Using NLP Enabled Text Search for Entity Disambiguation and Data Cleanup

Team: Kaitlin Coltin, Sumeet Siddhartha

Lead: Subadhra Parthasarathy

Contents

Applications for text search in current scenarios

Text search is a challenging problem affecting a variety of businesses and industries. Consider the following use cases.

- **Enterprise search:** Businesses leverage text search to provide relevant, impactful information to intranet users.
  Example: An employee uses an internal web browser to search for business resources. User input is prepared (for example, by making all text lowercase, removing punctuation and stop words, and lemmatizing verbs). The prepared input is compared to strings in an enterprise database. The strings which match most closely are returned to the user in order of similarity, so the user sees the most relevant results first.

- **Data analysis and disambiguation:** Data analysts rely on text search to identify patterns and mitigate ambiguity by grouping similar entities.
  Example: A census worker must group individuals by surname. However, the same name can have multiple different spellings or transliterations (e.g. Smyth, Smith). To account for such differences, the names are encoded based on their phonetic pronunciation and stored in a database. The worker can then find all names with equivalent pronunciations regardless of spelling.

- **Data cleaning and record linkage:** Text search helps businesses maintain data quality by identifying all records across a dataset that refer to the same entity.
  Example: A data science team wants to analyze sales of a specific drug year over year, but the same drug appears across multiple rows, columns, and tables under various brand names with various typos and abbreviations. The team implements a solution to standardize and compare drug names so they can search for records containing drug A. They find drug B in table 1 and drug C in table 2 are both equivalent to drug A, allowing them to link records across the two tables and eliminate duplicate information.

Case Study

Fuzzy matching added value to an engagement for a multinational news and publishing organization. The client was interested in extracting key events from news articles to share with its clients to increase profitability. Deloitte implemented a proof of concept demonstrating how key events, entities, and relationships between them could be extracted from articles to deliver information of value.

The volume and variety of articles presented a challenge because an entity of interest, such as a company or location, frequently appeared multiple times across multiple articles under several aliases. For example, "Amazon," "Amazon.com", and "Amzaon" all refer to Amazon.com, Inc. To extract accurate information and avoid duplicate events, there needed to be a way to disambiguate entities from one another and determine which names referred to the same entity.

Deloitte implemented a search solution using fuzzy matching to map company and location names extracted from articles to standardized entities in a database. Approximately 70% of extracted company names and 75% of extracted location names were successfully matched to standard entities. The search process improved data quality by checking entity names against accepted standard names and improved efficiency by helping identify duplicate events.

**Please do not share externally without permission*

In each case, text must be searched and matched to return relevant information to the user. Searches should not miss any relevant results, but users also should not have to sift through very many results to find something relevant. Furthermore, search results should be relevant even if user input contains typos, spelling errors, abbreviations, or other irregularities.

Our understanding of the problem

The key to successful text search is fuzzy matching, a natural language processing (NLP) technique for correctly matching pairs of equivalent but not necessarily identical strings. We present a solution framework for text search problems using fuzzy matching and compare the benefits and drawbacks of various fuzzy matching algorithms. Although the techniques we present are by no means exhaustive, we hope to lay the groundwork for further exploration by offering a robust and scalable solution while sharing our findings and recommendations.

Implementing a robust text search solution can enable business users to maximize the value of their data. However, there are two main challenges in any text search problem.

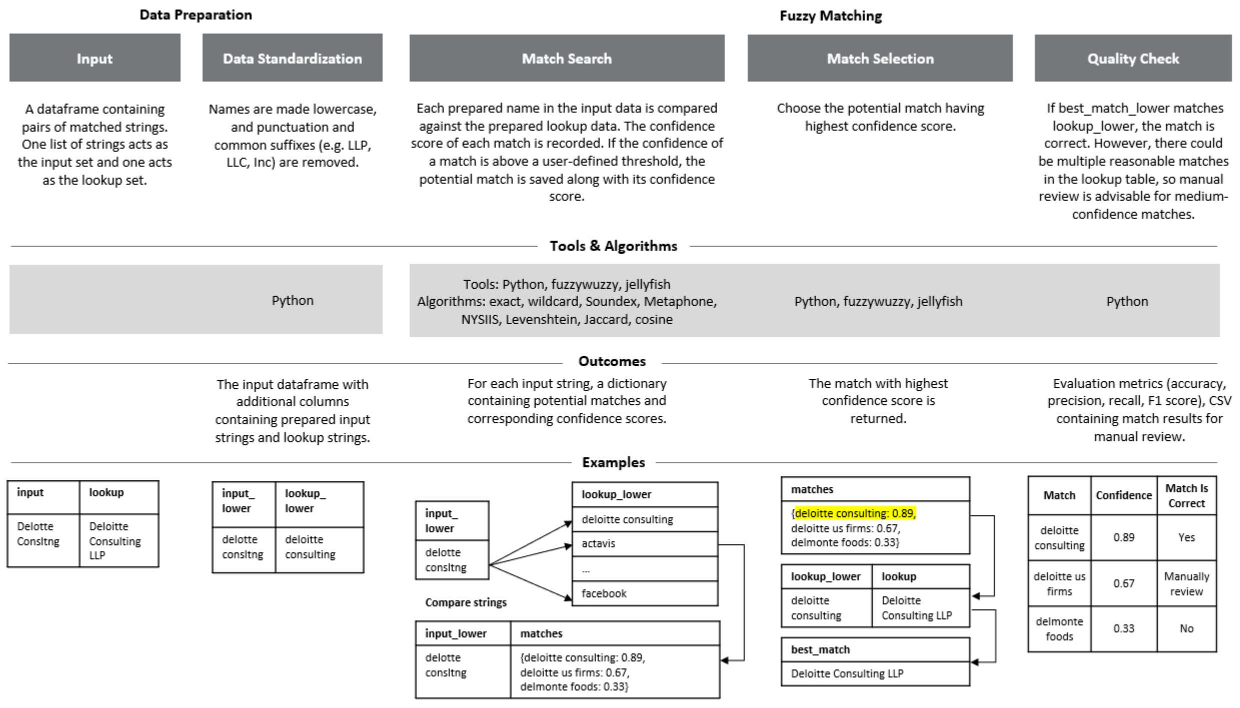
1. Trade-off between precision and recall: A good text search algorithm should be general enough to detect almost all equivalent versions of a string, but precise enough to return only matches that have a high probability of being relevant.

2. Computational complexity: Searching is challenging at scale because good fuzzy matching algorithms tend to be computationally expensive or slow on large datasets.

These challenges guided our comparison of the algorithms we implemented.

Solution framework

Our approach to text search involved two main phases: data preparation and fuzzy matching. A variety of algorithms were compared to determine the best approach. The framework is detailed in the below diagram, and descriptions of the algorithms used, dataset and programming environment, and orchestration pipeline follow.



Input: A dataframe containing pairs of matched strings. One list of strings acts as the input set and one acts as the lookup set.

Data Standardization: Names are made lowercase, and punctuation and common suffixes (e.g. LLP, LLC, Inc) are removed.

Match Search: Each prepared name in the input data is compared against the prepared lookup data. The confidence score of each match is recorded. If the confidence of a match is above a user-defined threshold, the potential match is saved along with its confidence score.

Match Selection: Choose the potential match having highest confidence score.

Quality Check: If best_match_lower matches lookup_lower, the match is correct. However, there could be multiple reasonable matches in the lookup table, so manual review is advisable for medium-confidence matches.

Tools & Algorithms: Python | Tools: Python, fuzzywuzzy, jellyfish; Algorithms: exact, wildcard, Soundex, Metaphone, NYSIIS, Levenshtein, Jaccard, cosine | Python, fuzzywuzzy, jellyfish | Python

Outcomes: The input dataframe with additional columns containing prepared input strings and lookup strings. | For each input string, a dictionary containing potential matches and corresponding confidence scores. | The match with highest confidence score is returned. | Evaluation metrics (accuracy, precision, recall, F1 score), CSV containing match results for manual review.

Examples:

input	lookup
Delotte Consltng	Deloitte Consulting LLP

input_lower	lookup_lower
delotte consltng	deloitte consulting

input_lower
delotte consltng

Compare strings

lookup_lower
deloitte consulting
actavis
...
facebook

input_lower	matches
delotte consltng	{deloitte consulting: 0.89, deloitte us firms: 0.67, delmonte foods: 0.33}

matches
{deloitte consulting: 0.89, deloitte us firms: 0.67, delmonte foods: 0.33}

lookup_lower	lookup
deloitte consulting	Deloitte Consulting LLP

best_match
Deloitte Consulting LLP

Match	Confidence	Match Is Correct
deloitte consulting	0.89	Yes
deloitte us firms	0.67	Manually review
delmonte foods	0.33	No

Algorithms

Fuzzy matching was the core of our text search solution. We implemented and compared the following fuzzy matching algorithms.

1. Distance algorithms (Levenshtein, Jaccard, cosine)
2. Phonetic algorithms (Soundex, Metaphone, NYSIIS)
3. Exact match (baseline)
4. Wildcard match (baseline)

Distance algorithms compare strings using some measure of distance. Levenshtein distance, for example, compares the edit distance of two strings, or the number of changes required to transform one string into another. The fewer changes required, the higher the similarity between the strings. Cosine similarity, on the other hand, involves representing strings as vectors (e.g., using TFIDF) and computing the cosine distance between them. We also experimented with the Jaccard index, which measures “distance” or similarity by treating strings as sets of letters and finding the ratio of the size their intersection to the size of their union. Intuitively, strings that are more similar will have a larger intersection and therefore a higher Jaccard index.

Distance algorithms involve a wide variety of approaches, but phonetic algorithms all follow the same basic logic. Strings are encoded according to how they are pronounced, and the encodings are compared. Phonetic algorithms are therefore able to match strings that sound the same but are spelled differently. We experimented with three phonetic algorithms: Soundex, Metaphone, and NYSIIS. Each algorithm encodes strings differently, so they produce somewhat different results.

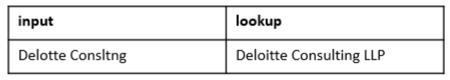
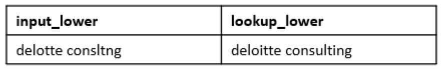
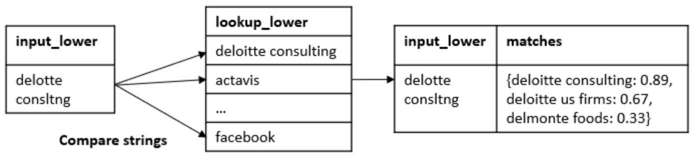
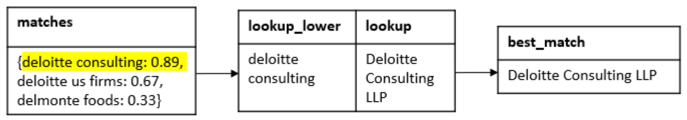
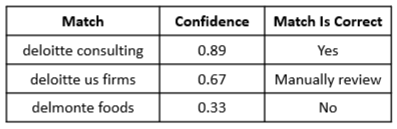
As a baseline for comparison, we also considered exact matches and wildcard matches. For two strings to be considered a wildcard match, one string must be a substring of the other.

Data and programming environment

Our experiments use a dataset of company names publicly available on GitHub. Our code is implemented in Python in Jupyter Notebook on an Ubuntu Linux virtual box with 8 CPUs. We used publicly available packages including fuzzywuzzy (Levenshtein distance) and jellyfish (phonetic algorithms) to implement the matching algorithms.

Orchestration pipeline

The below table outlines our approach and sample output. Our framework serves as a blueprint for approaching text search problems, because regardless of the use case the same basic steps apply.

Step	Description	Example
0. Input	A dataframe containing pairs of matched strings. One list of strings acts as the input set and one acts as the lookup set.	 input | lookup Delotte Consltng | Deloitte Consulting LLP
1. Standardize data	Names are made lowercase, and punctuation and common suffixes (e.g. LLP, LLC, Inc) are removed.	 input_lower | lookup_lower delotte consltng | deloitte consulting
2. Match names	Each prepared name in the input data is compared against the prepared lookup data. The confidence score of each match is recorded. If the confidence of a match is above a user-defined threshold, the potential match is saved along with its confidence score.	 input_lower delotte consltng Compare strings lookup_lower deloitte consulting actavis ... facebook input_lower | matches delotte consltng | {deloitte consulting: 0.89, deloitte us firms: 0.67, delmonte foods: 0.33}
3. Select the best match	Choose the potential match having highest confidence score.	 matches {deloitte consulting: 0.89, deloitte us firms: 0.67, delmonte foods: 0.33} lookup_lower | lookup deloitte consulting | Deloitte Consulting LLP best_match Deloitte Consulting LLP
4. Quality check the results	If best_match_lower matches lookup_lower, the match is correct. However, there could be multiple reasonable matches in the lookup table, so manual review is advisable for medium-confidence matches.	 Match | Confidence | Match Is Correct deloitte consulting | 0.89 | Yes deloitte us firms | 0.67 | Manually review delmonte foods | 0.33 | No

Learnings from implementing a selection of algorithms

We implemented the fuzzy matching algorithms to search for a set of input company names among a set of lookup company names. We ran each fuzzy matching algorithm three times, first on 1,000 records, then on 5,000 and finally on 10,000. For each iteration, we recorded the time in seconds, number of matches returned, number of matches correct, accuracy, precision, recall, and F1 score. We repeated each iteration twice and recorded the average of the two runtimes to account for variations in execution time. We analyzed the results and found distance algorithms outperformed phonetic algorithms, with cosine distance performing best overall. However, since each algorithm has its own strengths and weaknesses, it is usually best to combine multiple approaches. To demonstrate, we implemented three approaches in the following order and tested the hybrid approach on 2,000 records.

1. Check for exact matches
2. Apply the Metaphone phonetic algorithm to the remaining unmatched names
3. Apply the cosine distance algorithm to the remaining unmatched names

With the combined approach, we correctly matched 90% of input company names, more than was possible with any individual algorithm.

The following table contains the results of running each algorithm individually. The results reflect the two main challenges of text search: the trade-off between precision and recall, and growing time complexity as the number of records increases.

Algorithm Type	Algorithm	# of Records	Avg Time (s)	# of Matches Returned	# of Matches Correct	Accuracy	Precision	Recall	F1
Exact	Exact	1K	2.93	381	379	37.90%	99.48%	37.98%	54.97%
		5K	21.25	1829	1787	35.74%	97.70%	36.04%	52.66%
		10K	58.08	3751	3558	35.58%	94.85%	36.28%	52.48%
Wildcard	Wildcard	1K	61.38	512	492	49.20%	96.09%	50.20%	65.95%
		5K	1750.68	2577	2350	47.00%	91.19%	49.24%	63.95%
		10K	7058.43	5237	4536	45.36%	86.61%	48.78%	62.41%
Phonetic	Soundex	1K	4.37	823	533	53.30%	64.76%	75.07%	69.53%
		5K	61.3	4452	1615	32.30%	36.28%	74.66%	48.83%
		10K	189.29	9249	2352	23.52%	23.43%	75.80%	35.80%
	Metaphone	1K	4.54	463	459	45.90%	99.14%	46.08%	62.92%
		5K	62.8	2335	2192	43.84%	93.88%	45.13%	60.96%
		10K	199.26	4902	4399	43.99%	89.74%	46.32%	61.10%
	NYSIIS	1K	5.85	463	459	45.90%	99.14%	46.08%	62.92%
		5K	127.28	2234	2135	42.70%	95.57%	43.56%	59.84%
		10K	428.53	4645	4249	42.49%	91.47%	44.24%	59.64%
Distance	Levenshtein	1K	16.96	922	876	87.60%	95.01%	91.82%	93.39%
		5K	446.62	4808	4215	84.30%	87.67%	95.64%	91.48%
		10K	1737.3	9755	8115	81.15%	83.19%	97.07%	89.60%
	Jaccard	1K	5.16	891	746	74.60%	83.73%	87.25%	85.45%
		5K	86.17	4676	3245	64.90%	69.40%	90.92%	78.72%
		10K	303.62	9506	6023	60.23%	63.36%	92.42%	75.18%
	Cosine	1K	2.39	930	872	87.20%	93.76%	92.57%	93.16%
		5K	12.12	4707	4186	83.72%	88.93%	93.46%	91.14%
		10K	27.86	9518	8097	80.97%	85.07%	94.38%	89.48%

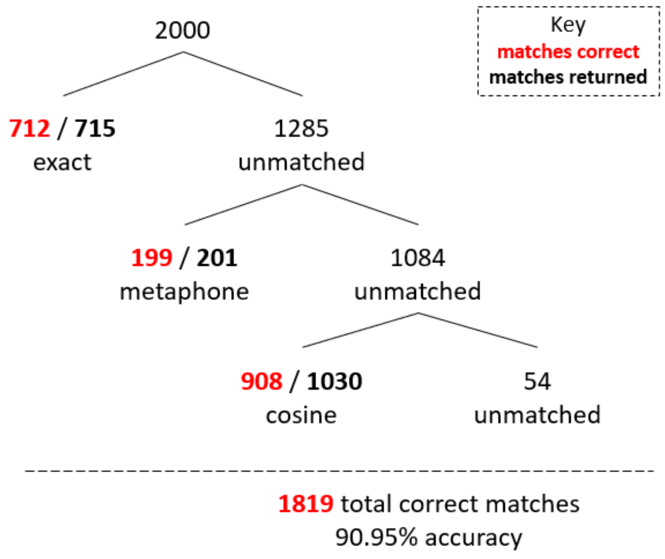
Overall, distance algorithms performed better than phonetic algorithms, with cosine distance achieving the best balance of speed and accuracy even as the size of the input data grew. Of the phonetic algorithms, Soundex was the least accurate because Soundex encodings have a maximum length of four characters, making it less effective at representing long strings. The Metaphone and NYSIIS phonetic algorithms do not have a maximum encoding length. Both Metaphone and NYSIIS achieved over 99% precision on 1,000 records, but their recall was under 50%. Some of the main reasons for false positives and false negatives were

1. Abbreviations (e.g. "intl" vs. "international") caused names to be encoded differently. A dictionary of abbreviations may help mitigate this challenge.
2. One company name was a substring of the other. The wildcard approach is designed to handle this case.
3. Minor spelling errors created a difference in pronunciation. Distance algorithms are better suited to address this challenge.

Our findings indicate that while phonetic algorithms can sometimes deliver high precision, distance algorithms allow greater flexibility in order to capture more potential matches. Phonetic algorithms are therefore most useful in situations where the cost of false positives is high, or for data like person names or location names, which may have multiple alternative spellings.

Both Levenshtein distance and cosine distance correctly matched over 80% of the company names in every trial. However, as the size of the input data increased, it became apparent that cosine distance was the top performer in speed and efficiency. Levenshtein distance performed similarly well in every metric, but cosine distance was significantly more time efficient. This was because the cosine algorithm utilized a TFIDF matrix as its underlying structure, whereas Levenshtein performed comparisons on pairs of strings. Although 10,000 is a very modest number of records considering real world implementation scenarios often involve millions of records, the results suggest cosine distance will continue to outperform Levenshtein in speed even as the size of the dataset increases.

Cosine distance was the top performer of any individual algorithm, but the hybrid approach proved most effective overall. We first checked for exact matches. Then we checked for phonetic matches using Metaphone because of its high precision and greater time efficiency compared to NYSIIS. Finally, we used cosine distance to find any remaining matches, since cosine distance had higher recall but lower precision compared to Metaphone. With the hybrid approach, we correctly matched 90% of company names. The tree diagram and table below illustrate the results.



Algorithm	# of Unmatched Input Records	Avg Time (s)	# of Matches Returned	# of Matches Correct	Cumulative Matches Returned	Cumulative Matches Correct	Cumulative Accuracy*
Exact	2000	4.76	715	712	715	712	35.60%
Metaphone	1285	6.04	201	199	916	911	45.55%
Cosine	1084	2.43	1030	908	1946	1819	90.95%

*Cumulative accuracy = $\frac{cumulative\ matches\ correct}{size\ of\ the\ dataset}$

Perhaps unsurprisingly, there is no silver bullet for text search problems. By testing and comparing a few approaches we arrived at a reliable hybrid solution.

Fuzzy matching as a source-to-target mapping solution

Our approach can be extended to any source-to-target string mapping problem, including data cleaning, record linkage, and entity disambiguation. In any such use case, there is a set of input strings and a set of target or lookup strings. The input must be standardized, such as by making all text the same case, removing punctuation and non-ASCII characters, standardizing common prefixes and suffixes, and in some cases stemming or lemmatizing verbs. The process then becomes a standard text search problem of looking for the input string among the lookup strings. Probable matches are found and saved. Depending on the use case, a "top match" may be selected based on some scoring criterion, or all the probable results may be returned for the user to review. Regular quality checks should be performed to maintain the system's accuracy over time.

Of course, real world data is often complicated and messy. Fortunately, fuzzy matching is well-suited to handle real world ambiguities. Using fuzzy matching, we were able to find a variety of match types that could not be found using an exact search. The below table contains a sampling of match types found.

Match Type	Input String	Lookup String	Algorithm
abbreviation	SNST PRPRTS INC	SUNSET PROPERTIES INC	Metaphone
abbreviation	Kr-US Exchng Cncl	Korea-US Exchange Council	cosine
typo	Fintech Advihsory	Fintech Advisory	Metaphone
spelling error	Dell Perot SystSems	Dell Perot Systems	cosine
missing space	PrescientLogistics, LLC	Prescient Logistics, LLC	cosine
truncated/ missing words	Willis Med Cen	Willis-Knighton Medical Center	cosine
substring	California State Coastal	California State Coastal Conservancy	cosine
different suffix	BRIGHTON MARINE HEALTH CENTER	Brighton Marine Hospital	cosine
reversed order	BARASH ASSOC, PETER	Peter Barash Assoc	cosine

Although any algorithm has its weaknesses and will not cover all possible cases, fuzzy matching greatly improves text search results by accounting for common types of variations between strings. Fuzzy matching is therefore a robust, NLP enabled approach for text search problems, and for source-to-target text mapping problems more generally.

Scalability and suggestions for further exploration

Since our approach assumes little about the data other than that it includes input strings and lookup strings, it can be applied to virtually any text search scenario. Our framework and experiments can therefore serve as a blueprint for approaching text search problems. The following are suggestions for practitioners interested in further exploring NLP enabled text search.

- Apply clustering techniques before searching names to narrow down the number of string comparisons that must be performed to improve speed and efficiency
- Explore additional fuzzy matching techniques, such as hashing, graph algorithms, ontology, and machine learning models
- Scale the implementation to handle larger amounts of data in a production environment

Those who are interested in scaling our approach will need to utilize infrastructure capable of handling big data, for example leveraging distributed computing across multiple CPUs or GPUs within a virtual box. It may also be possible to leverage inline SQL queries in Python to improve speed. For the case study described in the "Applications for text search in current scenarios" section of this paper, inline SQL was used within a Python script to perform exact and wildcard matches in a Google BigQuery database. The SQL results could be retrieved quickly and fed into a pandas dataframe to manipulate with Python. A similar approach could be used with any relational database provided a connection can be opened in Python. Finally, a simple UI can be built in Flask for users to conduct searches.

Text search in a challenging problem, especially at scale. However, with a clear approach and a few robust algorithms, it is possible to deliver high value text matching solutions.

References / resources

Codebase and dataset are available on Teams.
https://amedeloitte.sharepoint.com/:f:/r/sites/AIIEEmergentCapabilities/Shared%20Documents/04.%20NLP%20-%20NUG/Text%20Search?csf=1&web=1&e=DFImLT

Dataset source: https://github.com/bradhackinen/nama/tree/master/trainingData

“An Overview of Fuzzy Name Matching Techniques,” *Rosette Text Analytics*, Dec. 12, 2017. https://www.rosette.com/blog/overview-fuzzy-name-matching-techniques/ (accessed Nov. 17, 2020).

B. Mohr and K. Whalen, “Overview of Soundex - bradandkathy.com,” *BradandKathy.com*, Apr. 15, 2013. https://bradandkathy.com/genealogy/overviewofsoundex.html (accessed Nov. 17, 2020).

C. Ostertag, “Build a Scalable Search Engine for Fuzzy String Matching,” *Medium*, Oct. 26, 2019. https://towardsdatascience.com/build-a-scalable-search-engine-for-fuzzy-string-matching-751bd09f6074 (accessed Nov. 17, 2020).

C. van den Berg, “Super Fast String Matching in Python,” *van den Blog*, Oct. 14, 2017. https://bergvca.github.io/2017/10/14/super-fast-string-matching.html (accessed Nov. 17, 2020).

F. J. Carrera Arias, “Fuzzy String Matching in Python - DataCamp,” *Datacamp.com*, Feb. 06, 2019. https://www.datacamp.com/community/tutorials/fuzzy-string-python (accessed Nov. 17, 2020).

“How do Phonetic Coding Algorithms work?,” *IBM.com*, Jun. 01, 2005. https://www.ibm.com/support/pages/how-do-phonetic-coding-algorithms-work (accessed Nov. 17, 2020).

J. Turk, *jamesturk/jellyfish*. 2020. https://github.com/jamesturk/jellyfish
“Jaccard Index / Similarity Coefficient - Statistics How To.” https://www.statisticshowto.com/jaccard-index/ (accessed Nov. 17, 2020).

Piyush Sagar Mishra, “An Ensemble Approach to Large-Scale Fuzzy Name Matching,” *Medium*, Mar. 28, 2019. https://medium.com/bcggamma/an-ensemble-approach-to-large-scale-fuzzy-name-matching-b3e3fa124e3c (accessed Nov. 17, 2020).

“Soundex System,” *National Archives*, 5/302007. https://www.archives.gov/research/census/soundex (accessed Nov. 17, 2020).

“Text Matching: Cosine Similarity,” *Kanoki*, Dec. 27, 2018. https://kanoki.org/2018/12/27/text-matching-cosine-similarity/ (accessed Nov. 17, 2020).