Detecting Nearly Duplicated Records in Location Datasets

Yu Zheng, Xixuan Fen, Xing Xie
Microsoft Research Asia,
4F, Sigma Building, No.49 Zhichun Road, Haidian District, Beijing 100190, China
{yuzheng, xingx}@microsoft.com,

Shuang Peng, James Fu
Search Technology Center, Microsoft,
4F, Sigma Building, No.49 Zhichun Road, Haidian District, Beijing 100190, China
{speng, jamesfu}@microsoft.com

ABSTRACT

The quality of a local search engine, such as Google and Bing Maps, heavily relies on its geographic datasets. Typically, these datasets are obtained from multiple sources, e.g., different vendors or public yellow-page websites. Therefore, the same location entity, like a restaurant, might have multiple records with slightly different presentations of title and address in different data sources. For instance, 'Seattle Premium Outlets' and 'Seattle Premier Outlet Mall' describe the same Outlet located in the same place while their titles are not identical. This will cause many nearly-duplicated records in a location database, which would bring trouble to data management and make users confused by the various search results of a query. To detect these nearly duplicated records, we propose a machine-learning-based approach, which is comprised of three steps: candidate selection, feature extraction and training/inference. Three key features consisting of name similarity, address similarity and category similarity, as well as corresponding metrics, are proposed to model the differences between two entity records. We evaluate our method with intensive experiments based on a large-scale real dataset. As a result, both the precision and recall of our method exceeded 90%.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - *data mining, Spatial databases and GIS.*

General Terms

Algorithms, Performance, Experimentation.

Keywords

Local search, Geographical information systems, Data quality, name similarity, address similarity.

1. INTRODUCTION

With the combination of geographical information systems and the Web, users are increasingly relying on mobile/location search services, such as Google and Bing Maps, to find destinations like shopping malls or restaurants. The quality of these kinds of search engines heavily depends on their location datasets, which annotate the digital map with landmarks, businesses and points of interest.

In this kind of dataset each location entity consists of a set of properties shown in Table I. We call each row a record, and name a column as a field. The location entities are usually classified as "point of interest" (POI) entities or "yellow page" (YP) entities. POI entities are often created by users with mobile, GPS-enabled devices. Accordingly, the GPS coordinates for such entities tend to have a high degree of accuracy. Other fields of a POI entity (e.g., name, address, etc.), however, tend to be less accurate as the entity-creating user may not enter those fields with a great degree of care. YP entities are often created by the businesses or locations that they identify, and may be captured for the dataset by, e.g., crawling the Internet. Because YP entities are often created by businesses or locations having a strong desire to be found, name and address fields of the entities are likely to be highly accurate. GPS coordinates for YP entities are then geo-coded based on the address field and vary in quality based on the accuracy of the address field.

Table 1. Properties of a location entity

Name	Address	GPS Position	Phone Num.	Category	Type
The Matt's Bar	701 5th Ave Seattle, WA	116.325, 35.364	1-56987452	Café	YP
Silver Cloud Inn	314 7th Ave Redmond, WA	116.451, 35.209	1-25698716	Hotel	POI

Typically, these datasets are obtained from a variety of sources, such as purchased from multiple professional data providers and crawled from the Web. Intuitively, different people would pay different attention when manually recording a location entity in the physical world. Meanwhile, some fields of an entity could change as time goes by. For example, a restaurant could change its name, and a street can be renamed. Therefore, a location entity could have multiple records with slightly different presentations in different data sources. For instance, 'Seattle Premium Outlets' and 'Seattle Premier Outlet Mall' describe the same Outlet located in the same place while their titles are not identical. We call this phenomena nearly duplicated entities, which means two records from a (multiple) location dataset(s) describe the same real-world entity with slightly different (very similar) presentations.

These nearly duplicated entities will bring trouble to the management of a location database. Also, users would be confused by the multiple results associated with a query. For instance, when searching for "Matt Bar", three results ("The Matt's Bar", "Matt's Bar", "The Matt Bar") with different geo-positions could be retrieved for the user. As a result, the user would have no idea which one is the correct answer. However, by merging these nearly duplicated records, we can get the following two benefits. First, we can save lots of storage by normalizing the nearly duplicated records and make our location database more consistent. Second, if a POI record and a YP record are detected

GIS '10, 03-NOV-2010, San Jose CA, USA
Copyright © 2010 ACM 978-1-4503-0428-3/10/11…$10.00

as nearly duplicated, we are able to improve the geo-position accuracy of the YP entity using the POI entity's GPS position.

However, it is very challenging to identify whether or not two records are nearly duplicated. First, the two records might have different presentations in the same field. Therefore, we cannot make a decision by exactly matching the fields of two records. Second, each field of a record might not be very accurate. (As we mentioned before, the geo-position of a YP entity would not be very precise and the address information of a POI entity might be less accurate.) Consequently, it is very difficult to define some thresholds to determine the similarity between two records. For instance, it is hard to say two records are nearly duplicated if the geo-distance between them is less than 50 meters and the edit distance between their names is less than 2. Third, metrics like the edit distance and Euclidian distance cannot well model the difference between two records. For example, 'Sunshine Day coffee' and 'Sunny Day coffee' are different restaurants located in the same street while the edit distance between their names is very small and their locations are very close.

In this paper, we propose a machine-learning-based approach, which can detect nearly duplicated records from a (or multiple) location dataset(s). It is a step towards improving the data quality of geographical information systems for mobile/local search services. In this approach, we first select a small set of similar candidates for each given location entity using a preprocessing algorithm. Then, for each pair of similar candidates, we investigate the similarity of each field, such as name similarity, address similarity and category similarity, between them, and identify a set of features representing these similarities. Later, these features are fed in a classification model, which has been trained in advance using human labeling datasets, to attain prediction results (duplicated or non-duplicated). The contribution of this work lies in the following three aspects:

- We employ the philosophy of machine learning to infer the similarity between two entities rather than using exact match or rule-based heuristics. Meanwhile, instead of checking each single field, multiple fields of an entity are considered comprehensively when performing the inference.
- We identify a set of key features including name similarity, address similarity and category similarity, which can be used to effectively differentiate nearly-duplicated from non-duplicated records. Also, we define corresponding metrics that can better model these similarities beyond existing distance measures, such as the edit and Euclidian distances.
- We evaluate our approach using a large-scale real dataset and justify its effectiveness with intensive experiments.

As the technology has been transferred into Microsoft Bing Local Search, we will not introduce our approach in detail.

The structure of the rest of this paper is organized as follows. In Section 2, we give an overview of our approach. Then, in Section 3 we present some key modules of our method, including candidate selection and feature extraction. In Section 4, we conduct intensive experiments on the real dataset and report on some major results followed by a discussion. Finally, we draw our conclusion and present future work in Section 5.

2. SYSTEM OVERVIEW

In this Section, we first introduce some basic concepts used in our method and then briefly describe the architecture of our approach, consisting of online inference and offline learning.

2.1 Preliminary

Definition 1. Record. A record R in a location database is a set of descriptions of different properties of a location entity O from the real world. Each record has a set of fields F={*name, Address, geo-position, category, phone, type*}, where a field $f \in F$ represents a property of the location entity O.

Definition 2. Nearly duplicated records. Two records R_1 and R_2 describe the same location entity O, while $\exists f \in F, R_1.f \neq R_2.f$.

Definition 3. City structure C. City structure is a tree-based hierarchy specifying the parent-child relationships between locations of different granularities. Nodes appearing on a higher level have a larger geo-scale than those occurring on a lower level. This structure can be built based on the zip codes of a city or some predefined schema offered by governments.

Figure 1 demonstrates the city structure of New York City. $C.l_i$ denotes the i-th level of the city structure C, e.g., $C.l_4$ represents the 4th level (could be streets, in a semantic meaning) of the city. Of course, different cities have different structures, and some cities may only have a three-level structure (such as city → area code → zip code).

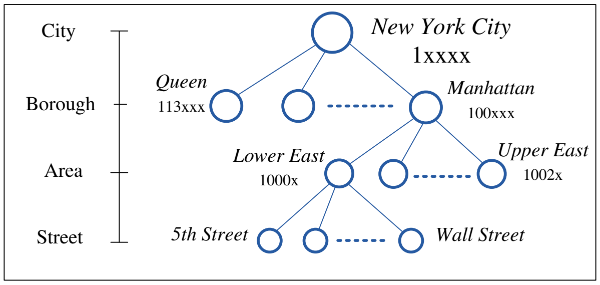

Figure 1. The city structure of New York City

Definition 4. Category Hierarchy CH: Category hierarchy is a tree-based structure describing the parent-child relationship between location entity categories of different granularities. The children nodes denote the sub-categories of their parent node; therefore, they have a relatively finer granularity.

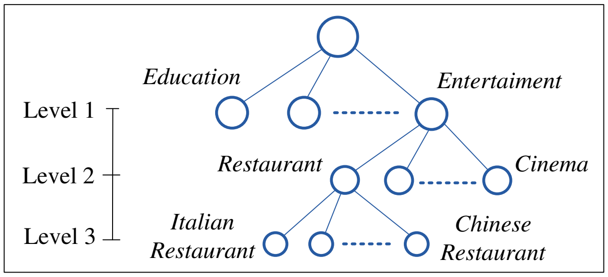

Figure 2. Category hierarchy

Figure 2 depicts an example of category hierarchy. On the first level, location entities can be partitioned into categories of education, entertainment, sports and government, etc. Further, the categories on the second level, e.g., entertainment, can be divided into finer subcategories, such as restaurant, coffee, and Movie Theater. Later, the category of restaurant can be classified into Chinese restaurant and Italian restaurant, etc., on the third level.

2.2 Architecture

Figure 3 shows the architecture of our approach, which is comprised of offline learning and online inference operations.

Offline Learning: First, for each record R from a location dataset $\mathbb{R}$, we filter obviously irrelevant ones using some simple heuristics and select some similar candidates $\mathbb{S}$, where $\mathbb{S} \subset \mathbb{R}$; we call this operation candidate selection. For example, if the names of two entities are totally different (or the geo-positions of two records are far away from each other), they would not describe the same entity in the physical world. In order to improve the efficiency of the selection, a spatial index is built over the whole dataset, i.e., only the records within a distance threshold to the given record will be considered as similar candidates.

Second, given a record R_1, we formulate some record pairs, e.g., $P =< R_1, R_i >, R_i \in \mathbb{S}$, between R_1 and its similar candidates. Then, we investigate the similarity between the records in each pair P by respectively checking the similarity between their names, addresses and categories. Later, a set of features are extracted to represent these similarities. With regard to the name similarity, the idea of inverse document frequency (*idf*) is used to weight different terms of an entity name according to their occurring frequencies in the name fields of the total corpus. Regarding the address similarity, we project the addresses of these two records to the city structure by parsing their addresses. The lower the level two entities encounter, the higher the address similarity these two entities share.

Third, the extracted features are employed to train a classification model with human labeled ground truths. Later, this model will be used to answer online queries.

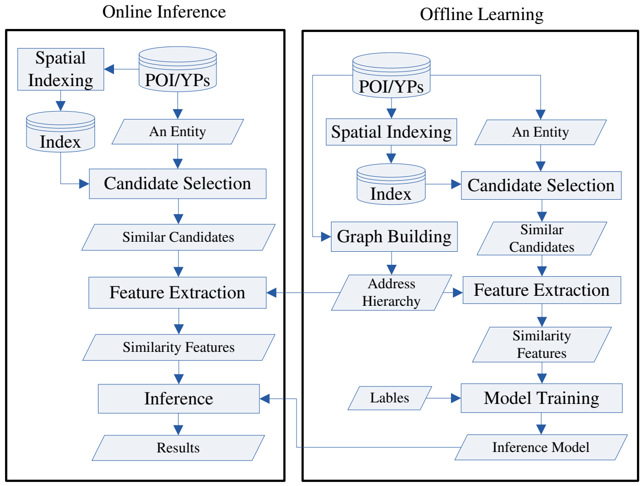

Figure 3. Architecture of our approach

Online Inference: Given a record, we first select a set of similar candidates using the same method as the offline learning. Then, the same features are extracted from each record pair, consisting of the record and one of its similar candidates. We feed these features into the inference model learned in the offline process and obtain a predicted result.

3. Detailed Methodology

This section presents some key components of our method, including candidate selection and feature extraction.

3.1 Candidate Selection

The candidate selection aims to filter some obviously irrelevant entities in terms of some insights generated from commonsense knowledge; therefore improving the efficiency of our approach.

Insight 1: *If the name fields of two records are totally different, we believe they describe different location entities.*

Typically, people might make some mistakes, such as typos, when recording a location entity in the physical world. However, these mistakes would not be that huge so as to totally change the name of a POI entity.

Insight 2: *If the geo-distance between two entities exceeds a threshold, we believe they are impossible to be nearly duplicated ones.*

Although the accuracy of a YP entity's geo-position varies in the quality of a geo-coder and that of its address, the inferred geo-position could not be very far away from its real location, e.g., 10 km away.

Insight 3: *If the level-1 category fields of two records are different, they are impossible to be nearly duplicated records of the same physical entity.*

Sometimes, people may fail in differentiating a European restaurant from US ones; however, they would not regard a movie center as a café. Therefore, if two entities pertain to different categories on a high level, we believe they are different records.

3.2 Feature Extraction

Herein, we identify three key features, consisting of name, address and category similarities, and calculate these similarities with corresponding metrics.

3.2.1 Name Similarity

Normally, people use edit distance [3] to measure the difference between two strings. The edit distance is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. Basically, the bigger the edit distance between two strings is, the more different these two strings are. However, this distance metric cannot be used directly to measure the similarity between the names of entity locations.

Record names	Same part	Difference	Edit Dist.	Results
Galaxies Cafe Galaxies Coffee House	Galaxies	Cafe Coffee House	9	Same
Espresso Darer Espresso Diana	Espresso	Darer Diana	4	Diff.

Figure 4. Name similarity and edit distance

As shown in Figure 4, the edit distance between the names ('Galaxies Cafe' and 'Galaxies Coffee House') shown in the first example is 9, which is greater than that of the second example ('Espresso Darer' and 'Espresso Diana'). However, the records shown in the first example are nearly duplicated while those shown in the second case describe different entities. Obviously, this observation is against and beyond the nature of the edit distance.

In our method, we first detect the shared and differing words between two names. As shown in Equation (1) and (2), the shared words could formulate a vector $V_s = \langle w_1, w_2, \ldots, w_i \rangle$ and the different parts are stored in anther vector $V_d = \langle w_1, w_2, \ldots, w_i \rangle$, where w_i is a term which might contain several words.

$$V_s = Same(R_1.name, R_2.name); \qquad (1)$$

$$V_d = Diff(R_1.name, R_2.name); \qquad (2)$$

Regarding the first instance depicted in Figure 4, V_s =<Galaxies> and V_d =<Café, Coffee House>, and in the second example, V_s =<Espresso> and V_d=<Darer, Diana>. Intuitively, terms like '*the*' frequently appearing in the name field of other entities might be too general to represent the meaning of an entity, while terms that rarely occur in entities' names would be more important to distinguish an entity from others.

Following this observation, we employ the idea of *idf*, which is a statistic metric used in information retrieval to determine how important a word is to a document. Inverse document frequency can be regarded as a factor diminishing the weight of terms occurring frequently in many names and increasing the weight of terms occurring rarely. As shown in Equation (3), N denotes the number of entities in the corpus and the denominator is the number of entities whose name contains the term w_i.

$$idf(w_i) = log\frac{N}{|\{R_j, w_i \in R_j.Name\}|}; \qquad (3)$$

From the detected V_s and V_d, we identify two features S_1 and S_2 according to Equation (4) and (5). In short, we aggregate the weights of the same parts and find out the maximum different parts between two names. These two features represent the similarity between names of two entities and will be fed into the inference model with other features.

$$S_1 = \sum_{i=1}^{|V_s|} idf(w_i \in V_s)\ ; \qquad (4)$$

$$S_2 = max_{w_i \in V_d}\, idf(w_i); \qquad (5)$$

3.2.2 Address Similarity

As we mentioned before, the address field of a record could be somehow insufficient and inaccurate. For instance, the following two strings describe the address of the same building.

"*79 Beaver St, New York, NY 10005-2812*", and

"*92 Water St, New York, NY 10005-3511*"

At first glance, the location corresponding to these two addresses is totally different in terms of edit distance or presentation. However, these two locations pertain to the same node (zip code: 10005) in the city structure, i.e., they may be located very closely in geographical space.

The observation is "*the geospatially closer two records are located, the higher the probability these two records might be nearly duplicated*". Following this observation, we first project each location entity to a node of a city structure and find out the lowest parent node two records share. In short, the address similarity is represented by the level of the lowest co-parent of two records. The lower the level shared by two records is the more similar these two records might be.

As shown in Figure 5, we parse the address of each YP record (line 2) into several phrases and insert this record into a city structure C (refer to Definition 1) according to its zip code. Later, we calculate the average (latitude and longitude) coordinates of each node on the bottom level of the city structure. The returned city structure with the average coordinates will later be used for address similarity calculation.

```
Algorithm BuildCityTree ($\boldsymbol{C}, \boldsymbol{Y}$)
Input: city structure $C$ and the collection of yellow page entities $Y$
Output: city structure $C'$ with mapped entities
1. Foreach record $R \in Y$
2.     $\theta$=AddressSegmentation($R.Address$); //address segmentation
3.     $\boldsymbol{C'}$=InsertAddress($\theta$, $\boldsymbol{C}$)
4. Foreach node $n \in \boldsymbol{C'}.l_{max}$ // the lowest level of the city structure
5.     $Sum$=0;
6.     Foreach record $R \in n$
7.         $Sum$+= $R.GPS$;
8.     $n.coord = Sum/|n|$; //calculate the average coordinates of a node
9. Return $C'$.
```

Figure 5. The algorithm for building a city tree

As shown in Figure 6, given two records R_1 and R_2, we find the nodes these two records belong to and return the lowest co-parent they share in the city structure C' built above. Figure 7 demonstrates some examples of co-parent search, and the node search algorithm is detailed in Figure 8.

```
Algorithm AddressSimlarity ($C'$, $R_1$, $R_2$)
Input: city structure $C'$ with mapped entities, two records $R_1$ and $R_2$
Output: An integer representing address similarity between $R_1$ and $R_2$
$n_1$=SearchNode($C'$, $R_1$);
$n_2$=SearchNode($C'$, $R_2$);
$n_p$=SearchCoParent($n_1$, $n_2$, $C'$); //search for the lowest co-parent
Return $n_p.level$;                 //return the level node $n_p$ lies in.
```

Figure 6. Measuring the address similarity between records

For instance, as depicted in Figure 7 A), the co-parent node n_p of R_1 and R_2 is on the second level, and the co-parent nodes demonstrated in Figure 7 B) and C) are on the third and fourth level respectively. Obviously, the case presented in Figure 7 C) is more likely to be nearly duplicated.

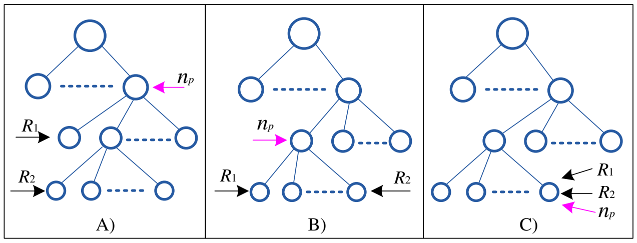

Figure 7. Examples of co-parent

Since the addresses of YP entities are relatively accurate, we determine the nodes they pertain to by using their address (Figure 8, lines 1~3). However, with regard to a POI entity, we search for the nearest node that is geospatially close to the POI on the bottom level of C' (Figure 8, lines 4~10). The measurement here is geographical distance calculated based on GPS coordinates rather than text matching.

```
Algorithm SearchNode ($C'$, $R$)
Input: city structure $C'$ with mapped entities and a record $R$
Output: the node the record $R$ pertains to

If $R.type$ = YP                    //Yellow page entity
    $\theta$=NameSegmentation($R.Address$);
    $n'$=ProjectToNode($\theta$, $C'$);  //map $R_1$ to a node in terms of its address.
Else                                //POI entity
    minDist=0;
    Foreach node $n \in C'.l_{max}$  // the lowest level of the city structure
        dist=Distance($R.GPS$, $n.coord$); //distance between $R$ and node $n$
        If dist < minDist
            minDist←dist;
            $n'$=$n$;
Return $n'$.
```

Figure 8. Mapping a location entity to a city structure

3.2.3 Category Similarity

Similar to address similarity, we map each location entity to a category hierarchy *CH* in terms of the category they pertain to and find the lowest co-parent node on the *CH*. The lower the level their co-parent node is found on the more similar these two records might be. Sometimes, the category of location entity with multiple functionalities is really difficult to determine, e.g., some shops usually provide coffee, lunch and wine simultaneously. Therefore, different people would classify these shops into different categories. So, it is not reasonable to make a binary decision whether two records are nearly duplicated based on exactly matching their categories.

4. Experiments

In this Section, we first present the experimental settings and then report on some major results, followed by a discussion.

4.1 Settings

4.1.1 Dataset

We evaluate our approach based on the real location dataset of Beijing, which contains 0.64 million location entities (0.25 million POIs and 0.39 million YPs). From this dataset, we manually labeled 1600 entity pairs consisting of 800 nearly duplicated pairs and 800 non-duplicated ones. These entity pairs were randomly selected and used as an evaluation dataset.

With the manually labeled ground truth, we can evaluate our approach as a classification problem. Meanwhile, to investigate the stability of our method, we perform the experiment on different scales of dataset step by step. If the performance does not vary in the scale of the test datasets, our method is stable and scalable. Table II shows four evaluation datasets of different scales. For instance, in dataset D1, we select 200 entity pairs as the training set and use 200 pairs as the test. In both the training and test datasets, the number of nearly duplicated cases is the same with that of non-duplicated pairs.

Table II. Evaluation Datasets

Datasets	Training Set	Test Set	Total
D1	200	200	400
D2	400	400	800
D3	600	600	1200
D4	800	800	1600

4.1.2 Inference Model

Our inference model employs a decision tree as a classifier and uses bootstrap aggregating (bagging) [7] as a meta-algorithm to improve the accuracy by reducing variance and over-fitting. All the experiments related to inference are conducted by using Weka APIs [17], a well-known open toolkit for machine learning.

4.1.3 Baselines

We compared our method with two baselines. One is the *exact match*, in which two records are detected as nearly duplicated if their names and addresses are identical. This method would have a relatively high precision with a very low recall as a trade-off. The other is a *rule-based* method, in which two records are detected as nearly duplicated if the edit distance between their names is less than a threshold and the geo-distance between them is less than a certain value.

4.2 Results

4.2.1 Single Feature Study

Figures 9 and 10 depict the precision and recall of our method using each single feature. Herein, S_1 and S_2 were defined in Equation (4) and (5); S_3 denotes address similarity and S_4 represents category similarity.

First, S_1 and S_2 are more effective beyond S_3 and S_4 in identifying nearly duplicated entities. In short, the information contained in a location entity's name is more important than the observations from other fields of this entity. Meanwhile, the address similarity is more powerful than category similarity in terms of the inference precision. However, category similarity has a very high recall capability in retrieving the nearly duplicated entities over others.

Second, we can see that the precision and recall of our method using each feature as a stand-alone does not vary too much in the increasing scale of the evaluation dataset. This phenomenon justifies the scalability of our method and the stability of those features.

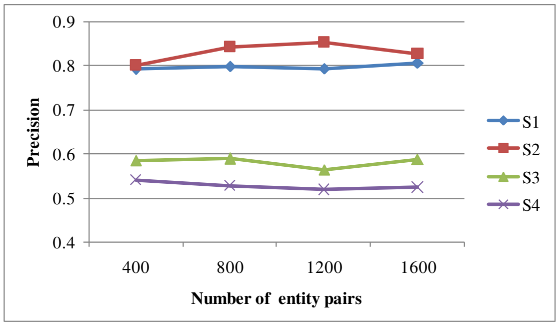

Figure 9. Precision of identifying nearly duplicated entities using a single feature

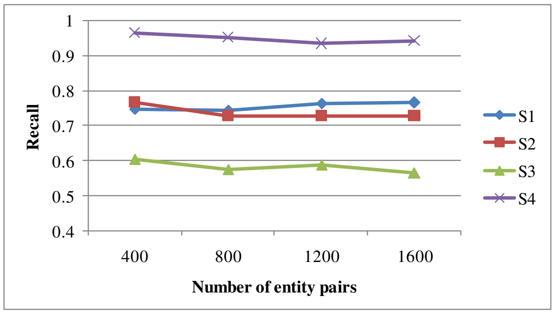

Figure 10. Recall of identifying nearly duplicated entities using a single feature

4.2.2 *Feature Combination*

Table III presents the performance of our methods using different feature combinations, such as $S_1 + S_2 + S_3$, which means combining name similarity with address similarity. All the results were obtained based on dataset D4. From the reported data, we can see the inference performance clearly increases when adding address and category similarities. When these four features were used together, we obtained the highest accuracy.

Table III. The performance of feature combinations

Features	Duplicated		Non-duplicated		Overall
	Pre.	Rec.	Pre.	Rec.	accuracy
$S_1 + S_2$	0.86	0.85	0.852	0.86	0.858
$S_1 + S_3$	0.80	0.76	0.746	0.81	0.782
$S_1 + S_2 + S_3$	0.86	0.85	0.853	0.86	0.861
$S_1 + S_2 + S_4$	0.86	0.85	0.853	0.86	0.861
$S_1 + S_2 + S_3$	**0.88**	**0.86**	**0.858**	**0.89**	**0.875**

Figure 11 depicts the performance of the rule-based baseline method, which estimates the near duplication according to geo-distance and edit distance. For instance, when Geo-dist=0.3 KM and edit distance=5, the accuracy of this baseline reached its peak (0.75). In short, if the geo-distance between two records is less than 0.3 KM and the edit distance between their name is smaller than 5, these two records are regarded as nearly duplicated.

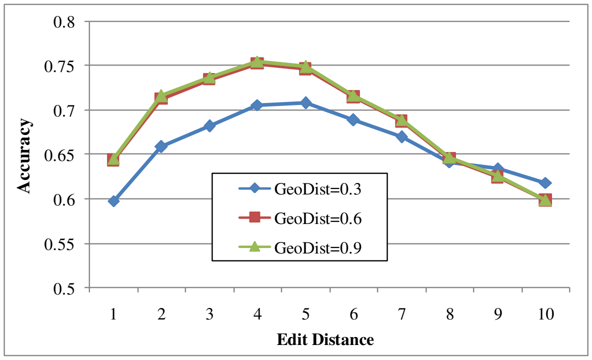

Figure 11. Overall accuracy of the rule-based baseline method

In Table IV, we compare the performance of different methods using dataset D4. Clearly, our approach outperformed the two baseline methods in terms of both precision and recall.

Table IV. Comparison of different methods

Features	Duplicated		Non-duplicated		Overall
	Pre.	Rec.	Pre.	Rec.	accuracy
Exact Match	1	0.183	0.558	0.100	0.598
Rule-based	0.78	0.701	0.736	0.808	0.755
Our approach	**0.88**	**0.866**	**0.858**	**0.891**	**0.875**

Further, Figure 12 shows the stability and scalability of our approach based on increasing datasets, where precision (Y) denotes the accuracy of inferred nearly duplicated and precision (N) means that of non-duplicated records. No matter what size of dataset we used, the precision and recall of our method are around 0.85. Also, the overall accuracy improved as the dataset increases. Therefore, our approach would be more effective on a large-scale dataset.

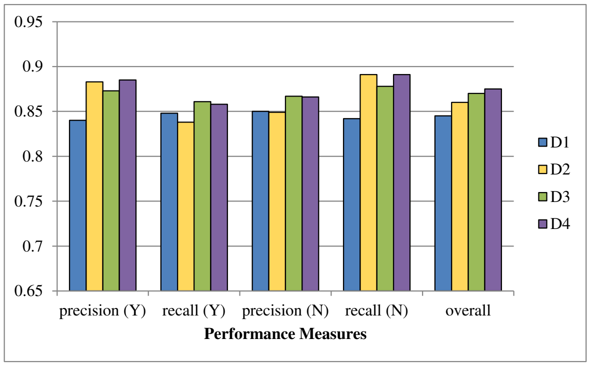

Figure 12. Performance of our approach over different datasets

5. Related Work

Actually, literature similar to our work is few. However, the idea of our work is related to the following three parts: similar document search in information retrieval, string distance and mining similar location entities.

5.1 Document Retrieval

Document retrieval is defined as the matching of some stated user query against a set of free-text records. These records could be any type of mainly unstructured text, such as newspaper articles, real estate records or paragraphs in a manual. User queries can range from multi-sentence full descriptions of an information need to a few words [2]. In this retrieval, a document was first parsed into some terms, which will be calculated term frequency-inverse document frequency (*tf-idf*).

Tf-idf is a statistical measure used to evaluate how important a word is to a document in a collection or corpus [15]. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus [12][13]. Then, each document is represented by a feature vector, each item of which denotes the *tf-idf* value of a term. Later, an inverse index between a term and documents containing this term can be built for online-search.

In our approach, we also employ the concept of *tf-idf* to weight the different parts of an entity's name. The terms that appear in one entity's name but that rarely occur in other entities' names would be more important than the rest of the name in differentiating the entity from others. However, instead of building an inverse index between terms and records, we convert the sum of the same parts' *idf* and the maximum *idf* of the different parts into two features. Therefore, we are able to use a machine learning strategy to infer the nearly duplicated entities.

5.2 String Edit Distance

The edit distance is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. Some excellent literature about string distance is available [3][5][14]. Several variants of the edit distance have been proposed, including the normalized distance [9] and the constraint edit distance [11].

Basically, the bigger the edit distance between two strings is, the more different these two strings are. However, this distance metric cannot be used directly to measure the similarity between the names of entity locations (refer to the example shown in Figure 4).

In our string distance, we consider not only the information of the two strings to be matched but also their context information, i.e., the *tf-idf* value of the words contained in the two strings in a context of the whole corpus. Meanwhile, the similarities we defined for an entity pair are far beyond the string distance, e.g., the address and category similarities are based on some kinds of tree structures.

5.3 Mining Similar Location Entities

Co-location pattern discovery is a newly developed trend in this direction. It aims to find classes of spatial objects that are frequently located together, e.g., restaurants and shopping malls are usually co-located [4][10][16].

Zheng et al. [8] detected spatial outliers from a location dataset using a data mining algorithm. Here a spatial outlier means that the geo-position field of a location entity's record is different from its real-world position. Thus, people will be misguided by the inaccurate results returned by a local search engine like Bing local search. Chang et al [1] proposed a system that can find out the geographic regions sharing a similar distribution of POIs.

These techniques mentioned above focus on clustering different location entities in terms of some similar properties. However, we aim to detect the same entities (in the real world) but having nearly duplicated presentations in different data sources.

6. Conclusion

In this paper, we propose an algorithm that detects nearly duplicated records in a (multiple) location dataset(s) using a machine-learning-based inference paradigm. In our approach, we identify name similarity, address and category similarities between two records as features, and define corresponding metrics that measure these similarities well. Using a small set of human labeled ground truths, we train a classification model, which can be later used for inferring large-scale datasets. We evaluated our approach on a real location dataset and test the stability and scalability of our method with different sizes of data. As a result, our approach clearly outperformed baseline methods by obtaining an overall inference accuracy of 0.875. Meanwhile, no matter what size of dataset we used, our approach achieved an accuracy of about 0.85. These results justify the stability and scalability of our approach in processing large-scale datasets.

7. REFERENCES

[1] Chang, S., Zheng, Y., and Xie, X. Answering Top-k Similar Region Search Queries. In Proceedings of Database Systems for Advanced Applications (2010), Springer Press: 186-201

[2] Document Retrieval. http://en.wikipedia.org/wiki/Document_retrieval

[3] Hall P., and Dowling G., Approximate string match. ACM Computing Surveys, 12, 4 (1980): 381-402.

[4] Huang, Y., Shekhar, S., and Xiong, H.. Discovering co-location patterns from spatial datasets: A general approach. TKDE, 16(12):1472-1485, 2004

[5] Kukich, K. Techniques for automatically correcting words in text. ACM Computing Surveys, 24 (1992): 377-439

[6] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (1966):707–710.

[7] Leo Breiman. Bagging Predictors. Machine Learning. 1996

[8] International Patent. MS 326249.01. Detecting spatial outliers in a location entity dataset. 2009

[9] Marzal, A., and Vidal, E. Computation of normalized edit distance and applications. IEEE Trans. PAMI 15, 9 (1993): 926-932

[10] Morimoto, Y. Mining frequent neighboring class sets in spatial databases. In Proceedings of SIGKDD, 2001, ACM Press: 353-358.

[11] Oomman, B. Constraint string editing. Information Sciences 40 (1986)

[12] Salton, G. and Buckley, C. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24**,** 5 (1988): 513–523.

[13] Salton, G., Edward, A. and Wu, H. Extended Boolean information retrieval. Communications of the ACM, 26**,** 11 (1983): 1022–1036.

[14] Sankoff, D. and Kruskal, J. Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. AddisonWesley, Reading, MA, 1983.

[15] Spärck Jones, Karen. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28, 1 (1972): 11–21

[16] Xiao, X., Xie, X., Luo, Q. Density-based co-location pattern discovery. In Proceedings of ACM SIGSPATAIL conference on Geographic Information Systems, 2008. ACM Press: 11-20.

[17] Weka. http://www.cs.waikato.ac.nz/ml/weka/.