Industrial Paper: Large-scale Record Linkage of Web-based Place Entities

Vinícius M. R. Cousseau[1,2], **Luciano Barbosa**[2]

[1]In Loco, Recife – PE – Brazil

[2]Centro de Informática, Universidade Federal de Pernambuco, Recife – PE – Brazil

vinicius.cousseau@inloco.com.br, luciano@cin.ufpe.br

Abstract. *Extracting data about entities from the Web has become commonplace in the industry and academia alike. Web-based entities, however, are inherently noisy and, as such, introduce several normalization issues which must be attended to in order to maintain a clean database. Record linkage, which refers to the detection of replicated datum from possibly multiple sources, is one of the most critical of those issues. This paper presents a practical approach for solving the record linkage problem in the places data domain at an industrial scale, displaying both a model which reaches a normalized Gini coefficient of 0.92, and an architecture that supports large-scale processing.*

1. Introduction

The creation and management of databases built upon web-based data is a process which suffers from several issues pertaining to a noisy and unstructured nature. While the amount of information covered by the web grows steadily, only 52% of its websites display their data in some structured manner[1]. On top of that, one may also encounter well-structured but ill-defined *datum*, which not only is harder to treat, but also lessens its informational and commercial value.

One of the most prominent normalization issues that must be addressed in this context is record linkage, which refers to the detection and deduplication of redundant data in databases. This field of research has evolved considerably, since most of the Web is currently open to user input and there is usually a plethora of websites displaying information about the same entities but in different forms.

Our domain consists of data about places around the globe, gathered from a focused web crawler. We define a place in our database as a direct representation of a physical location in the real world which has well-defined boundaries, a purpose, a considerable size, and is a source of context.

The most relevant place fields for the work herein described are **ID** (a unique textual identifier), **Name**, **Geo Location** (expressed in latitude and longitude coordinates), **Address**, **Phone**, **Homepage**, **Labels** (which are the places' categories such as restaurant or store), and **Parentage** (which is an ID of another place which encompasses the current one, if any).

We should note that places are particularly complex entities, since they are tightly coupled with a location and several of its fields are subjective. The places' **Geo Location**

[1]https://w3techs.com/technologies/overview/structured_data/all

field, for instance, presents issues due to a common lack of precision resulting from the geocoding - address to latitude and longitude translation - strategies utilized by websites. Moreover, having redundant places in our database could result in catastrophic results for our company, due to resultant product malfunctions.

One common strategy for approaching the linkage problem, mainly in the industry, is the development of deterministic systems based on rule chaining [Berjawi 2017]. Albeit quick to implement and effective, these systems are domain-dependant and do not scale well, as the number of chained rules grows indefinitely. There are some techniques to reduce temporal constraints, such as record blocking [Christen 2012].

The work by [Dalvi et al. 2014] is the one that most closely resembles ours, since they also investigate the record linkage problem in the places domain, proposing a dynamic programming technique based on the detection of core words for each place's name in order to match pairs. Significant gains should be expected when using machine learning models for record linkage [Wilson 2011], but it is important to note that the record linkage problem has often highly imbalanced class distributions, akin to an anomaly detection task. Hence, any chosen statistical model and metrics must account for class imbalance.

On top of these issues, we also had to take into consideration the need for a resilient and scalable architecture. For reference, as of the writing of this paper, our database was being queried approximately 700 million times on a daily basis, and it held information about more than 28 million places globally. This greatly restricted available technique choices for us, and led to some interesting trade-offs.

This paper then describes our experience in dealing with duplicated places in an industrial scale. Its contribution is twofold: it unravels architecture choices and trade-offs in dealing with on-line and off-line approaches to the record linkage problem in parallel, and presents our algorithms and machine learning models applied to the task, alongside insights learned about this problem and its domain.

2. Proposed Method

In the places domain, the record linkage problem may be defined by Research Problem (`RP`) I, which is a relaxed version of `RPII`:

> **Research Problem I** *Given a pair of place entities, how do we detect if they represent the same real-world place?*
>
> **Research Problem II** *Given a set of place entities, how do we detect if they represent the same real-world place?*

In order to translate a model which works on `RPI` to solve `RPII`, one must simply represent each positive pair output as connected nodes in a graph, and negative outputs as unconnected ones. The connected components of said graph will represent the replicated data sets, thus fulfilling the requirements of `RPII`. The techniques described in this section solve `RPII` by applying this incremental procedure.

Our first approach was deterministic and utilized the concept of a place's name being usually defined by a set of core words and a set of background ones [Dalvi et al. 2014]. We name it *WordRelevanceHeuristics* or *WRH* for short. As we had to take both off-line and on-line contexts into consideration during our development process, the relative naivety of this approach allowed us to quickly deploy a working version to both fronts.

Table 1. *PairwiseRandomForest*'s features and their descriptions

Feature	Description
MostRelevantWordJaro	Jaro Winkler similarity between core words
NameSoftTfIdf	Soft TF-IDF between normalized names
AddressSoftTfIdf	Soft TF-IDF between normalized street names
LabelsJaccard	Jaccard similarity between label lists
Distance	Distance in meters between two geo locations
HomepageMatches	Exact homepage match
ParentageRelationship	Exact match between an id and a parentage
SiblingRelationship	Exact match between parentages

WRH derives the core word concept from the fact that those words are usually the most unique ones in a name, and thus, rarer. We also restrict the detection to a single core word. Translated to code, this means that each place's core word is the one with the least global term frequency value. These words, however, may simply be localized words, such as street names, neighborhoods or local folklore, which are not good candidates for core words. To account for that, our algorithm prunes localized words by applying experimentally defined thresholds to TF-IDF values calculated only inside each place's Geohash[2], instead of globally.

Afterwards, for each place pair, the core word of each of their members is compared via exact string matching. Geographic distance, parentage and label similarity rules are then taken into account to produce a final result, indicating whether or not a given pair represents duplicated places. This approach is still very sensitive to typing errors and acronyms, which are almost always detected as the core word due to their uniqueness. As is the case with deterministic systems, it also suffers from being hard to maintain without a good domain knowledge, which in turn is undesirable in an industrial setting.

During production usage, *WRH*'s recall was deemed too low, and we decided to follow a second approach which leveraged the capabilities of supervised learning. We opted for a Random Forest model, since it tends to excel at handling skewed data and finding a good bias-variance trade-off without feature scaling. We name this model *PairwiseRandomForest* or *PRF*, and are utilizing it in our current production environment.

The *PRF* model features are described at Table 1. These are all pairwise features, i.e. for each possible place pair, each feature is computed by comparing attributes from both of the places. As the reader may notice, we still make use of the place's core word, but this time only as a single feature in the model. We also include two Soft TF-IDF [Moreau et al. 2008] comparisons since the Soft TF-IDF algorithm compares words which surpass a secondary similarity threshold, thus being more resilient to typing errors, different verbal tenses and acronyms. Finally, parentage and homepage comparisons are included as one-hot encoded categorical features.

In conclusion, each place pair is represented as a row vector $v \in \mathbb{R}^{15}$, which is then processed by the model. The random forest's output and an experimentally defined classification threshold α are utilized in tandem to generate the final binary classification for each input *datum* (`RPI`), and the aforementioned connected components analysis is

[2]http://geohash.org/

applied to generate the replicated place clusters (RPII).

2.1. Pre-processing and Trade-offs

Given that our approaches attempt to solve RPI and then adapt the results to RPII, generating possible place pairs is one of the key pre-processing steps in our pipeline. With a database containing upwards of 28 million records, performing quadratic combinations over the whole database would be too computationally and economically expensive, and as such, we utilize distributed computing and opt for the implementation of blocking techniques to reduce the number of potential pairs.

A labeled diagram for our final architecture utilizing the *PRF* model, including its on-line version and translation from RPI to RPII, can be seen at Figure 1.

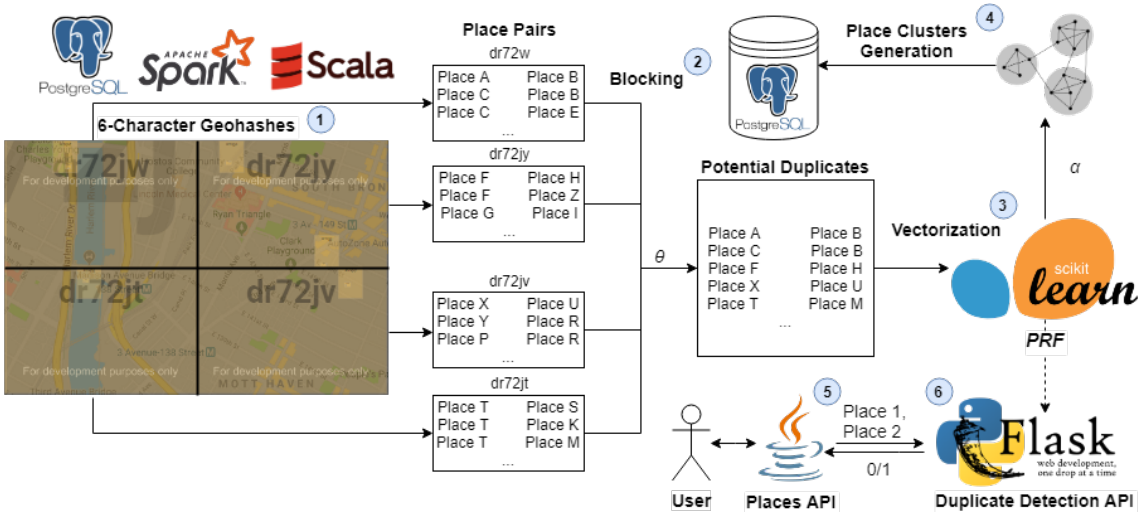

Figure 1. Diagram of our final architecture, including off-line and on-line approaches with labels. Map image made with http://geohash.gofreerange.com/.

Our blocking algorithm firstly subdivides the space into 6-character long Geohashes, each having an area of approximately $744m^2$ and a diagonal distance of approximately $1364m$, as seen in step (1). Then, for each Geohash, all possible place pairs are generated and filtered by comparing both place names with a Jaro Winkler string similarity, using a coefficient that is low enough to preserve the class distribution between duplicates and non-duplicates, and high enough to prune obvious non-duplicate pairs (2). This threshold was experimentally defined as 0.7. Place pairs are then converted into pairwise features in a separate feature engineering step, and categorized using *PRF* (3). The final off-line results are generated by the detection of connected components among the positive pairwise outputs and fed back to our database (4).

Performing quadratic operations when blocking on each Geohash would still not fulfill the needs of a real-time responsive service. Hence, in order to generate possible pairs in an on-line environment for each new place, we first query one of our APIs to find all nearby places in a $250m$ radius, limiting the query result to 150 places (5). The generated pairs are then passed through our name similarity blocking algorithm and sent to a dedicated API to respond to record linkage queries, using the trained model (6).

3. Experiments

The first step in our experiments was the definition of a silver standard over which we could iterate. With manual and crowdsourced approaches to this dataset construction having failed, we decided to utilize the same blocking technique described in the previous section to generate possible pairs for our dataset. For each of these pairs, we then filtered out all the ones without the **Phone** attribute, present in roughly 40% of the dataset entities, and utilized a direct match of the phone digits to indicate whether or not the pair was positive or negative. We then manually analyzed a few troublesome cases and fixed them.

This approach still preserved the estimated class distribution, and provided us with a dataset containing 597452 place pairs, 572560 of those being negative samples: a $24 : 1$ negative to positive ratio. It is important to note that we decided to run both of our experiments for Brazilian places only, since different languages could imply different feature importance values.

The dataset was then divided into training and test datasets using a stratified split of 70% and 30%, respectively. Afterwards, we undersampled the training set with Tomek links and then oversampled the positive samples with SMOTE. The *PRF* model was trained on top of this training dataset, while *WRH* did not require any kind of training. The hyperparameters for *PRF* all follow the default values from `scikit-learn v0.20.3`, apart from the maximum tree depth, which was manually optimized to 23. In these experiments, α is manually set to 0.95.

Table 2 shows the results for the execution of the two strategies on top of the test dataset. The balanced accuracy score metric takes the class imbalance issue into account, preventing the majority of negative samples from biasing the results. We added precision and recall as means of completeness, even though they are not good evaluation metrics for an imbalanced scenario.

Table 2. Results on top of our test dataset, with $\alpha = 0.95$

Metric	*WRH*	*PRF*
Balanced Accuracy Score	0.51	**0.87**
$F_{\beta=0.5}$ (Positive)	0.21	**0.42**
Precision (Positive)	0.56	0.56
Recall (Positive)	0.06	**0.22**

The $F_{\beta=0.5}$ score was chosen in spite of the $F1$ score to better reflect the increased importance of false positives in our context, when compared to false negatives. Regardless, it is still highly affected by the class imbalance, i.e. due to the higher density of negative samples, there will naturally be several more false negatives than false positives as the size of the dataset grows.

We also computed the normalized Gini coefficient for the *PRF* model. This score presents itself as one of the best ways to measure performance of classifiers in a highly skewed dataset, by measuring the degree of inequality among the results. The *PRF* model achieved a normalized Gini coefficient of 0.92, very close to the optimal value of 1.

By comparing both models, we clearly see the predicted effects of a much higher recall, a 366% gain. The absent precision gain for the second model with the chosen α

value is a result of the precision-recall trade-off that we were willing to adhere to. By adjusting α according to the model's learning curve, however, we are able to achieve a precision of 0.72, a 28% gain, with *WRH*'s recall value of 0.06.

Upon manual investigation, we discovered that rejecting address comparisons is detrimental, since there are many seemingly replicated places with the same name near each other that are actually two real separate locations from a chain store. Indeed, a feature importance analysis by permutation shows that that the **MostRelevantWordJaro**, **NameSoftTfIdf**, and **AddressSoftTfIdf** features are the most crucial ones, respectively.

Furthermore, to assess the scalability of our model, we recorded execution times of our distributed algorithms, which run on a recurring basis on top of all our database containing approximately 252Gb of data stored in the parquet format. We utilized 7 AWS m4.xlarge instances, each having 16Gb of memory and 4vCPUs. The pipeline achieved an average run time of 93 minutes to process the whole database, with the generation of over 2 million potential pairs to be processed by our model. The classification and clusterization steps were extremely fast, and always took less than a minute to process all generated pairs in each experiment.

4. Conclusions

In this paper, we presented the evolution of an approach to detect replicated records in an increasingly growing database of web-based place entities, with upwards of 28 million records. These records are inherently noisy, and we had to deal with several issues pertaining to this nature during the development of our methods.

As a result of our work, we were able to both fulfill the current company needs and overcome the technical challenges in the area, making our architecture fully operational in a production environment. We expect that this work should be of use to researchers and engineers in the academia and industry, whether they are dealing directly with places data or not. As future steps, we intend to dig deeper into more recent techniques, such as the development of spatial embeddings for usage in deep neural network architectures.

References

Berjawi, B. (2017). *Integration of Heterogeneous Data from Multiple Location-Based Services Providers: a Use Case on Tourist Points of Interest*. PhD thesis.

Christen, P. (2012). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555.

Dalvi, N., Olteanu, M., Raghavan, M., and Bohannon, P. (2014). Deduplicating a places database. In *Proceedings of the 23rd international conference on World wide web - WWW 14*. ACM Press.

Moreau, E., Yvon, F., and Cappé, O. (2008). Robust similarity measures for named entities matching. In *Proceedings of the 22nd International Conference on Computational Linguistics - COLING 08*. Association for Computational Linguistics.

Wilson, D. R. (2011). Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage. In *The 2011 International Joint Conference on Neural Networks*. IEEE.