The Journal of Open Source Software

Zoomerjoin: Superlatively-Fast Fuzzy Joins

Beniamino Green [1¶]

1 Yale University, USA ¶ Corresponding author

DOI: 10.21105/joss.05693

Software

- Review
- Repository
- Archive

Editor: Samuel Forbes

Reviewers:

- @cjbarrie
- @wincowgerDEV

Submitted: 06 June 2023
Published: 26 September 2023

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Researchers often have to link large datasets without access to a unique identifying key, or on the basis of a field that contains misspellings, small errors, or is otherwise inconsistent. In these cases, "fuzzy" matching techniques are employed, which are resilient to minor corruptions in the fields meant to identify observations between datasets. Most popular methods involve comparing all possible pairs of matches between each dataset, incurring a computational cost proportional to the product of the rows in each dataset $\mathcal{O}(mn)$. As such, these methods do not scale to large datasets.

Zoomerjoin is an R package that empowers users to fuzzily-join massive datasets with millions of rows in seconds or minutes. Backed by two performant, mutlithreaded Locality-Sensitive Hash algorithms (Broder, 1997; Datar et al., 2004), `zoomerjoin` saves time by not comparing distant pairs of observations and typically runs in linear ($\mathcal{O}(m+n)$) time. The algorithmic details are technical but the results are transformational; for the distance-metrics it supports, `zoomerjoin` takes seconds or minutes to join datasets that would have taken other matching packages hours or years.

Statement of Need

Fuzzy matching is typically taken to mean identifying all pairs of observations between two datasets that have distance less than a specified threshold. Existing fuzzy-joining methods in R do not scale to large datasets as they exhaustively compare all possible pairs of units and recording all matching pairs, incurring a quadratic $\mathcal{O}(mn)$ time cost. Perhaps worse, the most widely-used software packages typically also have a space complexity of $O(mn)$, meaning that a patient user cannot simply wait for the join to complete, as the memory of even large machines will be quickly exhausted (Robinson, 2020).

Zoomerjoin solves this problem by implementing two Locality-Sensitive Hashing algorithms (Broder, 1997; Datar et al., 2004) which sort observations into buckets using a bespoke hash function which assigns similar entries the same key with high probability, while dissimilar items are unlikely to be assigned the same key. After this initial sorting step, the algorithm checks pairs of records in the same bucket to see if they are close enough to be considered a match. Records in different buckets are never compared, so the algorithm takes $O(\max_{ij} m_i n_j)$ time to run (time proportional to the size of the largest bucket). In the ordinary case that each observation matches to few points in another dataset, the running time is dominated by the hashing step, and the program finishes in linear time using linear memory.

With this remarkable increase in speed comes two costs: Locality-Sensitive hashing is a probabilistic algorithm, so there is some probability that some true matches may be discarded by chance. This said, the chance that any matches are discarded by chance can be made arbitrarily low by changing parameters of the hash. Additionally, the LSH algorithms are more complex to implement than exhaustive searches; `zoomerjoin` only provides hashing schemes for two common distances, the Jaccard distance (for strings and other data that can be represented

as a set) and the Euclidean distance (for vectors or points in space).

Implementation Details:

Zoomerjoin allows users to fuzzily-join on the basis of two distance measures, the Euclidean distance and the Jaccard distance. The Jaccard similarity is defined over two sets, $\mathcal{A}$ and $\mathcal{B}$, as the cardinality of their intersection over the cardinality of their union. It can take values in the inverval [0,1].

$$sim(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}$$

Two strings can be compared by transforming them into sets of characters or sets of consecutive "shingles" of characters, then comparing them using the Jaccard distance.

The Euclidean distance, defined over two vectors, $\vec{a}$, and $\vec{b}$ is defined as the square root of the sum of squares of their componentwise distances, and can take values in the interval $[0, \infty)$:

$$dist(\vec{a}, \vec{b}) = ||\vec{a} - \vec{b}||_2$$

Integration between R and rust is managed by the `extendr` and `rextendr` Rust and R packages (Wilke et al., 2023). Instrumental to the package's fast performance is the relentlessly-optimized `dashmap` Rust crate (Wejdenstal, 2023), which provides a fast hash map that can be populated by many threads working in parallel. `Dashmap`'s concurrent hash maps are used to quickly store the hashes associated with each observation and allow multiple threads to compute hashes and write to the hash map at the same time. Parallel computation of the hashes is scheduled using the parallel iterators provided by the `rayon` crate (Matsakis & Stone, 2023). The Locality-Sensitive Hash implementation for the Jaccard distance is modeled after the textbook description by Leskovec et al. (2014), including the technique of storing the hashes of the tokens rather than the tokens themselves to save memory.

Benchmarks

I show how the runtime and memory consumption of the programs scale with the number of observations in the datasets being joined. I choose to include the `fuzzyjoin` package as a point of comparison as its superlative tidy syntax and fast underlying implementation backed by the multithreaded `stringdist` package (Loo, 2014) make it the de-facto standard for fuzzy-matching in R.

I compare both packages' time and memory usage joining two datasets using the Jaccard distance for strings and the Euclidean distance for points.

For the Jaccard distance benchmarks, I use both R packages to join rows of donor names from the Database on Ideology, Money in Politics, and Elections (DIME) (Bonica, 2016), a database of donors to interest groups, a dataset used to benchmark other matching / joining algorithms (Kaufman & Klevs, 2021).

For the Euclidean distance, I use the programs to link datasets of points drawn from a multivariate gaussian with 50 dimensions to a copy of this dataset with dataset with each observation shifted by adding a small ε along each axis. The exact code to generate this dataset can be seen below:

```
# R Code to create synthetic dataset for Euclidean Joining
n <- 10^5
p <- 50
X <- matrix(rnorm(n * p), n, p)
```

```
# First dataframe to join
X_1 <- as.data.frame(X)
# Second dataframe to join
X_2 <- as.data.frame(X + .000000001)
```

For both types of joins, I adjust the hyperparameters until the false-negative rate is less than .1%, meaning that fewer than .1% of all matches are discarded by random chance. More details, including the complete benchmarking code can be seen in the package's benchmarking vignette.

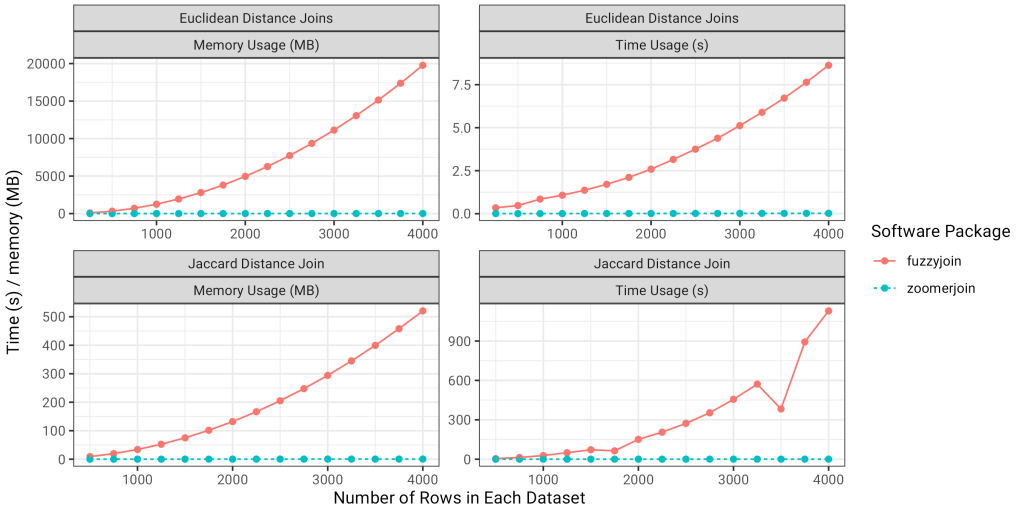

Figure 1: Memory Use and Runtime Comparison of Fuzzy-Joining Methods in R

Zoomerjoin achieves almost linear scaling in both runtime and memory, while `fuzzyjoin` scales quadratically in both quantities. Even for datasets with 2500 rows, `zoomerjoin` finishes in under a second. By contrast, the Jaccard-distance joins implemented in `fuzzyjoin` take over three minutes to join. For the largest Euclidean datasets, `fuzzyjoin` almost exhausts the 8GB memory capacity of the laptop used for benchmarking, while `zoomerjoin`'s memory rises above above 8 MB — a thousand-fold decrease.

Example Usage:

Zoomerjoin is designed to be easy to use for R users familiar with the popular dplyr grammar of data manipulation (Wickham et al., 2023). To give an example, I show how to use the `euclidean_inner_join` function (the fuzzy analogue of `dplyr`'s `inner_join`) to join the two datasets from the benchmarking example:

```
euclidean_inner_join(
        X_1, X_2,
        threshold = .1,
        n_bands = 90,
        band_width = 2,
        r = .1
        )
```

The first two arguments are exactly the same as those in the corresponding `dplyr` function, and should be familiar to most R users. The remaining arguments, `threshold`, `n_bands`, `band_width`, and `r` are specific to zoomerjoin, and determine how close units must be to be considered a match, as well as the performance / recall of the LSH scheme. As with `dplyr`, the package also allows users to perform other types of logical joins using the

`euclidean_(outer|left|right|full)_join` family of functions functions. A corresponding family of functions also exists for the fuzzy joins based on the Jaccard distance.

State of the Field:

To the best of my knowledge, no other packages exist for fuzzy-joining in sub-quadratic time in R. Two similar packages are the aforementioned `fuzzyjoin`, which provides fast, tidy joins for small to medium datasets, and the `textreuse` package (Mullen, 2020) which implements Locality-Sensitive Hashing, but does not offer a joining functionality, and is implimented mostly in R.

Zoomerjoin draws from both packages, and aims to synthesize and extend aspects of both to create a powerful joining toolset. The package combines the functionality of the tidy, dplyr-style `fuzzyjoins` provided by `fuzzyjoin` with the performance offered by a Rust implimentation the same Locality-Sensitive Hashing algorithm used in `textreuse`. The core of the package is written in performant Rust code, which makes the package suitable for datasets with hundreds of millions of observations.

Other Functionalities

The flagship feature of `zoomerjoin` is its fast, dplyr-style joins, but it also implements two other algorithms improved by locality-sensitive hashing: a fuzzy-string grouping function which is backed by locality-sensitive hashing, and an implementation of the probabilistic record-linkage algorithm based on the Fellegi-Sunter model (Fellegi & Sunter, 1969) developed by Enamorado et al. (2018).

The fuzzy-string-grouping algorithm provides a principled way to correct misspellings in administrative datasets by combining similar pairs of strings into groups with a standardized name. The probabilistic record-linkage algorithm described by Enamorado et al. (2018) provides a way to link entities between two datasets but involves comparing all possible pairs between each datasets. A simple pre-processing step with the Locality-Sensitive Hashing methods of `zoomerjoin` can drastically decrease the runtime by considering as potential matches units that have similar values of the blocking fields. This allows the algorithm to scale almost linearly with the size of the input datasets, at the cost of discarding a small amount of true matches excluded by the blocking scheme.

Limitations and Future Work

Zoomerjoin currently provides locality-sensitive hashing implementations for two distances: the Jaccard and the Euclidean distance. While these distance metrics are suitable for many if not most applications, researchers may wish to use other metrics, or bespoke combinations of distance metrics. Further work could extend the functionality of the package to also support LSH-backed joins based on other notions of distance such as the edit distance (Marçais et al., 2019) for strings, or the Manhattan distance for points.

Acknowledgements

I thank Jack Green, Cleo Falvey, John Cho, Matthew Dahl, and Amelia Malpas for their feedback and suggestions in designing the package and revising this manuscript.

References :

Bonica, A. (2016). *Database on ideology, money in politics, and elections: Public version 2.0 [computer file]*. https://data.stanford.edu/dime

Broder, A. Z. (1997). On the resemblance and containment of documents. *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*. https://doi.org/10.1109/sequen.1997.666900

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004, June). Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. https://doi.org/10.1145/997817.997857

Enamorado, T., Fifield, B., & Imai, K. (2018). Using a probabilistic model to assist merging of large-scale administrative records. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3214172

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*(328), 1183–1210. https://doi.org/10.1080/01621459.1969.10501049

Kaufman, A. R., & Klevs, A. (2021). Adaptive fuzzy string matching: How to merge datasets with only one (messy) identifying field. *Political Analysis*, *30*(4), 590–596. https://doi.org/10.1017/pan.2021.38

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139924801

Loo, M. P. J. van der. (2014). The stringdist Package for Approximate String Matching. *The R Journal*, *6*(1), 111–122. https://doi.org/10.32614/RJ-2014-011

Marçais, G., DeBlasio, D., Pandey, P., & Kingsford, C. (2019). Locality-sensitive hashing for the edit distance. *Bioinformatics*, *35*(14), i127–i135. https://doi.org/10.1093/bioinformatics/btz354

Matsakis, N., & Stone, J. (2023). *Rayon: Simple work-stealing parallelism for rust*. https://github.com/rayon-rs/rayon

Mullen, L. (2020). *Textreuse: Detect text reuse and document similarity*. https://CRAN.R-project.org/package=textreuse

Robinson, D. (2020). *Fuzzyjoin: Join tables together on inexact matching*. https://CRAN.R-project.org/package=fuzzyjoin

Wejdenstal, J. (2023). *Dashmap: Blazing fast concurrent HashMap for rust*. https://github.com/xacrimon/dashmap

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *Dplyr: A grammar of data manipulation*. https://CRAN.R-project.org/package=dplyr

Wilke, C. O., Thomason, A., Reimert, M. M., Kosenkov, I., Yutani, H., & Barrett, M. (2023). *Rextendr: Call rust code from r using the 'extendr' crate*. https://CRAN.R-project.org/package=rextendr