Deduplicating a Places Database

Philip Bohannon
Facebook
pbohannon@fb.com

Nilesh Dalvi
Facebook
nileshd@fb.com

Marian Olteanu
Facebook
marianolteanu@fb.com

Manish Raghavan
U. of California, Berkeley
manishraghavan@gmail.com

ABSTRACT

We consider the problem of resolving duplicates in a database of places, where a place is defined as any entity that has a name and a physical location. When other auxiliary attributes like phone and full address are not available, deduplication based solely on names and approximate location becomes an extremely challenging problem that requires both domain knowledge as well an local geographical knowledge. For example, the pairs "Newpark Mall Gap Outlet" and "Newpark Mall Sears Outlet" have a high string similarity, but determining that they are different requires the domain knowledge that they represent two different store names in the same mall. Similarly, in most parts of the world, a local business called "Central Park Cafe" might simply be referred to by "Central Park", except in New York, where the keyword "Cafe" in the name becomes important to differentiate it from the famous park in the city.

In this paper, we present a language model that can encapsulate both domain knowledge as well as local geographical knowledge. We also present unsupervised techniques that can learn such a model from a database of places. Finally, we present deduplication techniques based on such a model, and we demonstrate, using real datasets, that our techniques are much more effective than simple TF-IDF based models in resolving duplicates. Our techniques are used in production at Facebook for deduplicating the Checkins Places database.

1. INTRODUCTION

A key step in data cleaning and integration is the identification and removal of duplicate records, and a variety of technologies [8, 2] have been developed to address the problem. One of the core problems in duplicate identification is *scoring* a pair of records – computing the probability that the records are a match. This computation, in turn, depends on scoring individual attribute matches.

In this paper, we address the problem of scoring matches between pairs of noisy, user-generated business records. Many of these records are generated by a user in the process of "checking in" to a business on their mobile phone. Since the user is pressed for time, the business record often has only two attributes, a name and a location, where the name is typed by the user and the location is from the phone's GPS system. Unlike other domains with relatively *rich* attribute data, such as census data or bibliographic data, our data set has exceptionally *sparse* attribute data, making it absolutely critical to compute a high quality score based on the two attributes that are always present, even in the face of noise or errors in one or both attributes.

Note that, in the literature, matching business names is generally considered a hard problem, with significantly poorer quality scores obtained compared to other domains [15, 3]. Examples of some of the challenges that limit match quality Table 1. The problems include 1) large edit distances for the names of matching records, 2) small edit distances for records that do *not* match and 3) geographic knowledge that is needed to determine if a match exists.

A second source of difficulty in weighting terms come from the *spatial context* of the business name. As an example, "Bleecker Grocery and Convenience" and "Bleecker Delicatessen" are two businesses that are less than two blocks apart on "Bleecher Street" in New York City. In the places database as a whole, "Bleecker" is an infrequent word; highly rated by TF/IDF and similar metrics. However, due to the *local* popularity of the term "Bleecker", the disagreement between "Delicatessen" and "Grocery and Convenience" should be given much more weight. In contrast, if these two names appeared somewhere else, the fact that the *locally rare* term "Bleecher" appeared in both should be weighted heavily.

In this paper we make several contributions toward solving the challenge of matching business names:

1. We formally define the *core word* problem to capture the intuition of weighting "starbucks" and "subway" more than "restaurant" or "cafe", and introduce an unsupervised learning problem to assign weights to words in business names according to this model.

2. We define the problem of determining the *spatial context weight* to assign words in a business name matching system, and again introduce an unsupervised algorithm to determine such weights.

3. We define a *name matching function* that elegantly combines these two weights.

4. We demonstrate that the combination of accurate core word identification as well as contextual word weighting can dramatically improve precision and recall of a place-name matcher, compared to TF-IDF weighting. We are able to achieve a recall of 90% at a precision of 90% based solely on names.

Previous work on customizing string matching functions to specific tasks have fallen into two categories: token-weighting schemes [14] and machine-learned edit distances [10, 13, 4, 11]. Our *core words* technology is an extension of token-weighting schemes, but departs substantially from the use of frequency as the dominant component. In contrast, *spatial context* is more general, since it defines different weights to the same word in different contexts. Thus, this component cannot be modeled by a single fixed weighting as considered by published token-weighting schemes, and it will also be very difficult to simulate in a system for machine-learned edit distances, even with affine gaps. The point is that these techniques again attempt to learn a set of fixed weights for tokens and/or characters. A much more promising approach is to *extend* learned match-weighting systems by using our spatial and core word scores as *features*, and this is a topic of future work.

2. PLACES DATABASE

In this section, we describe our Places database, and illustrate the challenges of deduplication.

Our Places database consists predominantly of user generated content (UGC), which lets users search for places and "check in" to them [5]. The following three operations form a simple abstraction that captures the essence of our database.

create(`name`, `address`*, `phone`*, `category`*; `uid`, `lat`, `lon`)

search(`name`*; `uid`, `lat`, `lon`)

checkin(`placeid`; `uid`, `lat`, `lon`)

Every operation is implicitly associated with the *uid*, denoting the user performing the operation, and the *lat* and *lon*, consisting of the coordinates of the location at which the operation takes place. Optional fields are annotated by $*$.

The search operation, which is initiated by user `uid` at location (`lat`, `lon`), takes an single, optional argument that is the name of the place. It returns a set of potential matches, ranked by the likelihood of the given user to checkin to the place. When the `name` argument is omitted, it simply returns the popular places near the user. The checkin operation checks the user into a specific place, chosen from the search results. This operation records that the user was at the location, and usually alerts the users friends to their location as well. If the user does not find the place she wants, she can create a new place using the create operation, which requires the user to specify the `name`, and optionally other attributes like `address`, `phone` and `category`.

Source of Duplicates: We define a *search failure* as a situation in which a user calls search with the intention of checking in to some place p, but p does not appear in the result list, even though it actually exists in the database. Typically, a duplicate is created when a user has a search failure, and chooses to call create, producing a duplicate of p. The two main causes of search failure are

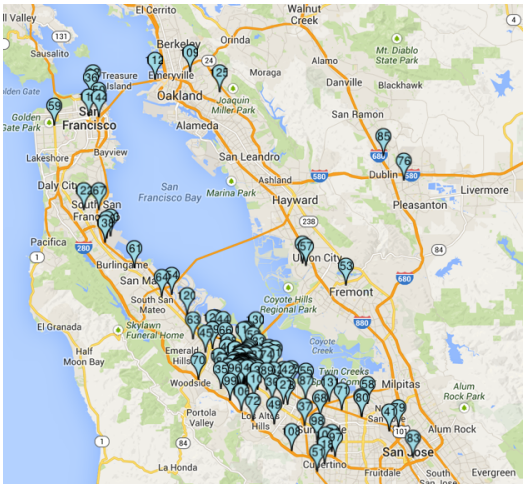

Figure 1: Duplicates of *Stanford University*

1. the user search query might have a typo, alternate spelling or abbreviation.

2. the user might try to checkin to a place from a physical location that is away from the actual place location, e.g., trying to check into a restaurant on the way back home. Again, this might result in the place not being ranked at the top in the search results.

In these situations, instead of refining or altering the search query, the user might simply choose to create a new place. Given the high volume of checkins, even if search failures are rare, the result can be a large number of duplicates. Note that this process effectively guarantees difficulty in matching the newly created place arose either from a location difference or a name difference that was not recovered by the spelling correction module on the search engine.

Finally, even if the search succeeds, the user may ignore the result and chose to create a new place in order to use the checkin as a medium of expression. For example, the user may create a place called "*The Happiest Place on Earth*" (for Disneyland) or "*New York Rocks!*". In these cases also, the name similarity between the new place and p suffers.

Users do not want to see multiple copies of places appearing in the search results. Duplicates often compete for user checkins, resulting in fragmentation of the checkin activity of places across several copies. Also, while business owners might create places with a rich set of attributes, the user created duplicates often lack detailed attributes, resulting in poor attribute coverage. As a result, in order to provide a good user experience, deduplication is an important problem to solve for any such database.

Challenges of Deduplication: While deduplication is a

	place_1	place_2	issue
(a)	*Fresca*	*Fresca's Peruvian Restaurant*	Large edit distance.
(b)	*Newpark Mall Sears Outlet*	*Newpark Mall Gap Outlet*	Places with high string similarity might differ in key terms.
(c)	*San Francisco Airport*	*San Francisco Airport Terminal 1*	Nested Places are hard.
(d)	*Central Park*	*Central Park Cafe*	In New York City, near Central Park, these may well be different. In most other places, they are almost certainly the same.
(e)	*Sittin' on my sofa eating chips*	–	Junk places or places of personal interest introduce lots of noise.

Table 1: Examples of Deduplication Challenges

ubiquitous problem, the fact that our database predominantly consists of user generated content (UGC) makes it an extremely challenging problem. We illustrate here some of the challenges.

First, a typical deduplication algorithm makes uses of a large number of attributes, i.e. `name`, `address`, `city`, `phone`, `website`, `category` etc. In our setting, we do not have the luxury of using rich feature set. When users create places, they do not necessarily add detailed attributes. E.g., a user trying to checkin to a restaurant might not know the full street address, the business phone or the website. Although we might have a copy of the same place, created by the place owner, with all the details, we do not have the corresponding attributes in the duplicates to match against. Thus, we primarily have to rely just on the place name, along with the location coordinates, since every place creation operation is associated with a `lat` and `lon`.

While the presence of location coordinates with each entity might suggest a very trivial deduplication task, in reality, location coordinates themselves are extremely noisy. This is because users often create places from a physical location that is distant from the actual place locations. Even when the user is physically present at the place at the time of creation, there are often large GPS inaccuracies [12] in the reported user location. As an example of the extent of this issue, Fig. 1 shows all the duplications of the *Stanford University* that we detected in our database, along with the locations where they were created. There are duplicates that were created several miles away from the true location of the *Stanford University*. Given that more than 400K users have checked in to Stanford, it is not surprising that a very small fraction of checkins resulted in the creation of several duplicates, throughout the San Francisco Bay Area.

User generated content also implies that there is no standardization in naming places. This includes descriptions like "*GocHi Japanese Fusion Tapas*" vs. "*Gochi Asian Restaurant*", abbreviations like "*JF Kennedy Intl. Arpt.*", misspellings and alternate spellings, as so on.

Table 1 captures some of the challenges of deduplicating places based on names. Example (a) shows two places that are duplicates, but have a large edit distance. Term frequency analysis can partially alleviate the problem, by suggesting that *Restaurant* is a frequent token. But *Peruvian* is a rare token, almost as rare as *Fresca*. As we show later, a *TF-IDF* based similarity measure does not give us an acceptable accuracy. Examples (b) and (c) shows the reverse problem, where two place names having a high string similarity are, in fact, not duplicates. Example (d) is especially interesting: two places named *Central Park* and *Central Park Cafe* near each other are most likely duplicates in most parts of the world, where users might refer to the cafe as simply *Central Park* when there is no ambiguity. However, in New York, near Central Park, they are most certainly different, where one refers to the actual park, and the latter refers to some cafe in the vicinity of the park. Thus, geographical context matters a lot when determining if two places are duplicates. The final example just shows that there are lots of junk/personal places which introduce lots of noise in deduplication.

Problem Definition Based on the motivation, we define the following problem:

PROBLEM 1. *Given two place names n_1 and n_2, and an approximate geographical location containing both of them, determine if the two names can refer to the same entity.*

Note that, as suggested by the above problem formulation, we are focusing only on name matching in this paper. In our full deduplication system, we use name matching as one of the features (albeit the most important one), in conjunction with the distance between the places, the set of checkins, and other features like phone and address when available.

Our Approach Our solution to the place name matching problem has three main components. The first is a novel generative language model for places that identifies the *core* of a place name. E.g., it identifies that in "*Fresca's Peruvian Restaurant*", *Fresca's* is the core and *Peruvian Restaurant* is the description. Section 3 describes this model. The second component is a language model that incorporates the spatial context, e.g., the presence of nearby landmarks, parks, malls, airports, major streets and the current city. The spatial model helps us with distinguishing "*Newpark Mall Gap Outlet*" from "*Newpark Mall Sears Outlet*" and matching/non-matching *Central Park* and *Central Park Cafe*. We present the spatial context model in Section 4. The third component of our solution is a place name similarity measure based on our language models, which is described in Section 5.

3. A SIMPLE NAME MODEL

Let W be a set of words, which we call the vocabulary. Each place name, $n \subseteq W$, consists of a set of words. Let N be a set of names. We assume that each name n consists of a set of tokens, which we call *core*, and rest, which we call *background*. We denote them by $core(n)$ and $back(n)$ respectively. E.g., for the name

$$n = \text{“}\texttt{Guggenheim Art Museum, Manhattan}\text{”},$$

$core(n) = \{\texttt{Guggenheim}\}$, since this identifies the place, and $back(n) = \{\texttt{Art}, \texttt{Museum}, \texttt{Manhattan}\}$, which describe the place properties. On the other hand, for the name `Guggenheim Starbucks`, the core consists of `Starbucks` and `Guggenheim` is the background.

Given a name, we want to identify the core and the background part of the name. We assume the following generative model for names. Let B and C be two probability distributions over W. First, a word is chosen from W from the distribution C to be the core of the name. Next, an integer $k \geq 0$ is picked from some distribution. Finally, k tokens are drawn from the distribution B as the background words. The final name consists of the union of the core and the background.

Learning Problem Given a set of names N, our objective is the learn (1) the distributions B and C, and (2) for each name $n \in N$ and each token $w \in n$, the probability that w is the core of n. Let $z(w, n)$ denote the binary event that w is the core of n, i.e., $z(w, n) = 1$ if $core(w) = n$ and 0 otherwise. Let $\mathbf{z}$ denote the collection of all such binary events. Then, the probability of observing the given data N is

$$\mathbf{P}(N \mid B, C, \mathbf{z}) = \prod_{n \in N} \prod_{w \in n} C(w)^{z(w,n)}.B(w)^{(1-z(w,n))}$$

We want to learn $B, C, \mathbf{z}$ that maximize the probability of $\mathbf{P}(N \mid B, C, \mathbf{z})$, or alternatively, its logarithm, given by:

$$L(B, C, \mathbf{z}) = \sum_{n \in N} \sum_{w \in n} z(w, n) \log C(w) + (1 - z(w, n)) \log B(w)$$

Note that the training data simply consists of the set of names N. If we knew the core of each name, i.e., the variables $\mathbf{z}$, then estimating B and C would have been straightforward by simple counting. However, in the absence of this information, we revert to the Expectation Maximization (EM) method to solve this optimization problem. In the E-step, given the current parameters $B^{(t)}$ and $C^{(t)}$, we compute the expected value of L over all choices of $\mathbf{z}$. Since L is a linear combination of $\mathbf{z}$, we can use linearity of expectations. We observe that

$$\begin{aligned}
\mathbf{E}(z(w, n) \mid B^{(t)}, C^{(t)}) &= \mathbf{P}(z(w, n) = 1) \\
&= \frac{C^{(t)}(w) \prod_{x \in n \setminus w} B^{(t)}(x)}{\sum_{w_2 \in n} C^{(t)}(w_2) \prod_{x \in n \setminus w_2} B^{(t)}(x)} \\
&= \frac{C^{(t)}(w)/B^{(t)}(w)}{\sum_{w_2 \in n} C^{(t)}(w_2)/B^{(t)}(w_2)}
\end{aligned}$$

Let $z^{(t)}(w, n)$ denote $\mathbf{E}(z(w, n) \mid B^{(t)}, C^{(t)})$. In the M-step, we compute the $B^{(t+1)}, C^{(t+1)}$ that maximizes the expectation of the objective function. This is a standard optimization problem, whose solution is given by

$$\begin{aligned}
C^{(t+1)}(w) &= \frac{\sum_{n|w \in n} z^{(t)}(w, n)}{\sum_{w' \in W} \sum_{n|w' \in n} z^{(t)}(w', n)} \\
B^{(t+1)}(w) &= \frac{\sum_{n|w \in n} (1 - z^{(t)}(w, n))}{\sum_{w' \in W} \sum_{n|w' \in n} (1 - z^{(t)}(w', n))}
\end{aligned}$$

The final algorithm is given in Algorithm 1. We illustrate with a simple example. Consider three place names given in the table below:

n_1 :	`Starbucks Coffee`
n_2 :	`Peets Coffee`
n_3 :	`Starbucks`

Our vocabulary, $W = \{\texttt{Starbucks}, \texttt{Peets}, \texttt{Coffee}\}$. The token `Starbucks` is as frequent as `Coffee`. However, `Starbucks` appears by itself in n_3, and hence we expect it to be likely a core term. Based on this knowledge, in n_1, we expect `Coffee` to be a background term. This, in turn, suggests that `Peets` should be a core term in n_2. Algorithm 1 captures this intuition. Initially, $B^{(1)}$ and $C^{(1)}$ are uniform distributions. Thus, we have $z^{(1)}(\texttt{Starbucks}, n_1) = z^{(1)}(\texttt{Coffee}, n_1) = \frac{1}{2}$, $z^{(1)}(\texttt{Peets}, n_2) = z^{(1)}(\texttt{Coffee}, n_2) = \frac{1}{2}$, and $z^{(1)}(\texttt{Starbucks}, n_1) = 1$. Based on this, we get $C^{(2)} = \{\texttt{Starbucks} \leftarrow \frac{1}{2}, \texttt{Peets} \leftarrow \frac{1}{6}, \texttt{Coffee} \leftarrow \frac{1}{3}\}$ and $B^{(2)} = \{\texttt{Starbucks} \leftarrow \frac{1}{4}, \texttt{Peets} \leftarrow \frac{1}{4}, \texttt{Coffee} \leftarrow \frac{1}{2}\}$. Thus, the core probability of `Starbucks` and the background probability of `Coffee` increases. The algorithm converges to the following solution:

$$\begin{aligned}
C &= \{\texttt{Starbucks} \leftarrow \frac{2}{3}, \texttt{Peets} \leftarrow \frac{1}{3}, \texttt{Coffee} \leftarrow 0\} \\
B &= \{\texttt{Starbucks} \leftarrow 0, \texttt{Peets} \leftarrow 0, \texttt{Coffee} \leftarrow 1\}
\end{aligned}$$

Thus, for this hypothetical example, the algorithm drives down the core probability of `Coffee` to 0 and background probabilities of `Starbucks` and `Peets` to 0. Note that the constraint of one core token in each name is crucial for learning to happen, since identifying a core token gives us (negative) information about all other tokens that co-occur with it. Without the constraint, both the core and the background models will converge to exactly the document frequencies.

Algorithm 1 EM (**Inputs:** Vocabulary W, names N)

1: $B^{(1)} \leftarrow$ uniform distribution
2: $C^{(1)} \leftarrow$ uniform distribution
3: **for** t = 1 .. N **do**
4: $\quad z^{(t)}(w, n) \leftarrow \frac{C^{(t)}(w)/B^{(t)}(w)}{\sum_{w_2 \in n} C^{(t)}(w_2)/B^{(t)}(w_2)}$
5: $\quad C^{(t+1)}(w) \leftarrow \frac{\sum_{n|w \in n} z^{(t)}(w,n)}{\sum_{w' \in W} \sum_{n|w' \in n} z^{(t)}(w',n)}$
6: $\quad B^{(t+1)}(w) \leftarrow \frac{\sum_{n|w \in n} (1-z^{(t)}(w,n))}{\sum_{w' \in W} \sum_{n|w' \in n} (1-z^{(t)}(w',n))}$
7: **end for**

4. SPATIAL CONTEXT MODEL

Place names often include the names of nearby landmarks, parks, malls, airports, major streets, the location city, and

so on. E.g. consider two places "`Newpark Mall Gap Outlet`" and "`Newpark Mall Sears Outlet`". While the two names look very similar textually, they are different businesses, both of which include the name of the mall they are located in. As another example, consider two places "`Copenhagen Cafe`" and `Copenhagen Bakery`". If both the candidates are in New York, they most likely refer to the same business. But in Copenhagen, Denmark, they most likely different places.

In this section, we describe an unsupervised method to identify the spatial context in any given region.

Backstorm et al. [1] have looked at the problem of how search engine queries vary across geographic regions. They posit a model where each query has a geographic center represented by a single point. This center corresponds to the point at which the query should occur most frequently, with frequency falling off with distance from the center. In addition to center, each query has two other parameters: a constant, C, giving the frequency at the query's center, and an exponent α determining how quickly the frequency falls off as one gets further away from the center. They assume that the probability of a random user at distance d issuing the query is proportional to $Cd^{-\alpha}$. Then they seek the parameters that best fit the data.

In principle, we can apply the same techniques to study the distribution of place names. However, in contrast to search engine queries that have high dispersion, we observe that the spatial contexts in place names are tightly concentrated in specific regions, corresponding naturally to the physical spans of the contexts. Figures 2,3,4 show all the places in our database containing the phrases "`Times Square`", "`Central Park`" and "`Broadway`" respectively, plotted on the map of New York. We see that `Times Square`, which is a point place, exhibits a natural center. On the other hand, `Central Park` has no center, and we see a constant density within the physical span on the park that decays off outside the span. For `Broadway`, we see a more extreme form of this behavior. The term is highly concentrated around the entire span of the Broadway street in New York. We even see two branches going across, which correspond to two other, lesser known, Broadway streets in the city.

In conclusion, we need to handle arbitrary spans of terms in modeling spatial context. Note that for our deduplication task, we don't need to solve the problem of identifying the span of any given term. Instead, we need to solve the reverse problem of given a location, what is the distribution of contextual terms in that location. Thus we take a simpler approach. We divide the entire world into tiles, and try to learn the distribution of terms in each tile.

Formally, a tile l is a geographical region. We assign each place a tile based on the location at which the place was created. Thus, each place is represented by a (n, l) pair, where n is the name and l is the tile. We want to learn a background contextual model $B[l]$ for each tile l that captures the probability distribution of background terms for each tile l. Suppose we are given the set of background terms, $back(n)$, for each place name n. The maximum likelihood

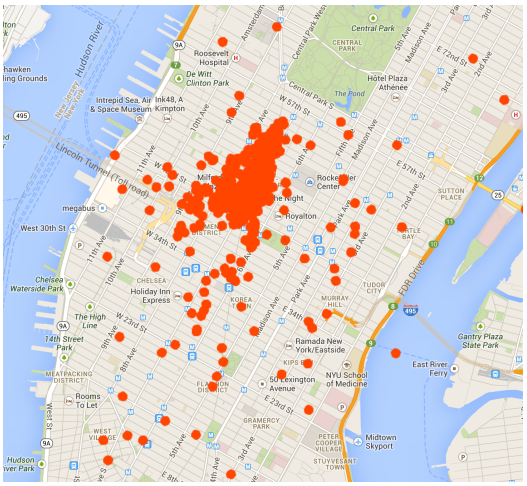

Figure 2: Times Square, NY

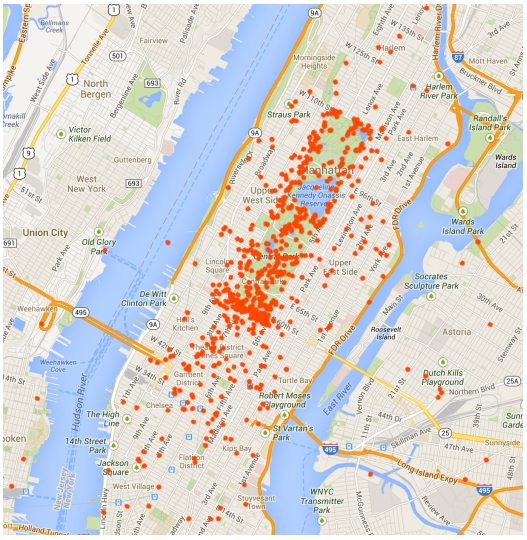

Figure 3: Central Park, NY

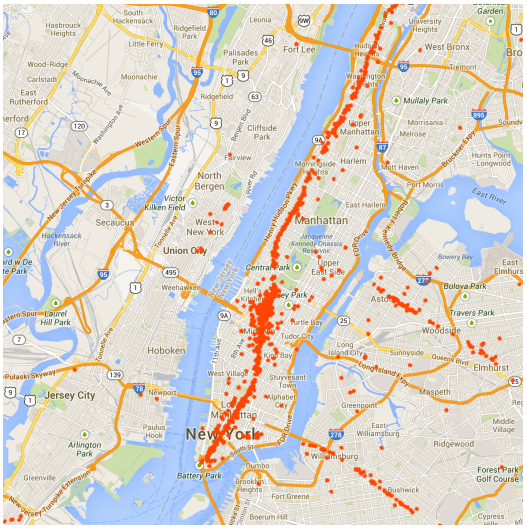

Figure 4: Broaway, NY

estimate for $B[l]$ is simply the relative counts,

$$B^l_{\texttt{ml}}(w) = \frac{count(w;l)}{\sum_w count(w;l)}$$

where $count(w;l)$ is the number of times token w appears in $back(n)$ in tile l. However, this can lead to overfitting. Therefore, we borrow techniques from Information Retrieval for smoothing document models. We linearly interpolate the maximum likelihood estimate for a tile with the global background model B, i.e.,

$$B[l](w) = \lambda B^l_{\texttt{ml}}(w) + (1-\lambda)B(w) \quad (1)$$

where B is the background model as described in Section 3.

4.1 Combining the Models

For defining the spatial model in previous section, we assumed that we already know the background terms in each name. We can first learn the name model as described in Section 3, and then use it to infer the spatial model. However, there is a cyclic dependency as the name model itself may benefit from a spatial model. E.g., consider a mall, called *Newpark Mall*, containing a large number of businesses containing the mall name. Without the spatial context, for each such place, the name model might infer that *Newpark* is the core term, as *Newpark* has a very low global frequency. Since the term *Newpark* doesn't appear in the background for any of the places, the spatial model doesn't learn that it is a contextual term for the location.

To resolve the cyclic dependency, we modify Algorithm 1 to jointly learn both models. We assume that we have a set of tiles L and a set of places P, where each $p \in P$ has a name $p.n$ and a tile $p.l$ containing the place.

Algorithm 2 EM (**Inputs:** Vocabulary W, places P, tiles L)

$B^{(1)} \leftarrow$ uniform distribution
$C^{(1)} \leftarrow$ uniform distribution
$B[l]^{(1)} \leftarrow$ uniform distribution for all $l \in L$
for t = 1 .. N **do**
  $z^{(t)}(w,p) \leftarrow \frac{C^{(t)}(w)/B[p.l]^{(t)}(w)}{\sum_{w_2 \in p.n} C^{(t)}(w_2)/B[p.l]^{(t)}(w_2)}$
  $C^{(t+1)}(w) \leftarrow \frac{\sum_{p|w\in p.n} z^{(t)}(w,p)}{\sum_{w'\in W}\sum_{p|w'\in p} z^{(t)}(w',p)}$
  $B^{(t+1)}(w) \leftarrow \frac{\sum_{p|w\in p.n}(1-z^{(t)}(w,p))}{\sum_{w'\in W}\sum_{p|w'\in p.n}(1-z^{(t)}(w',p))}$
  $B[l]^{(t+1)}_{\texttt{ml}}(w) \leftarrow \frac{\sum_{p|w\in p.n, p.l=l}(1-z^{(t)}(w,p))}{\sum_{w'\in W}\sum_{p|w'\in p.n, p.l=l}(1-z^{(t)}(w',p))}$
  $B[l]^{(t+1)}(w) \leftarrow \lambda B[l]^{(t+1)}_{\texttt{ml}}(w) + (1-\lambda)B^{(t+1)}(w)$
end for

In each iteration, in the E step, the algorithm computes the expected probability of each token being a background token based on the current name and spatial context model. In the M step, the algorithm updates the models based on the background token probabilities.

5. DEDUPLICATING PLACES

We next show how the language models from the previous sections, which gives the background probabilities for tokens in place names, can be combined with string edit operations to solve the task of places deduplication.

Our task is to determine, given two place names in a given location, whether they can refer to the same business. We formalize the problem as : given two places, determine if they have the same core. In Section 3, we worked with the constraint that exactly one word in each name is core. As we explained, the constraint was crucial for us to learn good core and background models. Now that we have the language models for the core and the background, we relax the constraint here, and allow names to have multiple core words. Given a place p, we assume that each token in the name $p.n$ is independently drawn from the core model C with probability α, and from the background model $B[p.l]$ with probability $1-\alpha$, where $p.l$ denotes the location of p. Let $core(w,p)$ denote the event that the word w in p is a core word. Then,

$$\mathbf{P}(core(w,p)) = \frac{\alpha C(w)}{\alpha C(w) + (1-\alpha)B[p.l](w)} \quad (2)$$

We use $c(w,p)$ to denote the probability $\mathbf{P}(core(w,p))$.

Let $core(p)$ denote the random variable containing the core tokens of place p. Given two places, p_1 and p_2, they have same core iff (1) all tokens belonging to their symmetric difference are from background, and (2) every token common to them is either a core in both or none. This can be written as $\mathbf{P}[core(p_1) = core(p_2)] =$

$$\left(\prod_{w\in p_1 \setminus p_2} 1 - c(w,p_1)\right) \times \left(\prod_{w\in p_2 \setminus p_1} 1 - c(w,p_2)\right) \times \quad (3)$$

$$\left(\prod_{w\in p_1 \cap p_2} c(w,p_1)c(w,p_2) + (1-c(w,p_1))(1-c(w,p_2))\right)$$

Incorporating Edit Operations So far we have focused on identifying the core of a given name. Next, we describe how to modify Eq. (3) to incorporate string edit operations. We assume that we have a set of edit operations E. Each operator takes as input a sequence of tokens, and produces a new sequence of tokens. Examples of edit operations that we use in our system include abbreviations, e.g. `North West` → `NW`, concatenations, e.g. `North West` →`Northwest` and character edits, e.g. `Guggenheim` → `Guggenheim`.

We consider the following generative model for place names. We are given a set of edit operations E, along with a probability function $\pi : E \to [0,1]$. Given a place p, each token in the name is independently drawn from the core model C with probability α, and from the background model $B[p.l]$ whit probability $1-\alpha$. Then, each edit $e \in E$ is chosen with probability $\pi(e)$ and applied to the resulting name. Given two places, we want to compute the probability that they have the same core.

As an example, consider two places, p_1="`Dish Dash Restaurant, Sunnyvale`" and p_2="`DD Sunnyvale`". There are several possible ways for the two places to have the same core. E.g. (1) `Dish Dash` is core in p_1 that gets edited to `DD` in p_2, rest are all background, (2) `Dish Dash Sunnyvale` is core in p_1, `DD Sunnyvale` is core in p_2 and `Restaurant` is background, (3) `Sunnyvale` is the common core and everything else is background, and so on. We want to find the proba-

bilities of all possible worlds where the two places have the same core.

We present an algorithm based on dynamic programming to solve this problem. Given an edit operation e, let $l(e)$ denote the number of tokens it consumes, and $r(e)$ denote the number of tokens it produces. E.g., for the edit operation that concatenates two tokens, $l(e) = 2$ and $r(e) = 1$. Similarly, for the misspelling operator, $l(e) = r(e) = 1$. We assume that for each type of edit operation e, we have a probability $\pi(e)$ associated with it.

Let p_1, p_2 be two given places, and let l_1 and l_2 be the number of tokens in their names respectively. Let $p.n[i, j]$ denote the subsequence of tokens in $p.n$ from token i to $j - 1$. To simplify the description of our algorithm, we define three new types of operators. For each $w \in p_1.n$, we define a $delete_w$ operator that deletes the word w, and has $\pi(delete_w) = 1 - c(w, p_1)$. For each $w \in p_2.n$, we define an $insert_w$ operator that inserts the word w, and has $\pi(insert_w) = 1 - c(w, p_2)$. Finally, for each $w \in p_1.n \cap p_2.n$, we define a $copy_w$ operator with $\pi(copy_w) = c(w, p_1)c(w, p_2)$. Clearly, $l(delete_w) = 1$, $r(delete_w) = 0$, $l(insert) = 0$, $r(insert) = 1$ and $l(copy) = r(copy) = 1$. The intuition behind these operators is that a copy operator asserts that the word is a core word in both names, delete asserts that the word is a background in the first name, and insert operator asserts that the word in a background in the second name. The final algorithm is presented below.

Algorithm 3 Similarity (**Inputs:** p_1, p_2)

1: **return**DP$(p_1, p_2, 0, 0)$

Algorithm 4 DP (**Inputs:** p_1, p_2, i, j)

1: **if** $i \equiv l_1$ and $j \equiv l_2$ **then**
2: **return**1
3: **end if**
4: $ret \leftarrow 0$
5: **for** $e \in E$ **do**
6: $W_1 = p_1.n[i, i + l(e)]$, $W_2 = p_2.n[j, j + r(e)]$
7: **if** $e(W_1) \equiv W_2$ **then**
8: $ret \leftarrow ret + \pi(e).DP(i + l(e), j + r(e))$
9: **end if**
10: **end for**
11: **return**ret

One can verify that Algorithm 4 generalizes Eq (3). In other words, when the set of edit operations E is empty (we still define the insert, delete and copy operators), then the score returned by the algorithm is exactly the expression in Eq (3). Indeed, for each token in the first name that doesn't occur in the second, the only choice is to delete it (with cost $1 - c(w, p_1)$. Similarity, for each token in the second name that does not occur in the first, the only choice is to insert it (with cost $1 - c(w, p_2)$). For each token in common, we can either copy it (with cost $c(w, p_1)c(w, p_2)$), or delete it and insert it again (with cost $(1 - c(w, p_1))(1 - c(w, p_2))$. Thus, we recover the expression in Eq (3).

6. EXPERIMENTS

We implemented our techniques on two datasets. The first dataset, which we call WikiMapia, consists of entities from WikiMapia [9], which is an open-content collaborative mapping project that aims to mark and describe all geographical objects in the world. The dataset has over 20M objects, and contains the name of the place, along with its latitude and longitude. We chose a subset of the data corresponding to entities in USA, to make the task of manual labeling and evaluation easy. The resulting dataset has around 1M entities.

The second dataset, which we call Places, consists of all entities from the Places Database [5] used in production at Facebook. Again, we restricted ourselves to entities in USA. For each entity, we use the name of the entity along with the latitude and longitude where it was created.

For both the datasets, we learn the name model as described in Section 3 as well as the context model as described in Section 4. For learning the context model, we divided the world map into geographical tiles based on the latitudes and longitude. For the task of learning language models, we use both the datasets for evaluation. For the task of deduplication, we restrict ourselves to Places, since WikiMapia is not interesting for the deduplication task.

We created two labeled sets for the deduplication task, one for training and another for evaluation. For each set, we randomly sampled 1500 entities from Places. For each sampled place, we generated candidate duplicates from Places with a very liberal distance and name similarity threshold. Each of the candidates was editorially inspected and labeled as a duplicate or not a duplicate. Each dataset contains around 7K candidate pairs, of which around 2K are labeled as true duplicates.

Note that the training of our language models is unsupervised, and hence done on the entire unlabeled dataset. In addition, our model uses a small set of parameters, e.g., the probabilities of individual edit operations, the granularity of tiles, and the weight for smoothing the context model (Eq. (1)). We use one set of labeled data to tune these parameters. The second labeled set is used for evaluation.

6.1 Name Model

We first evaluate the name model that we described in Section 3. For this task, we consider the task of classifying tokens as descriptive vs core. For training, we used the entire unlabeled dataset. For evaluation, we randomly sampled tokens from the vocabulary of the database, as manually labeled them as either a core token (e.g. `starbucks`, `guggenheim` and `maggianos`) or descriptive (e.g. `cafe`, `museum` and `italian`). Note that this evaluation is independent of the context of a name. E.g. there might be a place name with token `guggenheim` as a descriptive token. Context dependent evaluation will be the subject of the next section.

As a baseline, we consider an IDF classifier that classifies a token as descriptive if its frequency in the corpus is greater than some threshold. We vary the threshold, and plot its precision vs recall on the evaluation set.

For our method, we use the distributions B for background and C for core that we learn from the name model, along with a class prior of α for C. Thus, the posterior probability

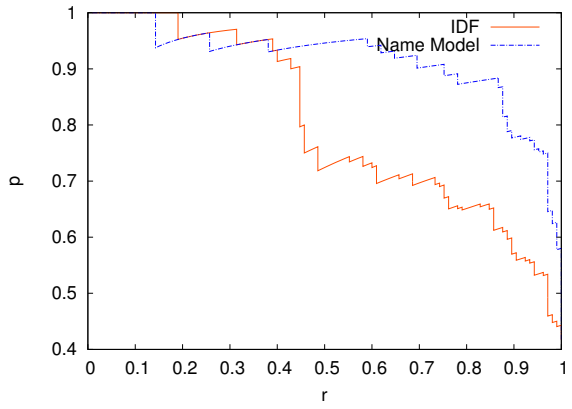

Figure 5: Accuracy of the core model on WIKIMAPIA

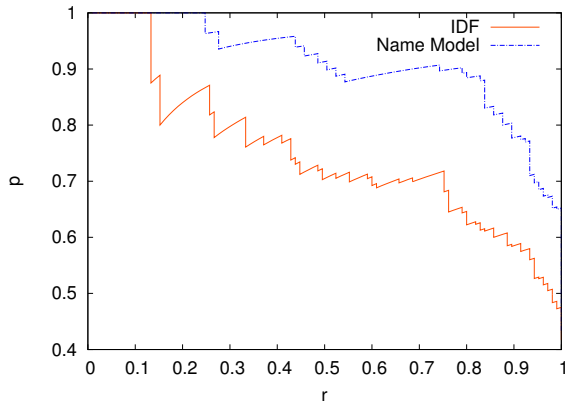

Figure 6: Accuracy of core model on PLACES

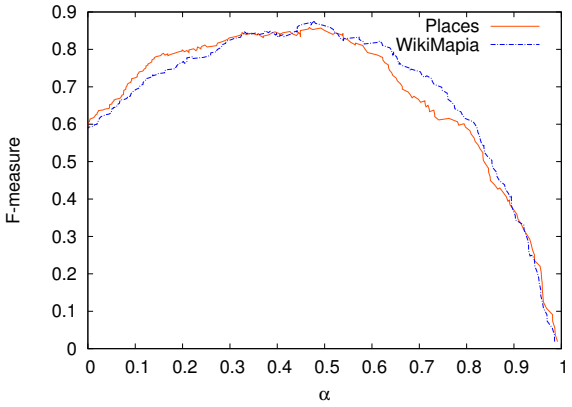

Figure 7: F-measure vs α

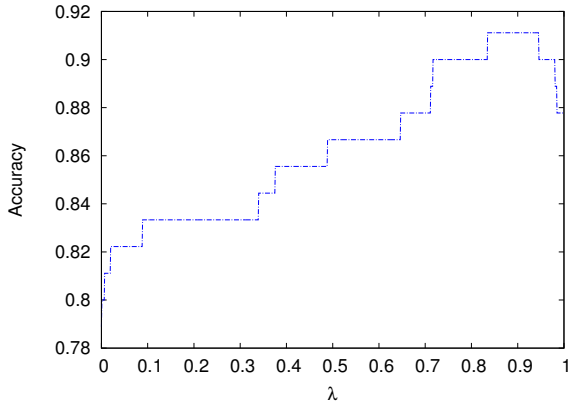

Figure 8: Accuracy of Identifying Core on PLACES as a function of λ

of a token w being core is

$$\frac{\alpha C(w)}{\alpha C(w) + (1-\alpha)B(w)}$$

A value of $\alpha = 1$ will make all words core with high probability and a value of $\alpha = 0$ will make all words background. Again, we vary α and study the precision and recall for this classifier. Figure 5 shows the plots for the IDF classifier and the name model on WIKIMAPIA dataset. Figure 6 shows the same plot on PLACES dataset.

We see that on both datasets, name model significantly outperforms IDF. The performance of IDF on WIKIMAPIA data is better than on PLACES data, possibly owing to the larger scale and diversity in PLACES, but the name model performance is at par on both.

The peak F-measure (harmonic mean of the precision and the recall) for our algorithm is around 0.85, which is achieved at $\alpha \approx 0.5$ on both the datasets. Figure 7 shows the F-measure as a function of α. This suggests a class prior of $\alpha = 0.5$, i.e., on average, names contain equal number of core and background tokens.

6.2 Context Model

Next, we look at the task of identifying core tokens in the context of a name. We learn a context model as described in Section 4 based on the entire unlabeled data. For evaluation, we randomly sampled a set of places, and labeled the core tokens in each. Then we used the context model to identify the core in each place name, taking into account the place location, using Equation (2).

A parameter that we need to choose is λ, defined in Eq. (1), which controls the smoothing between the local model and the global background model. For each choice of λ, we evaluate the accuracy of our model in identifying the core in the context of a name, where accuracy is simply the fraction of names were we correctly identified the core. Using $\lambda = 0$ amounts to not using the local model at all, while $\lambda = 1$ implies that the global model is completely ignored. Figures 8 and 9 show the accuracy of identifying core as a function of λ on PLACES and WIKIMAPIA respectively. We see that the

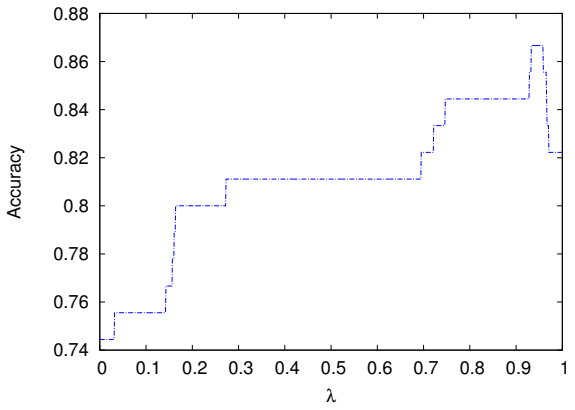

Figure 9: Accuracy of Identifying Core on WIKIMAPIA as a function of λ

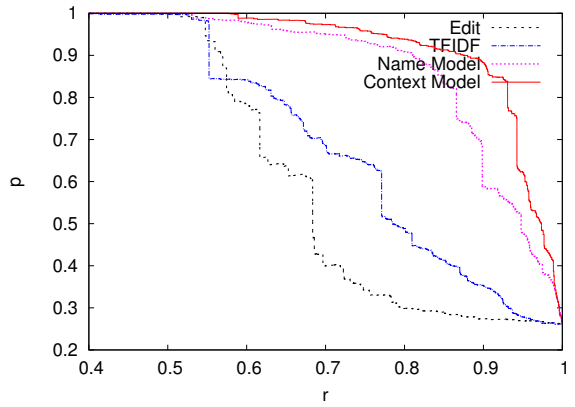

Figure 10: Deduplication Accuracy for Various Techniques on PLACES

peak accuracy is around 91% and 87% in the two datasets, corresponding to roughly the same value of $\lambda \approx 0.9$. Also note that context matters a lot, as the accuracies for $\lambda = 0$ are only 80% and 75% respectively.

6.3 Deduplication

Finally, we look at the our end-goal, which is place deduplication. We want to check if our effectiveness in identifying the cores of place names translates into accurate deduplication. We consider two baselines, *Edit*, that uses the Levenshtein distance between the two place names as the similarity metric, and *TF-IDF*, which uses the TF-IDF cosine similarity between the place names. We also consider two variants of our algorithm : *Context Model* refers to our complete model based on Algorithm 3, and *Name Model*, which is a variant of Algorithm 3 that does not use the local context. We observe that the *Name Model* significantly outperforms baselines based on edit distances and cosine similarities. Further, context helps a lot. We are able to achieve a recall of 90% at a precision of 90%.

7. RELATED WORK

String comparison for matching is a relatively well-studied field. In [7], the authors divide techniques into three categories, 1) edit-distance-like functions, 2) token-based distance functions and 3) hybrid functions. In [15], the authors describe a system for geographic name disambiguation, focusing on the names of towns and landmarks. The authors use traditional string match techniques - Jaccard, Jaro Winkler, and an Edit Distance. They note that their "highest F-measure is obtained with a threshold of .5" and observe that "many matching locations have significant differences in their names".

The simple name model of Section 3, the spatial-context model of Section 4 and the combined model are all techniques to produce token weights, and in Section 6, we compare our technique to the TF-IDF based similarity using the cosine similarity of the weighted scores as in [7] (and many others). Other weighting techniques have been proposed, including the "Learnable Vector-space Similarity" technique of [4]. This technique learns token weights by using the TF-IDF weights of matching tokens as input features to an SVM classifier to learn new weights. Fundamentally, the difference is that our weighting schemes compute *different* weightings for each word in each business name, based either on the role of the word or on the *spatial context*. Since, to our knowledge, all token weighting schemes in the literature including [4], but excepting [6], compute the *same* weight for a given token no matter where it is used in the matching task, our approach is not directly comparable to these techniques. In [6], the authors introduce a *context-sensitive* technique, but the form of context considered in their work is to assume a low probability of dupes within a particular *source* of data, quite orthogonal to the notion of *core* words or the *spatial* context of Section 4.

The place-similarity metric defined in Section 5 is properly a hybrid technique that incorporates both character edit weights and token copy, insert and delete weights. These techniques are similar to the notion of a *learned* customized matching functions, to which we now compare. Techniques that learn simple transition probabilities for edit models, including [13, 11] or with affine gaps as in the first technique of [4], are substantially different from our approach given the focus on a single set of parameters and the need for supervision. Like all other token schemes, this scheme gives a fixed weight to each token, unlike our technique that gives a variant weight based on context. Finally [10] also learns a discriminative edit distance, but this distance can be based on features of each word. An interesting topic of future work is to incorporate our core word and context metrics as features in an adaptation of this system.

In summary, we are not aware of any previous work that computes *core words* or term weighting based on *spatial context*, nor related work that achieves high accuracy on matching business names, especially not noisy ones.

8. CONCLUSION

In this paper, we described an approach to address the challenging problem of deduplicating places. Our approach consists of a novel unsupervised language model that captures both the domain knowledge as well as the spatial context. We experimentally showed the effectiveness of out techniques on real data. Our techniques are used in production at Facebook to maintain the Facebook Places data.

9. REFERENCES

[1] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *Proceedings of the 19th international conference on World wide web*, pages 61–70. ACM, 2010.

[2] K. Bellare, C. Curino, A. Machanavajihala, P. Mika, M. Rahurkar, and A. Sane. Woo: A scalable and multi-tenant platform for continuous knowledge base synthesis. *Proceedings of the VLDB Endowment*, 6(11), 2013.

[3] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *Intelligent Systems, IEEE*, 18(5):16–23, 2003.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[5] J. Chang and E. Sun. Location3: How users share and respond to location-based data on social networking sites. In *Proceedings of the International Conference on the Weblogs and Social Media (ICWSM?11)*, 2011.

[6] W. W. Cohen, N. Glance, C. Schafer, R. Tromble, and Y. W. Wong. *Data Integration for Many Data Sources using Context-Sensitive Similarity Metrics*. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2011.

[7] W. W. Cohen, P. D. Ravikumar, S. E. Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, volume 2003, pages 73–78, 2003.

[8] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

[9] A. Koriakine and E. Saveliev. Wikimapia. *Online: wikimapia. org*, 2008.

[10] A. McCallum, K. Bellare, and F. Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. *arXiv preprint arXiv:1207.1406*, 2012.

[11] J. Oncina and M. Sebban. Learning stochastic edit distance: Application in handwritten character recognition. *Pattern Recognition*, 39(9):1575–1587, 2006.

[12] B. W. Parkinson. Gps error analysis. *Global Positioning System: Theory and applications.*, 1:469–483, 1996.

[13] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.

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

[15] V. Sehgal, L. Getoor, and P. D. Viechnicki. Entity resolution in geospatial data integration. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 83–90. ACM, 2006.