, 10*(8), 707–710 (citeulike-article-id 311174).

Li, G. C., Lai, R., D'Amour, A., Doolin, D. M., Sun, Y., Torvik, V. I., Yu, A. Z., & Lee, F. (2014). Disambiguation and co-authorship networks of the U.S. patent inventor database (1975–2010). *Research Policy, 43*(6), 941–955. https://doi.org/10.1016/j.respol.2014.01.012

Miguélez, E., & Gómez-Miguélez, I. (2011). *Singling out individual inventors from patent data*. Research Institute of Applied Economics Working Paper. https://doi.org/10.2139/ssrn.1856875.

Morrison, G., Riccaboni, M., & Pammolli, F. (2017). Disambiguation of patent inventors and assignees using high-resolution geolocation data. *Scientific Data, 4*, 1–21. https://doi.org/10.1038/sdata.2017.64

Pezzoni, M., Lissoni, F., & Tarasconi, G. (2014). How to kill inventors: Testing the Massacrator© algorithm for inventor disambiguation. *Scientometrics, 101*(1), 477–504. https://doi.org/10.1007/s11192-014-1375-7

Raffo, J., & Lhuillery, S. (2009). How to play the "Names Game'': Patent retrieval comparing different heuristics. *Research Policy, 38*(10), 1617–1627. https://doi.org/10.1016/j.respol.2009.08.001

Trajtenberg, M., Shiff, G., & Melamed, R. (2006). *The "Names Game": Harnessing inventors' patent data for economic research*. National Bureau of Economic Research Working Paper 12479.

Ventura, S. L., Nugent, R., & Fuchs, E. R. H. (2015). Seeing the non-stars: (Some) sources of bias in past disambiguation approaches and a new public tool leveraging labeled records. *Research Policy, 44*(9), 1672–1701. https://doi.org/10.1016/j.respol.2014.12.010

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the Fellegi–Sunter model of record linkage. In *Proceedings of the American Statistical Association Section on survey research methods*. https://doi.org/10.1016/0140-7007(90)90071-4.

Yang, G. C., Liang, C., Jing, Z., Wang, D. R., & Zhang, H. C. (2017). A mixture record linkage approach for US patent inventor disambiguation. In *Advanced multimedia and ubiquitous engineering. FutureTech 2017, MUE 2017*, 2017. Lecture Notes in Electrical Engineering (Vol. 448, pp. 331–338). https://doi.org/10.1007/978-981-10-5041-1.

Yin, W., Schütze, H., Xiang, B., & Zhou, B. (2016). ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics, 4*, 259–272. arXiv:1512.05193

Zagoruyko, S., & Komodakis, N. (2015). Learning to compare image patches via convolutional neural networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2015. https://doi.org/10.1109/CVPR.2015.7299064.

Zbontar, J., & LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research, 17*, 1–32. https://doi.org/10.1103/PhysRevE.93.033307

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 2015 (pp. 1–9). arXiv:1502.01710