Learning to Rank for Spatiotemporal Search

Blake Shaw, Jon Shea, Siddhartha Sinha, Andrew Hogue
Foursquare
568 Broadway
New York, NY
{blake, jonshea, ssinha, ahogue}@foursquare.com

ABSTRACT

In this article we consider the problem of mapping a noisy estimate of a user's current location to a semantically meaningful point of interest, such as a home, restaurant, or store. Despite the poor accuracy of GPS on current mobile devices and the relatively high density of places in urban areas, it is possible to predict a user's location with considerable precision by explicitly modeling both places and users and by combining a variety of signals about a user's current context. Places are often simply modeled as a single latitude and longitude when in fact they are complex entities existing in both space and time and shaped by the millions of people that interact with them. Similarly, models of users reveal complex but predictable patterns of mobility that can be exploited for this task. We propose a novel spatial search algorithm that infers a user's location by combining aggregate signals mined from billions of foursquare check-ins with real-time contextual information. We evaluate a variety of techniques and demonstrate that machine learning algorithms for ranking and spatiotemporal models of places and users offer significant improvement over common methods for location search based on distance and popularity.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases; H.2.8 [**Database Applications**]: Data Mining; I.5.1 [**Pattern Recognition**]: Statistical

Keywords

spatial search, machine learning, learn to rank, spatiotemporal models, location data, data mining, geocoding, information retrieval, foursquare, human mobility, mobile devices

1. INTRODUCTION

Today's location-aware mobile devices have dramatically increased the availability and usefulness of location information. Many applications take advantage of the ability of these devices to report a user's location in terms of latitude and longitude, providing useful services to the user such as maps, driving directions, tourist guides, photo sharing, and more. Other services allow users to store content that has been "geotagged" with a location and share it with friends.

Several applications go one step further than raw coordinates, attaching a *semantically meaningful* name to a location. The coarsest form of this labeling involves using a reverse geocoding service [2] to find a city or neighborhood name for a given location. Examples include Twitter, which allows users to attach a location to a tweet, displaying it as a city or neighborhood, and Facebook, which tags most posts by default with a user's current city.

These coarse location names provide some context, but in many cases it is desirable to be significantly more granular when choosing a name for a location. For example, Instagram allows users to choose a specific location when sharing a photo, such as "Washington Square Park" or "The Blind Tiger." Similarly, foursquare, Path, and other mobile applications allow users to "check in" at a specific location such as a restaurant or museum. Tying users and their data to specific, semantically meaningful locations enables these services to provide richer experiences, such as showing the user relevant and timely information like a menu from a restaurant or a tip from a friend about what to order [31]. Moreover, identifying precise locations facilitates the sharing and aggregation of local information. For example, a user of foursquare can see not only recent places their friends have checked into, but also which of their friends have endorsed nearby venues.

Unfortunately, mapping a user's location to a database of known points of interest is complicated by several factors. While location services such as GPS, WiFi, and cell-tower triangulation can provide accuracies under 10 meters under ideal conditions [35], results in real-world environments are substantially worse, with median accuracies of 70 meters (see Figure 1). Additionally, many areas of the world are extremely dense in terms of semantically meaningful locations; in many urban environments, interesting locations may even be located directly above or below each other. Finally, while a user's history may be useful in determining their location [7, 18], the system must still provide accurate results even for users with little or no history.

Despite these difficulties, there are many benefits to an accurate system for finding specific, semantically meaningful names for a given set of coordinates. Tuning such a system for high recall in the top few results allows users to choose their location from a list of results and more easily share and

WSDM'13, February 4–8, 2012, Rome, Italy.

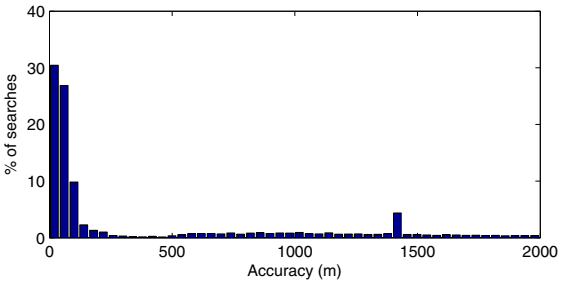


Figure 1: Distribution of reported location accuracy from 26,532 randomly sampled worldwide searches on foursquare. The median is 70 meters, and the mean is 551 meters.

retrieve local information. Alternatively, a system with high precision in the first position could be used to automatically label a user's location or content with a specific location name, or to tie together multiple pieces of content into a coherent grouping about a particular place. Furthermore, such a system could be used to trigger highly contextual notifications, such as transit information when a user enters a train station, or targeted deals when a user enters a store.

In this article, we describe the development, training, and evaluation of the spatial search engine that powers the foursquare application and API. We first introduce our approach to modeling people and places using check-in data, and then describe how these models can be used for retrieval and ranking. We focus on using machine learning techniques to create an optimal ranking function, which learns from large amounts of implicit feedback. Finally, we demonstrate the fully working system and evaluate its performance in a variety of real-world situations, showing significant improvement over common methods for location search and previously reported results.

1.1 Related Work

The growing ubiquity of GPS-enabled consumer mobile devices has only recently enabled the study of user location at the scale described in the current work. Lian and Xie [18] describe a similar system for mapping a user to a point of interest (POI) trained on data from 545 users in a single city using a database of approximately 16,000 points of interest. The system uses several features, such as distance, time, POI popularity, and user history, to train a model that predicts a user's location with 64.5% recall in the top 5 results. Our work improves these results by introducing several new features, more robust models, and significantly more training data (in terms of both size and diversity) to search a much larger database of locations.

Similar techniques regarding context and personalization have been applied in the area of local business search and web search. Lane et al. [17] report relevance improvements of up to ten times in local search relevance by including simple context and behavioral similarity features in a ranking model. Church and Smyth [10] demonstrate the efficacy of including user-, location-, and time-dependent features in mobile web search. Many techniques incorporate signals mined from query logs to improve search results [34, 32].

Outside of the realm of search, significant work has been done with respect to analyzing and describing GPS data

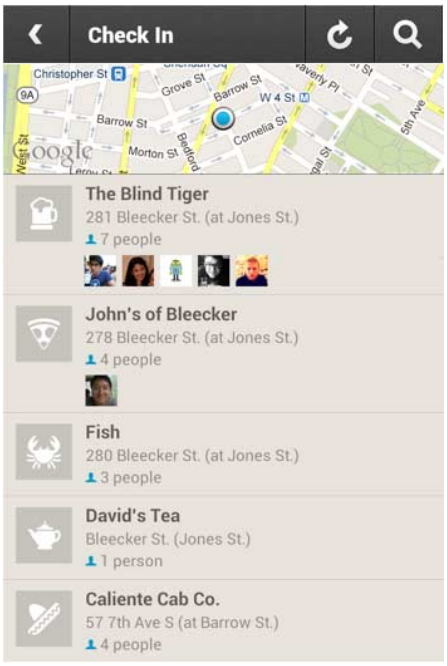


Figure 2: The check-in screen of the foursquare application where a user searches for nearby venues and selects a venue to check in to.

collected over time from users. Ashbrook and Starner [6, 7] cluster GPS locations to model significant locations and predict user movements using a Markov model. Gonzalez et al. [13] develop similar models to predict where a user will be based on their history. Liao et al. [19] predict significant locations for a given user using hierarchical conditional random fields, while Marmasse and Schmand [21] describe a similar system to collect GPS trails and prompt the user to label the significant locations. Furthermore, many techniques combine social and location-based signals to significantly boost the accuracy of models for predicting user location as well as determining attributes of social relationships [30, 8, 28, 20].

Geolocating other types of data has also received substantial attention. Systems for mapping text to a location have been described for tweets [9], blogs [11], and web pages [5] using a variety of lexicographic and user features. Similarly, Serdyukov et al. [29] describe techniques for mapping photos to geographic coordinates using language models and user annotations.

1.2 Venue Search in Foursquare

Foursquare is a location-based service which handles more than 5 million check-ins each day, mapping each to one of more than 40 million locations worldwide. Foursquare has accumulated over 2.5 billion historical check-ins.[1] These actions are enabled by a search engine that maps user locations into a database of points of interest with high accuracy. This service is also provided as an API for external developers to add location mapping to their applications [1].

Figure 2 shows the check-in screen from the foursquare

[1]As of August 2012.

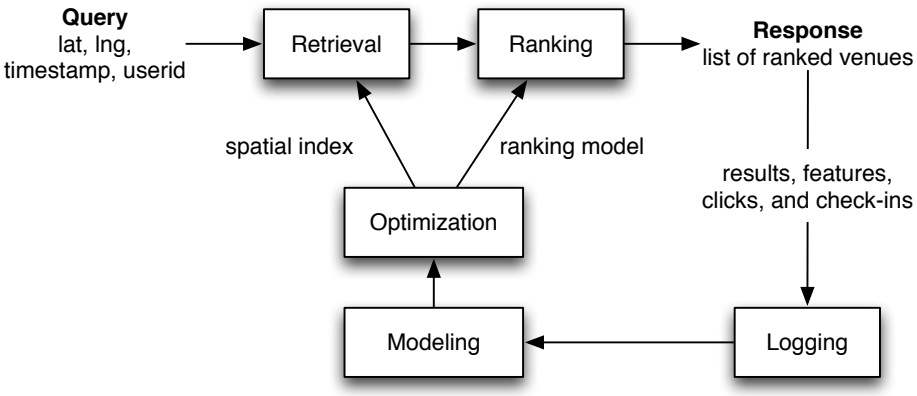


Figure 3: Overview of our system architecture. There are two main stages to responding to a query to produce a response: retrieval, where a spatial index is used to generate a set of candidate venues, and ranking where a machine-learned ranker is used to sort the response venues. Data is then logged from the search process and used offline for modeling and optimization.

application. Upon pressing the "Check in" button, the application sends the user's ID and current location to the server. The user ID is a unique identifier that is used to retrieve the user's history, friends, interests, and other personalized information used for ranking. The user's location is reported by the mobile device, and includes latitude, longitude, and a horizontal accuracy reading. A timestamp is generated at query time on the server in UTC, and is used to identify historical patterns in venue popularity for ranking.

The interface presents the user with a list of nearby locations, along with a map for context. The locations shown are chosen based on a variety of factors outlined below, including their popularity, distance from the user, and compatibility with the user's history. In particular, the current number of other users checked into each venue is displayed, which can help give the user an indication of the current popularity of the venues. For similar reasons, avatars for any of the user's own friends that are currently checked in are also shown. An icon indicating the location's primary category (e.g. Pub or Pizza Place) is also displayed to provide context for the user.

In certain situations, if the location in which the user is interested is not returned, they may perform an explicit textual query to find the correct venue. We term these searches *query* searches, to differentiate them from the *nearby* searches described above. We do not describe the system for query searches in this paper, but it is an extension of the techniques described here using standard text retrieval methods.

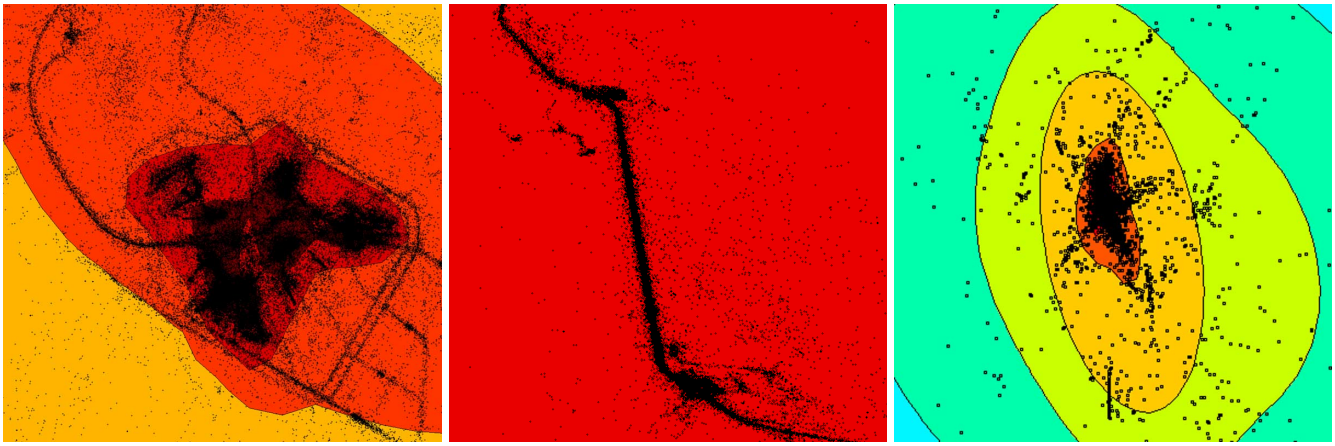
If the user still cannot find the desired location after a query search, they are offered the option of creating a new venue in the database. This venue is immediately included in the search engine and made available to the user (and others) to check in to. In this way, the system can adapt to new and missing locations and other recall failures.

Relative to other search systems, foursquare's check-in search has the benefit that the user provides direct feedback about the correct answer to their query by checking into that location after their search. Even in cases where the system fails to provide an accurate answer using nearby search, the user will often use query search to find the correct venue and check into it (26% of foursquare's searches include a text query). We use these sequences of user queries (also called *sessions*) as important training data when determining which features are the best predictors of the correct result.

2. ARCHITECTURE

The goal of a spatial search engine is to accurately retrieve and rank candidate venues which best match a query. Each query consists of a user, timestamp, and location $q_i = (u, t, l)$, where the location consists of a latitude, longitude, and an estimate of horizontal accuracy. Figure 3 gives an overview of our system architecture for mapping a query to a ranked list of venues. For each query we first generate a candidate set of venues $S_i = \{v_1, v_2, ...v_{m_i}\}$ using a coarse spatial index. We then compute features for each query-venue pair $F(q_i, v_j) = \mathbf{x}_{i,j} \in \mathbb{R}^d$, where the d features capture various relationships between each query and the candidate venue, such as distance, prior personal history of the user, etc. The response set R_i consists of the venues in the candidate set S_i rank ordered by some function of the features $G(\mathbf{x}_{i,j})$. For example, a simple linear model, represented as a vector of weights $\mathbf{w} \in \mathbb{R}^d$, would yield $G(\mathbf{x}_{i,j}) = \mathbf{w}^\top \mathbf{x}_{i,j}$. For each query-venue pair we then observe a value $y_{i,j} \in \{0, 1\}$ which indicates whether the user actually checked into venue v_j after performing query q_i.

In this section, we will focus on three main aspects of our system architecture. In Section 2.1 we discuss various ways for modeling places and people that are useful for both retrieval and feature generation. Then in Section 2.2 we describe how we generate the candidate set S_i for a given query. We then discuss in Section 2.3 how we construct features $F(q_i, v_j)$ and how we learn different ranking functions $G(\mathbf{x}_{i,j})$ by optimizing learn-to-rank algorithms using implicit feedback.



(a) JFK Airport (b) Golden Gate Bridge (close up) (c) The Blind Tiger

Figure 4: The spatial distribution of check-ins at different venues reveals complex patterns. The coordinates of check-ins for each venue are plotted in black. The colors represent a probability distribution modeled as a mixture of bivariate Gaussians, ranging from red (high probability) to blue (low probability). Although individual GPS estimates are noisy, in aggregate these estimates produce accurate reproductions of a venue's shape. Note the fine detail in the runways and terminals of JFK (left), and the clear outline of the Golden Gate Bridge (middle). Even for small venues such as the bar, The Blind Tiger (right), we see a non-uniform shape, that captures the nearby geography of streets and patterns in the noise associated with check-ins at that venue.

2.1 Modeling People and Places

Every check-in helps define exactly where a place is on the surface on the earth. Consider creating a spatial model for John F. Kennedy Airport in New York City from the 442 thousand historical check-ins that have occurred there. In Figure 4(a), we see that despite each individual GPS estimate being prone to noise, the aggregate of these estimates provides an accurate picture of the venue – one can easily make out the runways and terminals when the coordinates of all users' check-ins at the venue are plotted. A spatial model allows us to evaluate the probability that a set of location coordinates l belongs to a specific venue v: $P(l \mid v)$. Figure 4 shows the spatial models for a variety of different venues.

The simplest spatial model we compute from check-ins is based on the latitude and longitude of the center of the venue's check-ins, $l_{c(v)}$. We compute $l_{c(v)}$ by taking all check-ins to a venue, $c(v)$, removing outliers, and taking the mean of both the latitudes and longitudes. Then given a query location l, we approximate a distance-based spatial score as:

$$P(l \mid v) \approx \left(\frac{d}{\text{dist}(l, l_{c(v)}) + d} \right)^e$$

where we set $d = 50$ meters, $e = 4$, and $\text{dist}(l, l_{c(v)})$ is calculated by the Haversine formula. As we will show in Section 3.1, using a distance-based score by itself is a relatively poor predictor of the correct venue for a given set of location coordinates, particularly in dense urban areas. To address this problem, we also consider building probabilistic models of the shape of a venue and the distribution of the observed GPS noise as a mixture of bivariate Gaussians:

$$P(l \mid v) = \sum_k c_k g(l | \mu_k, \Sigma_k),$$

where $g(l, \mu_k, \Sigma_k)$ represents a bivariate Gaussian with mean μ_k and covariance Σ_k evaluated at location l, and c_k represents the mixing proportions. The parameters for this model can be learned via expectation maximization [23].

Each check-in also reveals information about the times at which places are popular. For example, Figure 5 shows the check-in rate over the course of a week for 3 different venues. We define this temporal signal as the probability that a check-in at a venue happens at a specific hour of the week, and estimate this probability using maximum likelihood:

$$P(t \mid v) = \frac{\#\text{ of check-ins at wh}(t)}{\text{total }\#\text{ of check-ins}},$$

where $\text{wh}(t)$ is the hour of the week corresponding to time t. This signal is vital for distinguishing between venues which are close in distance but popular at different times, for example a bar that is next to a bakery. We also calculate a variety of real-time signals about venues as well. We use an exponential moving average of check-ins/day to measure the current popularity of venues. We also maintain a count of the number of people who are currently checked into the venue, defined as those users who have been at the venue less than 3 hours and have not yet checked into another place.

People have been shown to be very predictable in terms of their mobility patterns [13], and Lian and Xie have shown that people's past check-in behavior is a strong predictor of future check-ins [18]. Furthermore, De Domenico et al. show that the locations of a user's friends is highly predictive of a user's location [20]. We observe that 73.1% of foursquare check-ins occur at venues to which the user has been before, and 62% of a user's check-ins are at places to which their friends have been before. Thus we also use personalized signals mined from a user's check-ins and other actions as well as their social connections to improve venue search for these users.

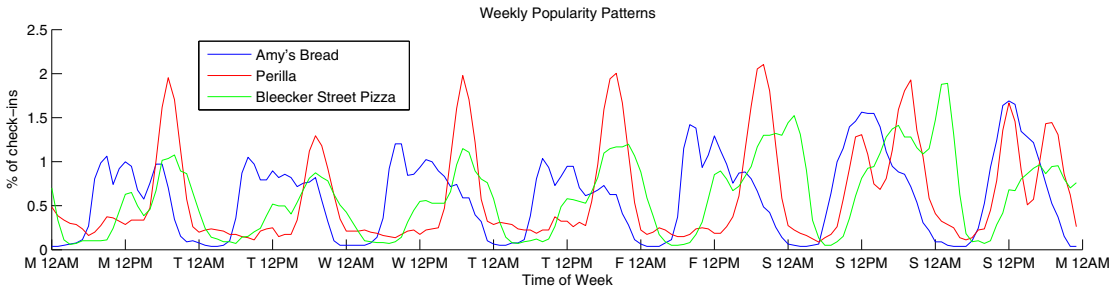


Figure 5: Places have distinct temporal signatures based on when they are popular during the week. Here we see normalized weekly check-ins for 3 venues within a 1 block radius in the West Village of New York City: Amy's Bread, a cafe which serves breakfast and lunch; Perilla, a nice restaurant that serves dinner and brunch on the weekends, and Bleecker Street Pizza, popular for both dinner and late night food on weekends.

2.2 Retrieval

The first step in our spatial search process is to retrieve a set of candidate venues for a given query. The primary source of these candidates is a geospatial index which we query for the m most popular venues closest to the user's query position. The current system uses $m = 150$ as a trade-off between recall and performance considerations.

Unfortunately, due to poor accuracy in the position reported at query time, this initial query can fail to retrieve the desired venue under a variety of different circumstances. The geospatial query is particularly problematic in dense urban areas. For example, in some areas of Manhattan retrieving the 150 venues closest to a given position will result in an effective search radius of less than 75 meters (approximately the median of reported location accuracy as reported in Figure 1). Additionally, because venues are indexed by their the center point (computed as described in Section 2.1), the geospatial query can fail when the desired venue itself is particularly large. In these cases, the query position might physically exist within the venue's boundary, but nonetheless still be a considerable distance from the venue's indexed center point. For example, a user who searches for an airport while sitting on the airport's runway might be 1000 meters or more from the position at which the airport is indexed, which is usually near the center of the airport terminal.

To mitigate these shortcomings, we retrieve supplemental candidates from three other sources. We query the last 6 months of the user's check-in history for any venues within 1000 meters of the user's current location. We include these venues in the candidate set even if they were not retrieved by the geospatial query. Similarly, we include any venue within 1200 meters at which the user has a friend who is currently checked in. Finally, we keep a list of several hundred of the most frequently checked into large venues in the world (such as airports), and include these in the candidate set if they are within a few kilometers of the query position.

Our geospatial index is built on top of the S2 Geometry Library, open sourced by Google.[2] S2 models spherical geometries by mapping the sphere to a cube, and then recursively dividing each face of the cube into four child squares, which it refers to as "cells". There are 31 levels of cells, starting with the faces of the cube itself, which are level 0. Each cell has a 64-bit identifier called its "cell ID". For a sphere the size of the Earth, a typical level 30 S2 cell has an area of less than 1 cm^2.

The S2 representation has several benefits. Mapping a latitude and longitude to any of the 31 S2 cell IDs which contain it is very efficient. Similarly, given two S2 cell IDs, determining containment is a single bitwise operation. Additionally, given a geometric region on a sphere, the S2 library can efficiently compute a set of S2 cells which cover that region. The number of cells in this "cover" can be tuned by choosing larger or smaller cells at a given point, at the expense of a less accurate representation of the given region.

Using the S2 library, it is easy to build a performant geospatial search system on top of any standard database or key-value store.[3] For each venue in our index, we add an indexed field to the key-value store which contains a list of the 31 S2 cell IDs which contain the venue, one for each S2 level. At query time, we compute the set of S2 cell IDs which cover the query region. We restrict the query to that region by adding those S2 cell IDs to the query as a boolean OR (Figure 6).

In order to find our target of approximately 150 candidate venues, we need to scale the size of the retrieval region based on the expected density of venues in that region. That is, we need to retrieve from a smaller region in areas where venues are dense (for example, metropolitan areas) than we do in areas where venues are sparse. To solve this problem we compute "venue density", which is defined as the number of venues per unit area in a given region. In practice, we choose the region to be a reasonably-sized S2 level (level 15, roughly 250 meters on a side), and calculate the venue density for every S2 cell at that level using an offline process. We use this density to automatically scale the query radius to retrieve approximately the desired number of venues given the density of the query location.

The complete venue retrieval process is thus:

1. Calculate the S2 cell at the chosen venue density S2 level for the user's reported location.
2. Look up the venue density for that cell, and calculate the estimated radius, r, that will retrieve approximately 150 venues for that density.
3. Calculate an S2 covering of the spherical cap within r of the query position.

[2]S2 Geometry Library, http://code.google.com/p/s2-geometry-library

[3]We have built our geospatial index on top of the Apache Solr project, http://lucene.apache.org/solr/.

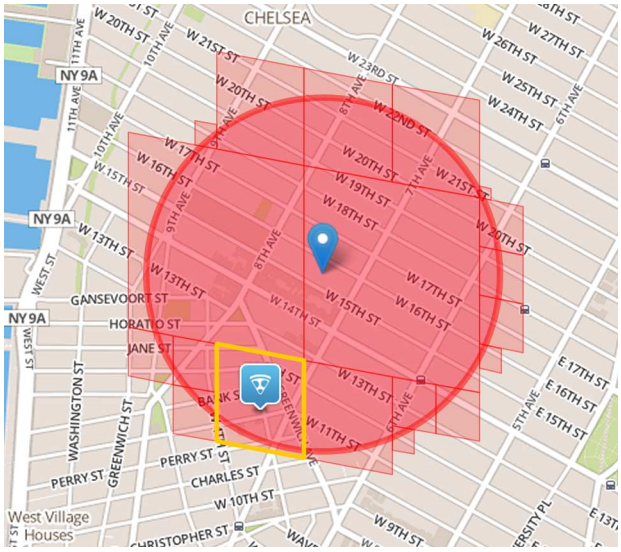


Figure 6: An example of an S2 cell-based retrieval. The query area is shown by the red circle around the query location. The set of S2 cells used for retrieval (known as a "cover") is shown as red polygons (note that an S2 cover is an approximation of the given region). The desired venue is shown as a pin on the map, with its enclosing S2 cell highlighted in yellow.

4. Query the index to retrieve all of the venues which are indexed with any of the cell IDs in the covering to get a list of nearby candidate venues.

5. Concatenate the nearby candidate venues with the supplementary retrieval sources (venues that the user has visited previously, venues that the user's friends are checked in at, and large venues that are nearby).

At the end of the retrieval process, we have between 150 and 250 candidate venues, along with supplementary information about the venues' history, shape, popularity over time, and other factors. In parallel, we have also retrieved the user's own history and preferences, as well as information about their friends and their friends' histories. All of this information is fed into the ranking system.

2.3 Ranking

Many state-of-the-art search systems rely on machine learning to create models for ranking [12, 27]. Traditionally, the data for training such models is generated by people explicitly annotating previously logged search results. There are a few drawbacks to this approach. First, it is time-consuming and expensive to collect explicit feedback at a large scale. Second, it is difficult for annotators to make judgements about personalized results, thus making this approach less desirable for use in a highly personalized search product such as foursquare's. Consider the case where two users perform a search from approximately the same location in a shopping mall. The desired result depends heavily on their personal preferences making it difficult for an annotator to label the correct result given only the list of nearby shops.

An alternative approach that has recently gained popularity is based on interpreting the implicit behavior of users as they are interacting with search results. Such implicit feedback is often available at a large scale and is easy to collect. Unfortunately, implicit feedback is ambiguous. Clicks, the most common form of implicit feedback in search, do not necessarily indicate that users have found the right result. Clicks are biased towards the top results and are often accidental. Users reformulate queries or abandon the search if the clicked results are not satisfactory. Joachims et al. have made great progress in interpreting clicks by extracting pairwise preferences [16] using query chains [15] and randomly modifying the presentation of search results [26].

In this work, we use a different form of implicit feedback: check-ins. A check-in at one of the venues in the search results is an unambiguous endorsement of that result by the user. Using this labeling technique, we can construct a training set consisting of features and labels for each candidate venue $\mathbf{x}_{i,j}$ and $y_{i,j}$ for $j = 1 \ldots m_i$ of each query q_i, where $y_{i,j}$ denotes the binary label for the jth result in the candidate set for the ith query. The label is 1 if the user checked into the venue and is 0 otherwise.

2.3.1 Data Collection

From the perspective of collecting data, a search query and a check-in are distinct and independent actions. Depending upon the version of the foursquare product, a check-in can immediately follow a query or it may have a few intermediate steps (e.g. viewing additional information about a venue prior to checking in). Each of these user actions is associated with independent calls to foursquare services running over a large pool of machines. Each call to a foursquare service results in an event log being written by the service to a distributed log collection mechanism.[4]

The candidate results R_i and the derived ranking features $F(q_i, v_j) = \mathbf{x}_{i,j}$ for $j = 1 \ldots m_i$ used to train our search algorithm are recorded in the server event logs whenever a user performs a search query q_i. Since many of our models and indices are updated continuously, regenerating the exact feature values that were used to rank the results for a query in the past is difficult. Instead, the candidate venues and their corresponding features are pushed to logs at the time of the query, removing the need for regenerating features altogether.

To derive training data from event logs, we use automated MapReduce jobs written in Scala [24] and Pig [25]. The first step in the process joins the independent logs into user "sessions" which contain all actions taken by a user during a given time frame (e.g. opening the application, performing a nearby check-in search, performing a query search for "The Blind Tiger", then checking in to a specific venue). Each session involving either a nearby or query search is then categorized as successful or not if it also includes a corresponding check-in.

Next, we sample a specified number of successful search sessions and split them into training, validation, and test sets. This process extracts from the logs all candidate venues, their features, and their original rank as presented to the user. It then joins these with other datasets containing exploratory features, allowing us to evaluate the efficacy of these features without implementing them at production scale. The training and test sets are then exported as files

[4]Apache Flume, https://cwiki.apache.org/flume/

Feature	Description
Spatial score	$P(l \mid v)$
Timeliness	$P(t \mid v)$
Popularity	Smoothed estimate of expected check-ins/day at the venue
Here now	# of users currently checked in to the venue at query time
Personal History	# of previous visits from the user at the venue
Creator	1 if the user created the venue, 0 otherwise
Mayor	1 is the user is the mayor of the venue, 0 otherwise[5]
Friends Here Now	# of the user's friends currently checked in at query time
Personal History w/ Time of Day	# of previous visits from the user at the venue at the same time of the day

Table 1: Overview of the features used for ranking. General signals (top) capture information about the user's proximity to a venue as well as information about when the venue is popular and to what degree. Personalized signals (bottom) capture past interactions between the user (and his friends) with the venue.

that can be used with various machine learning packages for model training and exploration.

2.3.2 Features

The primary features for ranking are derived from models of venues and users as described in Section 2.1. In addition to the scores derived from these models we construct features using contextual information about the users and venues at the time of the query. These contextual signals are updated in real time throughout our system. We also derive features using the social relationships between users. Each of these features is generated and logged at query time. In Table 1, we see an overview of the features used by our search ranking algorithm.

2.3.3 Metrics

We use 3 different metrics to evaluate the performance of a ranking R_i given query q_i and labels $y_{i,j}$: precision at position 1 (P@1):

$$\text{P@1} = y_{i,1},$$

recall at 5 (R@5):

$$\text{R@5} = \sum_{j=1}^{5} y_{i,j},$$

and normalized discounted cumulative gain (NDCG) [14]:

$$\text{NDCG@5} = \frac{1}{Z_i} \sum_{j=1}^{5} \frac{2^{y_{i,j}} - 1}{\log(1+j)},$$

where $y_{i,j}$ indicates that the result shown at position j for query i was the location the user checked into, and Z_i is the normalizing constant chosen such that a perfect ranking would result in an NDCG of 1 for each query. Each of these metrics can then be averaged over all n queries in our evaluation set to measure the performance of different ranking functions.

2.3.4 Optimization

We consider 3 training procedures for learning the function $G(\mathbf{x}_{i,j})$ which combines features into a rankable score: linear regression, coordinate ascent, and LambdaMART. For the linear regression model, we simply learn a set of weights $\mathbf{w}$ where $G(\mathbf{x}_{i,j}) = \mathbf{w}^\top \mathbf{x}_{i,j}$. The weights are learned such that the L2 loss is minimized:

$$\mathbf{w} = \text{a}rgmin_{\mathbf{w}} \sum_{i,j} ||\mathbf{w}^\top \mathbf{x}_{i,j} - y_{i,j}||_2.$$

The coordinate ascent training procedure similarly learns a set of linear weights but can be used to optimize metrics other than L2 loss and has thus proved to be very useful for search ranking [22]. LambdaMART has been shown to provide significant lift in search ranking tasks [33] by constructing an ensemble of decision trees which non-linearly combine features. We use MATLAB for our linear regression implementation and use RankLib for implementations of coordinate ascent and LambdaMART [3].

[5]On foursquare, the "mayor" of a venue is the person who has been there more days than any other foursquare user in the last 60 days.

3. EXPERIMENTS

We present a variety of experiments which evaluate different aspects of our search algorithm. We start by evaluating the impact of modeling places using spatiotemporal probability distributions, as compared to the most common baseline technique of simply using distance. We then demonstrate the performance of variants of our search algorithm which use different sets of features and optimization procedures. We compare these to baseline methods based on distance, popularity, and user history.

3.1 Spatiotemporal Models

To better understand how spatiotemporal models of venues improve unpersonalized venue search in dense urban areas, we present an experiment that compares different spatiotemporal models for predicting check-ins in a small 10-block radius of downtown Manhattan. Figure 7 shows a sample of the 143 thousand check-ins that have occurred at 127 popular venues near Bleecker and Jones St. color-coded by venue. We have removed any venues from this set with less than 100 check-ins (which represents less than 5% of the total number of check-ins). The dataset is then divided 90%/10% into training and testing sets.

Given only the latitude, longitude, and timestamp of each check-in, the goal is to predict which venue a user checks into (precision @1). We consider three models for this task. The baseline model simply selects the nearest venue based on distance. The spatial Gaussian mixture model uses a probabilistic model, as described in Section 2.1. The num-

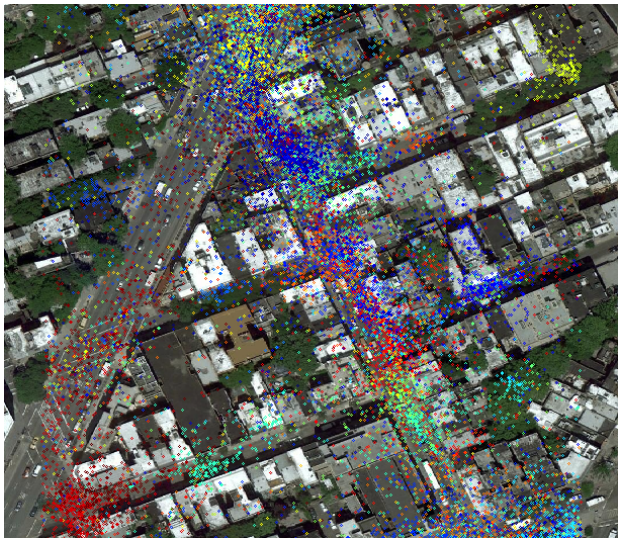


Figure 7: A scatterplot showing a sample of the 143 thousand check-ins that have occurred in the few blocks near the corner of Bleecker and Jones St. in Manhattan. Each check-in is color-coded by the venue to which it belongs. We see that although some popular venues can easily be distinguished from others nearby, there is a substantial overlap of check-ins in some areas which makes discrimination difficult.

ber of Gaussians was limited between 1 and 5 and selected via cross-validation. A venue is then predicted by finding the venue which maximizes the probability under this spatial distribution. The full spatiotemporal model uses the mixture of Gaussians, the timeliness feature, as well as popularity combined as a linear sum of log-likelihoods.

Table 2 shows the test accuracy of the 3 different models. We see that modelling venue shape offers 46% lift over a simple distance-based algorithm, and incorporating timeliness and relative popularity yields a 112% lift in performance.

3.2 Learning to Rank

In this section, we describe our experiments to determine the optimal features and training procedure for our search algorithm. Using the methodology discussed in Section 2.3.1, we collected 38 thousand examples of successful worldwide venue searches, randomly sampled from the week of 6/24/12, where a user selected the correct venue from a rank-ordered list presented to them. Our goal is to create a better ranking algorithm which ranks the correct venues at a higher position. The 38 thousand venue searches are comprised of over 4 million candidate venues and are split 70%/15%/15% into training, validation, and test sets respectively. The models for these venues were built using 282 million check-ins, collected over the 2 years prior to the query date.

We explore a variety of different combinations of features and models, including linear regression, coordinate ascent, and LambdaMART [33]. For the coordinate ascent model, we use domain knowledge to construct explicit cross-products that capture important non-linearities (such as spatial score $\times$ popularity). The LambdaMART algorithm automatically

Model	P@1
Baseline (nearest by distance)	0.130
Spatial Gaussian mixture model	0.193
Spatiotemporal models	0.277

Table 2: Comparison of different models of venues. We see that using more complex spatial models and incorporating temporal signals greatly improves the accuracy of the search algorithm.

Model	P@1
Random	0.009
Spatial only	0.201
User history only	0.358
Popularity only	0.143
Linear regression: spatial + temporal	0.230
Linear regression: spatial + temporal + popularity	0.251
Linear regression: all features	0.434
Coordinate ascent: all features w/ nonlinear pairs	0.493
LambdaMART: all features	0.531

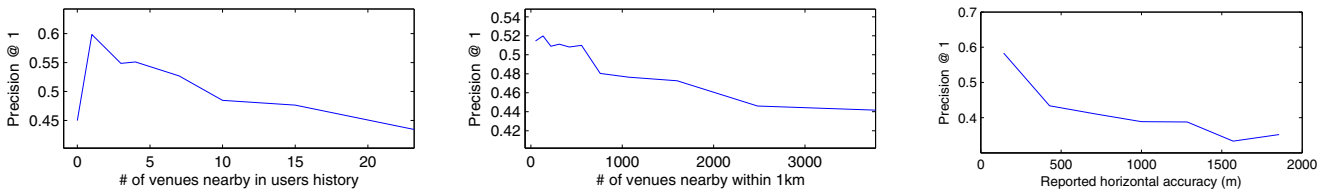
Table 3: The precision of various models and sets of features in ranking venue search results.

captures non-linearites by forming an ensemble of 2000 decision trees. Table 3 summarizes the performance of these different techniques in terms of precision at 1 on the held-out test set. The NDCG@5 for our best model is 0.686, and recall @5 is 0.822. Figure 9 shows our recall as rank increases. We see that using our best model, we find the correct result in one of the top 5 positions in 82% of searches and in the top 10 positions in 91% of searches.

The results in Table 3 indicate that distance and personal history are informative features; however their individual performance is poor when compared to a full model trained using a combination of many features. We also find that linear models can perform poorly for ranking when many features are combined. Carefully constructing non-linear pairs of features or using a robust non-linear model such as LambdaMART is crucial for combining many disparate signals into an accurate model.

3.3 Analysis

To better understand when venue search is difficult we more closely examine how the performance of our search algorithm degrades based on varying degrees of personal history, venue density, and reported horizontal accuracy from the GPS of the mobile device. In Figure 8(a), we see the performance of the linear model as a function of the number of nearby venues a user has been to when they perform a search. Because users are more likely to check into venues they have visited before, the model performs best when there are a small number of venues nearby that a user has previously visited. As the number of nearby venues in the user's history increases, this history has less predictive value, and the accuracy of the model decreases. In Figure 8(b), we see performance as a function of the number of venues within a 1km radius of the user's query. As expected, we observe



(a) User History vs. Precision (b) Venue Density vs. Precision (c) Reported Accuracy vs. Precision

Figure 8: Many factors influence the precision of venue search: the familiarity of users with nearby venues (left), the density of nearby venues (middle), and the reported horizontal accuracy from the mobile device (right).

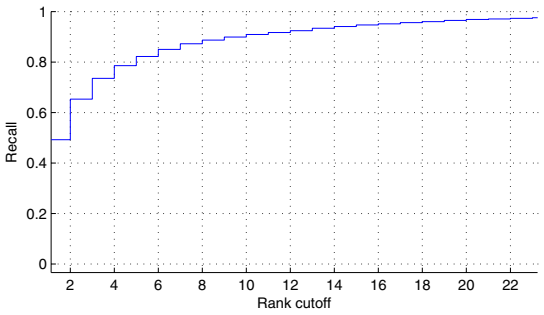


Figure 9: Recall as a function of the rank cut-off.

that performance is higher in areas with fewer venues, and degrades in dense urban areas. Similarly, Figure 8(c) shows that as the horizontal accuracy reported by the device increases, performance also degrades.

4. DISCUSSION

There are now over 5 billion consumer mobile devices in the world that are constantly reporting a signal of latitudes, longitudes, and timestamps [4]. As location data become more pervasive, the need to map these signals to semantically meaningful locations becomes increasingly important. In this article, we have demonstrated a spatial search engine for mapping noisy location estimates from mobile devices to points of interest. Furthermore, we show that simple baseline techniques for this task, such as those based on distance and popularity, are insufficient for accurately predicting a user's location. By leveraging machine learning algorithms for ranking and rich spatiotemporal models of places and users, our spatial search engine achieves significant lift in performance over these common baselines as well as previously reported results.

4.1 Future Work

We aim to incorporate more sophisticated models of user behavior by exploiting additional metadata such as categories of venues and demographics of users (e.g. age, gender, etc.). These additional signals would allow us to better distinguish between nearby venues that are frequented by different types of people (e.g. a nail salon next to barber shop). Furthermore, a better model of user dynamics could improve accuracy as well, by leveraging information about common transitions between locations (both for the user as well as in aggregate).

We plan to improve the modeling of venues in the retrieval stage of our search algorithm to better incorporate venue shapes (building on the Gaussian mixture models described in Section 2.1) instead of points, eliminating the need for supplementary retrieval of large venues. We also plan to incorporate weather information, modeling the popularity changes of venues under different situations (e.g. parks are more popular when it is sunny and art galleries are more popular when the weather is inclement).

We are striving towards a venue search system that can support not only millions of explicit nearby searches every day but millions of implicit searches as well, generated by users' mobile devices as users explore their cities. A passive venue search system would react to changes in a user's location and present relevant local information, such as details of nearby events, tips from friends, or suggestions about where to go next. Every location update can act as a search into local real-time information, surfacing highly contextual results that can help people better understand and navigate the real world.

5. REFERENCES

[1] foursquare API. http://developer.foursquare.com.
[2] Geonames.org reverse geocoding services. http://www.geonames.org/export/reverse-geocoding.html.
[3] Ranklib. http://people.cs.umass.edu/ vdang/ranklib.html.
[4] The World in 2011 – ICT Facts and Figures. http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf.
[5] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 273–280. ACM, 2004.
[6] D. Ashbrook and T. Starner. Learning significant locations and predicting user movement with gps. In *Proceedings of the Sixth International Symposium on Wearable Computers. (ISWC 2002).*, pages 101–108. IEEE, 2002.
[7] D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.
[8] Lars Backstrom, Eric Sun, and Cameron 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*, WWW '10, pages 61–70, New York, NY, USA, 2010. ACM.

[9] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 759–768. ACM, 2010.

[10] K. Church and B. Smyth. Who, what, where & when: a new approach to mobile search. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 309–312. ACM, 2008.

[11] C. Fink, C. Piatko, J. Mayfield, T. Finin, and J. Martineau. Geolocating blogs from their textual content. In *Working Notes of the AAAI Spring Symposium on Social Semantic Web: Where Web 2.0 Meets Web 3.0*. AAAI Press, 2009.

[12] J. Fürnkranz and E. Hüllermeier. Pairwise preference learning and ranking. *Machine Learning: ECML 2003*, pages 145–156, 2003.

[13] M.C. Gonzalez, C.A. Hidalgo, and A.L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[14] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 41–48. ACM, 2000.

[15] T. Joachims, L. Granka, Bing Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems (TOIS)*, 25(2), April 2007.

[16] T. Joachims, L. Granka, B. Pang, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information*, pages 154–161, 2005.

[17] N.D. Lane, D. Lymberopoulos, F. Zhao, and A.T. Campbell. Hapori: context-based local search for mobile phones using community behavioral modeling and similarity. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, pages 109–118. ACM, 2010.

[18] D. Lian and X. Xie. Learning location naming from user check-in histories. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 112–121. ACM, 2011.

[19] L. Liao, D. Fox, and H. Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *The International Journal of Robotics Research*, 26(1):119–134, 2007.

[20] A. Lima M. De Domenico and M/ Musolesi. Interdependence and predictability of human mobility and social interactions. *Proceedings of the Nokia Mobile Data Challenge Workshop*, June 2012.

[21] N. Marmasse and C. Schmandt. Location-aware information delivery with commotion. In *Handheld and Ubiquitous Computing*, pages 361–370. Springer, 2000.

[22] Donald Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, June 2007.

[23] T.K. Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996.

[24] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger. An overview of the scala programming language. Technical report, Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.

[25] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08. ACM, 2008.

[26] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 1406–1412, 2006.

[27] M. Richardson, A. Prakash, and E. Brill. Beyond pagerank: machine learning for static ranking. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, pages 707–715. ACM, 2006.

[28] Adam Sadilek, Henry Kautz, and Jeffrey P. Bigham. Finding your friends and following them to where you are. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 723–732, New York, NY, USA, 2012. ACM.

[29] P. Serdyukov, V. Murdock, and R. Van Zwol. Placing flickr photos on a map. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 484–491. ACM, 2009.

[30] M. Terry, E.D. Mynatt, K. Ryall, and D. Leigh. Social net: using patterns of physical proximity over time to infer shared interests. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*, pages 816–817. ACM, 2002.

[31] M. A. Vasconcelos, S. Ricci, J. Almeida, F. Benevenuto, and V. Almeida. Tips, dones and todos: uncovering user profiles in foursquare. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 653–662, New York, NY, USA, 2012. ACM.

[32] Petros Venetis, Hector Gonzalez, Christian S. Jensen, and Alon Halevy. Hyper-local, directions-based ranking of places. *Proc. VLDB Endow.*, 4(5):290–301, February 2011.

[33] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, June 2010.

[34] J. Yi, F. Maghoul, and J. Pedersen. Deciphering mobile search patterns: a study of yahoo! mobile search queries. In *Proceeding of the 17th International World Wide Web Conference (WWW)*, pages 257–266. ACM, 2008.

[35] P.A. Zandbergen. Accuracy of iPhone locations: A comparison of assisted GPS, WiFi and cellular positioning. *Transactions in GIS*, 13:5–25, 2009.