BLOCKING STRATEGIES TO ACCELERATE RECORD MATCHING FOR BIG DATA INTEGRATION

I.S. Kadochnikov[1,2 a], V.V. Papoyan[1 b]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie St, Dubna, Moscow Region, 141980, Russia*

[2] *Plekhanov Russian University of Economics, Stremyanny lane, 36, Moscow, 117997, Russia*

E-mail: [a] kadivas@jinr.ru, [b] vlpapoyan@jinr.ru

Record matching represents a key step in many Big Data analysis problems, especially leveraging disparate large data sources. Methods of probabilistic record linkage provide a good framework to find and interpret partial record matches. However, they require combining and therefore computing string distances for the records being compared. That is, the direct use of probabilistic record linkage requires processing the Cartesian product of record sets. As a result, a "blocking" step is used, when candidate record pairs are grouped by a categorical field, significantly limiting the number of record comparisons and computational cost. On the other hand, this method requires a high level of data quality and agreement between sources on the categorical blocking field. We propose a more flexible approach where blocking does not use a categorical column. The key idea is to use clustering based on string field values. In practice, we mapped the string field with TF-IDF into a latent vector space and then used Locality Sensitive Hashing to cluster records in this vector space. Apache Spark libraries were used to show the effectiveness of this approach for linking British open company registration datasets.

Keywords: record linkage, locality sensitive hashing, Apache Spark

1. Record matching for Big Data

Working with Big Data often requires integration, processing and analysis of large datasets from different sources. A critical step in making use of such combined data is record matching, that is, identifying records in two or more datasets that refer to the same entity, e.g. company, person, computer, etc. When data is coming from one source with an established unique identifier, record matching is trivial. With many disparate sources, a comparison of several identifying fields is necessary for record matching. Relevant columns depend on the nature of the dataset in question. As we discuss later, probabilistic record matching methods work with inexact value comparisons and can be adapted for large datasets.

Big Data creates several challenges to effective record matching. The number of records imposes strict performance limits on matching algorithms. The use of distributed Big Data processing tools such as Apache Spark somewhat mitigates this by increasing the computing resources available for analysis, as well as managing data access and providing libraries for distributed processing.

Correlating all the records relevant to the object of interest allows extracting new knowledge from integrated data. The combined master dataset can also be used for data enhancement on newly added records.

2. Probabilistic record matching

Probabilistic record matching aims to quantify the likelihood of a match between records based on their content. The Fellegi-Sunter model provides a flexible framework to construct and parametrize a record matching algorithm[1]. It is well understood and has many modifications and generalizations, especially for matching patient records in the medical field.[2]

The original Fellegi-Sunter model considers two base probabilities when comparing two records field by field. M-probability is the probability of two fields matching if the records refer to the same entity. Obviously, m-probability is closely related to the chance of an error or an out-of-date field value. U-probability is the probability of two fields matching when the records do not refer to the same entity. This chance depends mostly on the field value domain and the distribution of possible values within this domain. M- and u-probabilities can be estimated from the data and later optimized. Assuming that field matching probabilities are independent of each other, the log-likelihood ratio contributions of each matching and mismatched field can be combined into the total match likelihood score for the record pair $S = \sum_{k-th\ field\ match} \frac{m_k}{u_k} + \sum_{k-th\ field\ mismatch} \frac{(1-m_k)}{(1-u_k)}$

Any record pair with the likelihood score above the upper threshold S_{match} is considered a match; any pair with the score below the lower threshold $S_{mismatch}$ is dropped. Record pairs with the score in between are subject to human review. A variant without human review, where just one threshold is used to separate matches and mismatches, is possible [3].

2.1 String distance

The original Fellegi-Sunter model compares a pair of fields by an exact equality comparison. For lower-quality data, where a considerable number of typos or small format differences are expected, approximate field comparison extensions of this model are used [4]. The contribution of each field to the match score is based on the distance between the field values. An example of a distance metric that can be used is the edit distance (Levenshtein distance), that is, the number of one-character additions, deletions and substitutions necessary to transform one string into another. However, the optimal choice of the distance metric may depend on the field in question and the nature of errors expected.

2.2 Simple attribute-based blocking

Both probabilistic record linkage and its distance-based extension are computationally complex when applied to large datasets. As each record pair must be considered, the number of comparison operations is proportional to m×n, where m and n are the numbers of records in the datasets being matched. Usually this problem is solved by the blocking step. That is, grouping records in blocks based on some attribute, e.g. city, zip-code, date of birth; and only applying probabilistic matching within the blocks.

This blocking approach significantly reduces the number of comparison operations, but it is only applicable if there is a suitable attribute to apply blocking. The attribute must be present in both datasets and have the same semantics and format. Any random errors in the blocking field will lead to false mismatches. Therefore, it is better to use a categorical attribute with known categories, for example "country". Even this can create problems with differences in field domains. For example, the same entity can be recorded as being in Great Britain, the United Kingdom or England in different datasets.

3. Accelerating probabilistic matching

Mapping string fields into an n-dimensional vector space makes many vector-based data analysis methods available. This can be used to accelerate pre-filtering to find candidate pairs of records for probabilistic linkage.

3.1 TF-IDF

A popular method to transform an arbitrary string into a vector is using Term Frequency and Inverse Document Frequency (TF-IDF). It is usually applied to text documents to represent the relevance of terms within the document [5].

In the record matching problem, string fields are shorter than documents and, especially in the case of name fields, can contain arbitrary text values. TF-IDF can still be used, but with terms being computed as letter tuples (bigrams or trigrams). Since the result is used not for text search, but for approximating the string distance, the use of character tuples rather than words seems reasonable.

The cosine distance is the canonical metric used in the TF-IDF vector space. It is also possible to use the L2 (Euclidean) vector distance, or the Jaccard distance. The Jaccard distance obviously discards both the term frequency and document frequency information, using only the tuple presence or absence in the string.

These distance metrics between sparse vectors can be computed faster than edit distance functions between strings. However, as every pair of records still needs to be compared, simply using the vector distance to pre-filter candidate record pairs is not fast enough to provide adequate performance in the realm of Big Data.

3.2 Locality-sensitive hashing

For a given metric space (M, d), the Locality-Sensitive Hashing (LSH) scheme is a family of hash functions $h \in H$ that each map points in the space to buckets in such a way that any sufficiently close pair of points in M gets mapped to the same bucket with high probability, while any sufficiently far pair of points in M gets mapped to different buckets with high probability, that is:

$$\begin{aligned}&\forall a, b \in M,\\&d(a,b) \le r_1 \Rightarrow P\big(h(a) = h(b)\big) \ge p_1\\&d(a,b) \ge r_2 \Rightarrow P\big(h(a) = h(b)\big) \le p_2\end{aligned}$$

LSH functions can be "amplified" by combining random subsets of functions of the family. Randomly selecting k functions from H: ($h_1 \ldots h_k$) and defining each g in the G family so that g(x)=g(y) if and only if $h_i(x)=h_i(y)$ for *all* i=1,2…k creates an AND-amplified LSH family $g \in G$. Conversely, combining functions in such a way that g(x)=g(y) if and only if $h_i(x)=h_i(y)$ for *any* i=1,2…k creates an OR-amplified family $g \in G$. OR-amplification improves recall, but lowers precision, while AND-amplification improves precision at the cost of recall.

There are LSH families based on all three metrics discussed earlier: the bucketed random projection for the Euclidean metric, the random projection sign for the cosine distance and MinHash for the Jaccard distance [6]. Any of these families can be both OR- and AND- amplified.

With the record string field values mapped in the latent TF-IDF vector space, it is possible to use LSH for clustering data, achieving non-deterministic but very fast blocking. Unlike the traditional blocking for record matching, the LSH-based strategy can use any string fields that participate in probabilistic matching, with no requirement of a reliable shared categorical field. This approach was earlier proposed and tested on synthetic record matching datasets in [7].

4. LSH in Spark

Spark is a Big Data distributed analytics engine that provides many built-in libraries that cover every necessary step to implement record matching using this strategy. This includes reading datasets from files stored in the Ceph storage cluster, schema recognition, data profiling, initial filtering, string fields splitting into character n-grams and TF-IDF. After LSH-based blocking, the record match log-likelihood calculation was also implemented in Spark.

4.1 LSH in Spark-ML

Spark contains libraries implementing many machine-learning techniques, including LSH. Namely, BucketedRandomProjectionLSH and MinHashLSH, which implement the Euclidean distance and Jaccard distance hash families respectively [8]. However, the cosine distance hash family is not implemented and no AND-amplification is possible. Only the number of functions in the family, which are all OR-combined for clustering, and the bucket size for the bucketed random projection can be controlled.

In practice the stability of the Euclidean approximate similarity join implemented in Spark-ML was found to be disappointing, especially when matching large datasets. Since many records are projected to the 0^{th} bucket by default, the comparison of each record pair in this bucket takes a disproportionately long time. Evident silent failures were particularly troublesome, when no error was reported and from the client side, the spark job seemed to continue running.

4.2 ScANNS

The Scalable Approximate Nearest Neighbor Search (ScANNS) is an LSH library for Spark authored at LinkedIn[9]. It uses the older Resilient Distributed Dataset Spark API as opposed to the newer DataFrame API. Thus, using this library on DataFrames requires extra data transformation steps. ScANNS has several advantages compared to the built-in Spark-ML implementation.

The sign random projection is implemented in ScANNS; it is a hash based on the cosine distance, which is better suited for TF-IDF vectors. Both AND- and OR- amplification are controlled by the numHashes and signatureLength parameters. The problem of some buckets containing too many records is mitigated by sampling pairs within overcrowded buckets. This is especially useful in the Euclidean case, where the 0^{th} bucket is disproportionally full and does not actually provide a strong indication of a match.

Unfortunately, the practical comparison of the libraries is not ready to be reported.

5. Testing on real datasets

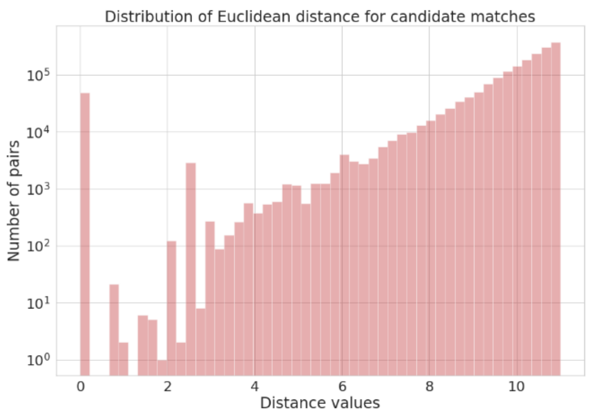

Figure 1 Distribution of L2 distances between TF-IDF vector representations of company name fields among close record pairs

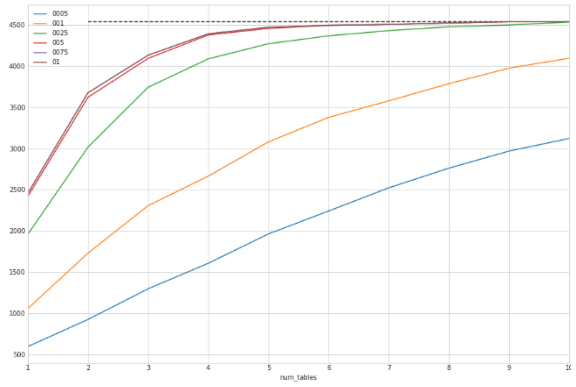

Figure 2 Measured recall of the bucketed random projection LSH depending on the bucket size and the number of OR-combined hashes

The proposed strategy was implemented on the Spark cluster situated at Plekhanov Russian University of Economics. Real open company datasets were used for testing. These were the Global Legal Entity Identifier Foundation (GLEIF) concatenated dataset [10] and the Companies House (CH) Free Company Data Product [11]. The CH dataset describes all registered British companies in a compressed csv format, and as of 2018-06-01 contained 4.2 million records. The GLEIF dataset represents all the companies worldwide with a registered Global Legal Entity Identifier in a compressed XML format, and as of 2018-06-14 contained 1.2 million records.

Company names from both datasets were mapped to the latent vector space using TF-IDF with character 3-grams as terms. Spark built-in LSH classes were used for Euclidean LSH blocking. The distance distribution among candidate pairs is shown in Figure 1. The number of hashes and the bucket size were varied, with blocking recall changing accordingly. The measured recall is shown in Figure 2.

6. Conclusion

A fast blocking strategy for probabilistic record matching based on string fields was proposed. It was implemented in Spark using two Locality-Sensitive Hashing libraries and tested on real-world open company datasets. The blocking operation performance and the false negative rate of LSH blocking, with their dependence on hashing family parameters, were measured for Spark-ML Euclidean LSH.

In the future, it would be useful to use a model dataset with known ground truth to compute the overall precision and recall of probabilistic record matching with LSH blocking, as well as compare vector distance filtering false negative rate to hash blocking false negative rate. The performance and accuracy of the cosine distance LSH implemented by ScANNS should also be investigated.

Another possible avenue for future research is using GPU computing resources to quickly directly calculate vector distances in the TF-IDF space using sparse matrix libraries.

7. Acknowledgements

The study was carried out at the expense of the Russian Science Foundation grant (project No. 19-71-30008).

References

[1] Fellegi, I.P., Sunter, A.B., 1969. A Theory for Record Linkage. Journal of the American Statistical Association 64, 1183–1210. https://doi.org/10.1080/01621459.1969.10501049

[2] Brown, A.P., Randall, S.M., Ferrante, A.M., Semmens, J.B., Boyd, J.H., 2017. Estimating parameters for probabilistic linkage of privacy-preserved datasets. BMC Med Res Methodol 17. https://doi.org/10.1186/s12874-017-0370-0

[3] Grannis, S.J., Overhage, J.M., Hui, S., McDonald, C.J., 2003. Analysis of a Probabilistic Record Linkage Technique without Human Review. AMIA Annu Symp Proc 2003, 259–263.

[4] DuVall, S.L., Kerber, R.A., Thomas, A., 2010. Extending the Fellegi-Sunter probabilistic record linkage method for approximate field comparators. J Biomed Inform 43, 24–30. https://doi.org/10.1016/j.jbi.2009.08.004

[5] Salton, G., Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Information Processing & Management 24, 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

[6] Wang, J., Shen, H.T., Song, J., Ji, J., 2014. Hashing for Similarity Search: A Survey. arXiv:1408.2927 [cs].

[7] Steorts, R.C., Ventura, S.L., Sadinle, M., Fienberg, S.E., 2014. A Comparison of Blocking Methods for Record Linkage. arXiv:1407.3191 [cs, stat].

[8] Extracting, transforming and selecting features - Spark 2.4.4 Documentation [WWW Document], n.d. URL https://spark.apache.org/docs/latest/ml-features.html#locality-sensitive-hashing (accessed 11.11.19).

[9] linkedin/scanns: A scalable nearest neighbor search library in Apache Spark [WWW Document], n.d. URL https://github.com/linkedin/scanns (accessed 11.11.19).

[10] Download the Concatenated Files – GLEIF Concatenated Files – LEI Data – GLEIF [WWW Document], n.d. URL https://www.gleif.org/en/lei-data/gleif-concatenated-file/download-the-concatenated-file (accessed 11.11.19).

[11] Companies House [WWW Document], n.d. URL http://download.companieshouse.gov.uk/en_output.html (accessed 11.11.19).