Parallel corpus approach for name matching in record linkage

Jeffrey Sukharev\textsuperscript{1,2}, Leonid Zhukov\textsuperscript{1,3}, and Alexandrin Popescul\textsuperscript{1}

\textsuperscript{1}Ancestry.com, San Francisco, USA
\textsuperscript{2}University of California, Davis, USA
\textsuperscript{3}Higher School of Economics, Moscow, Russia
\{jsukharev,lzhukov,apopescul\}@ancestry.com

Abstract—Record linkage, or entity resolution, is an important area of data mining. Name matching is a key component of systems for record linkage. Alternative spellings of the same name are a common occurrence in many applications. We use the largest collection of genealogy person records in the world together with user search query logs to build name-matching models. The procedure for building a crowd-sourced training set is outlined together with the presentation of our method. We cast the problem of learning alternative spellings as a machine translation problem at the character level. We use information retrieval evaluation methodology to show that this method substantially outperforms on our data a number of standard well known phonetic and string similarity methods in terms of precision and recall. Our result can lead to a significant practical impact in entity resolution applications.

Keywords—Record Linkage, Crowd Sourcing, Machine Translation

I. INTRODUCTION

Due to proliferation of very large collections of data in both commercial and government applications linking records from multiple sources becomes increasingly important. Healthcare, banking, and genealogy services are all facing the problem of identifying similar records using various fields with a non-unique identifier.

A person’s name, especially the family name, is the key field used in identifying person’s records in databases. Software tools and applications designed for entity resolution of a person’s records usually rely on the person’s name as a primary identification field. In particular, genealogy services provide user access to a person’s record databases and facilitate search for a user’s ancestors/relatives and other person/records of interest. A person’s records indicate some type of a life event including birth, death, marriage or relocation. Typically, records are indexed by some unique identifier and can also be searched by a combination of last/first names, geographical locations and event dates. Searching a record database is complicated by the user not knowing the exact spelling of the name in the record they are searching for. This task becomes even harder since databases often contain alternate spellings referring to the same person. This transpires due to many factors including optical character recognition errors when scanning the records, errors/misspellings of names in the records themselves, and name transliterations. For instance, a common last name “Shepard” has been also commonly spelled as “Shepherd”, “Sheppard”, “Shephard”, “Shepperd”, and “Sheperd”. Having methods that would provide users and the search engines with a list of highly credible alternative spellings of the query name would significantly improve the search results.

Knowing how to misspell names to find records of interest has always been a part of a professional genealogist’s domain expertise. In this paper we try to bridge this gap and bring this power to the average user by employing data-driven methods that rely on our unique data set\textsuperscript{1} Through Ancestry.com databases, we have access to the world’s largest genealogy data repository. The main function of a genealogy service is to facilitate discovery of relevant person’s records and the construction of family trees. Tree construction involves user attaching relevant scanned/digitized records, found in the organization’s databases, to user-generated tree node. Records attached to individual tree nodes give us candidates of alternative spellings. By leveraging user-provided links between individual user family tree nodes and scanned attached records, we generated a “labeled” dataset of name pairs of user supplied names and names from attached records. All user-identifying information except for last name pairs is discarded from the final dataset. We filter and pre-process this list of name pairs and use it to train a model using standard machine translation (MT) methods. We describe data pre-processing in Section V.

Additionally, we generated another dataset from company search logs. Often, users reformulate their search queries in hope of getting better results. By identifying logged-in user sessions and extracting names from user queries in sequential order from the same session in a specified time interval, we have been able to accumulate a large number of alternative spelling candidate pairs. To become a training set, the pairs are then processed similarly to the first dataset.

As a result of our experiments, we produce ranked candidate variant spellings for each query name. In addition to providing the translation model, we also propose a methodology, adapted from the information retrieval community, for evaluating the final candidate list and comparing it with other methods. Our results show that our methods perform significantly better in identifying a quality lists of alternative name spellings than other state-of-the-art methods in terms of precision and recall.

The remainder of the paper is organized as following. We discuss previous work in Section II. A detailed description of our training data is given in Section III. In Section IV we\textsuperscript{1}Training set is available for research purposes from the authors. "https://github.com/jeffsicdm14/name_pairs.git".
outline the machine translation method used in training our model. We then discuss our results and present comparisons with other methods in Section V and conclude in Section VII.

II. PREVIOUS WORK

The classic reference in the field of record linkage is a paper by Fellegi and Sunter [9] published in 1969. In their work, the authors have carefully presented the theory of record matching. Their work laid the probabilistic foundation of the record linkage theory. Since this seminal work, there has been a proliferation of work in this area. In the interest of brevity, we direct the reader to the outstanding 2006 survey paper by Winkler [23] and to the comprehensive work by Christen [7].

With the explosive growth of web data, it is becoming imperative to discover more accurate methods for record matching. Historically, methods focusing on name matching could be separated into two classes: sequential character methods and bag-of-words methods [15]. In this paper we deal with a single word last names only and, hence, we are not going to be discussing bag-of-words methods.

Phonetic similarity methods are an important category of sequential character methods. The key idea behind these methods is to map n-grams of characters into phonetic equivalents. The output of using these methods on string pairs is a binary decision and not a degree of similarity. The best-known method from this class is Soundex [19]. Over the years, numerous improvements of this approach have been made; in particular, for accommodating non-English names. Popular methods include Soundex [19], Double Metaphone[17], and NYSIIIS [21]. While these methods proved to be useful in improving performance in data matching applications, they do not solve the problem of relevance ranking of alternative spellings. Another important category of sequential character methods often used in conjunction with phonetic methods is the class of static string similarity measures. Similarity method based on Levenshtein edit distance [14] is the most well-known method of this type. Other common similarity measures include the Jaro [10] method which takes into account the number and order of common characters between two strings and the Jaro-Winkler [22] method which extends Jaro by accounting for the common prefixes in both strings [3], [7]. The static similarity measures described above, while useful in measuring similarity and ranking alternative spellings, are not capable of generating alternative spellings. This capability is typically absent from all methods that do not take a dataset’s statistical information into account.

Kernighan et. al. [12] proposed a method for spelling corrections based on noisy channels. The same formulation would later be used in the machine translation field. The basic idea was to find the best possible correction by optimizing the product of language model and correction model. For a comprehensive survey of spelling correction methods, the reader should look at the excellent chapter on this topic at Jurafsky and Martin [11].

In the last several decades, machine translation methods have gained significant traction and recently found their way into the problem of name matching. In 2007, Bhagat et. al. [2] implemented a transducer based method for finding alternative name spellings by employing a graphemes-to-phonemes framework. In 1996, Ristad and Yianilos [18] presented an interesting solution where the cost of edit distance operations is learned. Their model resulted in the form of a transducer. Because of the difficulty associated with obtaining the experimental data, many researchers build their own synthetic datasets by utilizing specialized tools for mining the web and extracting words that appear to be last names. The resulting names are used in forming artificial pairs using one or more similarity measure typically based on edit distance. Another popular alternative is to hire human annotators who create last name pairs based on their knowledge of other possible name spellings. These methods may lead to an undesirable bias. In our case we have literally millions of annotators producing training set and we believe that the “wisdom of crowds” present in our user-labeled dataset helps us overcome the bias associated with relying on artificial associations or the knowledge of a small number of annotators.

III. DATASET DESCRIPTION

Ancestry.com has over the years accumulated over 14 billion records and over 60 million personal family trees [1]. Most of the records in our company’s database originate from Western European countries, United States, Canada, Australia, and New Zealand. Scanned collections of census data, as well as personal (public and private) documents uploaded by company users comprise the bulk of the company’s datasets. One of the key features of Ancestry.com website is the facility for building personal family trees with an option for searching and attaching relevant documents from record databases to the relevant parts of family trees. For example, if a family tree contains a node for a person with the name “John Smith” it would be often accompanied by the birth record, relocation record, marriage record, and other records discovered by the owner of the family tree. Since most of the nodes in the deep family trees involve persons who are no longer living, death records can often be discovered and attached to the appropriate nodes.

This linkage between user-specified family tree nodes and the official records present us with a unique opportunity to assemble a parallel corpus of pairs of names, hand-labeled by the users themselves. Due to the availability of the labeled dataset in Ancestry.com database, we have a more direct way of generating training data. We avoided methodological complications associated with having to define negative examples needed for training when approaching the problem as binary classification problem. While it would have been possible to mine positive sets from user-labeled data, defining the process generating realistic negative examples is ambiguous at best. As a consequence of having this obstacle in front of us, we turned toward machine translation methods because only a parallel corpus was needed to train the translation model. Given the way Ancestry.com’s users interact with the genealogy service, we isolated two separate ways of collecting parallel corpus data that would later be used for training translation models and for testing.

The first process of assembling a parallel corpus consists of collecting all directed pairs of names drawn from user tree nodes and their attached anonymized records. We chose pair direction as following: last names on the left come from tree nodes and last names on the right come from the records. Since last names in records and tree nodes have different distributions, taking directionality into account is important when choosing the training set of pairs. A number of filtering steps have been applied in order to de-noise the datasets and
will be discussed in more detail in the later sections. The pairs are directed which implies that a pair “Johansson” - “Johanson” would be different from the reverse pair “Johanson” - “Johansson”. This would manifest in a separate co-occurrence count for each pair.

The second process for building a parallel corpus involves using recorded user search queries. Since Ancestry.com search query form asks the user for specific fields when searching for trees or records, we have been able to extract user queries containing names from a search log. By grouping users by their loginname, sorting the queries in chronological order, and fixing the time interval at 30 minutes, we have been able to extract directed pairs of names queried by users. Our built-in assumption is that frequently users do not find what they are looking for on their first attempt, and if that is the case, they try again. The resulting data set is also noisy and requires extensive filtering. Each pair has a direction from an initial name spelling to its reformulation. For example, if a user A searches for name “Shephard” at time $t_0$ and then searches for name “Shepperd” at time $t_1$ where $t_1 - t_0 < 30$ then the resulting pair will be: “Shephard” - “Shepperd”.

Table I provides an illustration of a sample of “records” dataset grouped by Levenshtein edit distance and sorted by co-occurrence count. The distribution of values of edit distances between names in each pair on Table II for each dataset. The share of deletes/inserts and replaces in distance $d$ for name “Johansson” is $33.9\%$, and “Thorogood” is $33.9\%$. n-gram model computes a probability of a character sequence where each subsequent character depends on $n-1$ previous characters in the sequence.

An alignment model is used in generating translational correspondences between names in our context and it can be best described by an example shown on Figure 2. Here the name “Thorogood” is aligned with the name “Thoroughgood”. Looking at the Figure 2 prefix “Tho” and suffix “good” demonstrate simple 1-to-1 alignment. Other more involve alignments are indicated with arrows. Letter ‘h’ in “Thorogood” does not align with anything in “Thoroughgood”. Estimating alignment model results in generation of alignment rules such as the ones that we just presented.

We are not only interested in a single best alternative spelling, but also in the ranked list of best suggestions that can be computed from the same distribution by sorting probabilities in decreasing order:

$$\arg\max_{name} P(t_{name}|s_{name}) = \arg\max_{name} P(t_{name})*P(s_{name}|t_{name})$$

where $\arg\max$ represents operator that finds top $K$ target names that maximize $P(t_{name}|s_{name})$. Finding $P(t_{name}|s_{name})$ accurately without using the equation above would be challenging. However, using Bayes’ rule and breaking down $P(t_{name}|s_{name})$ into language model $P(t_{name})$ and alignment model $P(s_{name}|t_{name})$ allows us to get a theoretically good translation even if underlying probabilities are not that accurate [4]. $P(s_{name})$ is fixed and does not depend on the optimization variable $t_{name}$ and hence, will not influence the outcome and can be discarded. We compute probability values for name and alignment model using machine translation tools replacing sentences with names and words with characters. endcenter For training of our language and alignment models we have chosen Moses, a widely known open-source statistical machine translation software package [13]. We chose to use Moses’ Baseline System training pipeline. It includes several stages:

**Preparing the dataset:** tokenization, truecasing, and cleaning.

### Table I: Example name pairs and statistics

<table>
<thead>
<tr>
<th>Levenshtein edit distance</th>
<th>name#1</th>
<th>name#2</th>
<th>co-occurrence</th>
<th>count#1</th>
<th>count#2</th>
<th>Jaro-Winkler</th>
<th>Jaro</th>
<th>Jaccard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>clark</td>
<td>clark</td>
<td>139024</td>
<td>1168804</td>
<td>335902</td>
<td>0.922</td>
<td>0.889</td>
<td>0.102</td>
</tr>
<tr>
<td></td>
<td>bailey</td>
<td>bailey</td>
<td>89910</td>
<td>725361</td>
<td>123012</td>
<td>0.922</td>
<td>0.889</td>
<td>0.119</td>
</tr>
<tr>
<td></td>
<td>parish</td>
<td>parish</td>
<td>77529</td>
<td>179308</td>
<td>138774</td>
<td>0.933</td>
<td>0.905</td>
<td>0.322</td>
</tr>
<tr>
<td>2</td>
<td>seymour</td>
<td>seymore</td>
<td>15583</td>
<td>90071</td>
<td>24127</td>
<td>0.907</td>
<td>0.810</td>
<td>0.158</td>
</tr>
<tr>
<td></td>
<td>schumacher</td>
<td>schumaker</td>
<td>6013</td>
<td>52769</td>
<td>12867</td>
<td>0.884</td>
<td>0.793</td>
<td>0.101</td>
</tr>
<tr>
<td></td>
<td>bohannon</td>
<td>bohanan</td>
<td>5902</td>
<td>44770</td>
<td>16252</td>
<td>0.854</td>
<td>0.738</td>
<td>0.107</td>
</tr>
<tr>
<td>3</td>
<td>arsenault</td>
<td>arsenaeu</td>
<td>1489</td>
<td>11455</td>
<td>4305</td>
<td>0.838</td>
<td>0.769</td>
<td>0.104</td>
</tr>
<tr>
<td></td>
<td>blackshear</td>
<td>blackshire</td>
<td>1269</td>
<td>9556</td>
<td>3049</td>
<td>0.884</td>
<td>0.793</td>
<td>0.112</td>
</tr>
<tr>
<td></td>
<td>grimwade</td>
<td>greenwade</td>
<td>781</td>
<td>1886</td>
<td>2480</td>
<td>0.764</td>
<td>0.611</td>
<td>0.218</td>
</tr>
<tr>
<td>4</td>
<td>sumarlidasson</td>
<td>somerledsson</td>
<td>671</td>
<td>674</td>
<td>1526</td>
<td>0.752</td>
<td>0.628</td>
<td>0.439</td>
</tr>
<tr>
<td></td>
<td>riedmueller</td>
<td>reidmiller</td>
<td>143</td>
<td>438</td>
<td>556</td>
<td>0.736</td>
<td>0.664</td>
<td>0.168</td>
</tr>
<tr>
<td></td>
<td>braunberger</td>
<td>bramberg</td>
<td>131</td>
<td>624</td>
<td>277</td>
<td>0.802</td>
<td>0.674</td>
<td>0.170</td>
</tr>
</tbody>
</table>

**IV. METHODS**

The problem of finding best alternative name spellings given a source name can be posed as maximization of conditional probability $P(t_{name}|s_{name})$, where $t_{name}$ is a target name and $s_{name}$ is a source name. Following the traditions of statistical machine translation methods [4], this probability can be expressed using Bayes’ rule as:

$$P(t_{name}|s_{name}) = \frac{P(s_{name}|t_{name}) * P(t_{name})}{P(s_{name})}$$

where $P(t_{name})$ is a name model (corresponds to language model in machine translation literature) and describes frequencies of particular name/language constructs. $P(s_{name})$ is a probability of a source name. $P(s_{name}|t_{name})$ corresponds to alignment model.

Name model can be estimated using character n-grams language model representation by finding the probabilities using the chain rule.

$$P(c_1c_2...c_m) = \prod_{i=1}^{m} P(c_i|c_{\text{max}(1,i-(n-1))}, ..., c_{\text{max}(1,i-1)})$$

where $c_i$ is an $i^{th}$ character in the sequence of characters that comprise a name of length $m$. n-gram model computes a probability of a character sequence where each subsequent character depends on $n-1$ previous characters in the sequence.
Tokenization involves including spaces between every character. Truecasing and cleaning deal with lowercasing each string and removing all non-alphabetic characters among other things.

**Language model training:** A language model is a set of statistics generated for an n-gram representation built with the target language. For this purpose we used IRSTLM [8], a statistical language model tool. As a result, we generated 2-gram through 6-gram language models (6 was the maximum possible). This step adds “sentence” boundary ("word" boundary here) symbols and, as in the Baseline System, uses improved Kneser-Ney smoothing. We follow a common practice in machine translation where all examples of the target language, not only forms present in parallel corpus translation pairs, are used to construct a language model. This can lead to an improved translation quality. In our experiments with search logs and tree attachment datasets we used their respective lists from 250,000 most frequent last name forms for language model estimation.

**Alignment model building:** Moses uses the GIZA++ package for statistical character-alignment [16] character (word)-alignment tools typically implement one of Brown’s IBM generative models [5] that are being used for determining translation rules for source language to the target language. We created an alignment model for each of the 2-gram through 6-gram language models created in the previous step. As in the Baseline System, the “alignment” option was set to “grow-diag-final-and” and the “reordering” option was set to “msd-bidirectional-fe”.

**Testing.** We tested decoding on test folds in a batch mode with an option “-n-best-list” to give top 1,000 distinct translations. This value was chosen large to well represent the high recall area on respective precision-recall curves.

We followed the Baseline System with the exception of tuning phase and replacing our source and target languages with sequences of characters and instead of sequences of words. The tuning phase consists of steps optimizing the default model weights used in the training phase. We have omitted this phase because based on our initial tests, it didn’t give immediate accuracy improvements on our datasets and it is relatively slow.

### V. Training Set Generation

Since we dealt with user-generated data, we had to devise an algorithm for treating the data and generating a high-confidence training set. We outlined the following procedure:

1) Initially, a “universe” of names was defined. All names in tests and training sequences came from this “universe”. A set of names was selected by taking top 250,000 most frequent names from both datasets (“search” and “records”).

2) For each pair selected using the procedure outlined in Section III we made sure that each name comes from our set of high-frequency names. This step resulted in 12,855, 829 pairs in the “search” dataset and 51,744, 673 pairs in the “records” dataset.

3) In order to de-noise the name pairs we selected the top 500k/250k pairs by co-occurrence for the “records” and “search” datasets respectively.

4) The remaining pairs were passed through the Jaccard index filter \( J(A, B) = \frac{|A \cap B|}{|A \cup B|} \) where \( A \) is a set of users linked with the first name from a name pair and \( B \) is set of users linked with the second name from a name pair. Users are identified by either their login session (“search” dataset) or by userid (“records” dataset). The reason that users were used in calculating Jaccard instead of just using co-occurrence counts and marginal counts has to do with preventing a few highly active users from skewing the results. This filter was used to remove name pairs that would be likely to co-occur by chance due to high frequency of each individual name involved in a pair. For instance, “Smith”-“Williams” pair would be filtered out. After filtering, we were left with 25k “record” and 12.5k “search” name pairs.

5) In the final step, we estimated the rate of “obvious” false positives based on manual checks and similarity measures cross checking. Looking at random samples stratified by edit distance, we manually evaluated these samples to estimate the false positive percentage. We estimated that the rate of obvious false positives is 1.5% in “search” dataset and 1.4% in “records” dataset. We specifically avoided using string-based similarity criteria when defining parallel corpus to prevent introducing bias. In principle, extra filters can be applied to training sets.

The flow of dataset preparation is summarized in Figure 1.

### VI. Results and Discussion

Comparing phonetic methods with similarity measures and with machine translation methods is not straightforward. Phonetic methods only allow for binary responses (match or mismatch) when applied to name pairs. Therefore, it is impossible to rank positive matches without introducing additional ranking criteria. Our machine translation method produces a score that...
we use in ranking. Similarity methods produce a similarity value that is also used in ranking. To get a meaningful comparison of these methods with phonetic methods, we had to make use of statistics that we gathered while processing datasets. Additionally, we devised a unified methodology that could be applied to all listed method types. In all our experiments, we used 10-fold cross validation for evaluating how the results of predictions generalize to previously unseen data. Training folds were used to train the MT models. The same test folds were used to test all methods, including MT-generated models, phonetic methods, and similarity measures. Generating results involves building a consistent metric that can be plotted and compared between different methods. We adapted a standard information retrieval performance metric: precision and recall. The methods with higher values of precision and recall when comparing with other methods for the same positions are superior.

Each test fold contains a source name and one or more target names associated with each source name. Each of our methods for each source name would produce its own list of target names. Since the number of suggested target names (or alternative spellings) can be large we needed to find a suitable method for ranking target names. For all target names for position/rank \( i \) in the range from 1 to \( N \) corresponding recall, and precision, are calculated. So, we had to agree on what precisely we mean by rank for phonetic methods, similarity methods and machine translation methods.

We decided to view ranking as the product of

\[
\text{rank}(s, t) = \text{alignmentScore}(s, t) \cdot \text{languageModelScore}(t)
\]

For the machine translation method (generated using Moses software library), we used model-applier output scores, which already contain the product of language model score and alignment score. For machine translation where character is a word:

\[
\text{rank}(s, t) = \text{mosesScore}(s, t)
\]

For phonetic algorithms \( \text{languageModelScore}(t) \) is the frequency of a name in the dataset (freq).

\[
\text{rank}(s, t) = \frac{\text{hasSameCode}(s, t)}{\text{freq}(t)}
\]

where \( \text{hasSameCode}(s, t) \rightarrow \{0, 1\} \) and \( \text{freq}(t) \) represents the frequency of name \( t \) in the dataset.

For similarity measures, we also used name frequency, but we had to experimentally find a suitable exponential constant \( \gamma \) to avoid over-penalizing low-frequency names.

\[
\text{rank}(s, t) = \text{sim}(s, t) \cdot \text{freq}(t)^\gamma
\]

where \( \text{sim}(s, t) \rightarrow [0, 1] \) represents the floating point similarity values and \( \gamma \) is the exponential constant used to control the frequency values. We used \( \gamma \) value 0.01. We identified this value using linear grid search where we ran our experiments using \( \gamma \) values from the range between 0.0 and 1.0.

After saving precomputed sorted lists (according to ranking) of alternative name spellings for each method (phonetic, similarity, MT methods), we computed Precision and Recall values for each position from 1 to \( N \) separately for each test fold. Resulting confidence bands shown on Figure 3 show at least 95% confidence bound. For description of how we calculated precision-recall curves’ confidence contours based on individual cross validation curves please see our technical report [20].

We ran 7o experiments on phonetic methods. Seven commonly used phonetic methods were selected for testing and these methods were applied on the same ten test folds. 90 experiments were conducted with distance metrics methods (Jaro, Levenshtein, Winkler-Jaro). We experimented with three values when choosing suitable \( \gamma \) parameter for distance measurement methods ranking.

Our main results are shown on Figure 3. Here, we present the comparison of all alternative name generating methods on precision-recall plots. It is clear that for both datasets MT methods perform better than all other methods and that similarity methods generally outperform phonetic methods.

Our results indicate general consistency when using test data from both datasets (“search” and “records”). NYSIIS phonetic method [21] significantly outperforms other phonetic methods. Phonex method appears to be the weakest performer of the phonetic methods we have looked at. Other phonetic methods lie in the middle and their confidence bands overlap. Because of the overlapping regions, we cannot definitively rank the performance of these methods. When choosing \( n \) in for n-gram language model for MT methods, we found that 5-gram slightly outperformed MT methods with other choices of \( n \). Therefore, we have selected 5-gram method to represent MT methods when comparing with phonetic and similarity methods.

VII. CONCLUSION

In this paper, we presented a novel way of approaching the alternative name spelling generation problem. The main conclusion of this work is that machine translation methods we have employed for finding ranked list of alternative last name spellings far-outperformed all other methods we tried. Our results, also, indicated that Jaro-Winkler similarity method together with the Levenshtein edit distance method performed better than the Jaro method. Additionally, the NYSIIS phonetic method significantly outperformed other phonetic algorithms and the Phonex phonetic method did not perform as well on our data.

The practical takeaway from our results can be two-fold. First, machine translation methods show obvious superiority to phonetic and similarity methods in terms of precision and recall and should be used if training data is available and if the most accurate list of alternative name spellings is required. Second, for applications where machine translation methods may not be utilized effectively or where a training set cannot be easily obtained, NYSIIS phonetic algorithm should be used in implementing blocking techniques for various applications and either Jaro-Winkler or Levenshtein methods should be used when selecting similarity measures.

ACKNOWLEDGMENTS

The Febrl library [6] implemented by Christen was used for calculating phonetic codes and string similarity values. Also
Fig. 3: MT moses-5-gram method clearly outperforms all other methods on both datasets. The general order of methods on “records” dataset looks similar to the order resulted from running on “search” dataset. Similarity measures such as Levenshtein, Jaro and Jaro-Winkler all perform better than phonetic methods. NYSIIS appears to significantly outperform other phonetic algorithms.

we would like to thank Mikhail Bilenko, prof. Nelson Max and the anonymous reviewers for their careful review of our work.

REFERENCES