



Pós-Graduação em Ciência da Computação

**Vinícius de Moraes Rêgo Cousseau**

**A linkage pipeline for place records using multi-view encoders**



Universidade Federal de Pernambuco  
posgraduacao@cin.ufpe.br  
<http://cin.ufpe.br/~posgraduacao>

Recife  
2020

**Vinícius de Moraes Rêgo Cousseau**

**A linkage pipeline for place records using multi-view encoders**

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

**Área de Concentração:** Banco de Dados  
**Supervisor:** Luciano de Andrade Barbosa

Recife  
2020

**FICHA**

**Vinícius de Moraes Rêgo Cousseau**

**“A linkage pipeline for place records using multi-view encoders”**

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 14/08/2020.

---

**Supervisor: Luciano de Andrade Barbosa**

**BANCA EXAMINADORA**

---

Profa. Dra. Valéria Cesário Times  
Centro de Informática / UFPE

---

Prof. Dr. Cláudio de Souza Baptista  
Departamento de Sistemas e Computação / UFCG

---

Prof. Dr. Luciano de Andrade Barbosa  
Centro de Informática / UFPE  
**(Orientador)**

I dedicate this work to my parents.

## **ACKNOWLEDGEMENTS**

First and foremost, I thank my mother, Grasiela, who throughout the years has given me a constant reminder of persevering through tough times, and has been putting up with a grumpy, overworked son for quite some time now.

I also thank Fernanda, for supporting me and providing a safe harbor throughout this and all other endeavors. She helps me become a better version of myself each passing day.

To Guilherme, who introduced me to proper machine learning and was my first teacher on the subject.

To the team at Inloco, especially Pedro and André, who supported me and provided the necessary means for this research to be conducted.

To my advisor, Luciano, who provided valuable insights and went above and beyond to help me during this research.

And to many others who have been a part of this journey.

## RESUMO

A extração de informações sobre entidades da Web é uma prática comum tanto na academia quanto na indústria. Em particular, dados sobre pontos de interesse destacam-se como uma fonte rica de informação geolocalizada e contexto espacial, servindo como base para uma variedade de aplicações. Estas entidades, porém, são inerentemente ruidosas e introduzem diversos problemas de normalização, que precisam ser resolvidos para que possa se obter uma base de dados limpa. Resolução de entidades, que se refere à detecção de dados replicados vindos potencialmente de diversas fontes, é um dos processos de limpeza mais críticos a serem realizados. Este trabalho apresenta uma solução original de resolução de entidades para dados de pontos de interesse oriundos predominantemente da Web em grande escala, sendo composto por três etapas: geração de potenciais pares de pontos de interesse duplicados, classificação de pares em duplicações ou não-duplicações, e geração de *clusters* a partir dos resultados da classificação de pares duplicados. A detecção de pontos de interesse duplicados destaca-se como o cerne da solução, sendo um problema complexo e pouco abordado. Como principal contribuição do trabalho, portanto, é apresentado um modelo baseado em uma arquitetura de redes neurais profundas, que utiliza *encoders* para os diferentes níveis de informação de nomes, endereços, coordenadas geográficas, e categorias. Cada *encoder* utiliza estruturas distintas para gerar vetores de representação, que são concatenados, comparados, e transportados para um espaço de *features* que representa duplicações e não-duplicações. Adicionalmente, são propostas alternativas de modelos de classificação para uso em tempo real por meio de APIs. A solução completa é analisada, sendo o modelo para a classificação de pares de pontos de interesse avaliado em dois conjuntos de dados distintos e comparado com o estado da arte na área. Como resultado, a solução proposta mostra-se capaz de lidar com grandes quantidades de dados em um ambiente de produção, e o modelo de classificação obtém performance superior à dos modelos comparados em ambos os conjuntos de dados, constituindo uma solução completa e eficaz para o problema.

**Palavras-chaves:** Pontos de Interesse. Resolução de Entidades. Aprendizado Profundo. Representation Learning.

## ABSTRACT

Extracting information about Web entities has become commonplace in the academy and industry alike. In particular, data about places distinguish themselves as rich sources of geolocalized information and spatial context, serving as a foundation for a series of applications. These entities, however, are inherently noisy and introduce several normalization problems, which need to be tackled in order to obtain a clean database. Record linkage, also known as entity resolution, refers to the detection of replicated data from potentially multiple sources, and is one of the most critical cleaning processes to be conducted in a database. This work presents a novel record linkage solution for large scale Web-based places data, being composed of three steps: generation of potential duplicate place pairs, place pair deduplication, and clusterization of the classification results. The detection of duplicate places is the solution's core, being a complex and seldom approached problem in this domain. Hence, the main contribution of this work is in the form of a model based on a deep neural network architecture, which utilizes encoders for different information levels of names, addresses, geographical coordinates, and categories. Each encoder uses distinct structures to generate representation vectors, which are concatenated, compared, and transported to a feature space that represents duplications and non-duplications. Additionally, this work proposes alternative classification models for real time usage by means of APIs. The complete solution is analyzed, with the classification model for place pairs being evaluated on top of two distinct data sets and compared against the state-of-the-art. As a result, the proposed solution is shown to handle large quantities of data in a production environment, and the classification model outperforms the baselines in both data sets, thus constituting a complete and efficient solution for the record linkage problem in the places data domain.

**Key-words:** Places. Record Linkage. Deep Learning. Representation Learning.



## LIST OF FIGURES

Figure 1 – Geohash with a precision of 6 characters contained inside a Geohash with 5 characters. . . . .	23
Figure 2 – Projection system utilized by H3. . . . .	24
Figure 3 – Sample overlay of H3 cells. . . . .	24
Figure 4 – Representations of words as vectors. . . . .	26
Figure 5 – Architecture of CBOW and Skip-gram Word2vec models. . . . .	27
Figure 6 – Overview of a traditional RNN processing input sequence $x$ . . . . .	28
Figure 7 – Internal structure of a LSTM network. . . . .	29
Figure 8 – Edge detection made by convolving a Sobel kernel with the original image. . . . .	31
Figure 9 – CNN architecture for extracting features from sentences. . . . .	31
Figure 10 – Common ways to input geo location fields of places in applications. . .	36
Figure 11 – Overview of our pipeline. . . . .	43
Figure 12 – Structure of the one-hot vectors utilized to represent the <code>parent_place_match</code> , <code>siblings_match</code> , and <code>homepage_matches</code> features in our ensemble classifiers. . . . .	49
Figure 13 – The APIs involved in our pipeline to solve record linkage in real time. .	52
Figure 14 – Diagram of our deep neural network architecture for classifying place pairs. . . . .	54
Figure 15 – Word encoder processing a pair of places $P_a$ and $P_b$ , producing intermediate embedding sequences $A^{P_i}$ and $B^{P_i}$ where $i \in \{a, b\}$ , and output embeddings $e_w^{P_a}$ and $e_w^{P_b}$ . . . . .	55
Figure 16 – Our character encoder consuming a pair of places $P_a$ and $P_b$ , and producing embeddings $e_{ch}^{P_a}$ . . . . .	56
Figure 17 – Our category level module, which produces embeddings $e_{ct}^{P_a}$ and $e_{ct}^{P_b}$ . .	58
Figure 18 – Place category distribution for the top five categories in $Places_{US}$ , $Places_{BR}$ , and $Places_{DB}$ . . . . .	66
Figure 19 – Heat maps of places in $Places_{US}$ and $Places_{BR}$ . . . . .	67
Figure 20 – Overlapping precision-recall curves in $Pairs_{BR}$ and $Pairs_{US}$ for all possible models. . . . .	79
Figure 21 – Summary plots of SHAP values for $PRF$ and $PLGBM$ in $Pairs_{BR}$ . . .	82

## LIST OF TABLES

Table 1 – Examples of entities and whether or not they fit the definition of a place in our work. . . . .	34
Table 2 – Description of fields for place entities, using the schema from our database. . . . .	35
Table 3 – Related works, their approaches, features, and limitations. . . . .	38
Table 4 – Pairwise features and their descriptions for our ensemble classifiers. . . . .	48
Table 5 – Contract for the spatial search endpoint of our Places API. . . . .	51
Table 6 – Simplified contract of our Places Playground API. . . . .	52
Table 7 – Places by country for the top 10 countries in <i>Places<sub>DB</sub></i> . . . . .	65
Table 8 – Place fields coverage in <i>Places<sub>US</sub></i> , <i>Places<sub>BR</sub></i> , and <i>Places<sub>DB</sub></i> . . . . .	65
Table 9 – Percentage of duplicate pairs among the first 100 randomly-select pairs of each similarity bucket for our blocking algorithm. . . . .	68
Table 10 – Characteristics of our <i>Pairs<sub>US</sub></i> and <i>Pairs<sub>BR</sub></i> silver truth sets. . . . .	69
Table 11 – Characteristics and issues of external place pair data sets. . . . .	70
Table 12 – Characteristics and issues of external raw place data sets. . . . .	71
Table 13 – Comparison of all models on test samples from <i>Pairs<sub>BR</sub></i> and <i>Pairs<sub>US</sub></i> . . . . .	78
Table 14 – Results of the ablation study for the Classification Model. . . . .	80
Table 15 – Comparison of <i>PRF</i> and <i>PLGBM</i> in regards to resampling. . . . .	81
Table 16 – Training and execution times for all models. . . . .	83
Table 17 – Execution times for the record linkage pipeline, averaged over 5 runs. . . . .	84
Table 18 – Hyperparameters of our baseline methods and the best values obtained in each data set. . . . .	97
Table 19 – Hyperparameters of our preliminary methods and the best values obtained in each data set. . . . .	98
Table 20 – Hyperparameters of our Classification Model and its ablated versions, with the best values obtained in each data set. . . . .	99

## LIST OF ABBREVIATIONS AND ACRONYMS

<b>API</b>	Application Programming Interface
<b>BOW</b>	Bags-of-words
<b>CBOW</b>	Continuous Bags-of-words
<b>CNN</b>	Convolutional Neural Network
<b>DF</b>	Document Frequency
<b>EM</b>	Expectation-maximization
<b>GRU</b>	Gated Recurrent Unit
<b>IDF</b>	Inverse Document Frequency
<b>LBS</b>	Location Based Services
<b>LGBM</b>	LightGBM
<b>LSTM</b>	Long Short-Term Memory
<b>MLP</b>	Multi-layer Perceptron
<b>NLP</b>	Natural Language Processing
<b>PCA</b>	Principal Component Analysis
<b>PLGBM</b>	Pairwise LGBM
<b>POI</b>	Points of Interest
<b>PRF</b>	Pairwise Random Forest
<b>REST</b>	Representational state transfer
<b>RNN</b>	Recurrent Neural Network
<b>SHAP</b>	Shapley Additive Explanations
<b>TF</b>	Term Frequency
<b>WRH</b>	Word Relevance Heuristics

## SUMMARY

<b>1</b>	<b>INTRODUCTION . . . . .</b>	<b>14</b>
1.1	OBJECTIVES . . . . .	16
1.2	WORK STRUCTURE . . . . .	16
1.3	LIST OF PUBLICATIONS . . . . .	17
<b>2</b>	<b>FOUNDATIONS . . . . .</b>	<b>18</b>
2.1	RECORD LINKAGE . . . . .	18
2.2	STRING MATCHING TECHNIQUES . . . . .	19
2.2.1	<b>Sequential Character Methods . . . . .</b>	<b>20</b>
2.2.2	<b>Bag-of-words Methods . . . . .</b>	<b>21</b>
2.2.3	<b>Mixed Methods . . . . .</b>	<b>22</b>
2.3	SPATIAL BLOCKING . . . . .	22
2.3.1	<b>Geohash Geocoding System . . . . .</b>	<b>23</b>
2.3.2	<b>H3 Indexing System . . . . .</b>	<b>23</b>
2.3.3	<b>Issues . . . . .</b>	<b>24</b>
2.4	REPRESENTATION LEARNING . . . . .	25
2.4.1	<b>Word Embeddings . . . . .</b>	<b>25</b>
2.4.2	<b>Word2Vec . . . . .</b>	<b>26</b>
2.4.3	<b>FastText . . . . .</b>	<b>27</b>
2.5	SEQUENCE-BASED NEURAL NETWORKS . . . . .	28
2.5.1	<b>Recurrent Neural Networks . . . . .</b>	<b>28</b>
2.5.2	<b>Convolutional Neural Networks . . . . .</b>	<b>30</b>
<b>3</b>	<b>BACKGROUND: PLACES . . . . .</b>	<b>33</b>
3.1	PLACE ENTITY DEFINITION . . . . .	33
3.2	CHALLENGES OF DEALING WITH PLACE FIELDS . . . . .	35
<b>4</b>	<b>RELATED WORK . . . . .</b>	<b>38</b>
4.1	TRADITIONAL APPROACHES . . . . .	39
4.2	DEEP LEARNING APPROACHES . . . . .	41
<b>5</b>	<b>PLACE RECORDS' LINKAGE PIPELINE . . . . .</b>	<b>43</b>
5.1	RECORD BLOCKING . . . . .	44
5.2	PRELIMINARY CLASSIFICATION MODELS . . . . .	45
5.2.1	<b>Heuristic Approach . . . . .</b>	<b>45</b>
5.2.2	<b>Supervised Learning Approach . . . . .</b>	<b>48</b>
5.3	DUPLICATE CLUSTERING . . . . .	49

5.4	APPLICATION PROGRAMMING INTERFACES . . . . .	50
<b>6</b>	<b>DEEP NEURAL NETWORK FOR CLASSIFYING PLACE PAIRS .</b>	<b>53</b>
6.1	WORD ENCODER . . . . .	54
6.2	CHARACTER ENCODER . . . . .	56
6.3	CATEGORY ENCODER . . . . .	57
6.4	GEOGRAPHICAL ENCODER . . . . .	59
6.5	AFFINITY GENERATION . . . . .	60
6.6	MODEL TRAINING . . . . .	61
6.7	FAILED ATTEMPTS IN OUR DEEP NEURAL NETWORK . . . . .	61
<b>7</b>	<b>EXPERIMENTAL DATA PREPARATION . . . . .</b>	<b>64</b>
7.1	DATA ANALYSIS . . . . .	64
7.2	NOVEL GROUND TRUTH SETS . . . . .	68
7.3	INADEQUACIES IN EXTERNAL GROUND TRUTHS . . . . .	69
<b>8</b>	<b>EXPERIMENTS . . . . .</b>	<b>73</b>
8.1	RUNTIME ENVIRONMENT . . . . .	73
8.2	PERFORMANCE METRICS . . . . .	73
<b>8.2.1</b>	<b>Gini Coefficient Interpretation . . . . .</b>	<b>74</b>
<b>8.2.2</b>	<b>F-score Interpretation . . . . .</b>	<b>74</b>
8.3	SETUP FOR BASELINE METHODS . . . . .	75
8.4	SETUP FOR PRELIMINARY PROPOSED METHODS . . . . .	76
8.5	SETUP FOR CLASSIFICATION MODEL . . . . .	76
8.6	RESULTS . . . . .	77
<b>8.6.1</b>	<b>Ablation Study . . . . .</b>	<b>79</b>
<b>8.6.2</b>	<b>On The Effects of Resampling . . . . .</b>	<b>80</b>
<b>8.6.3</b>	<b>Shapley Additive Explanations . . . . .</b>	<b>81</b>
<b>8.6.4</b>	<b>Training and Execution Times . . . . .</b>	<b>82</b>
8.7	DISCUSSION . . . . .	84
<b>9</b>	<b>CONCLUSIONS AND FUTURE WORK . . . . .</b>	<b>85</b>
9.1	CONCLUSION . . . . .	85
9.2	MAIN CONTRIBUTIONS AND LIMITATIONS . . . . .	85
9.3	FUTURE WORK . . . . .	86
	<b>REFERENCES . . . . .</b>	<b>87</b>
	<b>APPENDIX A – DUPLICATE DETECTION API BODY . . . . .</b>	<b>95</b>
	<b>APPENDIX B – IMPLEMENTATION OF THE GINI COEFFICIENT</b>	<b>96</b>

<b>APPENDIX C – HYPERPARAMETERS . . . . .</b>	<b>97</b>
---	-----------

## 1 INTRODUCTION

The creation and management of databases built with Web-based data has become commonplace in academia and industry alike. However, it is a process which suffers from several issues pertaining a noisy and unstructured nature. While the amount of information covered by the Web grows steadily, only 59.4% of its websites display their data in some structured manner<sup>1</sup>. On top of that, one may also encounter well-structure but ill-defined data, which not only is harder to treat, but also lessens the informational value of it.

Our domain consists of data about places around the world, gathered mostly from the Web by a focused web crawler. Each entity represents a real physical space with a given name, location, and several other attributes which are further presented in this work. Places are a rich source of location-sensitive information and geospatial context, however, they are also complex entities, since their locations display a lot of noise and several of its fields are subjective.

One of the most prominent normalization issues that must be addressed in this context is replicated data. Record linkage, also commonly known as entity resolution or deduplication, is a field which refers to the detection and deduplication of redundant data. The field of record linkage has evolved considerably since its inception, as the problem not only persists but is also enhanced, since the Web is mostly open to user input and there are usually a plethora of websites which display the same kind of information.

Despite the advances on information retrieval pipelines, including streamlined focused web-crawlers such as the one developed by Qiu et al. (2015), the task of linking records is not trivially solved without a full view of the database in question, and thus most are agnostic to the problem. Limiting the retrieval scope could also be considered a valid strategy to solve the problem by avoiding fetching duplicates at all, but replicated data are present even inside the same web domain at a large scale, as Dalvi et al. (2014) show.

A popular approach for record linkage and similar matching tasks in the recent years is utilizing machine learning and deep learning techniques to perform comparisons between records or representations of them (HU et al., 2014; GUO et al., 2016; XIONG; ZHONG; SOCHER, 2016; SANTOS; MURRIETA-FLORES; MARTINS, 2017; SANTOS et al., 2017; MARINHO, 2018; BARBOSA, 2018). More specifically, for the places data domain, Yang et al. (2019) explore learning place representations and creating a metric space in which duplicated places are near one another.

On the other hand, none of the previous works dealing with record linkage in the places Web data domain, to the best of our knowledge, provide an end-to-end solution for the problem in a production scenario. Thus, the following problem arises: given a Web-based

---

<sup>1</sup> <[https://w3techs.com/technologies/overview/structured\\_data](https://w3techs.com/technologies/overview/structured_data)>

database of places, how can one detect sets of place entities which represent the same real-world place?

This work proposes a novel pipeline for linking place records in a database built predominantly from Web data in a large scale. It does so in three steps:

- Record Blocking: first, it generates pairs of potential duplicate place entities from the raw database;
- Classification Model: second, it classifies these place pairs as duplicates or non-duplicates;
- Duplicate Clustering: third, it clusterizes the duplicate classification results.

In addition to that, we also show that one of our preliminary attempts at implementing a Classification Model can be ported to a set of APIs and be utilized to solve the record linkage problem in real time. In this way, we tackle the record linkage problem in both on-line and off-line fashions.

While the Record Blocking, and Duplicate Clustering steps of our pipeline bond together with the Classification Model to form a cohesive end-to-end solution for the problem, our research in the topic of classifying place pairs into duplicates or non-duplicates culminates in a model which outperforms both previous approaches on the same task and strong machine learning baselines. We then elect it as the main contribution of this work.

This model, named *CM*, is a deep neural network that utilizes four encoders to capture distinct information about place fields on different levels, transforming them into compact vector representations. More specifically, the network uses word and character encoders for place names and addresses, a category encoder for a place’s category list, and a geographical encoder, which operates on top of geographical distance metrics between places. The representation vectors for each place are then concatenated, compared, and transported to a vector space which represents duplications and non-duplications.

Differently from Yang et al. (2019), which attempt to build generic representations for places in an unsupervised manner with a costly smoothing process, and use then for the deduplication task, our network builds representations solely focused on place pairs. These representations are built during the duplicate detection process itself. While this hinders us from utilizing these representations for many other purposes, experimentation on two different data sets shows that:

- The proposed pipeline is able to operate on top of 28 million place records seamlessly;
- *CM* obtains performances surpassing those of competitive ensemble classifiers trained on the duplicate detection task in all testes scenarios;
- *CM* also obtains better results than previous heuristics and deep learning approaches on the same task;



- The preliminary proposed models are suited for on-line usage, having low execution times.

## 1.1 OBJECTIVES

The main objective of this work is providing an end-to-end solution for the record linkage problem in the places data domain, with a focus on detecting duplications among place pairs. In order to do that, the following tasks are required:

- Understand the database in question, and further our knowledge on place entities and their issues;
- Create a blocking strategy which prunes trivial non-duplications but keeps duplicates;
- Develop a deep neural network to capture multi-level information about place pairs and use it to detect duplications;
- Create a way to clusterize results from the classification step;
- Generate a ground truth set for the duplicate detection task;
- Evaluate the pipeline and, mainly, the deep neural network for duplicate detection.

## 1.2 WORK STRUCTURE

This work is comprised of 9 chapters, including this introductory one. Chapter 2 presents the fundamental concepts necessary for the understanding of this work, touching upon basic record linkage works, string similarity metrics and heuristics, as well as common spatial partitioning approaches. It also presents explanations on recurrent and convolutional neural networks, representation learning and embedding vectors.

Then, Chapter 4 discusses previous works related to ours. The main related works are those that deal with the record linkage problem directly in the places data domain, however, we also highlight some relevant works outside of this specific data domain.

Following that, Chapter 3 provides the definition of a place entity in the scope of this work, comparing it against previous ones. Furthermore, it described the fields present in each place in our database, and dissect the issues present in each of them.

Next, Chapter 5 dissects the proposed pipeline for linking place records, without going into details about our deep neural network. It first explains our record blocking strategy in Section 5.1, while Section 5.2 displays preliminary attempts at classifying place pairs with heuristics and traditional machine learning models. Finally, Section 5.3 explains the clustering technique utilized on top of the classification results, and Section 5.4 talks

about how we are able to solve the record linkage problem in real time by porting models to Application Programming Interface (API)s, providing technical details on it.

Our deep neural network for classifying place pairs is presented in detail in Chapter 6, with each of its encoders being presented in Sections 6.1 through 6.4, and the final affinity generation step, which compares the results from each encoder, being dissected in Section 6.5. Section 6.7 briefly discusses failed attempts we had during the development of this network, in order to aid other researchers facing similar issues.

In Section 7.1, the database utilized for the development of the proposed solutions and in the experiments is explained in detail, while Section 7.2 offers an explanation on the ground truth generation process. External ground truths which have been evaluated by us but deemed infeasible for our use case are described in Section 7.3.

Chapter 8 presents the experiments we conducted, their performance metrics and setup, and the obtained results, discussing them afterwards. Finally, Chapter 9 concludes the work by summarizing our results and describing steps for future work.

### 1.3 LIST OF PUBLICATIONS

During the development of this work, the following publication was made:

- COUSSEAU, V.; BARBOSA, L. Industrial paper: Large-scale record linkage of web-based place entities. In: *Anais Principais do XXXIV Simpósio Brasileiro de Banc ode Dados*. Porto Alegre, RS, Brasil: SBC, 2019. p. 181–186. ISSN 0000-0000. Available at: <<https://sol.sbc.org.br/index.php/sbbd/article/view/8820>>.

## 2 FOUNDATIONS

This Chapter describes the fundamental concepts to grasp our work. First, it describes the problem of record linkage, its seminal works, and how the problem is commonly structured. Next, common string matching and spatial blocking techniques utilized in record linkage works are detailed. Lastly, the Chapter offers overviews on representation learning and sequence-based neural networks, highlighting how they fit the record linkage problem. These last explanations assume a basic understanding of machine learning and deep learning.

### 2.1 RECORD LINKAGE

The field of record linkage, also known as entity resolution or duplicate record detection, has been studied in several contexts. One of the first recorded explorations of the problem was performed by Dunn (1946), who presented it in the public health record-keeping sector and proposed a simplistic deterministic approach to tackle it. In their work, Fellegi and Sunter (1969) develop a probabilistic foundation for the problem, where an agreement weight  $w_m$  is computed for each entity field by:

$$w_m = \ln \frac{m}{u} \quad (2.1)$$

and a disagreement weight  $w_u$  is computed by:

$$w_u = \ln \frac{1 - m}{1 - u} \quad (2.2)$$

where  $m$  is the probability that the given field is an exact match in a duplicate pair, and  $u$  is the probability that the given field is not an exact match on a non-duplicate pair. These probabilities may be calculated from a ground truth set. Then, to compute a score for a pair of records, each of their fields are compared. If they match, its agreement weight is added, and, if not, the disagreement weight is. This sum of weights can be compared against a decision threshold  $\theta$  to produce a final classification result.

This framework serves as a baseline, and has been studied and improved over the years. Notably, Wilson (2011) provides mathematical proof on the equivalence between the probabilistic model from Fellegi and Sunter (1969) and a naive Bayes classifier (LANGLEY; IBA; THOMPSON, 1998). Since naive Bayes classifiers are among the simplest models due to their hard independence assumptions, this proof implies that using more advanced machine learning models could provide significant improvements over the probabilistic record linkage framework.

Modern record linkage techniques commonly approach the problem in two main steps (STEFANIDIS et al., 2014): record blocking and entity matching or deduplication. Moreo-

ver, a third step, usually named as merging or grouping, is applied on top of the entity matching results (BERJAWI, 2017). The blocking step (CHRISTEN, 2011) is responsible for reducing the search space explored by the matching algorithm, since comparing all possible record pairs in a data set of size  $n$  would imply in a complexity of  $O(n^2)$ . When dealing with spatial records containing latitude and longitude information, these techniques usually employ grid subdivision strategies (CHRISTEN, 2011; DALVI et al., 2014; BERJAWI, 2017; COUSSEAU; BARBOSA, 2019), which are discussed in Section 2.3.

Next, the matching step is responsible for deciding whether or not two records represent the same real world entity. Traditional approaches for the matching task usually rely either on computing similarity measures between fields and applying a threshold to produce binary results (COHEN; RAVIKUMAR; FIENBERG, 2003; MOREAU; YVON; CAPPÉ, 2008), or combining several comparison metrics to do so. This metric combination strategy can be done through deterministic rules (BERJAWI, 2017; DENG et al., 2019) or through learning how to combine them (SANTOS; MURRIETA-FLORES; MARTINS, 2017; COUSSEAU; BARBOSA, 2019). In their work, Köpcke, Thor and Rahm (2010) label these two kinds of techniques as non-learned and learned.

Despite being quick to implement and relatively effective, non-learned techniques require previous specialized knowledge on the data domain in question. Moreover, they do not scale well in a real scenario, since the number of rules may grow indefinitely over time, leading to bug-prone code. With a high number of similarities being computed and rules being chained, it also becomes hard to visualize the exact rule chain a record pair goes through to be matched, which takes away from the maintainability of the system.

Adding to that, learned methods are able to capture more intricacies in the input data. For instance, as Santos, Murrieta-Flores and Martins (2017) show, even a basic form of combining string similarity metrics through state-of-the-art machine learning models is able to achieve good results. Thus, when data sets are complex enough, learned models are often favored.

Be it for the blocking or matching step, most record linkage solutions use some kind of string similarity metric in their pipeline. In light of that, we provide a description of common string matching techniques in the context of the problem.

## 2.2 STRING MATCHING TECHNIQUES

String matching methods receive two strings as input and return either a similarity value, a distance value, or a binary value for those, indicating whether or not and how closely they match one another. The works from Cohen, Ravikumar and Fienberg (2003), Moreau, Yvon and Cappé (2008) display and compare several string matching metrics and methods in the context of record linkage. More specifically, Moreau, Yvon and Cappé (2008) discern them into three types: sequential character methods, Bags-of-words (BOW) methods, and special measures which mix both others.

### 2.2.1 Sequential Character Methods

Among character based methods, the most prolific ones in record linkage works are the Levenshtein edit distance (LEVENSHTEIN, 1966) and Jaro-Winkler similarity (WINKLER, 1990). The former is a metric which indicates how many character insertions, deletions, or substitutions are needed to transform one string into another, with lower distances being better and a distance of 0 meaning exact matches.

Using an example from Moreau, Yvon and Cappé (2008), for the two strings *kitten* ( $x$ ) and *sitting* ( $y$ ), two substitution and one addition operations must be performed, totalling in a Levenshtein distance of 3:  $d(kitten, sitting) = 3$  ( $k \mapsto s, e \mapsto i, \varepsilon \mapsto g$ ). Then, the normalized Levenshtein similarity metric is calculated by  $1 - d/\max(|x|, |y|)$ . In the case of the sample strings, this similarity would be equal roughly to 0.57. Albeit intuitive to understand and implement, this metric is inefficient for long strings of characters.

The other previously mentioned common character based approach is the Jaro-Winkler similarity metric, which is a variation of the original Jaro metric (JARO, 1989). It adds a prefix scale that weighs strings whose initial characters match more heavily. The original Jaro metric is computed for two strings  $x$  and  $y$  by:

$$\text{Jaro}(x,y) = \frac{1}{3} \cdot \left( \frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{m} \right) \quad (2.3)$$

where  $m$  is the number of matching characters between  $x$  and  $y$ , such that they are the same and are at most  $D$  characters apart, with:

$$D = \left\lfloor \frac{\max(|x|, |y|)}{2} \right\rfloor \quad (2.4)$$

Then, given a constant scaling factor  $p$  and the number of characters  $l$  that match in the start of both strings, up to a maximum of 4, the Jaro-Winkler metric is computed by:

$$\text{Jaro-Winkler}(x, y) = \text{Jaro}(x, y) + l \cdot p \cdot (1 - \text{Jaro}(x, y)) \quad (2.5)$$

In the context of place entities matching, this prefix boosting makes sense for the English language, where most relevant words are usually found at the start of names, but other languages may not present this pattern. The Portuguese language is an example of that: the string *Café do João* translates roughly to *John's Coffee* in English, where *Café*  $\equiv$  *Coffee*. Moreover, different tasks may present different placement of most relevant tokens. Thus, the prefix boosting is not always desirable.

To account for that, two variations of the Jaro-Winkler metric have been proposed by Christen (2006): sorting and permutation. In the sorted version, strings are tokenized and their tokens are sorted in lexicographical order. In the permuted version, the Jaro-Winkler similarity is calculated for all possible permutations of the strings' tokens and the maximum value is returned. Santos, Murrieta-Flores and Martins (2017) further propose

a version of the metric which reverses both strings to be compared before computing the Jaro-Winkler metric, to boost suffixes instead of prefixes.

While all variations of the metric are valid, Santos, Murrieta-Flores and Martins (2017) point out that string similarities are highly task-dependant, and there is not a generic way to pick the best among them. While one could argue that the permuted version is prone to be the best, since it attempts all possible combinations of token orderings, it is computationally expensive to perform and may lead to higher average values of similarities, which may be undesirable for some tasks.

### 2.2.2 Bag-of-words Methods

Differently from character-based methods, BOW methods represent each string to be matched as a set of tokens, and compute similarities between those sets. Among those, the most common one in the record linkage task is calculating the cosine similarity between TF-IDF vectors. TF-IDF, in turn, refers to computing the Term Frequency (TF) of each word in strings to be matched by the Equation 2.6:

$$\text{TF}(w, s) = \frac{n_{w,s}}{\sum_{w' \in s} n_{w',s}} \quad (2.6)$$

where  $n_{w,s}$  is the number of times the word  $w$  appears in string  $s$ ; computing the Inverse Document Frequency (IDF) of each word in the to-be-matched strings by Equation 2.7:

$$\text{IDF}(w) = \log \left( \frac{|S|}{|\{s \in S \mid w \in s\}|} \right) \quad (2.7)$$

or one of its many variations (BAEZA-YATES; RIBEIRO-NETO et al., 1999); and finally multiplying them to achieve a final result for each word, as Equation 2.8 shows.

$$\text{TF-IDF}(w, s) = \text{TF}(w, s) \cdot \text{IDF}(w) \quad (2.8)$$

Having computed the TF-IDF values for each matching word of two strings to be matched, one may build two sparse feature vectors from these values, transporting them to a vector space where each word composes a dimension. Then, the cosine similarity between two of those vectors  $u$  and  $v$  can be calculated by Equation 2.9:

$$\text{cosine}(u, v) = \frac{u \cdot v}{\|u\| \|v\|} \quad (2.9)$$

This approach is more flexible about token positioning in a string, however, small differences in words may lead to big differences in results, since only exactly matching words are taken into consideration. For instance, the strings *Record Linkage* and *RecordLinkage*, when tokenized by spaces, have no common tokens and receive a cosine similarity of 0 by default.

### 2.2.3 Mixed Methods

To address problems pertaining to the previous methods, Cohen, Ravikumar and Fienberg (2003) propose a method named Soft TF-IDF, which leverages the best characteristics from both sequential character and BOW methods. It does so by modifying the TF-IDF cosine similarity algorithm to account for non-exact matches in words; given two strings  $s$  and  $s'$ , instead of only computing the cosine similarity for strictly equal words, a set  $CLOSE$  of close words is first populated with all tokens  $w$  from  $s$  such that there is a token in  $s'$  with  $\text{sim}(s, s') \geq \theta$ , where  $\text{sim}$  is a secondary character based similarity and  $\theta$  is a threshold. Putting it into an equation, we have:

$$CLOSE(w, s', \theta) = \{v \in s' \mid \text{sim}(w, v) \geq \theta\} \quad (2.10)$$

Then, a weighted cosine similarity between  $s$  and  $s'$ , which is the final Soft TF-IDF value  $S$ , is computed by Equation 2.11:

$$S(s, s') = \sum_{\{w, w' \mid CLOSE(w, s', \theta) \neq \emptyset \wedge w' \in CLOSE(w, s', \theta)\}} \text{cosine}(w, w') \cdot \text{sim}(w, w') \quad (2.11)$$

Using the same example from Section 2.2.2, the strings *Record Linkage* and *RecordLinkage* would have a Soft TF-IDF greater than 0, since their Jaro-Winkler similarity is 0.99. While there are other mixed methods in the literature, such as the Monge-Elkan measure (MONGE; ELKAN, 2001), the Soft TF-IDF has been shown to outperform them (COHEN; RAVIKUMAR; FIENBERG, 2003), and is thus more commonly utilized in the record linkage task.

## 2.3 SPATIAL BLOCKING

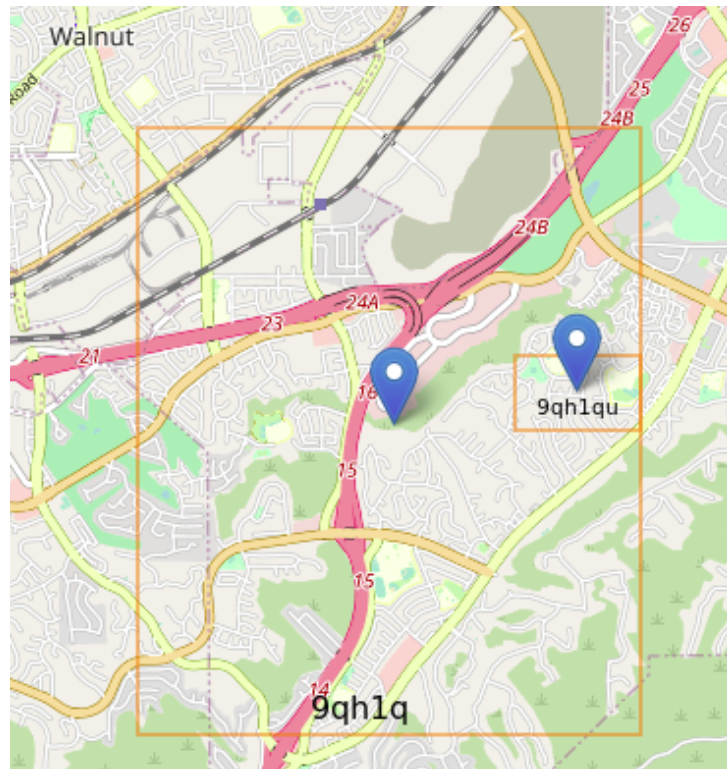
To perform record blocking for spatial data, i.e. data which contain location fields such as latitude and longitude pairs, many works utilize a grid subdivision approach (CHRISTEN, 2011; DALVI et al., 2014; BERJAWI, 2017; COUSSEAU; BARBOSA, 2019). The intuition behind this is that dividing entities into buckets and then comparing all pairs for each of those buckets avoids having to compute trivial non-duplications in the data set, thus alleviating the time complexity of the computation.

In the specific case of latitude and longitude coordinates, a simple grid subdivision approach consists of choosing a desired precision based on degrees. An approximation shows, for instance, that latitude and longitude values with digits up to the fourth decimal place can distinguish a length of 11 meters in the equator. Thus, to create grids with lengths of 11 meters in the equator, latitude and longitude values could be binned by increments of 0.0001. This simplistic approach poses two main issues: first, choosing a grid size is not straightforward, and second, grid sizes fluctuate greatly in proportion to distance from the equator, due to the non-Euclidian nature of the calculations.

### 2.3.1 Geohash Geocoding System

The Geohash geocoding system<sup>1</sup> (MORTON, 1966) improves the aforementioned simplistic approach, offering an intuitive way to define hierarchical grids. As Figure 1 shows, in this system, each grid (Geohash) is defined by a string of characters. The more characters a Geohash has, the more granular it is, and by truncating characters from the Geohash string, one is able to obtain a new, less granular grid which encompasses this one.

Figure 1 – Geohash with a precision of 6 characters contained inside a Geohash with 5 characters.



Source: (M. R. Cousseau, 2020)

The shape of each Geohash varies with its character precision: while character strings of even lengths describe square grids, Geohashes with an odd number of characters are shaped like a rectangle. This may prove troublesome for some use cases. Moreover, Geohashes do not tackle the issue of varying grid sizes according to the distance from the equator. More specifically, Geohashes are not proper geographical projections, and as such, they disregard the non-Euclidian geometry involved.

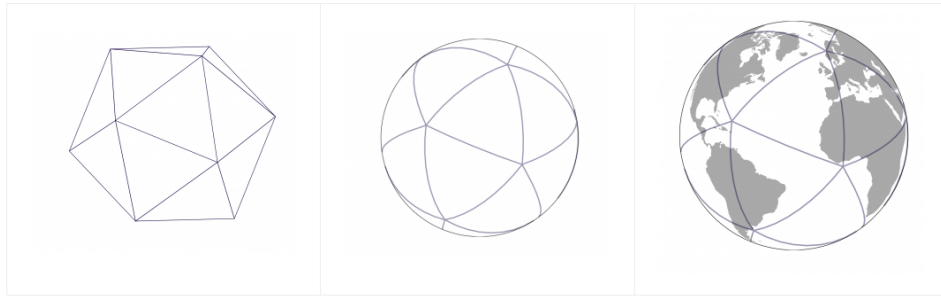
### 2.3.2 H3 Indexing System

To tackle these issues, the H3 indexing system (BRODSKY, 2018) approximates Earth's surface as a sphere and wraps it in an icosahedron, using a gnomonic projection (SNYDER,

<sup>1</sup> <<https://web.archive.org/web/20080221111539/http://blog.labix.org/author/admin/>>



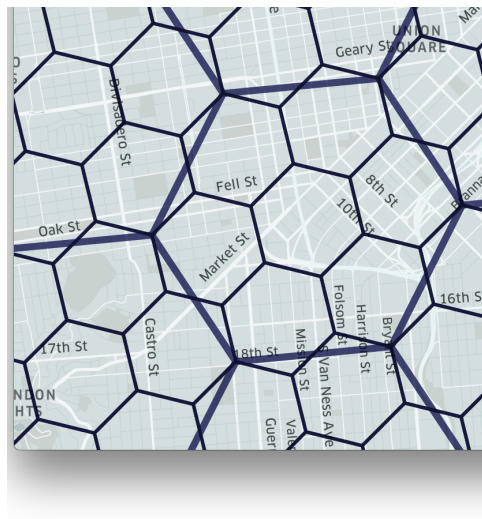
Figure 2 – Projection system utilized by H3.



**Source:** (BRODSKY, 2018)

1987) to project Earth as a spherical icosahedron, as Figure 2 shows. This projection generates 20 different panes (faces of the icosahedron), upon which a hexagonal grid of 122 cells is laid out. Using hexagons as a base shape provides an interesting geometrical property: the distance between the centroid of neighboring hexagons is always the same, which simplifies some distance calculations. Finally, Figure 3 shows H3 cells over a map.

Figure 3 – Sample overlay of H3 cells.



**Source:** (BRODSKY, 2018)

Each hexagon in the H3 system is represented by a string of characters, similarly to Geohash strings. Also akin to Geohashes, each H3 string might be truncated to obtain a less granular hexagon which encompasses the current one. Due to having a full-blown projection system included in its pipeline, the H3 system avoids distortion issues.

### 2.3.3 Issues

Each of the previously presented methods, however, do not account for different entity densities in the record linkage task. In the places data domain, for instance, the distribution of places around the world is not uniform, since first and foremost about 71% of

Earth’s surface is water-covered<sup>2</sup> and places tend to be concentrated in a few high density areas around the world. Moreover, they also suffer from edge cases, where two potentially duplicated entities to be matched may be close to one another but still be partitioned during the grid subdivision.

## 2.4 REPRESENTATION LEARNING

Choosing how to represent data or extract features is one of the key components in the performance of machine learning models (BENGIO; COURVILLE; VINCENT, 2013). Because of that, traditional attempts to employ machine learning to solve problems require a big time investment in understanding, pre-processing, and extracting relevant features from data, a process which is commonly referred to as feature engineering. This process, however, is labor intensive and requires prior domain knowledge. Its own existence, as Bengio, Courville and Vincent (2013) point out, highlights the inability of traditional algorithms to extract and organize relevant information from the data.

In that sense, efforts have been made to develop methods which learn relevant representations from data as a step to solve complex problems, such as Speech Recognition and Natural Language Processing (NLP), spawning a research area in of itself named Representation Learning. As a way to generate expressive and progressively abstract representations, deep learning has been successfully employed in Representation Learning tasks (HINTON; OSINDERO; TEH, 2006). Albeit presenting advantages, this approach is particularly difficult because it raises the question of what is a good objective for training those representations, which can lead back to the original issues of feature engineering.

The field of NLP has seen a particular boom of Representation Learning advances, specially since NLP works often deal with textual Web data, which are massive in size, noisy, and possibly unstructured. In that context, techniques often attempt to build vector representations for words.

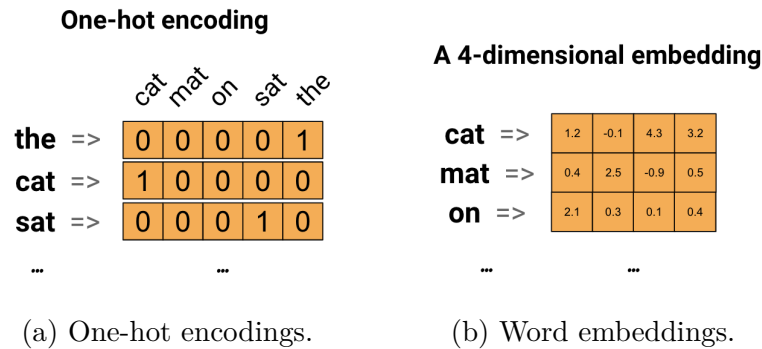
### 2.4.1 Word Embeddings

As exemplified in Figure 4b, word embeddings are a common name given to both processes which transport words to real-valued compact vector representations, and the vector representations themselves. The main concepts behind word embeddings are (1) reducing the dimensionality of the input data, in a way that machine learning models are able to consume a dense matrix of real values; and (2) creating a space where similar words have similar encodings.

A comparable approach to encode words is through one-hot encodings, shown in Figure 4a. However, it leads to extremely sparse representations, e.g. for a dictionary of 100,000 words, each word would be represented by a vector with 99.999% of empty spaces (zeroes).

<sup>2</sup> <<https://phys.org/news/2014-12-percent-earth.html>>

Figure 4 – Representations of words as vectors.



**Source:** Adapted from (TENSORFLOW, 2020)

Another possible approach would be to assign a unique integer value to each word in the dictionary. Albeit producing dense representations, the integer encodings have no explicit relationship, i.e. the index 2 is no more similar to the index 3 than the index 50,000. Moreover, some classifiers which do not account for feature scaling may fail to achieve meaningful results. Word embeddings tackle all of those issues.

#### 2.4.2 Word2Vec

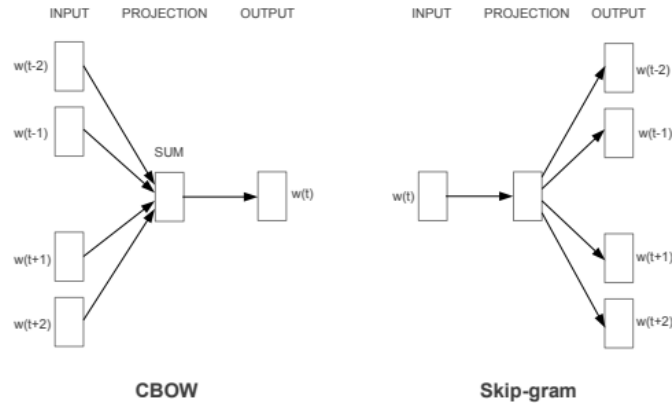
One of the most common approaches to build word embeddings are Word2Vec models (MIKOLOV et al., 2013; MIKOLOV et al., 2013). Word2Vec refers to a family of unsupervised neural network models which attempt to build contextual representations of words, transporting them to a compact space where similar words are near one another, and vector operations such as *King* - *Man* produce semantically meaningful results like *Queen*. In their core, the architecture of Word2Vec models consists of two layers, receiving randomly initialized vectors as input: one projection layer, which projects the input vectors to another space, and an output layer.

As Figure 5 shows, the shape of the input and output values vary among the two canonical Word2Vec models, which are named Continuous Bags-of-words (CBOW) and Continuous Skip-Gram (skip-gram). While the CBOW model receives a set of surrounding context words from a sentence as input and attempts to predict a current word, the skip-gram model receives the current word as input and attempts to predict its context. In this manner, the skip-gram model is often slower to train, but is able to work with less training data and represent rarer words better than the CBOW one<sup>3</sup>.

Mikolov et al. (2013) further improve training times of the skip-gram model for big data sets by proposing a negative sampling strategy. With it, instead of using the whole weight matrix with size equal to the vocabulary to calculate the output values, the network is able to utilize just a few negative contextual samples instead.

<sup>3</sup> <<https://groups.google.com/g/word2vec-toolkit/c/NLvYXU99cAM/m/E5ld8LcDxlAJ>>

Figure 5 – Architecture of CBOW and Skip-gram Word2vec models.



Source: (MIKOLOV et al., 2013)

### 2.4.3 FastText

An issue with Word2Vec models is that they do not account for the morphology of words, that is, they ignore the internal structures of the words and instead build agnostic representations for them. Bojanowski et al. (2017) indicate that this causes issues for learning representations of rare words, and learning representation of words in morphologically rich languages. Thus, the authors propose a new approach which extends the traditional skip-gram model with character representations, named FastText.

The FastText model represents each word as a bag of character n-grams, and learns vector representations for each of those n-grams instead of the word itself. Then, each word is represented by the sum of its character n-gram embedding vectors. The model also differentiates between n-grams and complete words, by including the full word into the bag of n-grams, and using special characters  $<$  and  $>$  denoting the start and end of sequences. For instance: using an n-gram size of 3, the bag of n-grams for the word *where* is composed of  $<wh, whe, her, ere, re>$ ,  $<where>$ . In this example, the authors note that the subword *her* for *where* differs from the word  $<her>$ .

Another benefit of capturing subword information is enabling the generation of embeddings for words which are not included in the initial vocabulary, by decomposing them into their character n-grams. For instance, assuming a vocabulary composed of the word *where*, FastText would be able to generate a word embedding for *there* by using the n-grams *her, ere, re*. One downside of the model is its slower training time, upwards of 1.5 times slower than the traditional skip-gram model with negative sampling as reported by its creators.

The FastText model has been thoroughly analyzed and compared against other word embedding approaches (MIKOLOV et al., 2018). It has also been trained in a *corpus* composed of CommonCrawl and Wikipedia data for 157 different languages. Both the pre-trained

vectors and the model itself are public to download, use, and extend<sup>4</sup>.

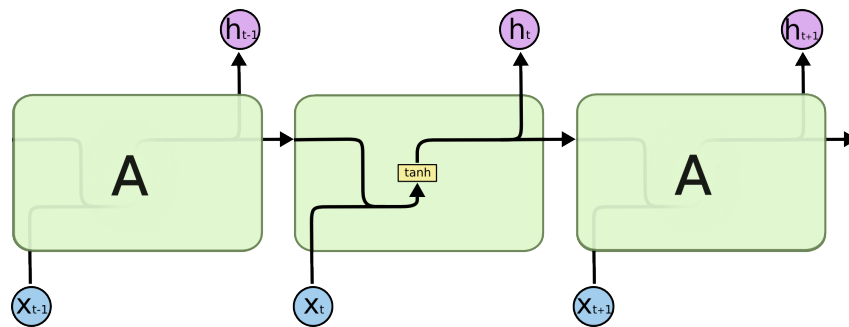
## 2.5 SEQUENCE-BASED NEURAL NETWORKS

In the context of record linkage on the web, most available data assume the form of textual sequences of some kind. Indeed, two of the main attributes in the places data domain are place names and addresses, both being short sequences of texts. In this context and more generally in the NLP area, two types of networks are ubiquitously utilized with success: Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN). This Section describes these two types of networks and their most common architectures.

### 2.5.1 Recurrent Neural Networks

Recurrent neural networks are neural networks that process inputs in multiple iterations, as Figure 6 shows. By doing so, they are able to store information about past inputs and utilize them to aid in the present task. As Olah (2015) points out, this idea is aligned with how humans think, since we continuously utilize past information to infer, for instance, the meaning of a text when reading it. When processing textual data, each word or character in a sentence can be fed sequentially to the RNN for it to learn relationships between those words.

Figure 6 – Overview of a traditional RNN processing input sequence  $x$ .



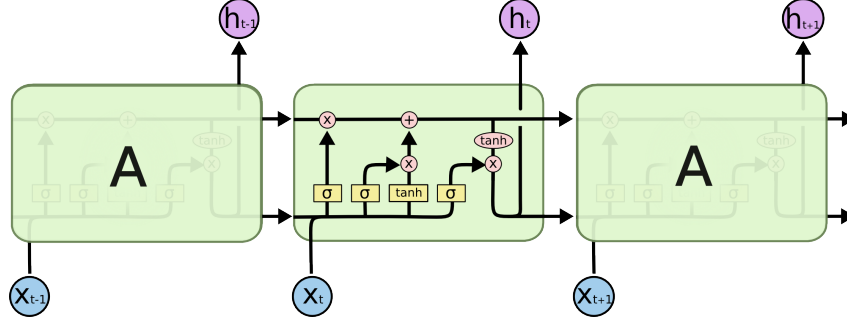
Source: (OLAH, 2015)

Even though they date back to 1990 (JORDAN, 1990), the popularization of RNNs was triggered by the development of the Long Short-Term Memory (LSTM) architecture (HOCHREITER; SCHMIDHUBER, 1997). LSTM networks account for the fact that traditional RNNs are not able to store relevant information for longer sequences (BENGIO; SIMARD; FRASCONI, 1994), by utilizing a cell state which is shared between all time steps in the network. This cell state may have information added or removed from it by a series of internal operations named gates: a forget gate, an input gate, and an output gate, as

<sup>4</sup> <<https://fasttext.cc/>>

Figure 7 shows. In the Figure, layers are represented in light yellow, pointwise operations in red, and vector operations by the arrows.

Figure 7 – Internal structure of a LSTM network.



Source: (OLAH, 2015)

The forget gate is responsible for reading the past hidden state and the current input to decide how much of the previous state should be kept in the cell state. Then, the input gate reads the current input and the last hidden state to decide which values to be updated in the cell state, and how to update them. In sample Figure 7, a hyperbolic tangent ( $\tanh$ ) function is applied to the input value and last hidden state to decide how to update them, but this activation function may be modified.

The cell state is updated with the outputs from the forget and input gate. Finally, the output gate reads the updated cell state and the current input to generate the hidden state of the current time step. The last hidden state of LSTM networks may be utilized as the final output for a sentence, but the hidden states from each cell may also be processed in some manner, like a max pooling operation, to achieve a final result.

Expressing a LSTM mathematically, we have:

$$\begin{cases} f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \\ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t = o_t * \tanh(C_t) \end{cases} \quad (2.12)$$

where  $f_t$  is the forget gate,  $i_t$  is the input gate,  $o_t$  is the output gate,  $C_t$  is the cell state at time step  $t$ , and  $h_t$  is the hidden state at time step  $t$ .

A widely used variation of LSTM networks is Gated Recurrent Unit (GRU) networks (CHO et al., 2014). Among the many changes made to the LSTM base cell, this model most notably merges the cell state and the hidden state, thus limiting the output values of each cell to a single vector, instead of two. In general, GRU networks have been found

to provide similar or better results than LSTMs for shorter sentences, and are trained faster.

While LSTM networks and its variations excel at handling long sequences of data and finding relevant patterns among each entry, some tasks require the network to focus on specific input steps rather than others. In the case of sentence translation, two LSTM networks are usually utilized: one serves as an encoder, which processes the original sentence to be translated, and the second one serves as a decoder, which receives the encoded representations as input and outputs translated words.

In this case, linguistic differences may cause some words from the original sentence to have an offset in the desired output sentence. When this happens, LSTM networks may fail to remember the necessary context from previous words to correctly translate them. In their work, Bahdanau, Cho and Bengio (2015) propose the concept of attentive training for translating sentences with RNNs. In short, the attention mechanism serves as an extra layer between the encoder and decoder which has its weights calculated based on comparisons between the states of the two networks at each time step. By multiplying those learned weights by the encoder outputs, the decoder is able to focus on different words at different time steps, independently of alignment with the original sentence.

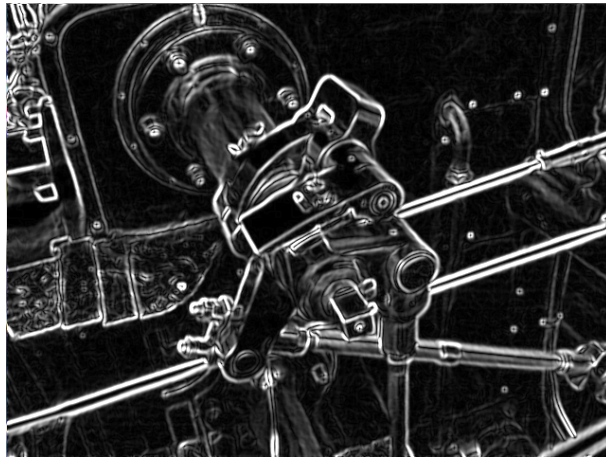
### 2.5.2 Convolutional Neural Networks

Convolutional Neural Networks are networks which utilize a combination of convolutional filter layers on top of the input data to capture a collection of local features from it. In the context of machine learning, a convolution is composed of a kernel, which is a matrix of weights that is applied to the input values. Convolutions have been widely utilized in the image processing field, where some known kernels detect features such as edges, as shown by the application of the Sobel filter (SOBEL, 2014) in Figure 8, when convolved with the pixel matrix of an image.

Apart from utilizing multiple convolutional filters, CNNs usually employ aggregation operations, such as max pooling, between each convolutional layer. The intuition behind applying these operations is that CNNs attempt to find distinguishing local features, so the most important information for them is knowing whether or not a certain feature appears in a sequence, and not necessarily how or where the feature precisely occurs in each step of the input data. Drawing from the edge detection sample again, a convolutional filter needs to detect where are the edges positioned and then highlight them, but does not need to do so with pixel-perfect precision, and does not need to produce partial matches for the rest of the image. Given the widespread usage of convolutions on research fields dealing with images, CNNs have also shown most usage and success on image-related problems (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

However, one dimensional CNNs have also been employed for textual data with great success (COLLOBERT et al., 2011; KIM, 2014). Each convolutional layer of a 1D-CNN re-

Figure 8 – Edge detection made by convolving a Sobel kernel with the original image.

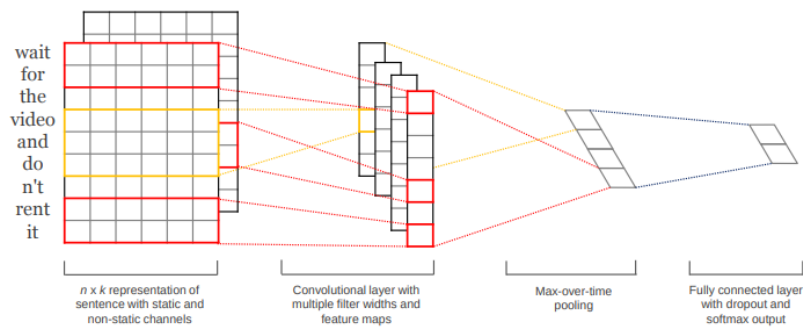


**Source:** (CONTRIBUTOR, 2008)

ceiving a sequence of size  $n$  may be defined by a kernel (window) of size  $k$ , a filter matrix  $f \in \mathbb{R}^{k \times d}$ , and an activation function. Then, sub-sequences of size  $k$  are extracted from the input, and  $n - k + 1$  vectors of  $d$  dimensions are generated, passed through the activation function, and returned.

In the work of Kim (2014), a 1D-CNN is trained on sentences following the architecture described by Figure 9. In this architecture, each word in a sentence is attributed to a pre-trained word embedding. Then, a convolutional layer using multiple filters with different window sizes extracts features from sub-sequences of words, which are defined by the concatenation of their respective word vectors. For instance, one of those filters with a window size of 2 may extract features from the sub-sequence  $\langle \text{wait}, \text{for} \rangle$ , while other one with a window size of 3 extracts feature from  $\langle \text{video}, \text{and}, \text{do} \rangle$ . These features are then passed through a max pooling over time, generating a final vector representation for the sentence.

Figure 9 – CNN architecture for extracting features from sentences.



**Source:** (KIM, 2014)

This network is trained for several different tasks involving text: sentiment analysis,



binary classification of positive or negative reviews, subjectivity classification, multi-label question classification, and opinion polarity detection. In most of them, the network is shown to surpass the results of other methods in the state-of-the-art with little amounts of training, which shows the power of 1D-CNNs when dealing with textual data.

### 3 BACKGROUND: PLACES

This work’s main goal is detecting replicated place entities in a database. Place databases built from Web data, in turn, are commonly created and populated following a fixed definition of what a place entity is, but due to the noisy nature of Web data, the entities present in these databases suffer from a series of data quality and consistency issues.

This Chapter serves as a foundation for understanding the data records dealt with by our proposed solution. It provides a thorough explanation of what a place entity is in the context of this work, which attributes are commonly associated with them, and issues pertaining to each attribute.

#### 3.1 PLACE ENTITY DEFINITION

Albeit having a clear definition in dictionaries, the “place” term is used ambiguously when referring to an entity type in a database. For instance, Google Places API<sup>1</sup> defines a place as “...establishments, geographic locations, or prominent points of interest”, while Yang et al. (2019) seem to utilize a broader concept that encompasses any page created in Facebook’s online service labelled as a place by the user. Dalvi et al. (2014) also utilize a broader concept, in which “...a place is defined as any entity that has a name and a physical location.”.

Adding to that, some place APIs and previous works also refer to places simply as Points of Interest (POI) (YANG et al., 2017; DENG et al., 2019; FACTUAL, 2020), and a few other authors also refer to places as toponyms in an interchangeable manner (SANTOS; MURRIETA-FLORES; MARTINS, 2017; SANTOS et al., 2017; MARINHO, 2018; KAFFES et al., 2019), mainly when dealing exclusively with place names. These inconsistencies derive from the intrinsic characteristics of each database and from the business interests involved in common place services. While we do not aim to tackle these differences in nomenclature, we consider it crucial to first provide a formal place definition in the context of our work:

**Definition 3.1.1. [Place entity].** A place entity in our database is a direct representation of a physical location in the real world which has well-defined boundaries, a clear purpose for visitation, and a considerable area.

Definition 3.1.1 is narrower than the ones previously described, and aims to only encompass entities that may not be ambiguously defined. We provide a few examples of entities and whether or not they fit out definition in Table 1.

---

<sup>1</sup> <<https://developers.google.com/places/web-service/intro>>

Table 1 – Examples of entities and whether or not they fit the definition of a place in our work.

Entity	Fits our place definition	Reason
A Beach	no	Boundaries are not well defined.
A store on a beach	yes	Meets all of the requirements.
A small statue	no	Does not have a considerable size.
The Statue of Liberty	yes	Meets all of the requirements.
A building floor	no	Has no clear purpose for visitation.
A random street	no	Has no clear purpose for visitation.
An office in a corporate building	yes	Meets all of the requirements.
A private mall parking lot	no	Has no clear purpose for visitation; is part of a shopping mall place.

**Source:** (M. R. Cousseau, 2020)

Some applications such as recommendation systems, engagement solutions, and review platforms are only able to function with information about places. To achieve this, they rely on places having a plethora of fields, some of which fall outside of this work’s scope.

Hence, we describe the most relevant place fields for record linkage solutions in Table 2, using the schema from our use case as an example. For ease of explanation, we overload the nomenclature of the *address* field with a composition of the *thoroughfare* and *sub\_thoroughfare* sub-fields, since they see most usage, and note that the *parent\_place\_id* field of a place represents another place in the database which encompasses it, e.g. a store inside a shopping mall. Additionally, for our use case, the *categories* field has 122 pre-defined values to choose from. Some examples of categories are *bar*, *restaurant*, *store*, *landmark*, and *corporate\_building*.

Table 2 – Description of fields for place entities, using the schema from our database.

Field	Sub-field	Type	Obligatory	Example
id	-	Textual	yes	507f1f77bcf86cd799439011
name	-	Textual	yes	Empire State Building
geo_location	-	Latitude and longitude coordinates	yes	40.7484629, -73.9856671
phone_number	-	Textual	no	+1 212-736-3100
homepage	-	Textual, URL	no	<https://www.esbnyc.com/>
categories	-	List of pre-defined values	no	[landmark, corporate_building]
parent_place_id	parent_place_id	Textual	no	507f1f77bcf86cd799438012
	country	Textual	no	US
	admin_area	Textual	no	New York
	sub_admin_area	Textual	no	New York City
	locality	Textual	no	Manhattan
	sub_locality	Textual	no	Midtown South
	thoroughfare	Textual	no	W 34th St
	sub_thoroughfare	Textual	no	20
	postal_code	Textual	no	10001

**Source:** (M. R. Cousseau, 2020)

### 3.2 CHALLENGES OF DEALING WITH PLACE FIELDS

This collection of fields presents several issues which may be encountered in any Web-based database in this domain. First of all, textual fields suffer from normalization problems, meaning that even if the sources from which they are gathered offer structured ways to insert the field, the value inside this structured field may have no strict ruling. This results, for example, in places named *Aeroporto Internacional de Guarulhos*, *Aeroporto Internacional de São Paulo - GRU* and *Aeroporto Governador André Franco Montoro* representing the same real world place. Moreover, textual fields are prone to typing errors and differences in the use of accents and special characters.

The *geo\_location* field is often attached to a place entity by manual tagging with pin-

Figure 10 – Common ways to input geo location fields of places in applications.

Address

Thoroughfare ⓘ	Sub Thoroughfare ⓘ
Avenida Cidade Jardim	191
Locality ⓘ	Sub Locality ⓘ
São Paulo	Itaim Bibi
Admin Area ⓘ	Sub Admin Area ⓘ
SP	São Paulo
Country ⓘ	Postal Code ⓘ
BR	01453-000

[Open with Google Maps](#)

(a) Receiving full address to be geocoded.

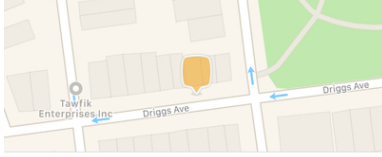
Verizon LTE 2:16 PM 81%

Cancel Add New Place

Name

Grocery

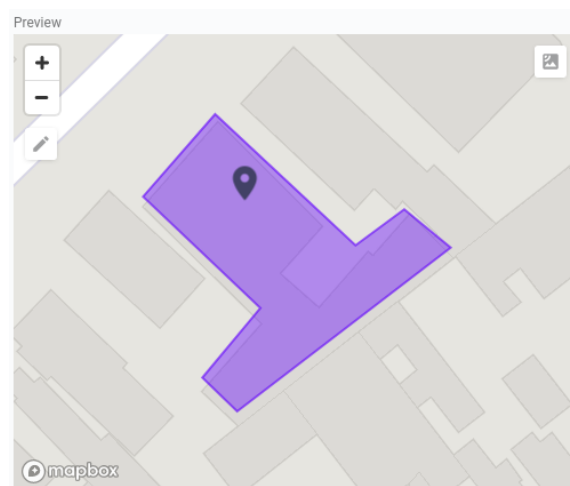
Location



Add a map pin >

Next Step

(b) Using GPS sensors.



(c) Manually pinning.

**Source:** (a,c) (M. R. Cousseau, 2020) (b) (FOURSQUARE, 2020)

point precision, shown in Figure 10c, or by a process named geocoding (CENTER, 2010), which converts textual addresses like those of Figure 10a into latitude and longitude pairs. In mobile applications, humans are often prompted for an address which is then geocoded, instead of manually pinning. In addition, some applications such as Swarm<sup>2</sup> - exemplified in Figure 10b - and Facebook allow their users to create new places using the latitude and longitude coordinates obtained from GPS sensors in smartphones and other devices.

All of the commonplace approaches to obtain geographical coordinates for a place entity may fail to consistently provide correct results with high precision due to these factors:

- For geocoded addresses, small modifications or lack of data in the original provided address, as well as failures in the geocoding system, may lead to different results;
- Locations obtained through GPS sensors in applications may vary because the GPS system is inaccurate in environments such as indoor ones, that pose interference, in which humans spent 87% of their time in 2001 (KLEPEIS et al., 2001). Users may also attempt to create a place far away from their current location and wrongly provide it as the place's (DALVI et al., 2014);
- Manual pinning, albeit precise, is seldom utilized in mobile applications due to user experience faults. It also may lead to confusion in cases where a place has no clear latitude and longitude due to its dimensions, e.g. a shopping mall, and lead users to create replicated places.

The address sub-fields present normalization issues in addition to those already pertaining to textual fields, due to different countries and regions having different conventions for administrative levels and abbreviations. For instance, while US street addresses often present their *sub\_thoroughfare* information at the beginning of the address, e.g. *34 First Avenue*, Brazilian street addresses usually contain the same information at the end of the address. This in turn hinders the development of heuristics for detecting address fields, and may affect deduplication models.

The *parent\_place\_id* and *categories* fields may be scraped from the Web or derived from other heuristics and models. For our use case, in short, a heuristic for parent place detection utilizes textual similarities between place names to detect cases such as *Jersey Gardens* being a parent of *Abercrombie - Jersey Gardens*. Meanwhile, categories are generated by a multi-label classification model, which accounts for places that may belong to several categories, such as *bar* which is also a *restaurant* during the day. Their main issue is exactly that: any approach developed for detecting parentage relationships or categories is prone to imperfections. Finally, problems with the *phone\_number* and *homepage* fields are less frequent, and their main issue is being rarer to find.

---

<sup>2</sup> <<https://www.swarmapp.com/>>

## 4 RELATED WORK

This Chapter offers an overview of previous similar works, drawing parallels to ours. In this analysis, we prioritize works which propose strategies for the duplicate detection (entity matching) task as a partial solution for the record linkage problem in the places domain, since the main contribution of this work resides there. We also highlight those which handle pre-detection or post-detection steps when applicable.

Table 3 – Related works, their approaches, features, and limitations.

Work	Features	Approach	Limitations
Dalvi et al. (2014)	names, geo locations	EM for core words detection with spatial contexts	Few features; does not learn relationships between words.
Berjawi (2017)	names, geo locations, addresses, categories	Combination of multiple string and spatial similarity metrics by a probabilistic approach; data fusion system	Deals directly with data from APIs; set of generational rules is hard to maintain
Deng et al. (2019)	names, geo locations, addresses, categories	Combination of multiple string and spatial similarity metrics by a probabilistic approach	Deals directly with data from APIs; tailored towards Chinese places; uses a small data set
Yang et al. (2019)	names, geo locations, addresses, categories, label sources	Unsupervised feature generation for places; metric learning on top of learned representations; novel loss function, sampling, attentive training, and label denoising	Depends on some constraints in the data set; trained on inaccessible <i>corpora</i> ; costly to build
Santos et al. (2017)	names	Siamese GRU networks for character sequences	Few features; results are biased towards transliterations
Marinho (2018)	names	Improvement of Santos et al. (2017) with attentive training and other techniques	Results are equivalent to those of the original work

**Source:** (M. R. Cousseau, 2020)

For the sake of legibility, we divide the related works into two categories, based on how they approach the duplicate classification task: (i) deep learning and (ii) traditional approaches. Table 3 sums up the main characteristic of each of them, and they are further described in the sections that follow.

#### 4.1 TRADITIONAL APPROACHES

The solution developed by Dalvi et al. (2014) approaches the problem of detecting duplicate places by using information from place names and latitude and longitude pairs to build a name model and a spatial context model. The main concept behind the work is that places names are sufficiently represented from a set of core words, the remaining ones being background words. For instance, a place named *Starbucks Coffee* is sufficiently represented by *Starbucks* alone, so when compared to another place named *Peete's Coffee*, the word *Coffee* should not have a big influence in similarity computations between both places.

The concept of core words guides the authors' process of generating a name model: they propose an Expectation-maximization (EM) algorithm (DEMPSTER; LAIRD; RUBIN, 1977) where a probability distribution of core words and a probability distribution of background words are progressively calculated from initial random states. This EM algorithm is expanded to account for a spatial context model. In this expansion, the authors perform a grid subdivision, and allocate each place to a tile. Then, a probabilistic background word distribution is calculated not only globally, but also for each tile during each step of the EM algorithm. The spatial context model tries to capture the fact that the relevance of each word is tied to its position on the globe: while the word *NYC* might not be relevant at all for places in New York City, the same word in Amsterdam might be highly relevant, as it would be more uncommon.

Finally, the work computes the probability of two places being duplicates by using the probability of the core word set from both of them being equal. They then expand this concept to include string edit operations by providing a dynamic programming algorithm. This approach, however, does not learn relationships between words and is constrained by a small feature set, leading to an algorithm which is prone to errors. As Chapter 8 shows, the EM algorithm is also slow to train.

In his thesis, Berjawi (2017) studies the problem of linking place records from different Location Based Services (LBS), such as Google Maps<sup>1</sup>, in a broad scope. Among others, he proposes a spatial blocking method to detect potential pairs, a similarity algorithm to match places, and a data fusion algorithm for the detected duplicates.

A key aspect of the thesis is dealing directly with LBS, instead of receiving a full-blown places database as input. Hence, the proposed spatial blocking strategy is based on

<sup>1</sup> <<https://www.google.com/maps>>



two specific queries made through each API of the LBS: for a given place, the first query returns all places in a given radius with the same category as the original one, and the second one returns all nearby places with at least one common token in their name. The union of those queries generates potential pairs of duplicate places.

The matching algorithm developed by Berjawi (2017), named Global Similarity, uses a probabilistic approach to combine multiple traditional similarity metrics such as Levenshtein (LEVENSHTein, 1966) and Trigram (ULLMANN, 1977) distances for multiples fields, and spatial distances comparing geo locations between place pairs. It follows the intuition that each similarity metric by itself presents specific pitfalls, and combining many of them alleviates this.

The probabilities resulting from the Global Similarity are processed by a decision algorithm that applies a threshold to produce a binary classification result. Finally, the classification results are merged through an algorithm that takes the global similarity for each pair into account, to not only expand the matching to sets of more than two places, but to choose the best values among them.

As limitations, dealing with data from LBS restricts the data to be matched, as seen with the proposed blocking algorithm. It also ensures a higher level of data quality than that of raw Web entities, such as those dealt with by our work. This work is also more focused on creating a new database rather than improving the quality of an existing one, and uses a set of probabilistic rules which is hard to maintain and visualize in a production system.

The work by Deng et al. (2019) tackles the record linkage problem for two different data sets obtained from Chinese LBS. Similarly to other traditional approaches, they use a multi-attribute matching strategy to classify duplicate places, having access to the name, address, geo location, and category of each place, but the core of their work resides in the combination strategy for these similarities, using an improved version of the Dempster-Shafer (D-S) evidence theory (DEMPSTER, 1967; SHAFER, 1976).

For computing spatial similarities, they use a normalized Euclidian distance between the places. Then, to compute name similarities, the authors utilize a modified version of the Levenshtein edit distance which accounts for differences in Chinese words. Akin to that, the address similarities are calculated by means of a feature extraction step based on pre-constructed Chinese word segmentation dictionaries and knowledge bases, and a subsequent cosine similarity calculation for each of the feature vectors. Finally, the similarity between categories is computed by a tree matching algorithm, which depends on a manual step to fuse category schemes from different LBS.

The similarity values for each attribute are then considered as independent evidences, and are used to construct basic probability models using the D-S evidence theory. Next, the authors improve the basic models by accounting for cases where evidence is highly conflicting, improving the D-S framework for their use case, and decision thresholds are

applied to produce final duplicate detection results. Their work is tested on top of data sets from two LBS, Baidu<sup>2</sup> and Gaode<sup>3</sup>, from which approximately 300 entities representing the same real-world place in the two data sets are manually detected and labeled.

Similarly to the work by Berjawy (2017), dealing directly with data from APIs of LBS implies in less noise and a more strict data quality than the Web’s. Furthermore, many of the similarity strategies proposed by their work are tailored towards Chinese places, and the category similarity assumes knowledge and manual fusing of the category schema from each source. Due to those reasons, albeit providing an insightful take on combining multi-attribute similarity metrics, most of the findings from the research are not applicable to our use case.

## 4.2 DEEP LEARNING APPROACHES

The work of Yang et al. (2019) draws inspiration from previous solutions for entity resolution and person re-identification tasks in different domains to create unsupervised representations of place entities from the Facebook Web service, noting that the user-facing nature of the service leads to place replications. They name this step as unsupervised feature generation, and utilize places’ names, addresses, locations, and categories to do so. Then, they apply a metric learning (LU; HU; ZHOU, 2017) step which transports the place representations into a metric space where duplicate places are close, and non-duplicated places are far apart.

For the unsupervised feature generation step, they utilize FastText (BOJANOWSKI et al., 2017; MIKOLOV et al., 2018) to create name word embeddings, using a *corpus* of 1.9 trillion words from public Facebook posts in the last 10 years, and a skip-gram Word2Vec (MIKOLOV et al., 2013) model trained on top of words from place addresses to create address word embeddings. Next, they incorporate category and geo location information by first applying a grid subdivision, then creating a places graph where places in the same grid or belonging to the same category are connected, and finally smoothing this graph by training a skip-gram objective function with negative sampling (PEROZZI; AL-RFOU; SKIENA, 2014).

This feature generation method is dubbed NF+AS+CS by the authors, and its results are passed through an MLP network which uses a novel pairwise contrastive loss, an adapted version of a triplet loss. This model, named PE, is further improved by batch-wise hard sampling (PEH), source-based attentive training (PEHA) and cluster-based label denoising (PEHAD) (GUO et al., 2017).

The batch-wise hard-sampling uses a secondary distance metric, which is mentioned but apparently not explained in their work, to discard samples below a certain percentile of distances in the batch from contributing to the training loss. Meanwhile, the attentive

<sup>2</sup> <<http://map.baidu.com/>>

<sup>3</sup> <<https://www.amap.com/>>

training aims to attribute different weights based on the source of each labelled place pair, due to the sources used in their data set having different quality levels. Finally, the proposed label denoising technique draws inspiration from self-training networks to create clusters for the positive and negative classes of samples, and updates their representatives during the training process. These clusters are then utilized to detect labelled pairs too distant from their respective clusters, lessening their effect on the model training process.

Their approach is similar to ours in many aspects, since our Classification Model also leverages embeddings to build vector representations for places using the same attributes. However, our solution is built from the ground up with a focus on detecting duplicate places, and, as such, the network generates embeddings in a pairwise and supervised manner, using unsupervised training only for initializing embeddings layers in the word encoder. Furthermore, the places graph generated during the embedding smoothing is costly to build and maintain, and the text *corpora* of 1.9 trillion words is inaccessible to most researchers and companies alike.

In their work, Santos et al. (2017) tackle the problem of toponym matching, which means that they attempt to detect duplicate places taking only their names into consideration. Their main argument is that using only word-level information for matching names has a performance plateau, due to them demanding common substrings to function. This, in turn, causes word matching approaches to fail during transliterations, typing errors, or slight semantic changes to toponyms over time.

Based on that, they propose a deep architecture with two layers of siamese GRU networks processing character sequences for two place names, generating an embedding for each place, and merging them through a series of operations. This result is passed through a feed-forward network with Dropout regularization (SRIVASTAVA et al., 2014) and a final sigmoid layer produces a probabilistic output indicating if the two places are duplicates.

They compare this solution to previous string matching heuristics and supervised learning approaches for combining string similarity metrics (SANTOS; MURRIETA-FLORES; MARTINS, 2017), showing significant improvements over all of them. This attests to the representational power of characters in the task of name matching, however, the results presented by the authors are biased by transliterations, a case which is uncommon in our database.

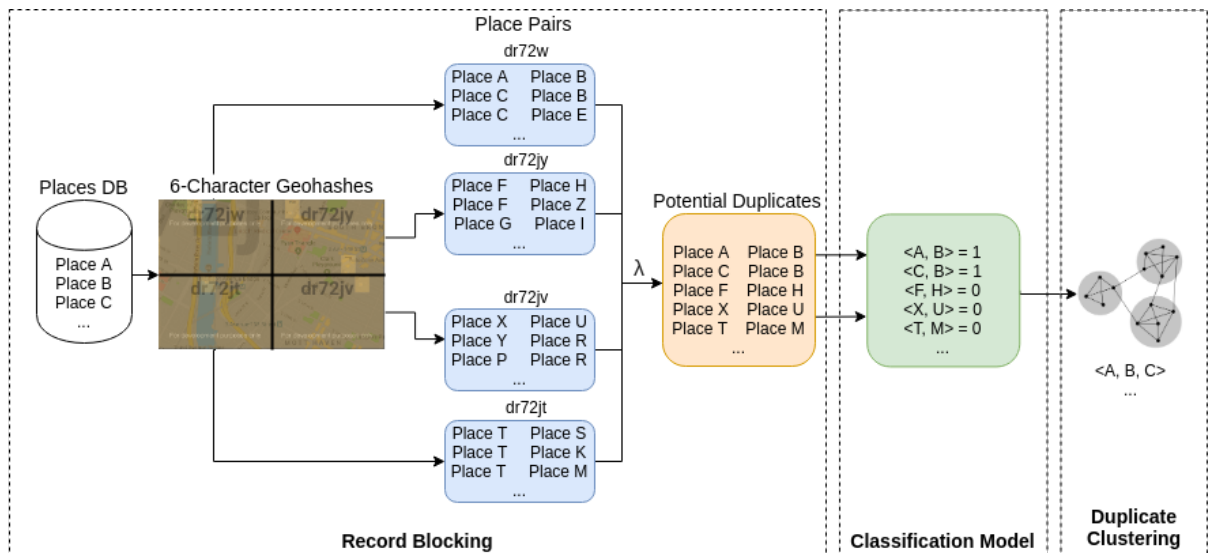
Marinho (2018) further attempts to enhance the work from Santos et al. (2017) by applying attention techniques, shortcut connections, and highway networks. They perform extensive experimentation on 4 data sets, but most of the results show little to no improvement of these techniques over the original model, which advocates for choosing the simplest and most effective solution instead of the most complex one, as Yang et al. (2019) also point out.

## 5 PLACE RECORDS' LINKAGE PIPELINE

In order to tackle the problem of record linkage for place entities, we propose an end-to-end pipeline that, given a database of places, outputs clusters of replicated entities in it. The pipeline is composed of three steps, as shown in Figure 11. First, the Record Blocking step is responsible for partitioning, blocking and filtering the database of places, transforming them into potential pairs of duplicates. It does so by computing similarities between place pairs for each Geohash and pruning trivial cases.

These pairs are then processed by a Classification Model that predicts duplicate places based on their geo location, categorical, and textual information. Finally, the Duplicate Clustering step creates clusters of replicated places by employing a graph-based approach.

Figure 11 – Overview of our pipeline.



Source: (M. R. Cousseau, 2020)

With this pipeline we are able to process a places database in its entirety, producing a list of replicated place entities to be processed in batches, and at the same time provide real-time duplicate detection via an API by porting our Classification Model. Duplicate places detected by this pipeline can then be cleaned up or merged by Data Fusion techniques (BERJAWI, 2017).

This Chapter provides an in-depth description of our pipeline steps in the same data flow presented in Figure 11. Thus, we firstly describe our Record Blocking algorithm in Section 5.1, and our preliminary attempts to develop a Classification Model in Section 5.2. Finally, dissections of the Duplicate Clustering step and of APIs using the Classification Model are performed in Sections 5.3 and 5.4, respectively. While this Chapter offers explanations on the first Classification Models developed by us, which serve as a baseline

for our final solution, the final deep neural architecture for the Classification Model is explained in Chapter 6.

## 5.1 RECORD BLOCKING

The first major objective in our places’ linkage pipeline is generating place pairs that need to undergo a duplicate detection process from our Classification Model. The simplest approach for solving this would be generating all place pairs in our database and then processing them, by using the rationale that the model should be good enough to discard obvious non-duplicates.

This, however, has some implications: (1) processing all pairwise combinations of places is computationally expensive, since its complexity is  $O(n^2)$ , where  $n$  is the size of the places’ database; and (2) there is a considerable imbalance between duplicates and non-duplicates, i.e., the data is highly skewed towards non-duplicates, which can compromise the performance of the duplicate classifier (HE; GARCIA, 2009).

To deal with these issues, and consequently make our solution scalable, we implement a two-step record blocking strategy. First, the places are grouped based on their geographical regions (partitioning phase) and then only places within the same region are compared in a pairwise fashion to identify candidate duplicates (filtering phase).

More specifically, the partitioning step prunes the search space by allocating the places in a spatial grid using the Geohash system<sup>1</sup> (MORTON, 1966), a public domain hierarchical geocoding system which subdivides the world into buckets of variable precision determined by a string of characters. We utilized a resolution of 6 characters, which was empirically defined in the early stages of our work. This resolution represents an area of approximately  $744m^2$  and a diagonal distance of approximately  $1364m$ , that vary depending on the actual location being partitioned due to the non-Euclidean nature of the calculations. This grid resolution aims to keep the place density in a grid cell high enough to encompass relevant pairs, but low enough to ease up on the computational costs.

In the filtering step, all possible place pairs within each Geohash  $g_i$  are generated with a complexity of  $O(|g_i|^2)$ , where  $|g_i|$  is the number of places in  $g_i$ . Place pairs are then filtered by comparing both place names with the Jaro Winkler string similarity (WINKLER, 1990), similarly to previous record linkage approaches (COHEN; RAVIKUMAR; FIENBERG, 2003; MOREAU; YVON; CAPPÉ, 2008; CHRISTEN, 2011; SANTOS; MURRIETA-FLORES; MARTINS, 2017).

Next, a threshold  $\lambda$ , which should be low enough to preserve duplicates, and high enough to prune trivial non-duplicate pairs, is applied to the pairwise comparisons. The filtering step thus reduces the amount of pairs to be further stored and processed, consequently reducing the complexity of other steps down in the pipeline.

<sup>1</sup> <<https://web.archive.org/web/20080221111539/http://blog.labix.org/author/admin/>>

We note, however, that this filtering step has a limitation since it may drop duplicate pairs with dissimilar names. More specifically, the Jaro Winkler similarity metric weights tokens in the start of each text more heavily. Thus, name pairs like *Starbucks Coffee* and *Coffee Starbucks* would display a very low similarity value: 0 in this specific case. Chapter 2 touched upon proposed alternative versions of the Jaro Winkler heuristic, such as ordering tokens before comparison. Since these variations also bring other error cases along with them, we use the original version of the heuristic, and leave improvements to this step as future work.

For both the partitioning and filtering steps, we use the Apache Spark Framework (ZAHARIA et al., 2016) for distributing computation among several machines. Since our records are partitioned first into Geohashes, thus gaining a partitioning key, and then must be processed and aggregated, our use case is a canonical one of the MapReduce programming model (DEAN; GHEMAWAT, 2004) implemented in Apache Spark.

## 5.2 PRELIMINARY CLASSIFICATION MODELS

Several methods were incrementally developed to try and solve the problem of binary classification for place pairs during the extent of this work. Since we reason that a heuristics-first approach could yield a better understanding of the data domain, as well as some preliminary results to base ourselves upon, we present it first. Given that it does not provide satisfactory results, however, we attempt more elaborate solutions using machine learning techniques, basing ourselves upon literature. These later approaches are subsequently explained.

### 5.2.1 Heuristic Approach

Similarly to Dalvi et al. (2014), the heuristic approach taken to deduplicate place entities relies on the concept of core and background words for a given place. Using the authors' own example, the place named *Peete's Coffee* is sufficiently defined by the word *Peete's* alone, while *Coffee* is simply adding context, i.e. the category of the place in this case. These concepts translate to the duplicate detection problem by providing a basis on which to compare place names upon. For instance: when faced with a place pair with names *Peete's Coffee* and *John's Coffee*, given  $core(Peete's\ Coffee) = \{Peete's\}$  and  $core(John's\ Coffee) = \{John's\}$ , most simple edit distance metrics would be able to assert non-duplication between the two cases by comparing core words only.

Moreover, the location-sensitive aspect of place entities causes some words to be more relevant to the composition of a core word set in some locations than others. Places containing street names or folklore references are exemplary of this behavior, e.g. for a place named *Bar 48* in a street named *48th Street*, *48* could be irrelevant if there are

several other establishments with *48* in their names, but the same name in a street named *Main Street* could have *48* as its most relevant word.

The method from Dalvi et al. (2014) utilizes the core word concept to deduplicate places, extracting them by probabilistic methods which take the aforementioned spatial context into consideration. Our approach differs from Dalvi et al. (2014), however, in the sense that it presents a simpler method of calculating the relevance of each word to a place’s name, while also restricting the cardinality of the core word set to 1. The full code for this approach, named Word Relevance Heuristics (WRH), is presented in Algorithm 1.

Given a word set  $W$  composed of all words in all place names, and a set of place names  $N$ , for each word  $w \in W$ , we calculate its Document Frequency (DF) value with Equation 5.1.

$$DF(w, N) = \frac{|n \in N : w \in n|}{|W|} \quad (5.1)$$

We choose to calculate the document frequency instead of its IDF to facilitate the selection of thresholds, and the TF value for each word is not taken into account due to place names with repeating words being uncommon. Then, to take each place’s location into account, we also calculate local DF values: given  $W$ ,  $N$ , and  $G$  - containing places partitioned by all 6-character Geohashes - for each  $w \in W$  in each  $g \in G$  we calculate a DF value with Equation 5.2.

$$DF(w, g, N) = \frac{|g|}{|n \in g : w \in n|} \quad (5.2)$$

The algorithm first calculates the core word for each place by tokenizing their names in line 6, and comparing each token’s local DF value to an usage threshold in line 8, which dictates how many times a word is allowed to appear in that specific Geohash. Additionally, a locality threshold  $L$  is utilized to check if appearances of a given word are tied to a specific Geohash, attempting to find localized words. An exact match comparison is then performed on the core word of each of the two places in line 10.

---

**Algorithm 1:** Word relevance heuristics (WRH).

---

**input :**  $dfs_{global} \leftarrow$  hash map of size  $|W|$   
 $dfs_{local} \leftarrow$  hash map of size  $|G| \times |W|$   
 $pairs \leftarrow$  list of place pairs  
 $U, L, D \leftarrow$  usage, locality and distance thresholds  
**output:** list of duplicate pairs

```

1  $duplicates \leftarrow []$ ;
2 for  $pair \in pairs$  do
3    $(first, second) \leftarrow pair$ ;
4   for  $place \in pair$  do
5      $place.core \leftarrow place.name$ ;
6      $tname \leftarrow \text{Tokenize}(place.name)$ ;
7     for  $word \in tname$  do
8       if  $dfs_{local}[place.geohash][word] < U$  and  $\frac{dfs_{local}[place.geohash][word]}{dfs_{global}[word]} < L$ 
9         then
10           $place.core \leftarrow word$ 
11   if  $first.core \neq second.core$  then break;
12   if  $\text{Distance}(first.location, second.location) > D$  then break;
13   // Similar () uses a manually defined mapping
14   if  $\text{not Similar}(first.categories, second.categories)$  then break;
15   if  $first.parentage = second.id$  or  $first.id = second.parentage$  then
16     break
17    $duplicates.append(pair)$ ;
18 return  $duplicates$ 

```

---

Lines 11 to 14 of the algorithm use additional fields to avoid false positives, such as *Peete's Coffee* and *Peete's Hospital*. In this sample case, both *Hospital* and *Coffee* could have been discarded as possible core words, but by having access to each place's category - *coffee* and *hospital*, respectively - our approach would be able to disregard the pair.

*WRH* depends on several manually defined thresholds, as well as a manually defined mapping of similar categories. Additionally, the comparison between core words is very simplistic, avoiding any kind of edit distance metrics. This all leads to a hard to maintain and easily breakable method, failing to handle typing errors or small modifications in a place's name. It also highlights the inherent complexity of developing heuristics involving place entities, due to their abundance of noisy information and outliers - pitfalls which are noted in the literature.



### 5.2.2 Supervised Learning Approach

To tackle the pitfalls encountered in our heuristic approach, we attempt a supervised learning solution for performing record linkage on place entities, since they have been reported to successfully produce relevant results for matching tasks (SANTOS; MURRIETA-FLORES; MARTINS, 2017). Additionally, Wilson (2011) proves that machine learning models expand the mathematical foundations of record linkage and thus are prone to offer better results. Our first model for the detection of duplicates among place pairs (COUSSEAU; BARBOSA, 2019) is trained with the pairwise features described in Table 4.

This model still relies on the place’s core word as a feature (`core_word_jaro`), which is detected by the logic in Algorithm 1. It also utilizes Soft TF-IDF (COHEN; RAVIKUMAR; FIENBERG, 2003) comparisons for the name (`name_soft_tfidf`) and thoroughfare (`address_soft_tfidf`) attributes, 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.

Table 4 – Pairwise features and their descriptions for our ensemble classifiers.

Feature	Description
<code>core_word_jaro</code>	Jaro Winkler similarity between core words
<code>name_soft_tfidf</code>	Soft TF-IDF between normalized names
<code>address_soft_tfidf</code>	Soft TF-IDF between normalized street names
<code>sub_thoroughfare_diff</code>	Difference between numeric sub thoroughfares
<code>categories_jaccard</code>	Jaccard similarity between category lists
<code>distance</code>	Distance in meters between two geo locations
<code>homepage_matches</code>	Exact homepage match
<code>parent_place_match</code>	Exact match between an id and parentage
<code>siblings_match</code>	Exact match between parentages

**Source:** (M. R. Cousseau, 2020)

Additionally, we add parentage id (`parent_place_match`) and sibling comparison features (`siblings_match`), which are decomposed into separate one-hot vectors, as shown in Figure 12. With the parentage feature, we aim to induce the model to learn that places in a pair which are encompassed by their counterpart, e.g. a store inside a shopping mall, have a lower chance of being duplicates.

In a similar manner, the sibling comparison feature indicates whether the two places in a pair are encompassed by the same place, which also tends to indicate that they are likely non-duplicates. There are some cases, however, of sibling places which are indeed duplicates, thus we expect the siblings feature to have a small impact after all.

Figure 12 – Structure of the one-hot vectors utilized to represent the `parent_place_match`, `siblings_match`, and `homepage_matches` features in our ensemble classifiers.

Both Missing	One Missing	Different Values	Same Value
0/1	0/1	0/1	0/1

**Source:** (M. R. Cousseau, 2020)

Furthermore, the model also relies on a homepage comparison represented as a one-hot vector feature (`homepage_matches`), which indicates whether or not the two places in a pair share the same normalized host name in their homepage URL. The homepage comparison feature, as noticed by us, presents some cases where places sharing the same homepage URL are not duplicates, but instead places of a same chain store. This lessens the feature’s discriminative power.

Summing up, each place pair is represented as a row vector  $v \in \mathbb{R}^{18}$ , which is then processed by a Random Forest model in the first implementation of the duplicate classifier. Random Forests are initially chosen as they tend to excel at handling skewed data and finding a good bias-variance trade-off without the need for feature scaling. The model itself returns a probability of a pair being a duplicate  $s \in [0, 1]$ , and we experimentally define a classification threshold  $t_c$  to dictate which results would be classified as duplicates, with  $s \geq t_c$ , and which would be classified as non-duplicates, with  $s < t_c$ . We name this model Pairwise Random Forest (PRF).

As we also have to solve the record linkage problem in an on-line fashion, we explore more lightweight alternatives in terms of performance and memory usage for our Random Forest model. We then train a LightGBM (LGBM) model (KE et al., 2017) with the same features described above, as it offers higher efficiency with lower memory usage whilst being capable of handling large-scale data. We refer to this model as Pairwise LGBM (PLGBM) in the following sections, and further discuss how using it instead of a Random Forest in a real-time constrained environment benefits our classification performance and memory footprint in Section 5.4.

### 5.3 DUPLICATE CLUSTERING

The results produced by our classification models are in the form of tuples of duplicate places  $\langle P_i, P_j \rangle$ . These results would be sufficient if there was at most one replication per place entity in a given database, however, acquiring places from different sources leads to an average replication factor that supersedes the former. As an example, if a database is built by sourcing the complete list of all Starbucks coffee shops in the US from  $p$  different providers, assuming that they are all correct, a single Starbucks Coffee shop would appear

as an entity  $p$  times.

Hence, the pairwise outputs produced by the detection of duplicate pairs are not enough for a full-blown record linkage system, as they alone do not provide client applications with all the necessary information to actuate on top of the database. For instance, we consider a sample data fusion system consuming pairwise outputs  $\langle A, B \rangle$  and  $\langle A, C \rangle$  in a sequential manner from classification models. This system chooses one or more records to delete and keeps one of them with the newly merged information. By first merging  $\langle A, B \rangle$ , thus deleting  $A$  from the database and keeping  $B$  with the new information, processing  $\langle A, C \rangle$  becomes infeasible since the record  $A$  no longer exists.

To relax our solution to tuples of arbitrary sizes, we first transform each tuple  $\langle P_i, P_j \rangle$  in the Classification Model's result set as connected nodes in an unweighted and undirected places graph, by using a their *id* fields as a representation. Then, we use a simple breadth-first search algorithm starting at each vertex to detect the graph's connected components.

A simple pruning step is then applied on top of the connected components, checking for degenerate cases where either a large connected component has been formed, or two members of the connected component belong to a manual blacklist of cases. A sample blacklisted case is a component where two places with a parentage relationship are present, due to some error in our models. In those cases, the component is discarded and logged to be verified by a human.

The remaining components, in turn, represent tuples of replicated place records. Using the same values of the previous example, the places graph would have tuples  $\langle A, B \rangle$  and  $\langle A, C \rangle$  form a connected component  $\langle A, B, C \rangle$ , which would transmit all necessary information for the data fusion system to properly function.

## 5.4 APPLICATION PROGRAMMING INTERFACES

By connecting the Record Blocking, Classification Model, and Duplicate Clustering steps, our pipeline is able to solve the places record linkage problem in an off-line and batch-processing manner. However, clients of this pipeline may also need to be able to detect replicated place records before they even get inserted in the database itself.

The detection of duplicate places in an online fashion enables not only the prevention of a degraded database waiting for a batch job to run and deduplicate its records, but also helps researchers and engineers alike to preview results. Thus, it facilitates debug processes and improves knowledge about the Classification Model itself. On the other hand, it is not feasible to run our whole pipeline for every single deduplication request.

Ergo, to solve record linkage in real time whilst dealing with the issues that accompany it, we design a Representational state transfer (REST) API named Duplicate Detection API, which is responsible for providing a way to query a Classification Model in real time, accepting HTTP bodies according to the JSON format displayed at Listing A.1 in

Appendix A. We utilize the *Flask*<sup>2</sup> library with *gunicorn*<sup>3</sup> as a WSGI server to create the communication layer of our API.

A first implementation of this API utilizes *PRF*, ported to the real-time environment with *joblib*<sup>4</sup>. The API converts the place pair information into the required pairwise features, queries *PRF*, and then returns the binary result to the caller. The Soft TF-IDF values are stored in an in-memory Redis<sup>5</sup> database. The *PRF* model, however, has a memory footprint upwards of 1.8Gb, and causes some latency spikes due to a lack of concurrency in its predictions. It is then substituted by *PLGBM*, which offers out-of-the-box concurrency support and has a memory footprint of only 30Mb, reducing the observed latency spikes.

While this API provides real time duplicate places detection, abstracting our blocking steps for a real time environment is still necessary. Thus, we create a library named Places Lib which makes use of both the Duplicate Detection API and another previously existing API in our landscape, named Places API. This API provides a spatial search endpoint, described in Table 5.

Table 5 – Contract for the spatial search endpoint of our Places API.

Method	Endpoint	Query Parameter	Description
GET	/places/nearby	lat	Latitude from which to draw radius.
		lng	Longitude from which to draw radius.
		radius	Search radius in meters.
		limit	Limit of places to return.

**Source:** (M. R. Cousseau, 2020)

The Places Lib receives a single place entity as input, and attempts to generate possible pairs with it by a best-effort approach: it queries the spatial search endpoint from Places API, creates all possible pairs, and then filters them by applying the same threshold filtering step from our Record Blocking. Finally, the Duplicate Detection API is queried with a user-chosen classification threshold. The Duplicate Detection API may also be queried as a stand-alone application, and it does not account for clusters of replicated places, as the off-line pipeline is responsible for batch detection.

Finally, manual inspection of our model’s results and other related works (YALAVARTHI; KE; KHAN, 2017; YANG et al., 2019) reveal that including human quality assurance

<sup>2</sup> <https://flask.palletsprojects.com/en/1.1.x/>

<sup>3</sup> <https://gunicorn.org>

<sup>4</sup> <https://joblib.readthedocs.io/en/latest/>

<sup>5</sup> <https://redis.io>

Table 6 – Simplified contract of our Places Playground API.

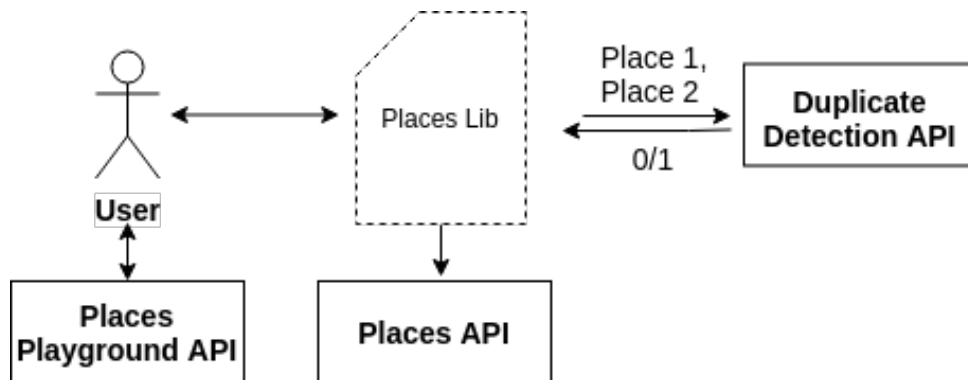
Method	Endpoint	Body Value	Description
POST	/duplicate_candidates	first_place_id	Id of the first place in the candidate pair.
		second_place_id	Id of the second place in the candidate pair.
	/duplicate_candidates/id/vote	vote_type	Value among <i>positive</i> , <i>negative</i> or <i>neutral</i> .

**Source:** (M. R. Cousseau, 2020)

(QA) input in the machine learning loop could help us improve the quality of our ground truth. We then create another API, named Places Playground API, to support this. As shown in Table 6, this API receives input in the form of votes for a given place pair, where a positive vote indicates duplicates and a negative vote indicates non-duplicates, with a skip (neutral) option also being available. An additional threshold  $t_\beta$  is defined for our models, and place pairs classified with a score  $s, t_\beta \leq s < t_c$  are sent up for voting.

We display a final diagram of the APIs in our ecosystem in Figure 13:

Figure 13 – The APIs involved in our pipeline to solve record linkage in real time.



**Source:** (M. R. Cousseau, 2020)

## 6 DEEP NEURAL NETWORK FOR CLASSIFYING PLACE PAIRS

Deep learning approaches have obtained state-of-the-art results for matching (HU et al., 2014; GUO et al., 2016; XIONG; ZHONG; SOCHER, 2016) and recognition tasks (SUN; WANG; TANG, 2015; TAIGMAN et al., 2014) in different data domains, ranging from facial images to textual *corpora*.

Drawing inspiration from these findings, we devise a Classification Model based on a deep neural network. The model captures multi-level information for each place using intermediate mappings and non-linearities, and then compares these representations in order to predict whether a place pair is a duplicate or not. It is crucial to highlight the fact that duplicate cases are uncommon, and thus handling class imbalance gracefully is important for the classifier’s performance.

The deep neural network architecture is laid out in Figure 14. For each place pair, the network receives its names, addresses, category lists, and geographical information in the form of latitude and longitude pairs. Then, to capture different views of the input, each place is processed by four different Encoders that transform its original representation into ones more suitable for the classification task. More specifically, word and character encoders are applied on each place’s textual fields (name and address), a category encoder on the category list, and a geographical encoder on the Haversine distance<sup>1</sup> calculated from the places’ latitude and longitude.

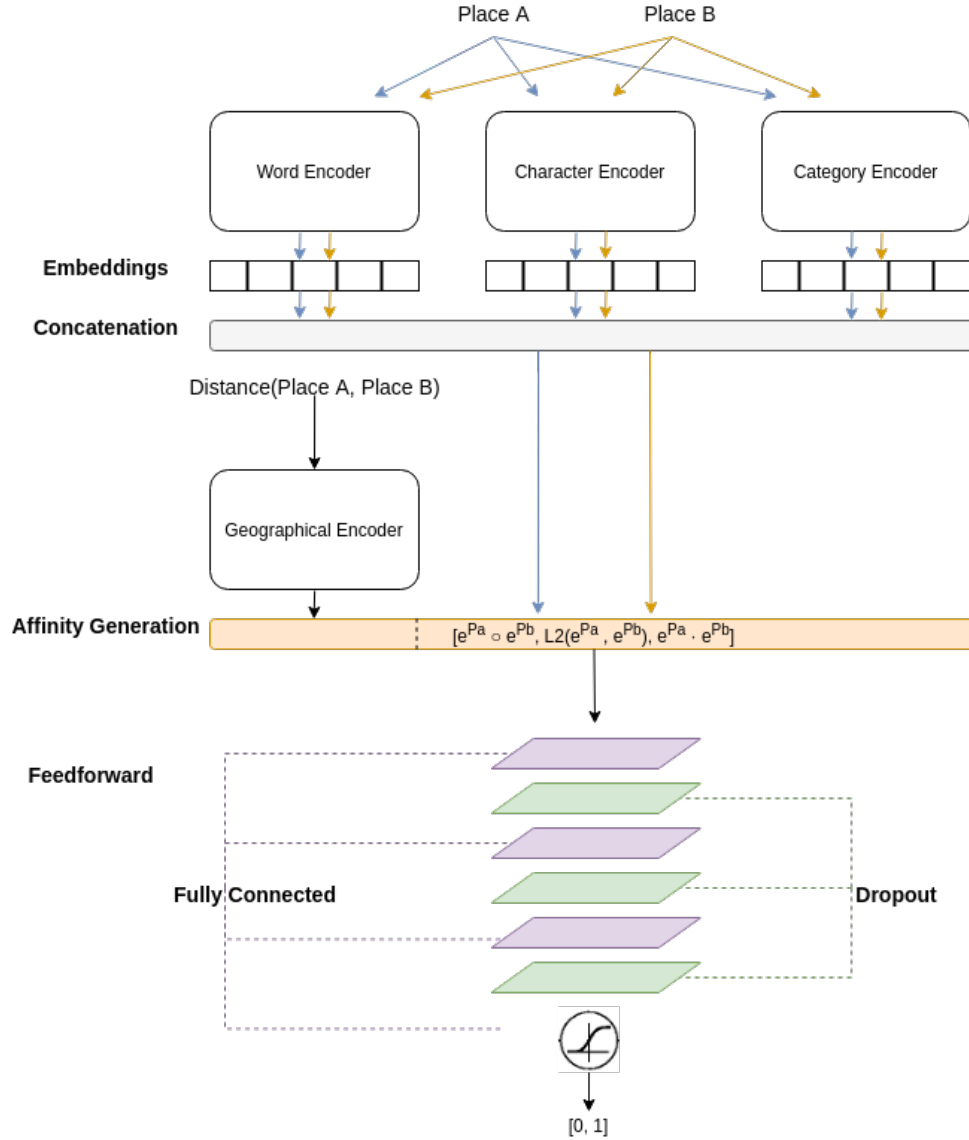
Next, the different representations produced by the Encoders for each place, with the exception of the geographical one, are concatenated and compared against each other in an Affinity Generation step in order to generate similarity values using different metric strategies. The resulting merged tensor is then concatenated with the geographical distance embedding generated by the geographical encoder, and passed through a feed-forward network with dropout regularization between each pair of layers. Finally, the final tensor is processed by a sigmoid-activated layer with a single neuron, generating a binary output.

In the sections that follow, we explain each encoder of the proposed architecture in more detail, alongside the final Affinity Generation steps. We also shed some light on failed attempts to improve our model, so as to achieve a better explanation of our thought process and assist other researchers. Value descriptions of all hyperparameters are left for Chapter 8, aiming to improve readability.

---

<sup>1</sup> <[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine\\_distances.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html)>

Figure 14 – Diagram of our deep neural network architecture for classifying place pairs.



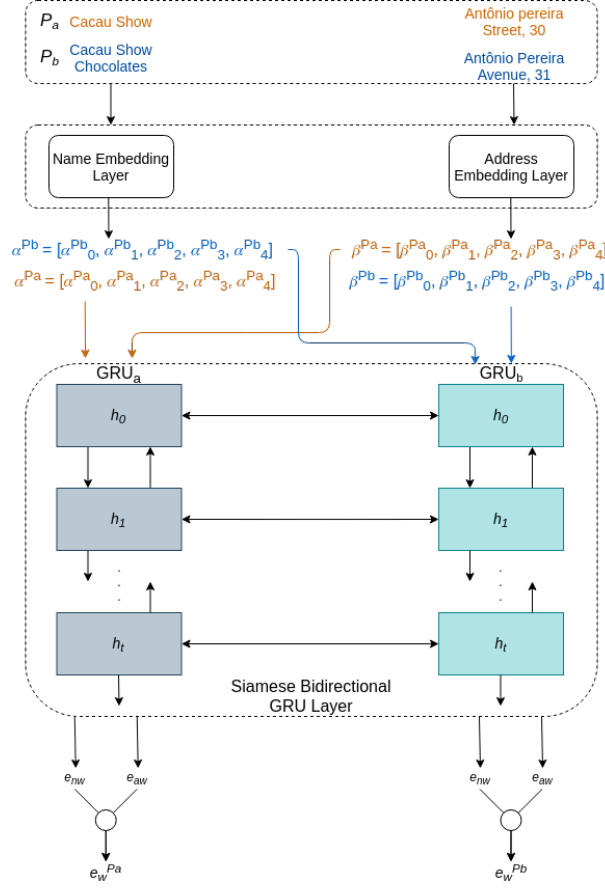
Source: (M. R. Cousseau, 2020)

## 6.1 WORD ENCODER

The word level representation encoder depicted in Figure 15 attempts to capture features of words in each place’s name and address, producing a word level place embedding  $e_w$ . By doing so, it is expected that words that appear in similar contexts, like *Cacau* and *Chocolates* in the example, reside closely in the embedded space, thus pulling both places in a pair closer.

Following Figure 15, we first transform all fields of places  $P_a$  and  $P_b$  into indexed padded sequences of size  $S^w$ . They are then transmitted to the word encoder, that feeds these sequences into separate embedding layers for names and addresses, taking advantage of transfer learning by being initialized with pre-trained embeddings:

Figure 15 – Word encoder processing a pair of places  $P_a$  and  $P_b$ , producing intermediate embedding sequences  $A^{P_i}$  and  $B^{P_i}$  where  $i \in \{a, b\}$ , and output embeddings  $e_w^{P_a}$  and  $e_w^{P_b}$ .



Source: (M. R. Cousseau, 2020)

- For **name embeddings**, a case-insensitive skip-gram model was trained on top of the *corpus* of all tokenized place names in our dataset;
- For **address embeddings**, we utilized case-sensitive embeddings trained with the FastText model on top of a Wikipedia and Common Crawl *corpus*<sup>2</sup>.

The reasoning behind the different embedding sources for names and addresses is that place names may contain unique words, not seen much in other *corpora* because they are relevant in establishing an identity for the place. Meanwhile, the presence of these unique words in addresses is unusual. This led us to believe and validate that training an address embedding on top of our database would not yield better results while imposing an additional computation step.

After producing sequences of embeddings  $\alpha^{P_i}$  and  $\beta^{P_i}$  - with every embedding vector  $\{\alpha, \beta\}_j^{P_i} \in \mathbb{R}^m$  - for places  $P_a$  and  $P_b$  respectively, the encoder passes them through a siamese bidirectional GRU (CHO et al., 2014) layer. This means that our network contains

<sup>2</sup> <https://fasttext.cc/docs/en/crawl-vectors.html>



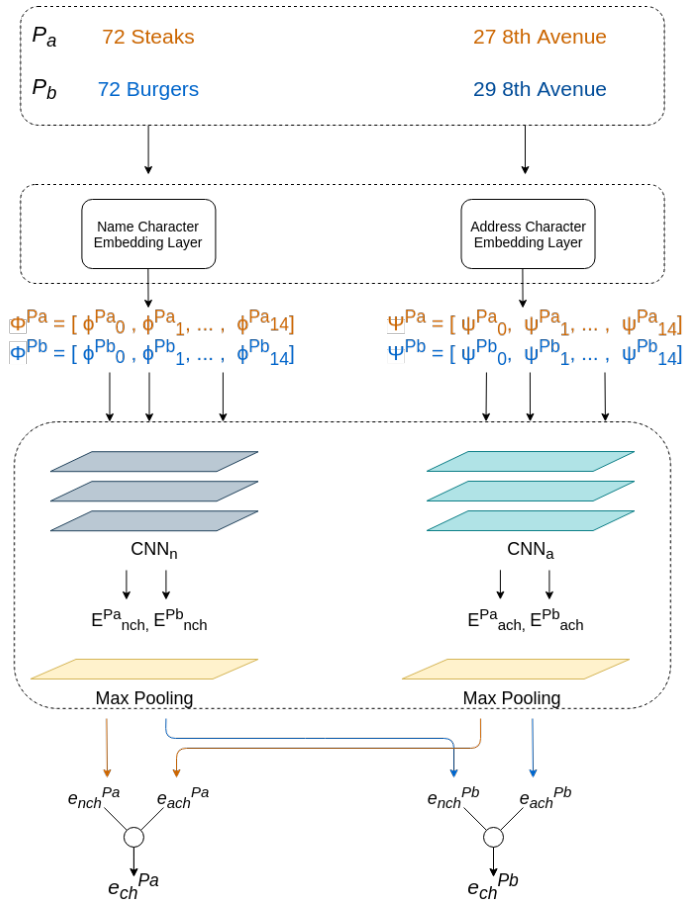
two GRUs:  $GRU_a$  and  $GRU_b$ , with shared weights and parameters. Embedding sequences for place  $P_i$  are passed through  $GRU_i$  both in order and in reverse order, and the concatenated hidden states from the last GRU cells for each ordering are used as output.

Thus,  $GRU_i(\alpha^{P_i})$  produces name word embeddings  $e_{n_w}^{P_i} \in \mathbb{R}^k$  and  $GRU_i(\beta^{P_i})$  produces address word embeddings  $e_{a_w}^{P_i} \in \mathbb{R}^k$ , where  $k = 2m$ . Finally,  $e_{\{a_w\}}^{P_i}$  and  $e_{\{n_w\}}^{P_i}$  are concatenated to form the place word level embedding  $e_w^{P_i}$ .

## 6.2 CHARACTER ENCODER

While our word level encoder attempts to build an embedded space in which places with similar words in their names or addresses are close to each other, it does not account for the fact that contextual word similarity does not always translate to duplicate places. For instance, if trained with a reasonable amount duplicate place samples whose names contain the contextually similar words *Steaks* and *Burgers*, the word encoder could consider the non-duplicate places named *72 Steaks* and *72 Burgers* as duplicates. *Burgers* and *Steaks*, however, are very different in a character level view.

Figure 16 – Our character encoder consuming a pair of places  $P_a$  and  $P_b$ , and producing embeddings  $e_{ch}^{P_a}$ .



Source: (M. R. Cousseau, 2020)

Figure 16 depicts our character level encoder, which aims to build representations for each character and then use those embeddings to build a character-based embedding  $e_{ch}$  for each place. Given places  $P_a$  and  $P_b$ , their names and addresses are first transformed into indexed padded character sequences of size  $S_n^{ch}$  for names and  $S_a^{ch}$  for addresses.

Akin to the word encoder, we build two separate embedding layers for name and address characters. Both embedding layers are initialized with random weights, which lets the character encoder receive each place pair’s indexed sequences as input.

Afterwards, the sequence of name and address character embeddings,  $\Phi^{P_i}$  and  $\Psi^{P_i}$  - with every embedding vector  $\{\phi, \psi\}_j^{P_i} \in \mathbb{R}^m$  - for places  $P_a$  and  $P_b$  respectively -, are each passed through distinct 1D-CNNs, with the same kernel size  $k_{cnn}$  and filters  $F \in \mathbb{R}^{k_{cnn} \times f_{cnn}}$ , but different random weight initialization.

For the sake of simplicity, we name the 1D-CNN which processes name characters as  $CNN_n$  and the 1D-CNN which processes address characters as  $CNN_a$ . When used with textual data, the 1-dimensional convolution operations performed by 1D-CNNs may be interpreted as window-based feature extractors, where the relationship between sub-sequences of words inside a sequence are analyzed instead of single words.

Since the sequence length is one of the main bottlenecks in RNNs and character sequences are naturally longer than their word counterparts, while having a lower number of dimensions, 1D-CNNs also present better time efficiency as character feature extractors. Furthermore, 1D-CNNs have proved to be efficient in dealing with sequences of noisy data (WANG et al., 2018; KIRANYAZ et al., 2019).

Consequently,  $CNN_n(\Phi^{P_i}) = E_{n_{ch}}^{P_i}$ , with  $E_{n_{ch}}^{P_i} \in \mathbb{R}^{(S_n^{ch}-k_{cnn}+1) \times f_{cnn}}$ , and  $CNN_n(\Psi^{P_i}) = E_{a_{ch}}^{P_i}$ , with  $E_{a_{ch}}^{P_i} \in \mathbb{R}^{(S_a^{ch}-k_{cnn}+1) \times f_{cnn}}$ . The internal representations produced by the 1D-CNNs are then passed through a global max pooling layer which generates the name character embeddings  $e_{n_{ch}}^{P_i}$  and address character embeddings  $e_{a_{ch}}^{P_i}$  for each place, with  $e_{\{n_{ch}, a_{ch}\}}^{P_i} \in \mathbb{R}^{f_{cnn}}$ . As a final step, we perform concatenations  $e_{n_{ch}}^{P_i} \circ e_{a_{ch}}^{P_i}, i \in \{a, b\}$  to generate our final character embeddings  $e_{ch}^{P_a}$  and  $e_{ch}^{P_b}$ , with  $e_{ch}^{P_i} \in \mathbb{R}^{2f_{cnn}}$ .

### 6.3 CATEGORY ENCODER

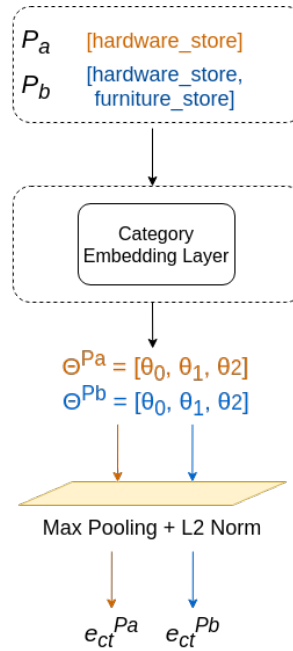
Our category encoder deals directly with the category attribute, which is a list of values pertaining to a fixed set of place categories. Its goal is generating a category level embedding  $e_{ct}$  for each place, following the intuition that places belonging to the same or similar categories have a higher change of being duplicates.

The importance of a category level representation in the deduplication context resides in the fact that even though places may present their category in their own names or addresses, e.g. *John’s Bar*, both the word and character encoders can be incapable of capturing the necessary latent patterns required in order to discern between places of different categories. Moreover, some places do not expose their category directly via their

names, e.g. *McDonald's*, and some, such as a sunglass chain store named *Chilli Beans*, may be misleading.

Category information, however, is not a standalone way of detecting duplicate places, because despite there being some duplicate place pairs which have the same or similar categories, there are also, by definition, non-duplicate pairs under the same situation, such as two different bars or restaurants. Moreover, since our category information is mostly gathered by a focused Web crawler and processed by multi-label classification model, it is prone to be imprecise. Thus, representing places in a category level should be seen only as a way to reinforce certain patterns and detect a limited set of features.

Figure 17 – Our category level module, which produces embeddings  $e_{ct}^{P_a}$  and  $e_{ct}^{P_b}$ .



**Source:** (M. R. Cousseau, 2020)

As Figure 17 shows, the categories of input places  $P_a$  and  $P_b$  are first transformed into index sequences of a fixed size  $S^{ct}$ . These sequences are then consumed by an embedding layer, producing embedding sequences  $\Theta^{P_a}$  and  $\Theta^{P_b}$ .

The embedding layer is initialized before training begins by using the pre-trained word embeddings for place names mentioned in Section 6.1, as it provides better initial results than using random initialization. More specifically, for each word in a category name, their embeddings are summed and L2-normalized to create an initial embedding space in which categories with contextually similar words are close. For instance, given the category *Hardware Store*, we construct an initial embedding  $\theta_{hardware\_store} = L2(pre_{hardware} + pre_{store})$ ,  $\theta_{hardware\_store} \in \mathbb{R}^m$ , where  $pre$  are the pre-trained word embeddings. We highlight that the aforementioned step is performed *a priori*, and does not depend on our network being trained beforehand.

After being generated, the embedding sequences  $\Theta^{P_a}$  and  $\Theta^{P_b}$  undergo a global max pooling operation, followed by an L2 normalization. This pipeline results in embeddings  $e_{ct}^{P_a} \in \mathbb{R}^m$  and  $e_{ct}^{P_b} \in \mathbb{R}^m$ .

## 6.4 GEOGRAPHICAL ENCODER

The last encoder in our architecture is the geographical level one, whose goal is generating an embedding  $e_g^{<P_i, P_j>}$  for each pair of places, which encapsulates the properties pertaining to the geographical distance between them. It operates on top of the Haversine distance<sup>3</sup> between the two places, which is already a pairwise metric. Because of that, the encoder’s output is only consumed right before the metric generation steps which will be described in Section 6.5.

Possible alternatives for inserting geographical context into a deep network include incorporating raw latitude and longitude pairs as features (WANG et al., 2018; YANG et al., 2019), or transporting latitude and longitude pairs to an embedded space which simulates the non-Euclidian distance calculation in the real world. While the former has been proven to have a negligible effect on deep networks due to their small dimensionality (YANG et al., 2019), the latter is troublesome due to the lack of any kind of sequential or additional spatial relationships in our use case. Furthermore, as Jiang et al. (2017) show, discretizing real values with a good subdivision strategy is sufficient for a network to capture the latent patterns of said value without losing information

Adding to that, in the place deduplication task, duplicate places usually do not have the same latitude and longitude information due to the errors described in Chapter 3, and places closer to each other should usually have a higher chance of being duplicates than places which are far apart. Thus, their surroundings should not matter as much. Take for instance the sample places *John’s Bar* and *Peete’s Restaurant*, which are both present in similar neighborhoods but are a few kilometers apart from each other. By taking both of their geographical contexts into account when building an embedded space, it is possible that both neighborhoods would be close to each other due to their similarity in the real world. This could in turn increase the likelihood of the two places being considered duplicates when comparing the embedding vectors, even with the places being a few kilometers apart.

So as to discretize the distances between places, we firstly calculate the maximum distance amongst all place pairs in the data set, which is in fact limited by our blocking strategy. By trivially assuming a minimum distance of 0 meters, we then create  $B$  distance buckets by equal-width discretization. Each bucket then represents an interval of distances  $[b_t, b_{t+1}[$ , and an embedding layer is randomly initialized for each of those buckets. Finally, for a pair of places  $P_a$  and  $P_b$ , their haversine distance is calculated, they are attributed

<sup>3</sup> <[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine\\_distances.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html)>

to the respective bucket, and then the geographical encoder outputs a final embedding  $e_g^{<P_a, P_b>} \in \mathbb{R}^m$ .

This embedding layer is trained alongside the other encoders in the network, leading to embeddings which hold the relationship between the distance of two places and they being duplicates or not. That is, for each distance bucket, its embedding will be able to indicate how likely is a place pair inside a given distance bucket a duplication.

## 6.5 AFFINITY GENERATION

With all of the embeddings  $e_w^{\{P_a, P_b\}}$ ,  $e_{ch}^{\{P_a, P_b\}}$ ,  $e_{ct}^{\{P_a, P_b\}}$ , and  $e_g^{<P_a, P_b>}$  produced by our encoders for a pair of places  $P_a$  and  $P_b$ , our deep neural architecture then proceeds to perform comparisons in order to learn a similarity metric between the two places. Before comparing the embeddings, however, the network performs a merging operation on the embeddings pertaining to a single place - namely every one but the geographical - to produce an appropriate embedding for each place. Hence, we form the place embeddings  $e^{P_i} = e_w^{P_i} \circ e_{ch}^{P_i} \circ e_{ct}^{P_i}$ , with  $e^{P_i} \in \mathbb{R}^{3m+2f_{cnn}}$  and  $i \in \{a, b\}$ .

Comparisons between the place embeddings are then performed by a series of operations in a merge layer:

- A concatenation of the two embeddings,  $e^{P_a} \circ e^{P_b}$ ;
- An element-wise L2 distance between the two embeddings,  $(e^{P_a} - e^{P_b})^2$ ;
- An element-wise multiplication of the two embeddings,  $e^{P_a} \cdot e^{P_b}$ .

These results are then concatenated into a tensor  $v_{me}^{<P_a, P_b>} = e^{P_a} \circ e^{P_b} \circ (e^{P_a} - e^{P_b})^2 \circ (e^{P_a} \cdot e^{P_b})$ , with  $v_{me}^{<P_a, P_b>} \in \mathbb{R}^{12m+8f_{cnn}}$ . Subsequently, the geographical distance embedding  $e_g^{<P_a, P_b>}$  is concatenated with the merge result since it already represents a merge operation itself, resulting in the final merged tensor  $v_{meg}^{<P_a, P_b>} = v_{me}^{<P_a, P_b>} \circ e_g^{<P_a, P_b>}$ , with  $v_{meg}^{<P_a, P_b>} \in \mathbb{R}^l, l = 13m + 8f_{cnn}$ .

Afterwards, the  $v_{meg}^{<P_a, P_b>}$  tensor is passed through a feed-forward network of  $D$  fully connected layers using the ReLU activation function (NAIR; HINTON, 2010). The first of these layers has  $H_{d_0}$  neurons, and each of the following  $d_i$  ones have  $H_{d_i} = \frac{H_{d_0}}{2^i}$  neurons, where  $i \in [0, |D|]$ . For ease of explanation, we name the results from each of the dense layers as  $v_{d_i}^{<P_a, P_b>}$ .

It is relevant to notice that each of those layers produce an increasingly compact representation of the learned similarity metric. Also, to reduce overfitting, we add Dropout regularization with a fixed ratio between each pair of fully connected layers in the feed forward network, as this regularization mechanism has proved to be quick and efficient (SRIVASTAVA et al., 2014).

Finally, the resulting tensor from the last dense layer in the feed forward network,  $v_{d_D}^{<P_a, P_b>}$ , is sent to a single neuron layer with a sigmoid activation function, that produces a binary classification result according to the following equation:

$$o^{<P_a, P_b>} = \begin{cases} 1.0 & \text{if } \sigma(W_l \cdot v_{d_D}^{<P_a, P_b>} + b) \geq t_c \\ 0.0 & \text{if } \sigma(W_l \cdot v_{d_D}^{<P_a, P_b>} + b) < t_c \end{cases} \quad (6.1)$$

Where  $W_l \in \mathbb{R}^{1 \times \frac{H_{d_0}}{2^{|D|}}}$  is the layer's weight matrix,  $b$  is the layer's bias, and  $t_c$  is the decision threshold. The sigmoid function outputs the probability of two places matching, and the decision threshold may be tuned according to some quality metric. A final output value of 1 means that the input pair is a duplicate, and a value of 0, otherwise.

## 6.6 MODEL TRAINING

To train the model while accounting for the expected high imbalance factor in the record linkage task, we utilize the Focal Loss training function (LIN et al., 2017), minimizing it. This function expands the traditional binary cross-entropy function by adding a modulating factor with a tunable focusing parameter  $\gamma$ . It leads the model to focus on samples which are presenting more error than the others, and reduces the contribution brought by easier, more abundant samples. Mathematically, the focal loss of a sample labeled as  $y \in \{0, 1\}$ , with prediction  $o$ , is defined by:

$$\text{FL}(o, y) = \begin{cases} -(1 - o)^\gamma \log(o) & \text{if } y = 1 \\ -(1 + o)^\gamma \log(1 - o) & \text{if } y = 0 \end{cases} \quad (6.2)$$

The class-balanced focal loss based on the effective number of samples, proposed by Cui et al. (2019), was also analyzed as an option, but failed to improve results during initial experiments. Furthermore, the contrastive loss proposed by Yang et al. (2019) depends on having two representations for places available, instead of a single pairwise one. Similarly, their proposed attentive training loss requires the data set to have labels from multiple sources as well as information on these sources for all entities, something which ours did not.

## 6.7 FAILED ATTEMPTS IN OUR DEEP NEURAL NETWORK

Throughout the development of our Classification Model, we attempted several approaches to build and improve our results that were met with failure. Thus, this section describes said failures, in order to help researchers facing similar issues. Even so, given that an exhaustive exploration of all of them could divert this work's focus, we highlight only the ones which had the most impact on our research.

The first failed attempt to highlight is using publicly available FastText embeddings (MIKOLOV et al., 2018) for place names. We expected these pre-trained embeddings to be a good match since they were trained on a Wikipedia and Common Crawl text *corpus*, and also expected its out-of-vocabulary token generation feature to account for missing tokens in its *corpus*. However, due to the unique nature of place names, the pre-trained embeddings were shadowed by the skip-gram model trained on our own *corpora*.

Next, we attempted different architectures of siamese GRU networks for the word encoder: separate GRUs for names and addresses, max pooling over GRU hidden states, and stacked GRUs with skip connections, similar to those in the work of Santos et al. (2017). The first of these decreased performance, with manual investigation showing that names and addresses shared a sufficient amount of words and patterns such that preventing a GRU from receiving data from both fields actually decreased its learning capacity. Max pooling over the hidden states also failed to improve our results, mainly due to the short length of place names and addresses. Finally, the increase in model complexity brought by stacked GRUs did not proportionally increase our representational power, which was a trigger for increasing variance.

Another attempt to improve our word level encoder was using the concept of co-attention, or attention over attention, following the intuition that it could be a natural extension of the concept of core words for places. More specifically, we adapted our word encoder to use the implementation from Xiong, Zhong and Socher (2016), and given context tensors  $c_n$  for names and  $c_a$  for addresses, the encoder produced a tensor  $v'_{meg}^{<P_a, P_b>} = v_{meg}^{<P_a, P_b>} \circ c_n \circ c_a$ . However, this implementation failed to improve our metrics and, in fact, lowered them, while slowing down the process of training the network and adding more complexity to the code. Marinho (2018) also reports little to no improvements brought by using attention in the similar problem of toponym matching.

Siamese GRU networks were the first attempt at implementing the character encoder, the main difference between them and those present in the word encoder being the average length of sequences, which is higher for characters. This imposed a bottleneck in the network, increasing training times tenfold, and did not improve the evaluation metrics. The main suspicion behind this resides in the higher convergence time needed by the character-level GRUs, which is cut short by early stopping criteria during training due to the other encoders converging faster. Thus, 1D-CNNs were used as an alternative, as described in Section 6.2.

Finally, albeit shown in section 6.4 that other alternatives to incorporate geographical context into the deep network do not fit our use case as nicely as the implemented one, first attempts to implement the geographical encoder included building an embedded space where similar regions are close to each other, and expanding raw latitude and longitude coordinates. The former was done by grid subdivision with the H3 system and building embeddings for each grid, but failed to improve our results due to the region embeddings

not translating to the duplicate detection case, and due to insufficient data in some grids. The latter, which followed the implementation of Wang et al. (2018), also failed to provide good results, mainly due to the original work using sequences with lengths upwards of 50, while ours had a fixed size of 2.



## 7 EXPERIMENTAL DATA PREPARATION

This work contributes to the state-of-the-art by both providing an end-to-end pipeline for performing record linkage of place entities and by proposing a deep learning model for classifying place pairs as duplicates or non-duplicates, which is the central issue in our record linkage pipeline. Thus, our experimental analysis had to be performed both on the pipeline’s efficiency and the classification model’s performance.

While evaluation of the pipeline itself does not require data preparation steps, because it consumes the whole data set available, measuring the performance of our deep neural network depends on having one or more ground truth sets of labelled place pairs. Adding this to the fact that our proposed solution for pairwise classification is our work’s main contribution among all other pipeline steps, this Chapter is dedicated to the explanation of our deduplication ground truth building process.

We first provide details on our places data set, from which the ground truth sets are created, in Section 7.1. This creation process is expounded in Section 7.2. Furthermore, we also search for external labelled data sets for us to perform additional experiments on, however, Section 7.3 describes how none of them are deemed adequate to our use case and, thus, lead us to solely utilize our own ground truth sets.

### 7.1 DATA ANALYSIS

The database utilized in this work is a snapshot, created *circa* 2018 from a places database which was being utilized in a production environment with upwards of 700 million daily queries. This database, referred to as  $Places_{DB}$  from here on, contains 28,467,486 place records from 227 different countries and annexed territories, with multiples languages and characteristics. It was created both by scraping structured data, such as *schema.org*<sup>1</sup>, from the web and by manual insertions from an internal company team. Table 7 presents an aggregation of number of records per country, as well as the total number of records in  $Places_{DB}$ .

As shown in Table 7, around 60.1% of all places in  $Places_{DB}$  are located either in the US or in Brazil, while remaining countries share small percentages of the database to fill the remaining 40%. Thus, we focused our efforts only on Brazilian and American places. Given that the Portuguese and English languages present different linguistic features and the record linkage task, by definition, does not need to consider records from two different countries, we further partition  $Places_{DB}$  into a data set of places from the US,  $Places_{US}$ , and a data set of places from Brazil,  $Places_{BR}$ .

---

<sup>1</sup> <<https://schema.org/>>

Table 7 – Places by country for the top 10 countries in *Places<sub>DB</sub>*.

Country	#records	Percentage
United States of America	12,320,561	43.28%
Brazil	4,788,454	16.82%
Japan	679,348	2.39%
Turkey	533,469	1.87%
Italy	345,437	1.21%
Russia	313,344	1.10%
Indonesia	295,057	1.03%
United Kingdom	263,541	0.93%
France	252,412	0.89%
Spain	251,291	0.88%
All	28,467,486	100%

**Source:** (M. R. COUSSEAU, 2020)

To verify that *Places<sub>BR</sub>* and *Places<sub>US</sub>* hold enough information for our pipeline to run on, we analyze the coverage of each place field for both sets, and display the results in Table 8 with the values of *Places<sub>DB</sub>* being added for the sake of comparison.

Table 8 – Place fields coverage in *Places<sub>US</sub>*, *Places<sub>BR</sub>*, and *Places<sub>DB</sub>*.

	<i>Places<sub>US</sub></i>	<i>Places<sub>BR</sub></i>	<i>Places<sub>DB</sub></i>
id	100%	100%	100%
name	100%	100%	100%
geo_location	100%	100%	100%
phone_number	91.01%	27.04%	50.21%
homepage	7.33%	2.60%	7.26%
categories	70.01%	67.83%	70.20%
parent_place_id	0.55%	1.07%	1.31%
address	97.50%	99.89%	86.54%

**Source:** (M. R. Cousseau, 2020)

As we may see from these results, the *id*, *name*, and *geo\_location* attributes present a coverage of 100% over all data sets, which happens trivially because these three are obligatory in our database. The fields which present the lowest coverage values for the

$Places_{DB}$  and  $Places_{BR}$  data sets are *parent\_place\_id*, *homepages*, and *phone\_number*, due to they being rarer to find in the Web and other sources.

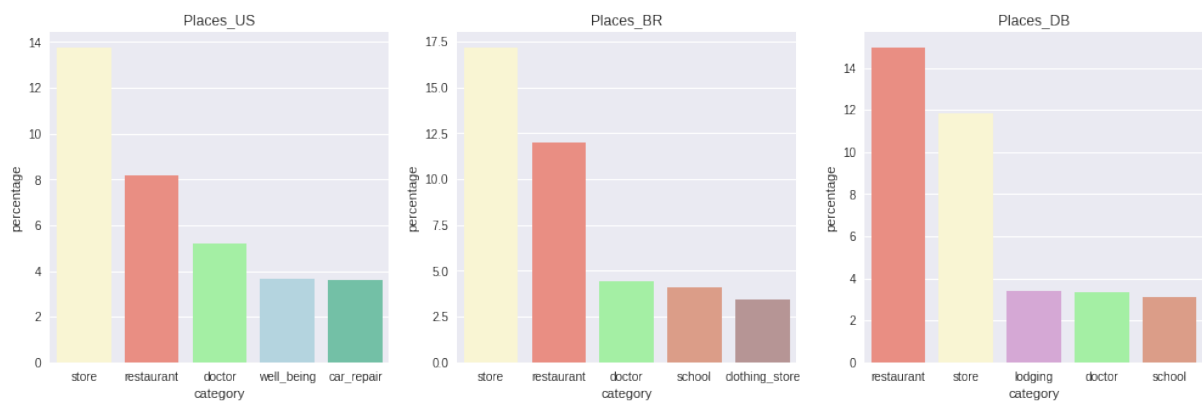
In the  $Places_{US}$  set, however, *phone\_number* coverage reaches 91.01%, which is more than three times the Brazilian set coverage, and almost double the global coverage. This indicates that sources for US places put more care into completeness than its counterparts. As later shown, we experience better results when training and evaluating our solution on top of the  $Places_{US}$  data set, and this completeness takes part in it since it is often accompanied by correctness in the provided data.

The low presence of parent places points to the fact that most places in the data sets are not encompassed by another one, which is an expected scenario in the real world. However, some of the missing data in this field may be attributed to failures in the algorithm that generated this association, as described in Chapter 3.

In a similar way, the *categories* field being covered in about 70% places of all sets implies that the algorithms utilized to classify each place into one or more categories had imperfections. This field receives a special treatment during this analysis because the *point\_of\_interest* category is a default value, thus we consider places with only that category as having no category at all.

Moreover, our set of 122 categories did not provide enough granularity for classifying all places into at least one category: categories for factories and corporate buildings, for instance, were not taken into account by the category list at the time of the snapshot creation.

Figure 18 – Place category distribution for the top five categories in  $Places_{US}$ ,  $Places_{BR}$ , and  $Places_{DB}$ .



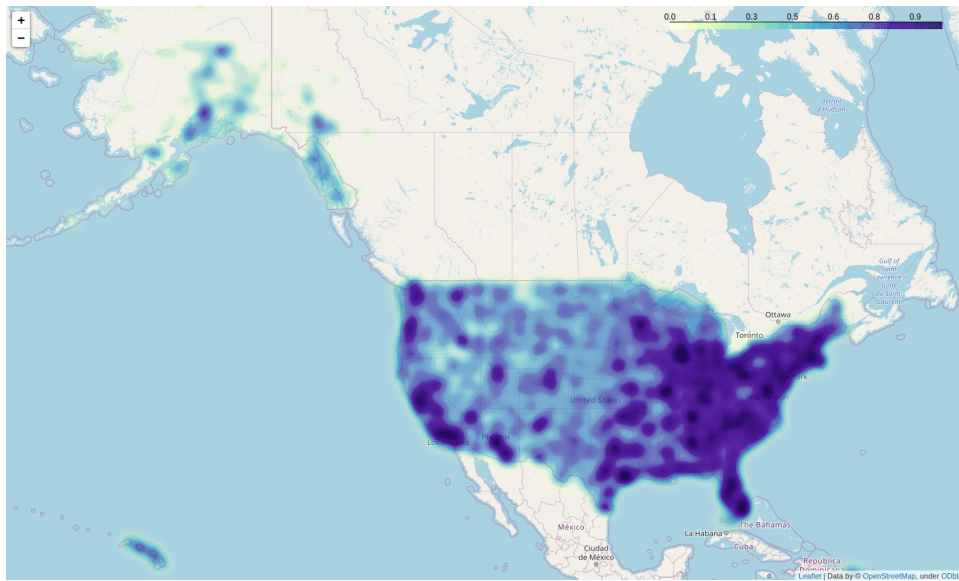
Source: (M. R. Cousseau, 2020)

Figure 18 displays the distribution of places on the top-5 categories in  $Places_{BR}$ ,  $Places_{US}$ , and  $Places_{DB}$ , excluding *point\_of\_interest*. The goal behind this analysis is asserting that the data sets are not being represented by just a small set of categories out of the 122 possible. Alas, the category distribution indicates that even though all data

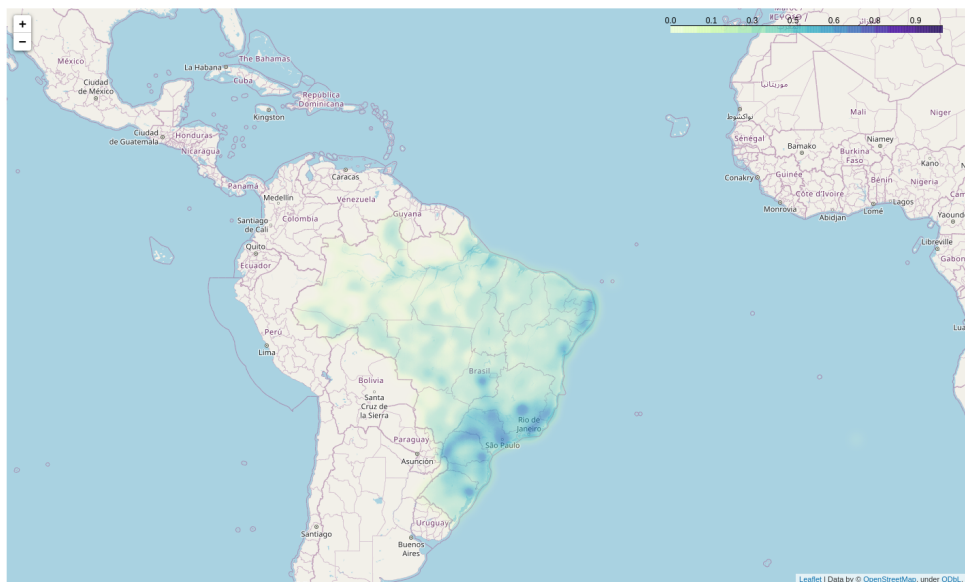
sets share a similar set of popular types of venues, the top two of them being *store* and *restaurant*, none of them represents a big share of places by itself.

Overall, the  $Places_{BR}$  and  $Places_{US}$  data sets present similar coverage values for each place field between themselves, with the exception of the *phone\_number* field. To ensure that the places in each set are not concentrated on a few specific spots, thus biasing algorithms and models on properties of a specific location, we also compute heat maps for places in both data sets, as Figure 19 shows.

Figure 19 – Heat maps of places in  $Places_{US}$  and  $Places_{BR}$ .



(a) Heat map for  $Places_{US}$ .



(b) Heat map for  $Places_{BR}$ .

**Source:** (M. R. Cousseau, 2020)

Apart from some specific areas, places in  $Places_{BR}$  and  $Places_{US}$  are well distributed

over each country, which means that they tend to follow the population distribution in the real world and cover all possible states. For instance, New York City has one of the highest density of firms in the US (BUREAU, 2020), while Alaska has a vast wilderness, both characteristics which show themselves in Figure 19a. Likewise, the Amazon rainforest is mostly depopulated and thus has a small number of places being shown in Figure 19b, but São Paulo and Rio de Janeiro, some of the most populated cities in Brazil, present a high density of places according to the heat map.

## 7.2 NOVEL GROUND TRUTH SETS

The first step towards building our ground truth is generating pairs from the places in our database. In order to do that, we utilize the same blocking algorithm described in Section 5.1 on top of  $Places_{US}$  and  $Places_{BR}$ , as it guarantees training and evaluating our model with the same data distribution as a production environment would present.

To select the  $\lambda$  Jaro Winkler similarity threshold used in the blocking step, we divide the place pairs from the  $Places_{BR}$  and  $Places_{US}$  sets into similarity buckets and manually inspect 100 random pairs from each of them. As Table 9 shows, a  $\lambda$  value between 0.8 and 0.9 is a good trade-off between cleaning up trivial non-duplicates and avoiding duplicates exclusion. We choose  $\lambda = 0.8$  to perform blocking, and this process results in an order of magnitude of  $10^7$  pairs for the  $Places_{US}$  data set, and  $10^6$  for the  $Places_{BR}$  one.

Table 9 – Percentage of duplicate pairs among the first 100 randomly-select pairs of each similarity bucket for our blocking algorithm.

Interval	Duplication
[0.5, 0.6[	0%
[0.5, 0.6[	0%
[0.6, 0.7[	0%
[0.7, 0.8[	0%
[0.8, 0.9[	4%
[0.9, 1.0]	28%

**Source:** (M. R. Cousseau, 2020)

Afterwards, we generate labels indicating duplication or non-duplication for each of the place pairs produced by the blocking algorithm. Albeit usually generating a higher quality of labels, manually labelling all of the generated samples is infeasible given our available resources.

We also considered using the MTurk<sup>2</sup> crowdsourcing marketplace to distribute the

<sup>2</sup> <https://www.mturk.com/>

labelling effort. However, the platform did not offer the necessary Portuguese language support at the time, and a previous work by Yalavarthi, Ke and Khan (2017) pointed out that aggregation processes had to be executed on top of the crowdsourced results to achieve a good label quality for harder tasks, such as the deduplication one.

In light of that, we utilize the *phone\_number* attribute of our database, present in 27.04% of places in *Places<sub>BR</sub>* and 91.01% of places in *Places<sub>US</sub>*, to generate labels for each pair, excluding it from every other step of our pipeline to avoid data leakage. This follows findings that most of the place pairs which share a phone number represent duplications in our data sets. Hence, place pairs with phone numbers are selected from each set, have their special digits such as “+” and “-” removed, and then are compared through exact match to produce a positive or negative label.

Next, we manually curate samples of these results, fixing certain erratic patterns such as different places inside a shopping mall or belonging to the same chain store having the same phone number. This generates the *Pairs<sub>US</sub>* and *Pairs<sub>BR</sub>* silver truth sets, whose characteristics are displayed in Table 10.

Table 10 – Characteristics of our *Pairs<sub>US</sub>* and *Pairs<sub>BR</sub>* silver truth sets.

Data set	#pairs	#places	#positive pairs	#negative pairs	Skew
<i>Pairs<sub>US</sub></i>	3,009,428	2,267,885	325,809	2,683,619	8.24
<i>Pairs<sub>BR</sub></i>	597,452	365,092	24,892	572,560	24.0

**Source:** (M. R. Cousseau, 2020)

Both sets preserve a low ratio of duplicates to non-duplicates. Also, we note that the *Pairs<sub>BR</sub>* set has almost three times the skew of the *Pairs<sub>US</sub>* one, meaning that the ratio of duplicates to non-duplicates in it is much higher. Since the exact ratio of duplicates in a database varies greatly over locations and over time, this difference in skew allows us to train and evaluate our models under different data quality scenarios. Moreover, the general differences in both sets make our evaluation stronger and broader.

### 7.3 INADEQUACIES IN EXTERNAL GROUND TRUTHS

To expand our experiments and more aptly compare our work against others, we searched for additional ground truth sets. We first looked for sets with already consolidated and labelled place pairs used by previous works, as Table 11 shows. Each of them, however, posed some kind of issue which made it impractical or impossible to use, for instance: albeit being very similar to ours in terms of available attributes and class distribution, the data sets from Dalvi et al. (2014) and Yang et al. (2019) are both proprietary to Facebook, and the ground truth from Deng et al. (2019) is also not reproducible.

Table 11 – Characteristics and issues of external place pair data sets.

Data set	Attributes	#records	Publicly available	Issues
Dalvi et al. (2014)	names, latitude and longitude pairs	14,000	No	Small size and unavailable to the public.
Deng et al. (2019)	names, addresses, categories, latitude and longitude pairs	323	No	Small size and unavailable to the public.
Marinho (2018) - Historical	names, distance	2,024	Yes	Small size and all positive pairs have a distance of 0.
Marinho (2018) - JRC	names	800,000	Yes	Contains mostly transliterated records. Uses names only.
Marinho (2018) - Geonames	names	5,000,000	Yes	Uses names only, duplications are mostly transliterations or normalizations, artificial distances.
Yang et al. (2019)	names, addresses, categories, latitude and longitude pairs	4,600,000	No	Unavailable to the public.

**Source:** (M. R. Cousseau, 2020)

In his work, Marinho (2018) performed the related task of toponym (location names) matching in three different data sets: one built on top of data from Geonames<sup>3</sup>, a web database of locations around the world, another one adapted from data of Ehrmann, Jacquet and Steinberger (2016) in the European Commission’s Joint Research Centre (JRC), and a last one composed of historical places. The three datasets had 50% of their pairs being duplicates and the other half being non-duplicates, which represented an unreal imbalance factor in the real world, and all of them utilized a different concept of place entities than our work’s.

Each entry in the raw Geonames database contained a name for a location alongside alternative names for it, which are in most cases transliterated or normalized versions of the original name. Thus, place pairs were generated from it by using the available

<sup>3</sup> <http://www.geonames.org>

transliterations and alternative names for each record to generate positive labels, and by picking similar names from random different locations to generate negative pairs. This created positive samples with a distance of 0, and negative samples with artificially high distance values. Furthermore, since our pipeline was not built to handle transliterations, most positive labels would be filtered during blocking steps.

The JRC data set contained pairs of names for organizations. It fell under some of the same issues present in the Geonames data set, since it contained only place names and the majority of them were just transliterations. Finally, the Historical data set from the same author was extremely small in size, and all positive pairs in it had a distance of 0 meters.

Given the issues in the previous pairwise data sets, we looked for raw public places data sets for us to generate labelled pairs using some additional attribute, similarly to our process of generating the novel ground truth sets. Our search resulted in the place data sets described by Table 12, which did not enable any way of automatically generating duplicates and, in fact, did not present any duplicate records after manual inspection of their samples.

Table 12 – Characteristics and issues of external raw place data sets.

Data set	Attributes	#records	Issues
Yelp <sup>4</sup>	names, addresses, latitude and longitude pairs, categories, working hours	192,608	No automatic way to generate duplicates.
Cho, Meyers and Leskovec (2011) - Gowalla <sup>5</sup>	names, latitude and longitude pairs	120,997	No automatic way to generate duplicates.
Weeplaces <sup>6</sup>	latitude and longitude pairs	7,658,369	No place names: check-in data set.
Facebook Kaggle competition <sup>7</sup>	latitude and longitude	-	No place names: check-in data set.
Gazetteer <sup>8</sup>	names, addresses, latitude and longitude pairs	50,000	Different concept of place entities, no automatic way to generate duplicates.

**Source:** (M. R. Cousseau, 2020)

<sup>4</sup> <https://www.yelp.com/dataset>



Not having an automatic way to generate duplicate place pairs from these data sets meant that this process would have to be performed manually, which in turn was an endeavor that we did not have the necessary resources to partake in. Moreover, the data sets from Weeplaces and the Facebook Kaggle competition contained only latitude and longitude pairs for check-in events.

Due to all of the inadequacies in both pairwise and raw place data sets encountered during our research, we opted to use our  $Pairs_{BR}$  and  $Pairs_{US}$  to evaluate our models. Our experiments using these ground truth sets are described in detail at Chapter 8.

---

<sup>5</sup> <https://snap.stanford.edu/data/loc-gowalla.html>

<sup>6</sup> <https://www.yongliu.org/datasets/>

<sup>7</sup> <https://www.kaggle.com/c/facebook-v-predicting-check-ins/>

<sup>8</sup> <http://gazetteer.org.uk/purchase>

## 8 EXPERIMENTS

So as to assess the quality of the techniques presented in Chapters 5 and 6, each of them is evaluated. While the full record linkage pipeline is measured in regards to its execution time on top of the whole *Places<sub>DB</sub>*, this Chapter focuses mostly on the comparison of the proposed supervised Classification Model against our preliminary versions and baseline methods. The reasoning behind this follows our work’s main objective of contributing to the state of the art in the detection of duplicate place records to be linked.

Both of our silver truth sets are split into training, validation, and test sets, using a fixed split of 70%, 20%, and 10%, respectively, preserving the class distribution in each of them. Following recommendations from Ng (2019), we also sample some entries from the validation set and used them as our Eyeball validation set, which allows us to probe for error patterns in our models’ results so as to look for data insights.

In the following sections, we explain the chosen metrics and baseline methods, the setup for each of the models and algorithms being experimented, and implementation details. Finally, we present the results observed and discuss them.

### 8.1 RUNTIME ENVIRONMENT

The experiments for deep neural networks are executed in machine instances provided by Google Colab<sup>1</sup>, containing 13Gb RAM, an Intel Xeon CPU, and a mix of NVidia K80, T4, P4, and P100 GPUs. Meanwhile, the CPU-bound tests make use of an instance with 16Gb RAM, and an Intel Core i7-7500U CPU. In addition to that, the full pipeline experiments, as well as any steps requiring distributed computation, utilize 7 AWS m4.xlarge instances, each having 16Gb of memory and 4vCPUs.

### 8.2 PERFORMANCE METRICS

Since the problem of detecting duplicate places suffers from class imbalance, both generally and in the *Pairs<sub>BR</sub>* and *Pairs<sub>US</sub>* data sets, the chosen metrics need to account for that. By choosing a standard metric such as accuracy, for instance, any classifier considering all pairs as non-duplicates would achieve a score of 0.958 for the *Pairs<sub>BR</sub>* data set and 0.891 for the *Pairs<sub>US</sub>* one, which is misleading.

Ergo, each model is evaluated on the task of detecting place duplications according to two metrics. The first of them is the normalized Gini coefficient (GINI, 1912; DIXON et al., 1987; DAMGAARD; WEINER, 2000), for whom there are several interpretations and ways to compute. Computationally, it attempts to measure how far apart are its probabilistic results from random guessing.

---

<sup>1</sup> <https://colab.research.google.com>

It may do so by the following process: first, we sort the predictions  $p$  into ascending order, then this ordering is used to sort the labels  $y_i \in \{0, 1\}$ , generating a new label list  $y'$ . By counting the number of necessary swaps between adjacent labels in  $y'$  to transform  $y'$  back to  $y$ , then dividing by the number of swaps necessary to transform a random list of zeroes and ones into  $y$ , one reaches the normalized Gini coefficient. Section 8.2.1 offers another interpretation of it.

The second metric is the  $F_{\beta=0.5}$  score, also referred to as e-measure (RIJSBERGEN, 1979), described in Equation 8.1.

$$F_{\beta=0.5} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision}} = 1.25 \cdot \frac{\text{precision} \cdot \text{recall}}{0.25 \cdot \text{precision}} \quad (8.1)$$

Optimizations are performed in regards to the normalized Gini coefficient first, and then the probabilistic outputs are transformed into binary labels by applying a threshold  $t_c$ , chosen by optimization of the  $F_{\beta=0.5}$  score in a separate grid search. The tuned hyperparameters for each model and their best values for each data set are described by Tables 18, 19, and 20 in Appendix C.

We also provide the time consumed by each classification model during training and execution to highlight this positive aspect about simpler approaches, even if the metrics provided by them end up proving to be lower in general. Finally, the full proposed pipeline is evaluated by the means of aggregated execution time, split by each of its steps.

### 8.2.1 Gini Coefficient Interpretation

While the swap-counting explanation for the Gini coefficient is intuitive, it relies on a bubble sort operation that has a quadratic worst-case complexity, to count the number of necessary swaps. Thus, another way to compute the Gini coefficient is by implementing Equation 8.2, proposed by Damgaard and Weiner (2000):

$$G = \frac{2 \sum_{i=1}^n i y'_i}{n \sum_{i=1}^n y'_i} - \frac{n+1}{n} \quad (8.2)$$

where  $n$  is the number of samples being evaluated and  $y'$  is a list of labels sorted according to the predictions given to them in ascending order, like in the first approach. An implementation of this Equation in Python is provided in Appendix B.

### 8.2.2 F-score Interpretation

Even though the normalized Gini coefficient is well-suited for problems with class imbalance, it does not provide an intuitive explanation on how frequently a given model will classify a sample correctly. Furthermore, it requires probabilistic outputs to function correctly.

The  $F_{\beta=0.5}$  score tackles these issues, and may additionally be broken up into its precision and recall components to improve understandability. Although the  $F_{\beta=0.5}$  score

and its precision and recall components are not optimally suited for imbalanced problems, they provide a clearer indication on how the model performs in the case of positive and negative pairs.

The  $\beta$  coefficient for the F-score is set as 0.5 to account for the fact that precision, i.e. correctly detecting a duplication when given a true one, outweighs recall, i.e. detecting all possible duplications, in our use case. This stems from place pairs incorrectly classified as duplicates place leading to possibly unrecoverable states in a database.

### 8.3 SETUP FOR BASELINE METHODS

In these experiments, we utilize two works as baselines:

- (DALVI et al., 2014): this approach utilizes EM to build probability distributions for words of a place’s name, splitting them into core and background words. The core words are used to compare if two places are duplicates.
- (YANG et al., 2019): this work proposes an unsupervised learning step to generate place embeddings. Then, a Multi-layer Perceptron (MLP) is trained on top of the place embeddings, and is gradually improved with: a novel contrastive loss function (PE), batch-wise hard-sampling (PEH), source-attentive training (PEHA), and label denoising (PEHAD).

According to our literature review, these are the most prolific works dealing directly with places deduplication, and specifically with Web data. None of these works, however, provide implementations for their methods, so we implement our own versions.

The model from Dalvi et al. (2014) is implemented in Python 3 with the numpy<sup>2</sup> library, and the probability of a place pair representing duplicate places is calculated directly by the core word comparison probability derived in their work. The authors also propose a dynamic programming algorithm to take edit operations into account during comparisons, but the provided code seems to be non-functional.

Our implementation of Yang et al. (2019) is adapted from the PE model, outputting the euclidean distance between the two places in the metric space, whose complement is then interpreted as positive class probability. We implement PE instead of the full model, PEHAD, because the additional improvements were either missing details or did not fully fit our data set and use case. Furthermore, we contacted the authors regarding the parameters for the model parameters, who recommended a search on a given range of values for the negative sampling ratio in the smoothing step, among others. All steps are implemented with Keras 2.3.1 and Tensorflow 1.15.2, with the aid of gensim<sup>3</sup>

<sup>2</sup> <<https://numpy.org/doc/stable/index.html>>

<sup>3</sup> <<https://radimrehurek.com/gensim/>>

We utilize `optkeras`<sup>4</sup>, which is a wrapper over `optuna`<sup>5</sup>, to tune the parameters of PE, and a grid search for the model of Dalvi et al. (2014), both using validation data. The Expectation-maximization algorithm from Dalvi et al. (2014) is executed for 10 iterations in  $Pairs_{BR}$  and 5 iterations in  $Pairs_{US}$ , while PE runs for 100 `optuna` trials in  $Pairs_{BR}$  and 50 trials in  $Pairs_{US}$ , using a median pruner after 5 warm-up trials and 3 warm-up steps. Geohashes are utilized as a means for the creation of tiles in all methods, and all hyperparameters are described in Table 18.

## 8.4 SETUP FOR PRELIMINARY PROPOSED METHODS

Both  $PRF$  and  $PLGBM$  are evaluated on top of the split  $Pairs_{BR}$  and  $Pairs_{US}$  data sets in terms of their classification performance. The  $WRH$  algorithm is unable to be executed in the  $Pairs_{US}$  data set due to access to its code being lost.

In addition, the effect of resampling techniques on these models is studied in the scope of the  $Pairs_{BR}$  data set, by simultaneously applying Tomek Links (TOMEK, 1976) for undersampling non-duplicates too similar to positive cases, and SMOTE (CHAWLA et al., 2002) for oversampling duplicate pairs, resulting in a data set named  $Pairs_{BR}^R$ . This resampling is applied directly into the features, and not the place entities themselves.

The  $PRF$  model is implemented using the `scikit-learn` and `pandas` libraries for Python 3 (PEDREGOSA et al., 2011; TEAM, 2020), while  $PLGBM$  uses the `LightGBM` package for Python 3 (KE et al., 2017). Meanwhile, the  $WRH$  algorithm and the feature generation step for the other models are implemented in Scala<sup>6</sup> with `Apache Spark` for distributed computation (ZAHARIA et al., 2016).

While no time is spent automatically tuning  $WRH$  model due to its access being lost, both other preliminary models are tuned with `optuna` for 100 trials in the scope of the  $Pairs_{BR}$ ,  $Pairs_{US}$ , and  $Pairs_{BR}^R$  validation data sets. All of their hyperparameters are described in Table 19, with any parameter not listed assuming the default value provided by their respective libraries.

## 8.5 SETUP FOR CLASSIFICATION MODEL

To implement our Classification Model, we utilize the ReLu activation function (NAIR; HINTON, 2010) in the  $D = 3$  feed-forward network layers, the first of them having  $H_{d_0} = 256neurons$ . Glorot uniform initialization (GLOROT; BENGIO, 2010) is utilized for the character and geographical encoder embedding layers, while the embedding layers from other encoders are initialized with pre-trained embeddings, as mentioned in Chapter 6. For the CNN layers in the character encoder and the RNN layers in the word encoder, the

<sup>4</sup> <<https://github.com/Minyus/optkeras>>

<sup>5</sup> <<https://optuna.readthedocs.io/en/stable/index.html>>

<sup>6</sup> <<https://www.scala-lang.org/>>

hyperbolic tangent is utilized as an activation function. Finally, regarding the sequence lengths noted in Chapter 6,  $S^w = 5$ , and  $S^{ct} = 3$  for both sets,  $S_n^{ch} = 42$ ,  $S_a^{ch} = 32$  for  $Pairs_{BR}$ , and  $S_n^{ch} = 26$ ,  $S_a^{ch} = 23$  for  $Pairs_{US}$ . We use  $B = 100$  distance buckets in the geographical encoder.

The network is implemented with Keras 2.3.1 and Tensorflow 1.15.2, with the aid of gensim. To use the pre-trained FastText embeddings in our model, their dimensionality is reduced beforehand by means of a Principal Component Analysis (PCA) (PEARSON, 1901) dimensionality reduction script<sup>7</sup>. In order to improve reproducibility, we also fix Tensorflow’s, numpy and Keras random seeds as 6810818.

The model is tuned with optkeras, a wrapper over optuna, for 50 trials on top of the  $Pairs_{BR}$  validation data set and 20 trials in the  $Pairs_{US}$  one, with a median pruner after 3 warm-up trials and 1 warm-up step. An additional early stopping callback with a patience of 2 epochs and a minimum change of  $10^{-3}$  is also added as insurance against degenerate cases not detected by the median pruner. Training is performed with the ADAM optimizer (KINGMA; BA, 2015), with a batch size of 64 for  $Pairs_{BR}$  and 1024 for  $Pairs_{US}$ , and all hyperparameters and their values are described in Table 20.

## 8.6 RESULTS

The results of comparing each of our models with the baseline methods in the  $Pairs_{BR}$  and  $Pairs_{US}$  test data sets are displayed in Table 13. Figure 20 expands on that by displaying plots of the precision-recall curves for all possible models on both data sets.

Our Classification Model achieves the best values for both the  $F_{\beta=0.5}$  and the Normalized Gini scores in both data sets, also achieving the best precision in  $Pairs_{BR}$  and the best recall in  $Pairs_{US}$ . By comparing  $CM$  with  $PLGBM$ , which achieves the second-best results in both data sets in terms of  $F_{\beta=0.5}$ , we see relative gains upwards of 7.14% in the  $Pairs_{BR}$  data set and 5.61% in the  $Pairs_{US}$  data set.

Analyzing Figures 20a and 20b further shows that the Classification Model offers an improvement over its competitors in almost all precision-recall ratios, that is: it is able to be more thoroughly tailored to attend specific demands of precision or recall ratios. For instance, fixing precision at 0.8,  $PLGBM$  has a recall of less than 0.7 while  $CM$  reaches a value close to 0.8 in the  $Pairs_{US}$  data set. These results prove that  $CM$  is able to better capture the place features with its encoders, independently of the size of the training data set or its imbalance factor.

However, by analyzing the precision-recall curves, we see that  $PLGBM$  has equivalent metrics to  $CM$  at some points. Furthermore,  $PLGBM$  is able to constantly surpass the methods from Dalvi et al. (2014) and Yang et al. (2019) in all evaluated metrics. Summing these results with the simpler approach taken by  $PLGBM$ , they confirm the  $PLGBM$ ,

<sup>7</sup> <[https://github.com/facebookresearch/fastText/blob/master/reduce\\_model.py](https://github.com/facebookresearch/fastText/blob/master/reduce_model.py)>

Table 13 – Comparison of all models on test samples from  $Pairs_{BR}$  and  $Pairs_{US}$ .

Data set	Model	Precision	Recall	$F_{\beta=0.5}$	Normalized Gini
$Pairs_{BR}$	Dalvi et al. (2014)	0.300	0.240	0.285	0.601
	PE - Yang et al. (2019)	0.590	0.246	0.462	0.778
	$WRH$	0.560	0.06	0.210	-
	$PRF$	0.498	<b>0.581</b>	0.513	0.915
	$PLGBM$	0.616	0.412	0.560	0.924
	$CM$	<b>0.617</b>	0.542	<b>0.600</b>	<b>0.929</b>
$Pairs_{US}$	Dalvi et al. (2014)	0.750	0.240	0.526	0.693
	PE - Yang et al. (2019)	0.837	0.547	0.757	0.896
	$PRF$	<b>0.844</b>	0.543	0.760	0.930
	$PLGBM$	0.801	0.652	0.766	0.938
	$CM$	0.837	<b>0.712</b>	<b>0.809</b>	<b>0.959</b>

**Source:** (M. R. Cousseau, 2020)

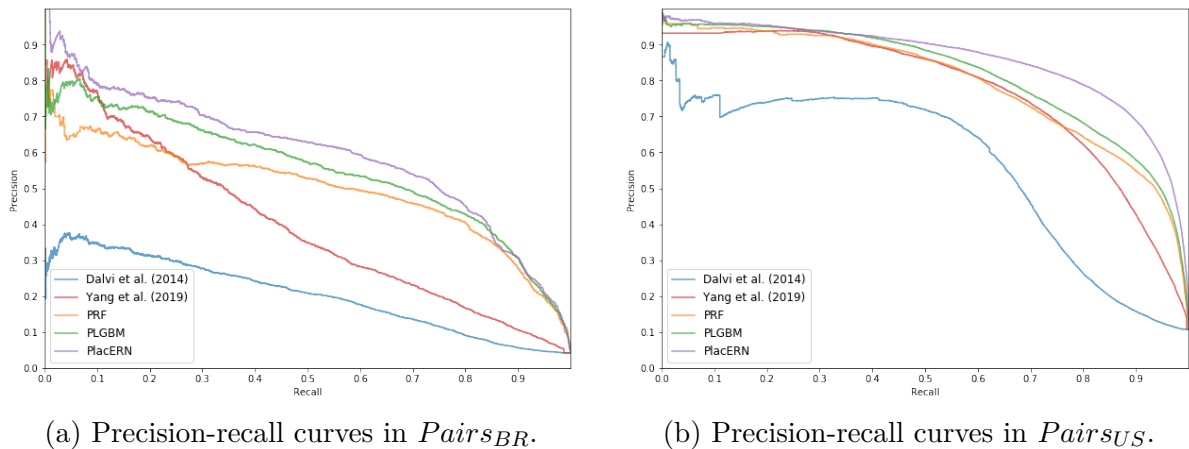
while being an overall worse choice than  $CM$  for the record linkage task, may be a good alternative for a quick and competitive solution for the problem.

Regarding  $PRF$ , Table 13 and Figure 20a show that it lags behind  $PLGBM$  in  $Pairs_{BR}$  by a considerable margin. For the  $Pairs_{US}$  data set, however,  $PRF$  provides a performance closer to that of  $PLGBM$  in all evaluated metrics. This points to the fact that  $PLGBM$  is better able to handle class imbalance with fewer data points than a traditional Random Forest model.

We also notice that  $WRH$  is surpassed in almost all metrics by the other methods, due to its simplistic and constrained approach. More specifically, while it provides a precision value of 0.56, which is close to the best result in the data set, its recall rate is only 0.06, proving that in order to achieve a relatively good precision, it must adhere to an unreasonably low recall value.

Finally, Table 13 shows that both baseline methods achieve lower  $F_{\beta=0.5}$  and normalized Gini coefficient values when compared to our preliminary models and the final Classification Model itself, with PE surpassing the heuristic approach of Dalvi et al. (2014)

Figure 20 – Overlapping precision-recall curves in  $Pairs_{BR}$  and  $Pairs_{US}$  for all possible models.



Source: (M. R. Cousseau, 2020)

in every evaluated metric. According to the precision-recall curves for the Brazilian data set, the PE model from Yang et al. (2019) has consistently lower metrics than all of our proposed solutions. Namely, we observe a difference of 0.151 in the normalized Gini coefficient when comparing  $CM$ , the best-performing model, with PE in the  $Pairs_{BR}$  data set, and 0.146 when comparing  $PLGBM$  to PE in the same set. Comparing  $F_{\beta=0.5}$  further exacerbate this pattern.

Meanwhile, the normalized Gini coefficient achieved by PE in the  $Pairs_{US}$  set is only 0.063 points below the best performer’s value, with its  $F_{\beta=0.5}$  score being 0.052 lower. Given that its performance on  $Pairs_{BR}$  was lackluster, this may indicate that the PE model, without the additional modifications proposed by its authors, is unable to handle class imbalance as well as the other models, and relies on a bigger data set to learn. We acknowledge, however, that the full PEHAD model proposed by Yang et al. (2019) displays a significant improvement over PE in their own work, so perhaps a full-blown implementation could have similar or even better results than our best performer in this data set.

### 8.6.1 Ablation Study

In order to verify the impact of each encoder in the Classification Model, we gradually remove each of them from the final architecture: the category encoder ( $CTE$ ) is removed first, followed by the geographical one ( $GE$ ), then the character encoders for both names and addresses are removed ( $CHE$ ). Thus, the basic model is composed only of the word encoder ( $WE$ ). Each of these models has its parameters optimized. The results for this ablation study in all data sets are presented in Table 14.

From the results, we see that the full Classification Model,  $CM$ , achieves the best



Table 14 – Results of the ablation study for the Classification Model.

Data set	Model	Precision	Recall	$F_{\beta=0.5}$	Normalized Gini
$Pairs_{BR}$	$WE$	0.577	0.508	0.562	0.9
	$WE + CHE$	0.588	<b>0.573</b>	0.585	0.92
	$WE + CHE + GE$	0.6	0.557	0.591	0.921
	$CM$	<b>0.617</b>	0.542	<b>0.600</b>	<b>0.929</b>
$Pairs_{US}$	$WE$	0.822	0.694	0.793	0.946
	$WE + CHE$	0.826	0.678	0.791	0.95
	$WE + CHE + GE$	<b>0.849</b>	0.668	0.805	0.956
	$CM$	0.837	<b>0.712</b>	<b>0.809</b>	<b>0.959</b>

Source: (M. R. Cousseau, 2020)

$F_{\beta=0.5}$  and normalized Gini scores in both data sets. Each encoder adds up to the model’s performance, making the full model present an increase of 0.038 in the  $F_{\beta=0.5}$  score and 0.029 in the normalized Gini coefficient when compared to  $WE$  only, in the Brazilian set.

The most impact brought by the addition of an encoder may be seen in the  $WE + CHE$  version, where adding the character encoder boosts the  $F_{\beta=0.5}$  by 0.023. These results are even less noticeable in the  $Pairs_{US}$  set, due to the larger amount of training data. The fact that none of the encoders display a considerable impact on the score by itself points to the fact that the word encoder is a very powerful classification tool by itself, and each of the additions to the model serves to improve it in some difficult cases.

### 8.6.2 On The Effects of Resampling

Additional experiments are conducted to study the effect of resampling in the  $PRF$  and  $PLGBM$  models, with the goal of discovering if applying such techniques could improve our models. As Table 15 shows, resampling both lowers the normalized Gini coefficient and the  $F_{\beta=0.5}$  score for the  $PLGBM$  model by 0.012 and 0.032, respectively, and lowers the normalized Gini coefficient by 0.003 while increasing the  $F_{\beta=0.5}$  score by 0.005 for the  $PRF$  model.

The impact brought by resampling thus ranges from minor trade-offs in terms of

Table 15 – Comparison of *PRF* and *PLGBM* in regards to resampling.

Model	Data set	Precision	Recall	$F_{\beta=0.5}$	Normalized Gini	Training Time (s)
<i>PRF</i>	<i>Pairs<sub>BR</sub></i>	0.498	<b>0.581</b>	0.513	<b>0.915</b>	<b>27.23</b>
	<i>Pairs<sub>BR</sub><sup>R</sup></i>	<b>0.517</b>	0.522	<b>0.518</b>	0.912	103.44
<i>PLGBM</i>	<i>Pairs<sub>BR</sub></i>	<b>0.616</b>	0.412	<b>0.561</b>	<b>0.924</b>	<b>7.93</b>
	<i>Pairs<sub>BR</sub><sup>R</sup></i>	0.545	<b>0.473</b>	0.529	0.912	8.5

**Source:** (M. R. Cousseau, 2020)

metrics to all-around worse results. Adding to that, training the *PRF* models on *Pairs<sub>BR</sub><sup>R</sup>* took 76.21 more seconds when compared to the *Pairs<sub>BR</sub>* version. While this 3.8 times increase is not very significant in this work’s data sets, it could impact larger data sets in a more relevant way. Based on these results, no resampling techniques are used in other experiments.

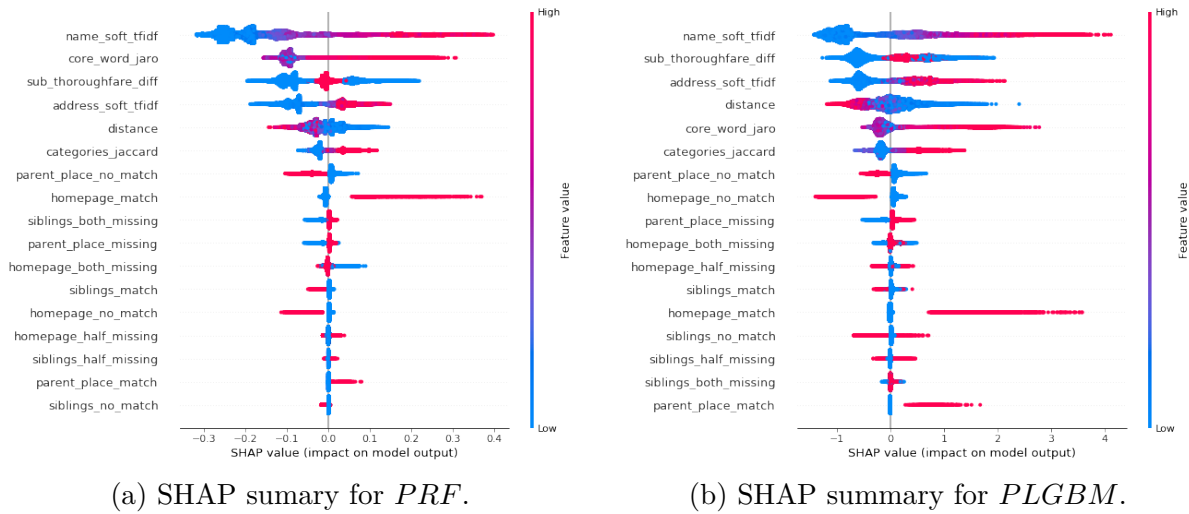
### 8.6.3 Shapley Additive Explanations

Shapley Additive Explanations (SHAP) (LUNDBERG; LEE, 2017) are conducted on the feature set for the *PRF* and *PLGBM* models, to infer the importance of each place attribute and transfer this knowledge to the development of other models such as *CM*. SHAP is a game-theoretic approach which assigns an importance value to each feature for a particular prediction, and is able to aggregate those importance values to produce a general analysis on the effect of each feature in the model.

Figure 21 presents the SHAP summary plots for each of the features in the *PRF* (a) and *PLGBM* (b) models in the scope of validation data from *Pairs<sub>BR</sub>*. Each dot represents a sample, and a higher absolute SHAP value indicates that a feature has more influence on the model classification, with dots in red representing higher feature values and dots in blue representing otherwise. In the Figure, features are ordered from top to bottom according to the sum of their absolute SHAP values for all samples.

Results show that the name\_soft\_tfidf, sub\_thoroughfare\_diff, address\_soft\_tfidf, and core\_word\_jaro features have the most overall impact on the models. These textual features are correlated with positive outputs, i.e. higher values for the features imply duplicate pairs for the model. The distance feature, as expected, is inversely correlated with positive samples, i.e. lower values push the model towards non-duplicates. The core\_word\_jaro feature provides relevant SHAP values for positive samples, but is shown to be unreliable for negative samples, due to the mixture of colored dots closer to negative SHAP values. These patterns indicate that the most important information to detect re-

Figure 21 – Summary plots of SHAP values for *PRF* and *PLGBM* in *Pairs<sub>BR</sub>*.



Source: (M. R. Cousseau, 2020)

plicated places are present in their names and addresses, with the distance between them being a good additive.

Furthermore, one is able to assert that the categorical features are either unreliable descriptors by themselves, as is the case with `homepage_both_missing`, or are reliable descriptors of a single class, such as `parent_place__match`. In all cases, the magnitude of their influence on the model is much lower than their counterparts'. The analysis of the SHAP values leads us to focus the development of other models on place names, addresses and geographical coordinates, which is the approach taken in our Classification Model.

#### 8.6.4 Training and Execution Times

Since the end goal of the models for classifying place pairs into duplicates or non-duplicates is being part of a full record linkage pipeline, we analyze the training and execution times for each model in Table 16, without accounting for pre-processing or post-processing steps. Analyzing training time is relevant to our use case because place databases are often dynamic, so recurrent training guarantees the quality of results over time. In a similar way, execution time matters for real-time use cases.

Whilst the Classification Model is the best performer in both data sets in terms of classification scores, it takes 3149 seconds to train in the *Pairs<sub>BR</sub>* data set. When compared to *PLGBM*, for instance, its training time is roughly 397 times higher.

Similarly, the training times of the model from Dalvi et al. (2014) and the PE model stand out as being at least one order of magnitude higher than the other models', especially in the *Pairs<sub>US</sub>* data set, where they reach upwards of 36 hours to train. It is relevant to note that most of the training time for PE comes from the embedding smoothing process, which relies on the construction of a places graph. This process was taken into account

Table 16 – Training and execution times for all models.

Data set	Model	Full training time (s)	Execution time (s)
<i>Pairs<sub>BR</sub></i>	<i>WRH</i>	-	3600.00
	<i>PRF</i>	27.23	0.31
	<i>PLGBM</i>	<b>7.93</b>	<b>0.19</b>
	<i>CM</i>	3149	80.83
	Dalvi et al. (2014)	7879.00	4.83
	PE - Yang et al. (2019)	8598.00	2.97
<i>Pairs<sub>US</sub></i>	<i>PRF</i>	364.26	1.56
	<i>PLGBM</i>	<b>32.05</b>	<b>1.28</b>
	<i>CM</i>	542.00	232.42
	Dalvi et al. (2014)	131610.00	25.44
	PE - Yang et al. (2019)	130742.00	9.40

**Source:** (M. R. Cousseau, 2020)

because, in a real scenario, these smoothed embeddings for each place would need to be re-generated to account for new entities and temporal changes in past ones. While training time does not disqualify a model for usage in a production environment, any researcher or engineer aiming to use it would have to dispense more resources to do so.

In regards to execution time, *PLGBM* takes only 0.19 seconds to classify the 60925 test samples from *Pairs<sub>BR</sub>* and 1.28 seconds to classify the 307017 test samples from *Pairs<sub>US</sub>*, surpassing every other evaluated model. More noticeably, *CM* takes longer than *PLGBM*, *PRF*, and each baseline method to execute, without using batch predictions. This points to the fact that *CM* is more suited to a batch-wise or off-line case than a real-time, latency-constrained one. These results also do not account for concurrency in real-time services, which may present itself as a bottleneck. In that regard, the LGBM library offers out-of-the-box concurrency and distribution support.

Finally, the execution times in seconds for the record blocking and duplicate clustering steps of the record linkage pipeline, averaged over 5 runs in the full *Places<sub>DB</sub>* and without considering the classification model utilized, are shown in Table 17. Although we do not offer comparisons of these run times to other alternatives, they serve to show that the full pipeline using any of our proposed models is able to be executed in less than 3 hours for a database with up to 28 million records, attesting to applicability to production environments.

Table 17 – Execution times for the record linkage pipeline, averaged over 5 runs.

	Record Blocking (s)	Duplicate Clustering (s)	Total (s)
Mean	7301.80	2617.20	9919.00
Std. Dev.	187.44	150.00	163.88

**Source:** (M. R. Cousseau, 2020)

## 8.7 DISCUSSION

By considering the results presented in the previous section, one can conclude that our linkage pipeline for place records using a multi-view classifier, *CM*, is able to successfully detect replicated places in large quantities of data in a reasonable time. This is sustained by experiments which show that: (i) *CM* outperforms our preliminary heuristic and supervised learning approaches in detecting duplicate places on all tested data sets; (ii) *CM* surpasses baseline approaches from other authors in the state-of-the-art in terms of all evaluated metrics; (iii) The full linkage pipeline is able to process up to 28 million place records in under 3 hours; (iv) *PLGBM* also obtains better results than the baseline methods, and is extremely fast to train and execute, proving itself as a good alternative for solving the linkage problem in an on-line fashion with the proposed set of API's.

Our results also show that resampling does not influence supervised learning techniques so much in the places record linkage task, even with a high imbalance factor. This points towards the fact that the models utilized by us are able to inherently deal with imbalance during their training process.

Furthermore, the experiments attest that the place's name and address information are the two most important fields to extract relevant information from, when attempting to detect duplicate places. Latitude and longitude values also serve as a good way to improve results. These findings are particularly fortuitous, because these are the most common attributes present in places data from the Web.

In regards to the architecture of *CM*, we find that the word encoder for names and addresses is already a powerful duplicate detection tool by itself. The additional encoders further increase the model's performance by small increments, making it easier to detect edge cases. Given the impact which classification errors in the record linkage task may cause in a database, each of these increments is a relevant addition.

## 9 CONCLUSIONS AND FUTURE WORK

This chapter concludes this work by presenting a summary of the obtained results, as well as its main contributions and limitations. Furthermore, some direction is given in terms of future works that may improve or extend this one.

### 9.1 CONCLUSION

This work presented an end-to-end pipeline for linking place records using a record blocking step, a deep neural network to detect duplicate places, and a duplicate clustering step. The network utilizes four different encoders to capture multi-level information about places: a word encoder, a character encoder, a category encoder, and a geographical encoder. Each of these encoders learns a representation for place pairs, which are then aggregated and compared to generate an affinity between two places, indicating whether they are duplicates or not. Other supervised learning approaches in the form of a pairwise Random Forest model and a pairwise LGBM classifier, alongside a set of APIs to expose them, were offered as alternatives for real-time environments.

Experimentation on these models and the full pipeline, on top of two data sets with varying characteristics, indicated that our final solution was able to outperform previous attempts and baseline methods from the state-of-the-art on the detection of duplicate places. Likewise, we showed that our preliminary models also surpassed other competitors in this task, and are able to be ported to an on-line environment with latency constraints. In addition to that, the full pipeline is also shown to handle large quantities of data in a reasonable amount of time.

### 9.2 MAIN CONTRIBUTIONS AND LIMITATIONS

We consider the main contributions of this work to be:

1. A novel classifier using multiple encoders is proposed for the task of detecting duplicate places;
2. This classifier outperforms previous attempts and baseline methods on the same task;
3. Preliminary supervised learning approaches to the problem also surpass baseline methods, and have better training and execution times;
4. Record blocking and duplicate clustering steps are proposed, composing an end-to-end solution for the record linkage problem in the places data domain;

5. A set of APIs is described in order to solve the linkage problem in an on-line fashion.

In terms of limitations, we highlight the fact that the the final classification model does not seem suitable for on-line environments, with the preliminary approaches tending to be better. Furthermore, the record blocking step drops some relevant duplicate cases, and could be improved.

### 9.3 FUTURE WORK

As possible improvements or extensions to this work, we suggest the following:

- Exploring the usage of additional encoders for other place attributes such as their parentage information;
- Analyzing better similarity metrics for the record blocking step;
- Utilizing reinforcement learning to improve the proposed model, and make it less susceptible to temporal changes in the data;
- Exploring self-attention in the context of the word and character encoders.

## REFERENCES

- BAEZA-YATES, R.; RIBEIRO-NETO, B. et al. *Modern information retrieval*. [S.l.]: ACM press New York, 1999.
- BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.
- BARBOSA, L. Learning representations of web entities for entity resolution. *International Journal of Web Information Systems*, Emerald, Dec 2018. Available at: <<http://dx.doi.org/10.1108/IJWIS-07-2018-0059>>.
- BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, USA, v. 35, n. 8, p. 1798–1828, Aug. 2013. ISSN 0162-8828. Available at: <<https://doi.org/10.1109/TPAMI.2013.50>>.
- BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, IEEE Press, v. 5, n. 2, p. 157–166, Mar. 1994. ISSN 1045-9227. Available at: <<https://doi.org/10.1109/72.279181>>.
- BERJAWI, B. *Integration of Heterogeneous Data from Multiple Location-Based Services Providers: a Use Case on Tourist Points of Interest*. Phd Thesis (PhD Thesis) — Ecole doctorale d’informatique et mathématique de Lyon, 09 2017.
- BOJANOWSKI, P.; GRAVE, E.; JOULIN, A.; MIKOLOV, T. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, v. 5, p. 135–146, 2017. ISSN 2307-387X.
- BRODSKY, I. *H3: Uber’s Hexagonal Hierarchical Spatial Index*. 2018. <<https://eng.uber.com/h3/>>. [Online; accessed 3-June-2020].
- BUREAU, U. S. C. *2017 SUSB Annual Data Tables by Establishment Industry*. 2020. <<https://www.census.gov/data/tables/2017/econ/susb/2017-susb-annual.html>>. [Online; accessed 7-June-2020].
- CENTER, A. R. *What is geocoding?* 2010. <<http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/002500000001000000.htm>>. [Online; accessed 7-June-2020].
- CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 16, n. 1, p. 321–357, Jun. 2002. ISSN 1076-9757.
- CHO, K.; MERRIËNBOER, B. van; GULCEHRE, C.; BAHADANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1724–1734. Available at: <<https://www.aclweb.org/anthology/D14-1179>>.



- CHRISTEN, P. A comparison of personal name matching: Techniques and practical issues. in *The Second International Workshop on Mining Complex Data (MCD'06)*, 12 2006.
- CHRISTEN, P. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, v. 24, 01 2011.
- COHEN, W.; RAVIKUMAR, P.; FIENBERG, S. A comparison of string metrics for matching names and records. *Proc of the KDD Workshop on Data Cleaning and Object Consolidation*, 10 2003.
- COLLOBERT, R.; WESTON, J.; BOTTOU, L.; KARLEN, M.; KAVUKCUOGLU, K.; KUKSA, P. P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, v. 12, p. 2493–2537, 2011.
- CONTRIBUTOR, S. *Valve sobel*. 2008. <[https://commons.wikimedia.org/wiki/File:Valve\\_sobel\\_\(3\).PNG](https://commons.wikimedia.org/wiki/File:Valve_sobel_(3).PNG)>. [Online; accessed 26-July-2020].
- COUSSEAU, V.; BARBOSA, L. Industrial paper: Large-scale record linkage of web-based place entities. In: *Anais Principais do XXXIV Simpósio Brasileiro de Banco de Dados*. Porto Alegre, RS, Brasil: SBC, 2019. p. 181–186. ISSN 0000-0000. Available at: <<https://sol.sbc.org.br/index.php/sbbd/article/view/8820>>.
- CUI, Y.; JIA, M.; LIN, T.-Y.; SONG, Y.; BELONGIE, S. Class-balanced loss based on effective number of samples. In: . [S.l.: s.n.], 2019. p. 9260–9269.
- DALVI, N.; OLTEANU, M.; RAGHAVAN, M.; BOHANNON, P. Deduplicating a places database. In: *Proceedings of the 23rd international conference on World wide web - WWW 14*. ACM Press, 2014. Available at: <<https://doi.org/10.1145/2566486.2568034>>.
- DAMGAARD, C.; WEINER, J. Describing inequality in plant size or fecundity. *Ecology*, v. 81, p. 1139–1142, 04 2000.
- DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In: . [S.l.: s.n.], 2004. v. 51, p. 137–150.
- DEMPSTER, A. P. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 38, n. 2, p. 325–339, 1967. ISSN 00034851. Available at: <<http://www.jstor.org/stable/2239146>>.
- DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, v. 39, n. 1, p. 1–38, 1977.
- DENG; LUO; LIU; WANG. Point of interest matching between different geospatial datasets. *ISPRS International Journal of Geo-Information*, MDPI AG, v. 8, n. 10, p. 435, Oct 2019. ISSN 2220-9964. Available at: <<http://dx.doi.org/10.3390/ijgi8100435>>.
- DIXON, P. M.; WEINER, J.; MITCHELL-OLDS, T.; WOODLEY, R. Bootstrapping the gini coefficient of inequality. *Ecology*, v. 68, p. 1548–1551, 1987.

DUNN, H. L. Record linkage. *American Journal of Public Health and the Nations Health*, v. 36, n. 12, p. 1412–1416, 1946. PMID: 18016455. Available at: <<https://doi.org/10.2105/AJPH.36.12.1412>>.

EHRMANN, M.; JACQUET, G.; STEINBERGER, R. Jrc-names: Multilingual entity name variants and titles as linked data. *Semantic Web*, v. 8, p. 1–13, 04 2016.

FACTUAL. *Factual Places*. 2020. <<https://www.factual.com/products/factual-places/>>. [Online; accessed 3-June-2020].

FELLEGI, I. P.; SUNTER, A. B. A theory for record linkage. *Journal of the American Statistical Association*, Taylor & Francis, v. 64, n. 328, p. 1183–1210, 1969. Available at: <<https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049>>.

FOURSQUARE. *How do I add/create a place?* 2020. <<https://support.foursquare.com/hc/en-us/articles/201065050-How-do-I-add-create-a-place->>. [Online; accessed 8-June-2020].

GINI, C. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [Fasc. I.]*. Tipogr. di P. Cuppini, 1912. (Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari). Available at: <<https://books.google.com.br/books?id=fqjaBPMxB9kC>>.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, v. 9, p. 249–256, 01 2010.

GUO, J.; FAN, Y.; AI, Q.; CROFT, W. B. A deep relevance matching model for ad-hoc retrieval. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016. Available at: <<http://dx.doi.org/10.1145/2983323.2983769>>.

GUO, X.; GAO, L.; LIU, X.; YIN, J. Improved deep embedded clustering with local structure preservation. In: . [S.l.: s.n.], 2017.

HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, Ieee, v. 21, n. 9, p. 1263–1284, 2009.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 18, n. 7, p. 1527–1554, Jul. 2006. ISSN 0899-7667. Available at: <<https://doi.org/10.1162/neco.2006.18.7.1527>>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, Nov. 1997. ISSN 0899-7667. Available at: <<https://doi.org/10.1162/neco.1997.9.8.1735>>.

HU, B.; LU, Z.; LI, H.; CHEN, Q. Convolutional neural network architectures for matching natural language sentences. In: GHAHRAMANI, Z.; WELLING, M.; CORTES, C.; LAWRENCE, N. D.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014. p. 2042–2050. Available at: <<http://papers.nips.cc/paper/5550-convolutional-neural-network-architectures-for-matching-natural-language-sentences.pdf>>.

JARO, M. A. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, [American Statistical Association, Taylor & Francis, Ltd.], v. 84, n. 406, p. 414–420, 1989. ISSN 01621459. Available at: <<http://www.jstor.org/stable/2289924>>.

JIANG, X.; SOUZA, E. N. de; PESARANGHADER, A.; HU, B.; SILVER, D. L.; MATWIN, S. Trajectorynet: an embedded gps trajectory representation for point-based classification using recurrent neural networks. *ArXiv*, abs/1705.02636, 2017.

JORDAN, M. I. Attractor dynamics and parallelism in a connectionist sequential machine. In: \_\_\_\_\_. *Artificial Neural Networks: Concept Learning*. [S.l.]: IEEE Press, 1990. p. 112–127. ISBN 0818620153.

KAFFES, V.; GIANNOPOULOS, G.; KARAGIANNAKIS, N.; TSAKONAS, N. Learning domain specific models for toponym interlinking. In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (SIGSPATIAL '19), p. 504–507. ISBN 9781450369091. Available at: <<https://doi.org/10.1145/3347146.3359339>>.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In: *NIPS*. [S.l.: s.n.], 2017.

KIM, Y. Convolutional neural networks for sentence classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1746–1751. Available at: <<https://www.aclweb.org/anthology/D14-1181>>.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Available at: <<http://arxiv.org/abs/1412.6980>>.

KIRANYAZ, S.; AVCI, O.; ABDELJABER, O.; INCE, T.; GABBOUJ, M.; INMAN, D. J. 1d convolutional neural networks and applications: A survey. *ArXiv*, abs/1905.03554, 2019.

KLEPEIS, N. E.; NELSON, W. C.; OTT, W. R.; ROBINSON, J. P.; TSANG, A. M.; SWITZER, P.; BEHAR, J. V.; HERN, S. C.; ENGELMANN, W. H. The National Human Activity Pattern Survey (NHAPS): A resource for assessing exposure to environmental pollutants. *Journal of Exposure Analysis and Environmental Epidemiology*, v. 11, n. 3, p. 231–252, 2001. ISSN 10534245.

KöPCKE, H.; THOR, A.; RAHM, E. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, VLDB Endowment, v. 3, n. 1–2, p. 484–493, Sep. 2010. ISSN 2150-8097. Available at: <<https://doi.org/10.14778/1920841.1920904>>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 60, n. 6, p. 84–90, May 2017. ISSN 0001-0782. Available at: <<https://doi.org/10.1145/3065386>>.

- LANGLEY, P.; IBA, W.; THOMPSON, K. An analysis of bayesian classifiers. *Proceedings of the Tenth National Conference on Artificial Intelligence*, v. 90, 06 1998.
- LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet physics. Doklady*, v. 10, p. 707–710, 1966.
- LIN, T.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. Focal loss for dense object detection. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017. p. 2999–3007.
- LU, J.; HU, J.; ZHOU, J. Deep metric learning for visual understanding: An overview of recent advances. *IEEE Signal Processing Magazine*, v. 34, p. 76–84, 11 2017.
- LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017. p. 4765–4774. Available at: <<http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>>.
- MARINHO, A. *Approximate String Matching and Duplicate Detection in the Deep Learning Era*. Master's Thesis (Master's Thesis) — Instituto Superior Técnico - Universidade de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal, 2018.
- MIKOLOV, T.; CHEN, K.; CORRADO, G. S.; DEAN, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Available at: <<http://arxiv.org/abs/1301.3781>>.
- MIKOLOV, T.; GRAVE, E.; BOJANOWSKI, P.; PUHRSCHE, C.; JOULIN, A. Advances in pre-training distributed word representations. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), 2018. Available at: <<https://www.aclweb.org/anthology/L18-1008>>.
- MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. S.; DEAN, J. Distributed representations of words and phrases and their compositionality. *ArXiv*, abs/1310.4546, 2013.
- MONGE, A.; ELKAN, C. An efficient domain-independent algorithm for detecting approximately duplicate database records. 08 2001.
- MOREAU, E.; YVON, F.; CAPPÉ, O. Robust similarity measures for named entities matching. In: *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*. USA: Association for Computational Linguistics, 2008. (COLING '08), p. 593–600. ISBN 9781905593446.
- MORTON, G. H. A computer oriented geodetic data base and a new technique in file sequencing. In: . [S.l.: s.n.], 1966.
- NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: FÜRKNRANZ, J.; JOACHIMS, T. (Ed.). *ICML*. Omnipress, 2010. p. 807–814. Available at: <<http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10>>.
- NG, A. *Machine Learning Yearning: Technical strategy for ai engineers, in the era of deep learning*. [S.l.: s.n.], 2019.

OLAH, C. *Understanding LSTM Networks*. 2015. <<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>>. [Online; accessed 19-July-2020].

PEARSON, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, v. 2, p. 559–572, 1901.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COUNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PEROZZI, B.; AL-RFOU, R.; SKIENA, S. Deepwalk: online learning of social representations. In: *KDD '14*. [S.l.: s.n.], 2014.

QIU, D.; BARBOSA, L.; DONG, X.; SHEN, Y.; SRIVASTAVA, D. Dexter: Large-scale discovery and extraction of product specifications on the web. *PVLDB*, v. 8, p. 2194–2205, 2015.

RIJSBERGEN, C. J. V. *Information Retrieval*. 2nd. ed. USA: Butterworth-Heinemann, 1979. ISBN 0408709294.

SANTOS, R.; MURRIETA-FLORES, P.; CALADO, P.; MARTINS, B. Toponym matching through deep neural networks. *International Journal of Geographical Information Science*, v. 32, p. 1–25, 10 2017.

SANTOS, R.; MURRIETA-FLORES, P.; MARTINS, B. Learning to combine multiple string similarity metrics for effective toponym matching. *International Journal of Digital Earth*, Informa UK Limited, v. 11, n. 9, p. 913–938, Sep 2017. Available at: <<http://dx.doi.org/10.1080/17538947.2017.1371253>>.

SHAFER, G. *A Mathematical Theory of Evidence*. Princeton: Princeton University Press, 1976.

SNYDER, J. P. *Map projections: A working manual*. Washington, D.C., 1987. (Professional Paper). Report. Available at: <<https://doi.org/10.3133/pp1395>>.

SOBEL, I. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, JMLR.org, v. 15, n. 1, p. 1929–1958, Jan. 2014. ISSN 1532-4435.

STEFANIDIS, K.; EFTHYMIU, V.; HERSCHEL, M.; CHRISTOPHIDES, V. Entity resolution in the web of data. In: *Proceedings of the 23rd International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2014. (WWW '14 Companion), p. 203–204. ISBN 9781450327459. Available at: <<https://doi.org/10.1145/2567948.2577263>>.

SUN, Y.; WANG, X.; TANG, X. Deeply learned face representations are sparse, selective, and robust. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 2892–2900, 2015.

- TAIGMAN, Y.; YANG, M.; RANZATO, M.; WOLF, L. Deepface: Closing the gap to human-level performance in face verification. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 1701–1708.
- TEAM, T. pandas development. *pandas-dev/pandas: Pandas*. Zenodo, 2020. Available at: <<https://doi.org/10.5281/zenodo.3509134>>.
- TENSORFLOW. *Word embeddings*. 2020. <[https://www.tensorflow.org/tutorials/text/word\\_embeddings](https://www.tensorflow.org/tutorials/text/word_embeddings)>. [Online; accessed 19-July-2020].
- TOMEK, I. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 7(2), p. 679–772, 1976.
- ULLMANN, J. R. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *The Computer Journal*, v. 20, n. 2, p. 141–147, 01 1977. ISSN 0010-4620. Available at: <<https://doi.org/10.1093/comjnl/20.2.141>>.
- WANG, D.; ZHANG, J.; CAO, W.; LI, J.; ZHENG, Y. When will you arrive? estimating travel time based on deep neural networks. In: *AAAI*. [S.l.: s.n.], 2018.
- WILSON, D. R. 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, 2011. Available at: <<http://dx.doi.org/10.1109/IJCNN.2011.6033192>>.
- WINKLER, W. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. *Proceedings of the Section on Survey Research Methods*, 01 1990.
- XIONG, C.; ZHONG, V.; SOCHER, R. Dynamic coattention networks for question answering. *ArXiv*, abs/1611.01604, 2016.
- YALAVARTHI, V. K.; KE, X.; KHAN, A. Select your questions wisely: For entity resolution with crowd errors. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2017. (CIKM '17), p. 317–326. ISBN 9781450349185. Available at: <<https://doi.org/10.1145/3132847.3132876>>.
- YANG, C.; BAI, L.; ZHANG, C.; YUAN, Q.; HAN, J. Bridging collaborative filtering and semi-supervised learning: A neural approach for poi recommendation. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2017. (KDD '17), p. 1245–1254. ISBN 9781450348874. Available at: <<https://doi.org/10.1145/3097983.3098094>>.
- YANG, C.; HOANG, D. H.; MIKOLOV, T.; HAN, J. Place deduplication with embeddings. In: *The World Wide Web Conference on - WWW '19*. ACM Press, 2019. Available at: <<http://dx.doi.org/10.1145/3308558.3313456>>.
- ZAHARIA, M.; XIN, R. S.; WENDELL, P.; DAS, T.; ARMBRUST, M.; DAVE, A.; MENG, X.; ROSEN, J.; VENKATARAMAN, S.; FRANKLIN, M. J.; GHODSI, A.; GONZALEZ, J.; SHENKER, S.; STOICA, I. Apache spark: A unified engine for big data processing. *Commun. ACM*, Association for Computing Machinery, New

York, NY, USA, v. 59, n. 11, p. 56–65, Oct. 2016. ISSN 0001-0782. Available at:  
<<https://doi.org/10.1145/2934664>>.

## APPENDIX A – DUPLICATE DETECTION API BODY

Listing A.1 – Sample JSON body format accepted by our Duplicate Detection API.

```
1 {
2   "places": [
3     {
4       "id": "string",
5       "name": "string",
6       "geo_location": {
7         "lat": double,
8         "lng": double
9       },
10      "address": {
11        "thoroughfare": "string",
12        "sub_thoroughfare": "string"
13      },
14      "labels": ["string"],
15      "homepages": ["string"]
16    },
17    ...
18  ],
19  "threshold": double
20 }
```



## APPENDIX B – IMPLEMENTATION OF THE GINI COEFFICIENT

Listing B.1 – Python code for computing the Gini coefficient.

```

1 import numpy as np
2
3 def gini(y_actual, y_pred):
4     """Calculates the Gini coefficient.
5
6     Keyword arguments:
7     y_actual -- numpy array with truth values (0 or 1)
8     y_pred -- list-like object with predictions in [0.0, 1.0]
9     """
10    total_predictions = len(y_actual)
11    sorted_y_actual = y_actual[np.argsort(y_pred)][::-1]
12    cumulative_y_actual = sorted_y_actual.cumsum()
13    cumulative_y_actual_normalized = (cumulative_y_actual
14                                     / sorted_y_actual.sum())
15    gini_coeff = (cumulative_y_actual_normalized.sum()
16                 - (total_predictions + 1) / 2.0)
17    return 2. * gini_coeff / total_predictions
18
19 def gini_normalized(y_actual, y_pred):
20     """Calculates the normalized Gini coefficient.
21
22     Keyword arguments:
23     y_actual -- numpy array with truth values (0 or 1)
24     y_pred -- list-like object with predictions in [0.0, 1.0]
25     """
26    return gini(y_actual, y_pred) / gini(y_actual, y_actual)

```

## APPENDIX C – HYPERPARAMETERS

Table 18 – Hyperparameters of our baseline methods and the best values obtained in each data set.

Model	Hyperparameter	Values	Value <i>Pairs<sub>BR</sub></i>	Value <i>Pairs<sub>US</sub></i>
Dalvi et al. (2014)	$\lambda$	0.0, 0.9	0.0	0.0
	Geohash characters	None, 5, 6	None	None
	$\alpha$	0.1, 0.3, 0.5, 0.7, 0.9	0.3	0.9
	$t_c$	0.5, 0.6, 0.7, 0.8, 0.9, 0.95	0.5	0.5
PE - Yang et al. (2019)	$\alpha$	[0.1, 0.9], increments of 0.1	0.4	0.5
	smoothing negative sampling	5, 10, 20	20	5
	smoothing random walk length	10	10	10
	smoothing random walks	100,000	100,000	100,000
	smoothing epochs	10	10	10
	smoothing minimum frequency	1	1	1
	smoothing half window size	5	5	5
	first layer neurons	256, 512	512	512
	second layer neurons	128, 256	128	128
	third layer neurons	128, 256	128	128
	training batch size	512, 1024	512	1024
	$t_c$	[0.5, 1.0], increments of 0.05	0.75	0.75

**Source:** (M. R. Cousseau, 2020)

Table 19 – Hyperparameters of our preliminary methods and the best values obtained in each data set.

Model	Hyperparameter	Values	Value $Pairs_{BR}$	Value $Pairs_{US}$	Value $Inloco_{BR}^R$
<i>WRH</i>	<i>U</i>	5	5	-	-
	<i>L</i>	0.1	0.1	-	-
	<i>D</i>	200 meters	200	-	-
<i>PRF</i>	n_estimators	100, 110, 120, 130, 140, 150	110	150	110
	max_features	sqrt, log2	log2	sqrt	log2
	max_leaf_nodes	None, 50, 100, 150	150	150	150
	min_samples_split	2, 3, 4, 5	3	5	3
	class_weight	balanced	balanced	balanced	balanced
	oob_score	True	True	True	True
	$t_c$	[0.5, 1.0], increments of 0.05	0.9	0.9	0.9
<i>PLGBM</i>	lambda_l1	log uniform in [ $10^{-7}$ , 10.0]	$1.247 \cdot 10^{-8}$	$1.132 \cdot 10^{-8}$	$4.919 \cdot 10^{-8}$
	lambda_l2	log uniform in [ $10^{-7}$ , 10.0]	0.659	0.247	8.708
	num_leaves	[2, 256]	95	256	243
	feature_fraction	[0.4, 1.0]	0.5	0.62	0.678
	bagging_fraction	[0.4, 1.0]	1.0	1.0	0.759
	bagging_freq	[0, 7]	0	0	3
	min_child_samples	[5, 100]	5	20	100
	class_weight	balanced	balanced	balanced	balanced
	early_stopping_rounds	10	10	10	10
	iterations	100	100	100	100
	$t_c$	[0.5, 1.0], increments of 0.05	0.5	0.5	0.8

Source: (M. R. Cousseau, 2020)

Table 20 – Hyperparameters of our Classification Model and its ablated versions, with the best values obtained in each data set.

Data set	Hyperparameter	Values	Value <i>CM</i>	Value <i>WE</i>	Value <i>WE+ CHE</i>	Value <i>WE + CHE+ GE</i>
<i>Pairs<sub>BR</sub></i>	learning rate	[0.001, 0.01], increments of 0.001	0.001	0.005	0.001	0.002
	focal loss $\gamma$	0.0, 1.0, 1.5, 2.0	1.5	2.0	2.0	0.0
	dropout rate	[0.05, 0.4], increments of 0.05	0.15	0.05	0.1	0.35
	GRU dimensions	50, 100, 150, 200	150	100	200	200
	$f_{cnn}$	64, 128, 256, 512	512	-	128	512
	$k_{cnn}$	2, 3, 4, 5	4	-	5	4
	$t_c$	[0.5, 1.0], increments of 0.05	0.5	0.5	0.5	0.4
<i>Pairs<sub>US</sub></i>	learning rate	[0.001, 0.01], increments of 0.001	0.001	0.001	0.001	0.001
	focal loss $\gamma$	0.0, 1.0, 1.5, 2.0	2.0	2.0	1.0	1.5
	dropout rate	[0.05, 0.4], increments of 0.05	0.2	0.25	0.05	0.15
	GRU dimensions	50, 100, 150, 200	100	150	150	100
	$f_{cnn}$	64, 128, 256, 512	128	-	512	256
	$k_{cnn}$	2, 3, 4, 5	3	-	4	5
	$t_c$	[0.5, 1.0], increments of 0.05	0.55	0.55	0.65	0.6

**Source:** (M. R. Cousseau, 2020)