

**Geographic Coordinate Analysis and Estimation Using Public  
Geocoding and Social-Local Data**

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Adrian Mummey

December 2016


© 2013-2016


Adrian Mummey

ALL RIGHTS RESERVED




APPROVED FOR THE COMPUTER SCIENCE PROGRAM

 Dec 7, 2016  
Advisor: Dr. Richard Wasniowski Date

 12/7/16  
Dr. Brian Thoms Date

 Dec 7, 2016  
Dr. Peter Smith Date

APPROVED FOR THE UNIVERSITY

 12-7-16  
Dr. Gary A. Berg Date



## Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Geo-Coordinate Analysis and Estimation Using Public Geocoding and Social-Local Data

Title of Item

Geocoding, Social Media, Clustering

3 to 5 keywords or phrases to describe the item

Adrian Mummey

Author(s) Name (Print)

Author(s) Signature

12/26/16

Date



*Master Thesis by Adrian Mummey*

**Geo-Coordinate Analysis and Estimation Using Public  
Geocoding and Social-Local Data**

by  
Adrian Mummey

Computer Science Program  
California State University Channel Islands

## **Abstract**

Geocoding is a complex but important task and its accuracy is crucial to many industries. Although geocoding providers such as Google generally provide accurate coordinates, they often have limitations in their accuracy. The goal of this thesis is to analyze the data of popular geocoding services and to improve on their results.

This thesis will focus on the development of PlayPin, a software platform for generating geocoding data. It will demonstrate that aggregating results from public geocoding services and location-tagged social media data can generate very accurate coordinates. Initially, a location's coordinates are gathered from public geocoding APIs, these results are processed using a clustering algorithm to identify a cluster and the centroid of the cluster is found. Using this centroid as the geographic point of reference, social media APIs are queried for proximal and contextually relevant data around the point. This social-local data is then processed again using a clustering algorithm and a centroid is generated for the final point estimation.

Using a set of over four thousand locations, PlayPin was able to generate a coordinate within 50m of the correct position over 80% of the time, compared to the best public geocoder, which only achieved this accuracy 61% of the time

## **Acknowledgements**

I would like to thank Dr. Richard Wasniowski for his patience and support in completing this thesis. I am thankful to Dr. Brian Thoms and Dr. Peter Smith for reviewing my thesis and providing helpful feedback. I would also like to thank my wife Juliana for encouraging and supporting me through this process and my mom, Carmen, who provided proofreading, editing and moral support. Lastly, I would like to thank my colleagues at MomentFeed for their inspiration and support in this project.



TABLE OF CONTENTS

<b>Chapter 1 : Introduction</b> .....	<b>11</b>
<b>1.1 The Anatomy of a Location</b> .....	<b>12</b>
<b>1.2 Accuracy in Coordinates</b> .....	<b>13</b>
<b>1.3 Location Based Engagement</b> .....	<b>14</b>
<b>1.4 Remaining Chapters</b> .....	<b>15</b>
<b>1.5 Key Terms</b> .....	<b>17</b>
<b>Chapter 2 : Field Overview</b> .....	<b>18</b>
<b>2.1 Geographic Overview</b> .....	<b>18</b>
2.1.1 Projections .....	18
<b>2.2 APIs</b> .....	<b>19</b>
2.2.1 API Rate Limiting.....	20
2.2.2 Inconsistent/Limited API Results.....	20
<b>2.3 Geocoding APIs</b> .....	<b>21</b>
<b>2.4 Backfill APIs</b> .....	<b>23</b>
<b>Chapter 3 : Problem Formulation</b> .....	<b>25</b>
<b>3.1 Approaches to the Problem</b> .....	<b>25</b>
<b>3.2 Proposed Solution - PlayPin</b> .....	<b>27</b>
<b>Chapter 4 : Technical details of the work</b> .....	<b>31</b>
<b>4.1 Overview</b> .....	<b>31</b>
<b>4.2 Use Cases</b> .....	<b>31</b>
4.2.1 Marketers/Managers for Stores with Physical Locations .....	32
4.2.2 Researchers .....	32
4.2.3 Geocoding Providers.....	32
<b>4.3 Functional Requirements</b> .....	<b>33</b>
<b>4.4 Systems Implementation</b> .....	<b>34</b>
4.4.1 Languages .....	34
4.4.2 Databases.....	35
4.4.3 Puma Server .....	35
4.4.4 Frameworks and Libraries.....	35
<b>4.5 Architectural Overview</b> .....	<b>39</b>

4.5.1 Ruby on Rails.....	40
4.5.2 Model Classes.....	40
4.5.3 Sidekiq.....	42
4.5.4 Sidekiq Workers .....	44
4.5.5 Helpers.....	47
<b>Chapter 5 : Experiments .....</b>	<b>48</b>
5.1 Experimental Steps Overview .....	48
5.2 Data Set .....	49
5.3 Account creation and data import.....	49
5.4 Training and Generation of Weights.....	52
5.4.1 Training Data Geocoding.....	53
5.4.2 Distance Calculations.....	54
5.4.3 Weight Processing .....	55
5.5 Coordinate Estimation.....	57
<b>Chapter 6 : Experiment &amp; Analysis of Results.....</b>	<b>65</b>
6.1 Geocoding processing .....	65
6.2 Geocoding Results .....	69
6.3 Clustering and Centroid Estimation First Pass.....	71
6.3.1 Centroid Calculation .....	72
6.3.2 Centroid Results.....	74
6.4 Backfilling .....	75
6.4.1 Backfill Results .....	78
6.5 Clustering and Centroid Estimation Second Pass.....	78
6.6 Experimental Discussion .....	80
<b>Chapter 7 : Conclusions .....</b>	<b>82</b>
<b>Chapter 8 : Future Work.....</b>	<b>84</b>
8.1 Usability.....	84
8.2 Foundational.....	85
8.3 Additions .....	86



TABLE OF FIGURES

Figure 1.	Anatomy of a Location .....	12
Figure 2.	Satellite View of Restaurant from Google Maps.....	13
Figure 3.	Mobile Facebook Location Page .....	15
Figure 4.	Google API Rate Limits.....	20
Figure 5.	Table of geocoding APIs .....	23
Figure 6.	Table of Backfill APIs .....	24
Figure 7.	Possible approaches to bulk geo-coordinate acquisition .....	26
Figure 8.	Traditional Geocoding .....	28
Figure 9.	PlayPin approach to geocoding.....	29
Figure 10.	Systems Diagram .....	39
Figure 11.	Database Schema .....	41
Figure 12.	Sidekiq control panel .....	43
Figure 13.	Table of Sidekiq Workers .....	46
Figure 14.	Table of Helper Utilities .....	47
Figure 15.	Sample Location Data.....	49
Figure 16.	Playpin Account Creation Interface.....	50
Figure 17.	Location Import Interface .....	51
Figure 18.	List of locations in PlayPin .....	52
Figure 19.	Google Geocode Job Interface.....	53
Figure 20.	Distance Calculation Job Interface .....	54
Figure 21.	Weight Processing Interface .....	56

Figure 22.	Training Results for Outback location data .....	57
Figure 23.	A view of geocoding query results for a sample location.....	58
Figure 24.	A zoomed in view of the bottom left cluster from Figure 23 .....	58
Figure 25.	Centroid Calculation Interface.....	60
Figure 26.	Weighted centroid in middle of cluster represented by green hexagon....	61
Figure 27.	Location Based Engagement Backfill Interface .....	62
Figure 28.	Contextually and proximally relevant LBE around a location .....	63
Figure 29.	Backfill centroid and its cluster represented by pink hexagon. ....	63
Figure 30.	Comparison of backfill & weighted centroids.....	64
Figure 31.	Sample Geocoding Query (Google Maps API) .....	66
Figure 32.	Sample Google Maps API response .....	66
Figure 33.	Results from geocoding a single location .....	68
Figure 34.	Histogram of hits vs. misses of geocoding results.....	70
Figure 35.	Cluster of geocoded points with outlier .....	72
Figure 36.	Centroid Equation .....	72
Figure 37.	Weighted Centroid Equation.....	73
Figure 38.	Centroid of Geocoded points represented by orange hexagon .....	74
Figure 39.	Histogram of hits vs. misses of weighted centroid results.....	75
Figure 40.	A Tweet that happened in proximity to an Outback Steakhouse.....	76
Figure 41.	Histogram of hits vs. misses of backfill results .....	78
Figure 42.	Histogram of hits vs. misses of backfill centroid results .....	79
Figure 43.	Cartesian vs. Spherical Coordinate Systems.....	90
Figure 44.	Distance formula for two dimensional plane .....	91



Figure 45.	Haversine Formula.....	91
Figure 46.	Geocoded venues .....	93
Figure 47.	Distance matrix of geocoded venues .....	94
Figure 48.	Pruned Distance Matrix .....	95
Figure 49.	Outliers at k=3 .....	96
Figure 50.	Outliers at k=2 .....	97
Figure 51.	Final and Best Cluster.....	98

## Chapter 1 : Introduction

If you have ever searched for a place on Google Maps and followed the directions given only to end up at the wrong location, you have experienced the problem that has inspired this research. Many online providers of geo-location data suffer from inaccuracies. A study of 3000 residential homes showed the mean error for a commercial geocoder in rural areas was 614 meters, 143 meters for suburban areas and 58 meters in the urban areas [1]. One problem with inaccurate geo-location data is that it can lead consumers of the data to the wrong locations. For businesses, that can lead to frustrated customers and potentially lost business. Accurate geocoded data is also useful in things tracking diseases [2] and measuring air quality [3]. Because of the explosion of GPS capable smart phones in the hands of the population [4] this locational data is consumed more frequently and the problems with inaccuracy are affecting greater numbers of consumers [5].

A thought leader in the location-based digital marketing space has this to say:

*“The challenge is that lat/long inaccuracy is a large, systemic problem for all multi-location brands. It starts with bad internal data (store finders) and gets perpetuated through so many third parties including Google, Facebook, Foursquare, and others... The most important piece of data for store finders and local listings is not the street address or even the phone number. It’s the lat/long.”[6]*

## 1.1 The Anatomy of a Location

*“The anatomy of a physical place can be broken down into layers. First, it has a latitude and longitude. Aside from acts of God, these are permanently fixed. Next, there is a street address, complete with city, state, and zip code, as well as a physical structure, such as a building. These are less permanent than its GPS coordinates but much more fixed than the business or brand that inhabits the structure e.g. a McDonald’s, Walmart, or P.F. Chang’s. Finally these physical locations have digital representations.” [7]*

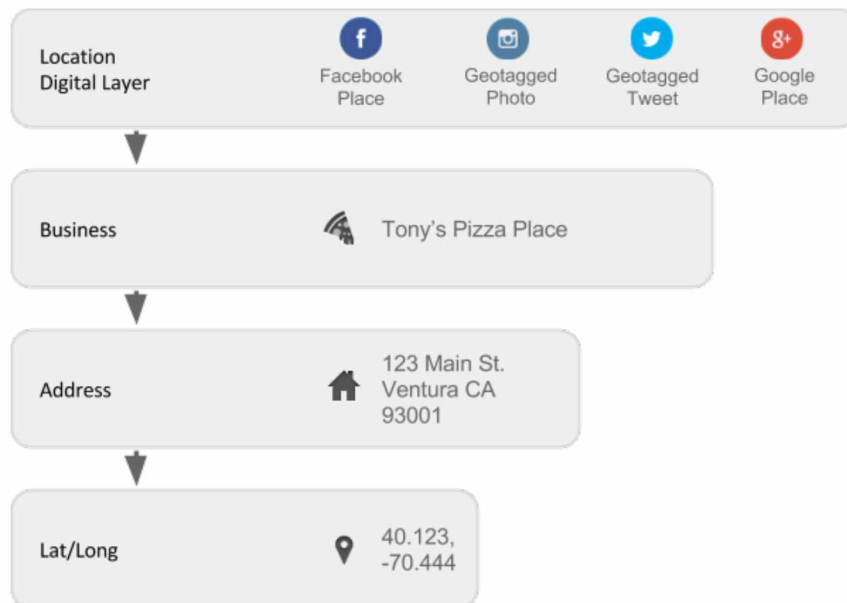


Figure 1. Anatomy of a Location

Figure 1 illustrates that a location is composed of many different layers and the most fundamental representation is a coordinate. When we use our smartphones to search for a location, we are asking for a coordinate and the accuracy of that coordinate is essential to us finding our way to that location.

## 1.2 Accuracy in Coordinates

It is important to define what is meant by accuracy when referring to coordinates. The most common way of measuring geospatial coordinates is with latitude and longitude (lat/long) [8]. This measurement defines a single point on the earth and generally is measured using GPS (Global Positioning System). As of writing this thesis, the best GPS systems have an accuracy of +/- 3 meters [9]. An issue that arises when discussing accurate coordinates is, how to best determine accurate coordinates for a location.



Figure 2. Satellite View of Restaurant from Google Maps

In Figure 2 we see an image of a restaurant taken by from the satellite view of Google Maps. There are several options for defining an accurate coordinate for this restaurant, we could take a GPS measurement from the road, or the parking lot, or we could walk inside the restaurant and take a measurement from there. There are a number of choices many of which can be valid and accurate depending on the purpose of the coordinate. Accurate coordinates depend on the ultimate purpose of the required task. A postal worker may want the coordinates of the mailbox, a customer may just want coordinates of the parking lot, and people on foot might just want the front door. There is also an issue with larger venues. Universities can take up many city blocks. The accuracy needed to get someone in front of the restaurant in the satellite image from Figure 2 may be in the tens of meters, however with a university, the accuracy needed to get a person to that institution may be far less, even in the hundreds of meters because the location spans so much more space.

For the purpose of this thesis, I will be investigating accuracies in GPS measurements. The main use case that will be considered is a customer searching for an accurate coordinate for a location of a business. This location is a structure that has a street address. We will limit the perimeter of our locations to tens of meters and will therefore define an accurate lat/long as one that can either get a person within the physical bounds of the location, or at minimum, within 10 meters to the outside of the location.

### **1.3 Location Based Engagement**

GPS enabled smartphones allow users to determine their current location with a high level of accuracy [10]. Because of this, mobile social media applications such as Twitter, Instagram and Facebook allow users to associate a particular location to the content they generate in a process known as “geotagging” [11]. Herein, I will refer to user-generated content that has been geotagged and publically shared on a social media provider as location-based engagement (LBE). One example of LBE is a geo-tagged photo. GPS enabled smartphones such as the iPhone can automatically geotag user photos by adding a lat/long coordinate in the photo’s metadata. Other examples of LBE include checking-in, leaving tips and reviews, redeeming offers and geo-tagging tweets [12].

Facebook users upload 350 million photos every day [13]. There are now more than 13 million local business pages on Facebook, with over 60% active monthly [14]. It is clear by this sheer volume of interaction that users are engaging with these digital locations on a massive scale. An example of using Facebook to engage with local business is shown in Figure 3.

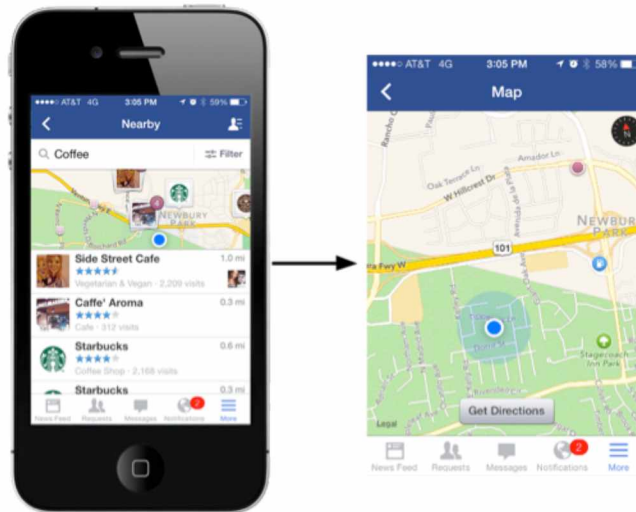


Figure 3. Mobile Facebook Location Page

## 1.4 Remaining Chapters

In Chapter 2 I present an overview of the field, including geographic concepts, geocoding APIs, social media APIs, and various technologies related to this project.

In Chapter 3 I show the formulation of the problem, possible and proposed solutions and a statement of the hypothesis.

In Chapter 4 I explore the technical details of the implementation of this project and the methodologies used in the experiments.

In Chapter 5 I present a detailed overview of the experiments run.



In Chapter 6 I cover the results from an analytical perspective. I will use graphs extensively to present the findings.

In Chapter 7 I provide a summary of the work and the conclusions reached.

Finally, in Chapter 8 I discuss potential and planned future work on the project.

## 1.5 Key Terms

- **Account** – a data object in PlayPin that is a group of locations which have some similarity or relationship.
- **API** - Application Programmer Interface - specifies how some software components should interact with each other. For this project it refers exclusively to web APIs accessed via REST services.
- **Backfill** - A spatially constrained query for LBE around a specific geo-coordinate. For example, find all photos taken within 500 meters of a point.
- **Centroid** - the center of mass of a geometric object of uniform density.
- **Clustering** - The task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.
- **Geocoordinate** - A point representing a specific place on earth defined by a latitude and longitude value.
- **Geocoding** - The process of converting location address data into a geo-coordinate.
- **GPS** - Global Positioning System, provides location data anywhere on earth.
- **LBE** - Location Based Engagement, social media actions that are associated with a location.
- **Nearest neighbor** - A clustering algorithm with the objects being assigned to the class most common among its  $k$  nearest neighbors.
- **PlayPin** - Software to aggregate and analyze location based data used in this thesis.
- **Rate Limiting** – Public APIs often limit their usage to discourage abusive use [15]. Many of the APIs explored here will detail exactly how much and how often requests can be made.
- **Gold Standard [coordinate] (GS)** – The correct or “true” coordinate of a location. All of the gold standard coordinates have been cross-referenced with satellite maps and have been human verified.
- **Hits and Misses** – For the purpose of this thesis we will define a coordinate as a “hit” if it falls within 50m of the GS coordinate and a “miss” otherwise. A “miss” may also include a coordinate with no data, which may occur when a geocoding process fails to find a match for an address.

## **Chapter 2 : Field Overview**

### **2.1 Geographic Overview**

Geographic coordinates are defined by latitude and longitude (lat/long), which represent two points measured in angular degrees from the center of the earth. Latitude is measured as the angle between the line that connects the center of the earth and the point of interest and the line that connects the center of the earth and the equator. Longitude is measured as the angle between the line that connects the “Prime Meridian” and another meridian that passes through the point. These points are measured in a spherical coordinate system that describe angular distance on a globe (see Appendix B).

#### **2.1.1 Projections**

When we look at a map, we are seeing the earth’s features projected on a flat surface. Transforming the latitudes and longitudes of a globe onto a plane via projection is an important tool in not only maps, but also can be helpful in distance calculation. The downside of projection is that it is not possible to represent a globe’s features on a plane without the distortion of some of those features. Whenever we look at a map on the wall, we are seeing a distorted version of the real thing.

One of the more popular projections in use is called the Mercator projection. The Mercator projection can preserve the angles and the shapes of small objects but as the distance from the equator to poles increases, the Mercator projection distorts the size and shape of large objects. For instance, on a Mercator map, Greenland appears larger than Africa, when Africa’s area is 14 times greater [16].

If we first project the spherical geographic coordinates to a flat plane, we can use simple Cartesian formulas (Appendix B) to calculate distance. Although these distances can suffer from inaccuracies, at certain scales and certain distance ranges they can indeed suffice. Projecting to a flat surface and then utilizing Cartesian formulas can speed up distance calculations drastically.

## **2.2 APIs**

To gather geocoding and location based engagement (LBE) data it is necessary to use publicly available APIs. The APIs will allow software to make requests across the Internet to query useful information. Each one of these APIs was chosen for its ability to use geo-targeted information in its queries. This thesis will cover only free and publicly available application programming interfaces (APIs). Since the APIs are free, most of these services enforce certain limits in their usage called Rate Limits. I have divided the APIs into two types: Geocoders and Backfills. Geocoders process street address information and try to match it with a specific lat/long coordinate. Google Maps is a good example of this; you can type in a street address and it will try to present you with a map with a specific point that shows you the address you typed in. Backfills on the other hand, take a coordinate as input and return LBE data around that point. Instagram is a good example of this. Instagram is a popular photo application where users can upload photos that they take. Often, Instagram photos will include additional geo-tagging of a user's location. Instagram allows users to search around a point at a fixed radius and will return photos taken inside the space.

### 2.2.1 API Rate Limiting

 Google Maps Android API v2 	<input type="checkbox"/> OFF	
 Google Maps API v3 	<input checked="" type="checkbox"/> ON	Courtesy limit: 25,000 requests/day • <a href="#">Pricing</a>
 Google Maps Coordinate API 	<input checked="" type="checkbox"/> ON	Courtesy limit: 1,000 requests/day
 Google Maps Engine API 	<input checked="" type="checkbox"/> ON	Courtesy limit: 10,000 requests/day
 Google Maps Geolocation API 	<input checked="" type="checkbox"/> ON	Courtesy limit: 0 requests/day • <a href="#">Pricing</a>
 Google Maps SDK for iOS 	<input type="checkbox"/> OFF	
 Google Maps Tracks API 	<input type="checkbox"/> OFF	

Figure 4. Google API Rate Limits

For instance, Google has a whole page dedicated to enumerating the various rate limits in its free API access show in Figure 4. Google’s free Maps API is limited to 25,000 requests per day. Many APIs have even more frugal limitations than Google. Mitigating these rate limits is a very important problem to overcome when dealing with large amounts of location data.

### 2.2.2 Inconsistent/Limited API Results

One of the biggest drawbacks of using free, publicly available APIs is the potential for unreliable service, inconsistent data and limited results.

- Downtime in provider servers can lead to unreliable service.
- Changing API definitions and policy changes in data providers can lead to inconsistent data
- Data caps and rate limits can lead to only partial results being returned from the server.

## **2.3 Geocoding APIs**

There are many different free geocoders available and each has its own API. Furthermore, each geocoder expects queries differently than each other and each returns its results in a different data structure. For this thesis, only free and public geocoding APIs are used for the software developed for this thesis. All APIs are described in Figure 5.



Geocoder API	Description
Google Maps Geocoding API	The most popular geocoding service on the planet. They offer an API that delivers results in either XML or JSON and rate limit to 25,000 requests per day. To use the API a valid Google account is needed and the service needs to be enabled in the developer console. [17]
Bing Maps Locations API	Microsoft's offering into the online mapping ecosystem. Bing offers a geocoding API with many input parameters and delivers results in either XML or JSON. To use the API a Windows Live ID is required. Bing Maps rate limits you to 30,000 requests per day. [18]
Geocoder.us	A public service providing free geocoding of addresses and intersections in the United States. Their geocoder is based on The US Census Bureau data called TIGER/Line, which they claim is "notoriously incomplete". Geocoder.us limits you to 50,000 requests per day and returns results in XML format. []
Mapquest Geocoding API	Allows users to access the MapQuest geocoder through a simple HTTP request. The Geocoding API supports three major geocoding functions: address, reverse, and batch. Mapquest returns results in JSON and XML formats and limits you to 5000 requests per day. [19]
Yahoo YQL	Although Yahoo is promoting their paid geocoding service called BOSS, they still support their free geocoding service that uses YQL (Yahoo Query Language). Yahoo delivers geocoding results in JSON and XML and limits you to 2000 queries per day. [20]

Yelp	Although Yelp isn't a true geocoder, by using a combination of Yelp's search API, plus scraping it's website HTML includes geocoded information. Yelp returns results in XML and JSON and limits you to 10,000 requests per day. [21]
------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 5. Table of geocoding APIs

## 2.4 Backfill APIs

Backfilling is the consumption of LBE data. Generally backfilling involves querying an API with a coordinate and a fixed radius. The APIs return results that are contained within the geographic circle. The backfill APIs are listed in Figure 6.

Backfill API	Description
Facebook Places	Using the Facebook's graph search API, we can pull user-created and brand-created Facebook place pages [22]. These pages represent locations in Facebook's database.
Flickr Photos	Flickr is a photo hosting and sharing service. Flickr offers a public API that allows users to search for photos around geo-coordinates [23].
Google Places	Google Places is the location based business listing service provider by Google. It has a public API that allows users to search for businesses around geo-coordinates. [24]
Instagram	Instagram is a social media photo sharing service. They offer a public API that allows users to search for photos around a location. [25]
Twitter	Twitter is a social media messaging service. They offer a public API that allows users to search for "tweets" around a location. [26]

Figure 6. Table of Backfill APIs

## **Chapter 3 : Problem Formulation**

The goals of this study are twofold; first to show the inaccuracies in common geocoding for a given data set and then determine if it is possible to improve the accuracy using LBE data gathered from social media providers for certain sets of locations.

### **3.1 Approaches to the Problem**

It is trivial to take a single restaurant and figure out a lat/long. One could drive to the restaurant, park, take a GPS enabled smartphone to the front door or inside the building and determine the lat/long within about +/- 5 meters. But what about the food chain that has 15,000 locations worldwide and wants accurate coordinates for all of them? It may not be feasible for them to have someone use this technique to gather all this information.

Figure 7 identifies three possible approaches to the problem of gathering accurate coordinates at scale:

Approach	Pros	Cons
1. Go to each location, or have employees with a GPS device and determine the coordinates for each one.	Will get very accurate data.	Difficult to scale, coordinating the exercise over thousands of locations may be logistically difficult and expensive.
2. Use a geocoding service	Very straightforward generation of coordinates from street address data. Highly scalable. Inexpensive.	No way to ensure accuracy or be confident about results.
3. Use satellite imagery to determine coordinates of locations	Can be as accurate as the satellite images. Very scalable, satellite maps can be accessed online. Easier to ensure accuracy or coordinate.	Not all locations around the world have detailed satellite maps needed to gather accurate lat/longs. Locations that exist inside large buildings may be hard to find, especially in dense metropolitan areas. It is still a difficult process finding and making sure the location you are looking at is exactly the one you are looking for.

Figure 7. Possible approaches to bulk geo-coordinate acquisition

None of the approaches shown in Figure 7 is a perfect solution for the problem at hand. They each have problems ensuring both accuracy and efficiency at scale. Another possible approach would be to combine activities 2 and 3 from above. That is, when possible, use a geocoding service to find a coordinate for a location, then check and refine your coordinate using satellite maps. Some companies do exactly that. Businesses send a list of locations and use a combination of geocoding and satellite imagery to identify the correct coordinate for the business. This process is very accurate and somewhat scalable. However, it still requires a large amount of manual oversight, especially when the geocoders fail to find a match for the location and can be prone to human error.

### **3.2 Proposed Solution - PlayPin**

To solve the problem of automatically generating accurate geo-coordinates at scale, I propose a solution that combines existing publicly available geocoding APIs along with LBE data gathered from various social-media providers. The assumptions that inspired this method are:

1. Using a combination of geocoders to generate an initial reference point is more accurate because it will help mitigate the inaccuracies of any single geocoder.
2. LBE around a location is extremely accurate because it comes directly from within or around the location itself.
3. A combination of 1 and 2 will generate a point that is more accurate than any single geocoder.

From these assumptions, the hypothesis tested is that using a method that combines geocoders and LBE to generate geo-coordinates will, on average, be more accurate relative to the GS coordinate than the coordinates generated from any of the geocoding providers used.



To test this, the software I have created allows the geographic data to be analyzed by using a custom developed algorithm. This algorithm detects spatial outliers in groups of geographic points, and aggregates spatially and contextually relevant LBE. The clustering and outlier detection uses a modified k-nearest neighbors algorithm. Using a clustering algorithm to analyze geographic point information is not a new technique; it can be a very effective method for outlier detection in groups of geographic coordinates [27]. However, the novelties of my proposed method are its application in the context of geocoded data and the processing of contextually relevant LBE data.

At this point it is helpful to review the process of the geocoding providers. Figure 8 shows how address data is processed by a traditional geocoding service such as Google Maps.

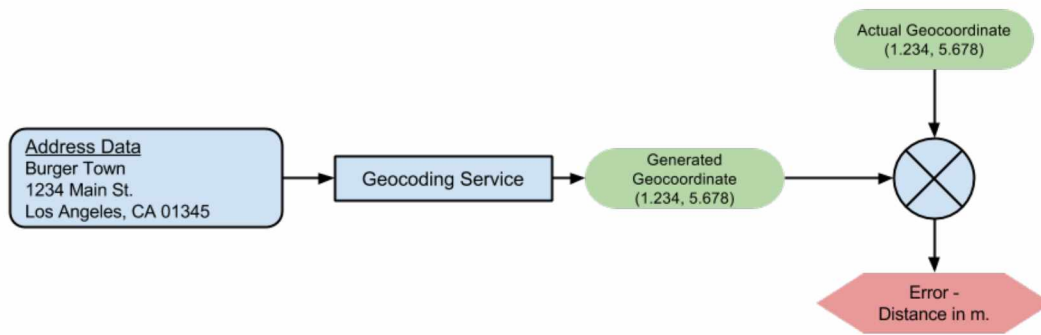


Figure 8. Traditional Geocoding

The address data is used as an input to the geocoding service, which in turn outputs a latitude and longitude (See Appendix A). The error is then calculated based on the distance from the generated point to the GS coordinate for the location.

As I will show in Section 5.4.3, this solution of using traditional geocoding providers can be prone to error, especially when searching for coordinates with accuracy within 50m of the GS location. Although most geocoding providers use proprietary algorithms, this research can still make improvements in the accuracy of geocoding.

To do so, my solution takes advantage of many geocoding providers and a large range of location-based data that is publicly available to generate very accurate coordinates. Figure 9 shows an alternate way to process address data.

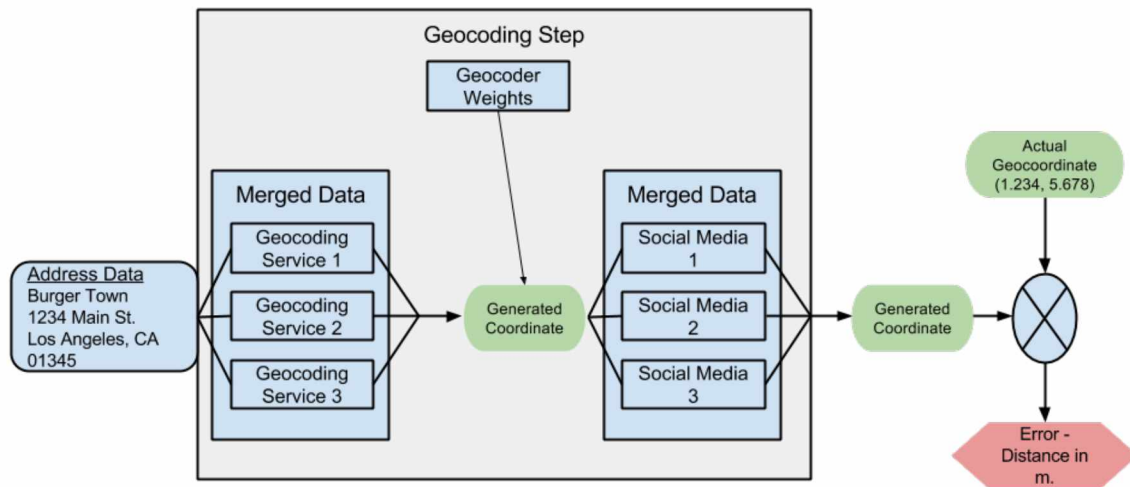


Figure 9. PlayPin approach to geocoding

The system I developed uses multi-step, aggregated approach to geocoding.

1. Use a small set of training data to determine the errors of the traditional geocoding providers and subsequently generate weights that represent the accuracy of each provider.
2. For each location in the working set of data, merge and process the results from various geocoding providers to generate an approximate coordinate based on the weights from training. (See Appendix A for more details)
3. The generated coordinate is used to gather location based engagement (LBE) data, which is then processed again to generate a final approximate coordinate.

The experiment described in the following chapters will show that this process improves accuracy of the resulting geo-coordinates compared to standard geocoding providers for a set of sample location data.

## **Chapter 4 : Technical details of the work**

### **4.1 Overview**

The software under development, PlayPin, allows a user to geocode and analyze structured location data and generate new coordinates. By analyzing the accuracy of various geocoding providers, we can craft a model for more accurate estimation of new coordinates.

To determine if this system can meet the goal of generating more accurate coordinates, we need to define a metric to measure against. The metric used to evaluate the system-generated coordinates against those from various geocoding providers is the distance to the Gold Standard (GS) coordinate. That is, if we have a GS coordinate C, and a certain geocoder gives us a coordinate A, then the distance from A to C will be the metric we use to rank that geocoder against others. There are two important assumptions here that we need to address for this to work.

1. A GS coordinate exists for each location.
2. A large sample of locations will provide a more accurate average measurement.

In this study, we have access to a large set of location data that includes both address and a GS coordinate for each address.

### **4.2 Use Cases**

PlayPin appeals to any user or organization that wishes to analyze structured location based data. I have identified several of the potential users here:

### **4.2.1 Marketers/Managers for Stores with Physical Locations**

The main goal of the marketing manager is to increase business for clients. Discoverability on the Internet should be a major concern for all businesses these days. If their customers search for their locations on the Internet and are directed to inaccurate locations, this could cause bad sentiment towards the brand. Section 1.1 discusses a whole layer in the anatomy of a location which is the digital representation of that location. If that location isn't accurate, the user misses out on engaging with that digital location.

In this case the marketers can upload a list of their client's locations, perform the geocoding on various providers and view the results. PlayPin can give them a detailed analysis of how well each geocoder performs for their locations as well as give them an accurate estimate of each locations coordinates.

### **4.2.2 Researchers**

PlayPin offers an interesting platform for research. It was designed to be very flexible in its approach to the collection and processing of this data. This could be valuable to researchers who are interested in processing geo-spatial data. For instance, PlayPin uses a fairly straightforward heuristic for its estimations; however, there are countless variations that could be developed that might outperform it.

### **4.2.3 Geocoding Providers**

Geocoding providers such as Google and Bing might be interested in a platform like PlayPin to compare the results of their geocoders to others. Using PlayPin like this could help the providers improve their own internal algorithms and generate more accurate geo-coordinates.

## **4.3 Functional Requirements**

PlayPin is a powerful system that is both an experimental platform and a turnkey solution. Although parts of the workflow are automated, each step allows for user configuration and tweaking. The functional requirements for PlayPin are listed below.

### ***Account Creation***

An account is simply a named group of locations that have some similarity or relationship. All the locations of an account will have a relationship to each other. For instance, a sample account could be “High Schools” or “Gas Stations”. Segregating locations by accounts allows us to aggregate the analysis around certain types of locations. PlayPin also allows for some account-level keywords to be designated. If locations are all Best Buy stores, a keyword could simply be “Best Buy”. These keywords allow us to filter LBE not only by proximity but also by context.

### ***Import Location Data***

A proper analysis of the problem space requires a large test set of locations. Therefore, one of the requirements for PlayPin is the ability to import large address datasets. Initially, the system will be able to handle formatted texts in the form of CSV or Excel files.

### ***Geocode Data***

When possible this system uses publicly accessible geographic data, one of these data sources will be public geocoding APIs. PlayPin therefore must be able to request geocoding data from public APIs and store this data in the database. This will need to be accomplished within a reasonable time frame and will need to be able to handle thousands of location requests.

### ***Processing and Viewing Geocoded Data***

PlayPin will collect coordinates returned from geocoding services and measure the distance from each to a corresponding reference point. The system should be able to calculate these distances accurately. A user-option to drill-down to specific locations and view summary data will be provided.

### ***Analysis of Geocoded Data***

PlayPin will be able to take all the data that has been imported and processed from the APIs and run summary statistics on it. Statistics should be available on an account level and a location-by-location level. An important way to analyze this data will be to see how well each geocoder performs by looking at the distances from a given reference point. PlayPin will do this through an interactive histogram. This interface allows users to define the distance of hits and misses, and specify which locations are included in the histogram.

### ***Gather LBE data around a point***

Once coordinates are estimated from the collected geocoding data, PlayPin will be able to make requests for LBE data around this point. The system will use public social media APIs to request data based on a certain radius around the point. These LBE data points will be able to be filtered by spatial and contextual relevancy.

### ***Estimate Coordinates***

The goal of PlayPin is to estimate an accurate coordinate using the acquired geocoded data. The system will be able to calculate this point using a variety of methods that can then in turn be analyzed for effectiveness.

## **4.4 Systems Implementation**

### **4.4.1 Languages**

Server-side development of PlayPin relies on the Ruby programming language. Ruby was chosen mainly for its robust set of 3rd party libraries (called Gems) and its ease of use. Ruby has an enormous developer network with thousands of open source options. In addition, there are already numerous libraries for interacting with geocoding and social media APIs as well as a robust library for handling geo-spatial data. Ruby coupled with the Rails framework allowed me to rapidly scaffold up this application in a short amount of time.

The front end was constructed using HTML, CSS and JavaScript. HTML5 and CSS3 provide the ability to create user-friendly interactive elements quickly.

#### **4.4.2 Databases**

One of the key decisions to be made in the implementation was the choice of database. Since there is the potential for hundreds of megabytes worth of geo-location and social media data to be collected and analyzed, it is essential to find a solution that can handle such loads efficiently.

##### ***PostGIS***

My choice of database was a special flavor of Postgres SQL called PostGIS. Postgres SQL is an open-source, relational database management system that is known for robust feature set and customization capabilities [28]. PostGIS adds additional geo-spatial data types along with many geo-spatial functions that are particularly helpful for my application [29]. Specifically, PostGIS offers a “point” data type that stores lat/long information. The database also offers distance calculations, spatial re-projections and analytical functionality.

##### ***Redis***

Redis is known to be a fast key-value store. In the scope of this project we are using Redis as the local storage of our Sidekiq worker queues (Section 4.4.4).

#### **4.4.3 Puma Server**

PlayPin uses the Puma web-server to serve the Ruby on Rails application. Unlike other Ruby webservers, Puma was built for speed and parallelism. Puma is a small library that provides a very fast and concurrent HTTP 1.1 server for Ruby web applications.[30]

#### **4.4.4 Frameworks and Libraries**

Ruby was chosen as the main language to implement this project partly because of its large collection of open source frameworks and libraries.



## ***Ruby on Rails***

Where Ruby is the language, Rails is the framework that provides a rapid prototype environment and full web application stack [31]. It uses a Model-View-Controller (MVC) paradigm working in conjunction with a webserver such as Apache or Puma. Rails also offers a database object-relational mapping feature called Active Record, which simplifies the access and manipulation of database records.

## ***Gems***

Gems are a way to package various libraries for Ruby.

### *RGeo Gem*

RGeo is a ruby toolkit for manipulating geo-spatial data. The most useful features for my project are the distance calculations and projection mappings.

### *Roo Gem*

Roo implements read access for all spreadsheet types and read/write access for Google spreadsheets.

### *Sidekiq Gem*

Sidekiq is a full-featured background-processing framework for Ruby. PlayPin uses Sidekiq to offload API calls and complex calculations into background tasks for efficient processing. It also offers a web-based dashboard for management of worker threads.

### *RubyFish Gem*

Rubyfish can do approximate and phonetic matching of strings. PlayPin uses it to process and filter search results based on contextual keywords.

### *GoogleMaps for Rails Gem*

Google maps interface for Rails.

### *Hierclust Gem*

Hierclust is a simple hierarchical clustering library for spatial data.

### *Nokogiri Gem*

Nokogiri is an HTML, XML, SAX, and Reader parser.

### *OmniAuth Gem*

OmniAuth is a library that standardizes multi-provider authentication for web applications.

## ***HTML/JS/CSS Frameworks***

PlayPin is an interactive application and uses several UI libraries. The system uses a web server which serves HTML to the user. To capture and respond to user events we need to use JavaScript. To provide a pleasing graphical interface, we use CSS to control things like styling, fonts, colors and layout of the elements on the screen. The following libraries and frameworks enable user interaction and styling by extending native JavaScript and CSS functionality.

### *jQuery*

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and AJAX much simpler with an easy-to-use API that works across a multitude of browsers. All the user interaction of PlayPin uses jQuery to capture and process events. [32]

### *Backbone*

Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface. [33]

### *Highcharts*

Highcharts is a charting library written in pure JavaScript. PlayPin utilizes this in its analytics interface. [34]

#### *Twitter Bootstrap*

Bootstrap is a sleek, intuitive, and powerful front-end framework for faster and easier web development. All the UI web elements in PlayPin are styled with Twitter Bootstrap styles. [35]

## 4.5 Architectural Overview

The system architecture, illustrated in Figure 10, is based around the Ruby on Rails stack. Rails provides the tools to connect with the databases and serve up a web application rendered using HTML and CSS. To consume and process huge amounts of data, a “worker” system is used to offload expensive computations.

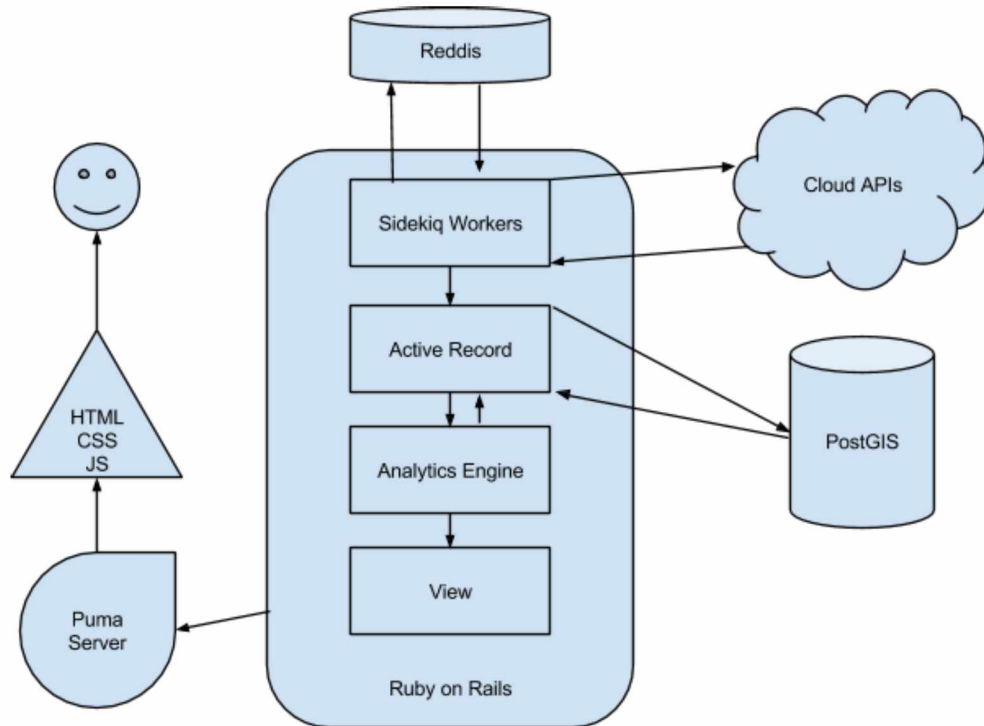


Figure 10. Systems Diagram

## 4.5.1 Ruby on Rails

### *Sidekiq*

API requests to various providers are handled using a RESTful interface. Since communication with APIs using REST can suffer from performance issues and errors, the process of making and responding to API requests is done using Sidekiq workers. This worker paradigm allows a single process to be spawned, run and retried if necessary. Since public API calls can fail for unforeseen reasons, it is important that processes handling them be robust and fault tolerant. If a single worker process fails it can be put to sleep for a set amount of time and retried in the future. This is especially helpful when APIs rate limit the amount of calls that can be made in a given time frame. Sidekiq also handles the processing of data resulting from these API calls. In addition to handling API requests, there are also Sidekiq workers to calculate distances between points, generate statistics from results, and process imported files. These are explored in more detail in Section 4.5.4.

### *Active Record*

The Object Relational Mapper (ORM) provides a useful layer of abstraction on top of the database. This allows for more compact Ruby code, which is then translated into SQL queries. Ruby on Rails uses an ORM implementation called Active Record that represents database tables as classes.

### *View Rendering*

The view rendering done by rails is responsible for presenting the data to the user in a readable format. This involves rendering and serving HTML pages that the user can interact with.

## 4.5.2 Model Classes

Active record exposes model classes that map directly to the database tables. This allows the software to seamlessly communicate with the underlying database without writing any SQL code. A diagram of these tables is shown in Figure 11.

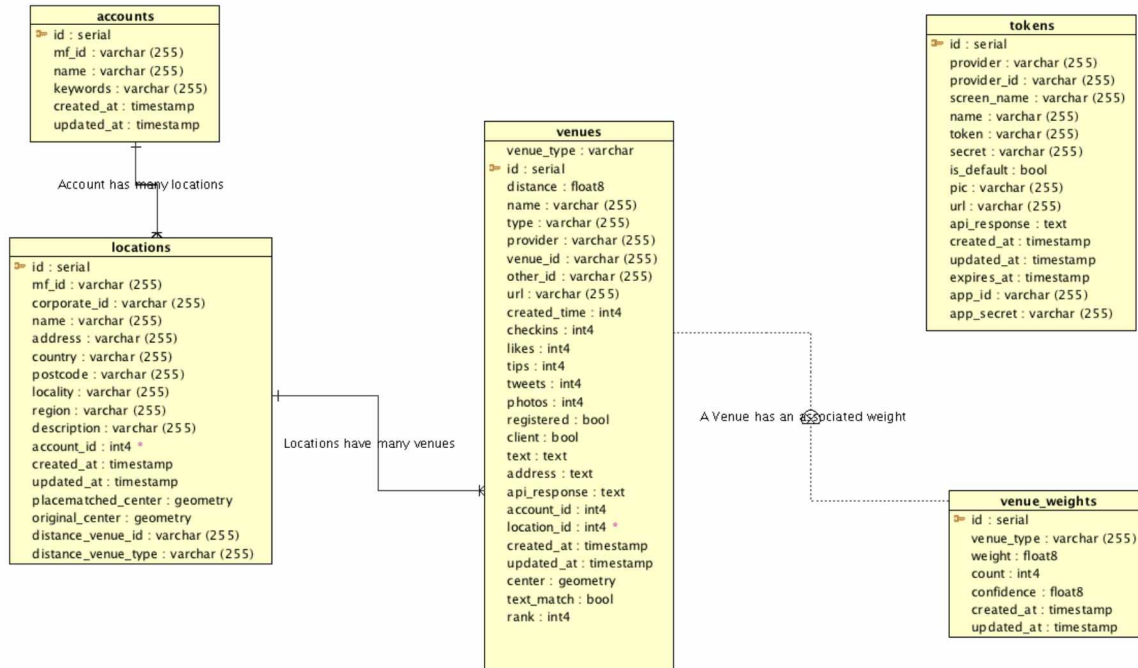


Figure 11. Database Schema

### **Account**

An account is the top-level object in the database. It binds together locations and venues. It also stores keyword metadata that is used by the workers.

### **Location**

The location class stores address data for each location and optionally the GS coordinate data.

### **Venue**

The venue class provides a wrapper for lat/long information generated from the workers. For each geocoding and backfill worker there is a corresponding venue object.

### ***Venue Weights***

These are the weights generated from the training data. For each geocoding provider or LBE provider there is an associated weight. For example, venues geocoded from Google Maps have a certain weight that is stored in this table.

### ***Token***

Tokens are the “passwords” that allow a user or program to interface with remote APIs. Since PlayPin communicates with many different APIs the management of these tokens is necessary.

### **4.5.3 Sidekiq**

Sidekiq is the background-processing framework that PlayPin uses for most of its calculations and API processing.

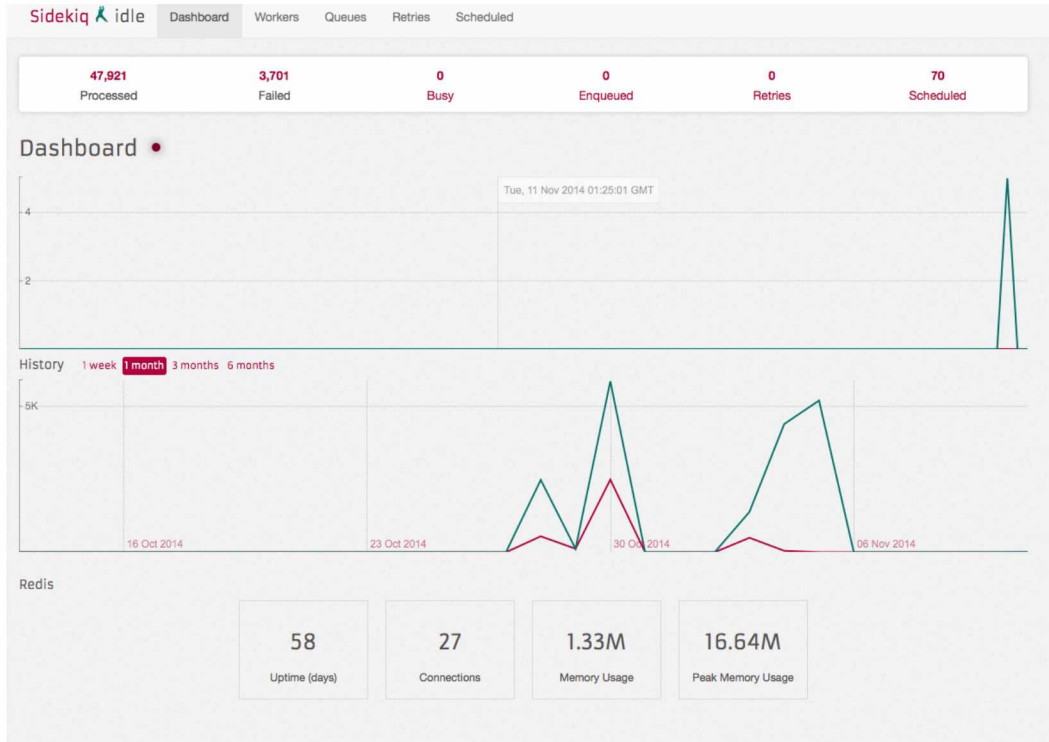


Figure 12. Sidekiq control panel

Sidekiq includes a dashboard and control interface where the status of the workers can be viewed. It also allows individual workers to be retried and inspected. Figure 12 shows the Sidekiq control panel. The graphs represent the how many workers are currently processing data over time. Each task the workers process can have different states including processed, failed, busy, enqueued and scheduled as seen in the top menu on the figure. When workers fail to process a task, the task can be retried a certain number of times. This is very important when the tasks include querying data from remote APIs. Often these requests can fail due to unpredictable factors such as network errors, server problems, and rate limits. Having the ability for the workers to automatically retry these failed tasks leads to a very robust processing system.



#### **4.5.4 Sidekiq Workers**

The Sidekiq workers are a core component of the system. They are responsible for all the asynchronous API calls and data processing. When dealing with processes that are error prone, such as making remote API requests, it is helpful to have an architecture that is robust enough to handle things like networks errors and API rate limiting. Sidekiq provides a means to allow failed tasks like API calls to be automatically retried a certain number of times. In addition, it provides a graphical interface that allows the user to monitor the status of all the data processing. When processes fail, Sidekiq allows you to interrogate the specific inputs and outputs around the process. For example, we have workers that handle only geocoding tasks. This involves taking address data and making a remote API request. If any of these tasks fail, Sidekiq will log both the input data and the failed results of the request for further inspection.

Sidekiq Worker	Description
GenericWorker	This is the base class for all workers
LocationImportWorker	It is responsible for parsing CSV and Excel files and importing locations into each account.
GeocodeWorker	This is the base geocoding class that contains functions shared by all inheriting classes. It has shared logic to retry failed API calls after a certain amount of time. In addition, it will intelligently batch certain geocoding calls as needed to save on computational and network resources.
BingGeocodeWorker	This worker processes geocoding using the Bing Search API.
GeocodeUsWorker	This worker processes geocoding using the geocode.us API.
GoogleGeocodeWorker	This worker processes geocoding using the Google Maps API.
MapQuestGeocodeWorker	This worker processes geocoding using the MapQuest API.
YahooYqlGeocodeWorker	This worker processes geocoding using Yahoo YQL Geocoder API.
YelpSearchWorker	This worker processes geocoding using a combination of Yelp API and screen scraping. Screen scraping is done with the Nokogiri Gem to retrieve actual lat/long results from Yelp venues.
BackfillWorker	The backfill worker is the abstract class that contains shared functionality for all backfill workers. All backfill workers take at minimum a geo-coordinate to search for LBE.

FacebookLocationWorker	Search for Facebook place pages using the Facebook Graph Search API.
FlickrPhotoWorker	Searches for Flickr photos around a location, processes keyword matches on photo meta data.
GooglePlacesWorker	Searches for Google Places around a coordinate using the Google Place Page API.
InstagramPhotoWorker	Searches for photos around a coordinate using the Instagram search API. Optionally filters photos using keywords.
TwitterTweetWorker	Searches for tweets containing keywords around a location using the Twitter search API.
VenueRecalcWorker	This processing worker recalculates the distance between two venues. It becomes useful when analyzing various distances.
CentroidWorker	This processing worker calculates a centroid, and optionally a weighted centroid for a location. It takes any number of venues and weights as parameters. It uses a heuristic along with a modified nearest neighbor algorithm to exclude outliers from the centroid processing.

Figure 13. Table of Sideiq Workers

### 4.5.5 Helpers

Helpers are various utilities that perform specific tasks.

Helper	Description
Outlier Detection	Used in conjunction with the CentroidWorker, this helper class uses clustering algorithm to detect outliers in a set of venues.
Mercator Projection	This helper function allows us to project a lat/long coordinate to the Mercator projection. This enables us to use Cartesian distance calculations that are much faster than spherical calculations.
Centroid Calculation	Uses the outlier detection along with the heuristic described in section 6.3 to find a cluster of venues, of which the centroid is calculated.

Figure 14. Table of Helper Utilities

## **Chapter 5 : Experiments**

### **5.1 Experimental Steps Overview**

The primary goal of this project is to generate accurate geo-coordinates at scale given structured location address data. To do this, several preliminary steps must be completed within PlayPin.

1. Create an account on PlayPin that will be used to associate the location data and relevant keywords relating to locations themselves.
2. The list of location addresses is imported into the database and converted into a standardized format.
3. Each location's data is processed through various public geocoding APIs.
4. Clustering techniques are used to exclude outlier data and find the weighted centroid of the reduced point set.
5. With this centroid Backfilling is used to query public social media APIs to find relevant and spatially constrained location based engagement around the centroid.
6. Using this new and relevant set of points, the points are again clustered into a more spatially constrained set and the weighted centroid is generated.
7. The resulting output is the final point estimation.
8. The generated points for all locations are compared to the various geocoding APIs and to the true, GS geo-coordinates.
9. These comparisons will ultimately determine if the system-generated coordinates are more accurate than any of the individual geocoders used.

## 5.2 Data Set

The data set we are using for our experiment is a list of 769 Outback Steakhouses and 3,518 Sonic Drive Through locations representing a total set of 4,287 locations. The addresses were gathered from publicly available address data. A sample row of location data looks like this:

name	address	locality	region	postcode	country	latitude	longitude
Outback Steakhouse	101 W. 34th	Anchorage	AK	99503	USA	61.1895	-149.883

Figure 15. Sample Location Data

Figure 15 shows a sample of location data which includes a location name, street address, city, state, zip code and country. In addition, we have the latitude and longitude. This coordinate is the Gold Standard (GS) coordinate, which will be used to measure the accuracy of the system. It is important to note that this coordinate will only be used to evaluate the accuracy of PlayPin.

## 5.3 Account creation and data import

For each set of data, we create an account record to hold relevant metadata and to be able to analyze each set independently. This is done through the account creation screen. Here we can add an identifying ID, account name and relevant keywords that will help filter the LBE data.

**PlayPin** Accounts Locations Analytics Jobs Tokens Sidekiq

## Edit Account

**Mf**  
5140f445619ff9507c00077f

**Name**  
Outback Steakhouse

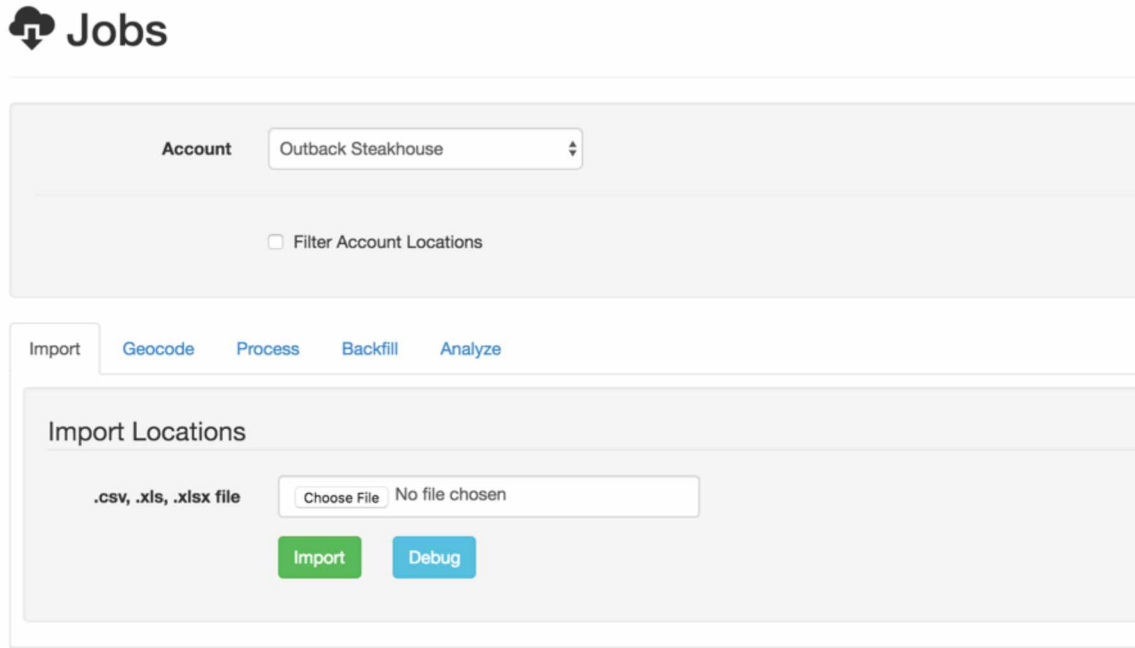
**Keywords**  
Outback

[Update Account](#) [Cancel](#)

© Adrian Mummey 2013

Figure 16. Playpin Account Creation Interface

Following the account creation, we upload the structured data into the system so that it can be copied into the local database



© Adrian Mummey 2013

Figure 17. Location Import Interface

Using the Import Locations interface, we can upload a structured location data file in CSV or MS Excel format. The file will be processed and the locations will be imported into the database.



## 📍 Locations

1 2 3 4 5 ... Next > Last >

Id	Name	Address	Account	Original center	Created at	Actions
9227	Outback Steakhouse	60 W. 23rd St.	Outback Steakhouse	POINT (-73.9915 40.7424)	Sun, 25 Aug 2013 05:34:17 +0000	Edit Delete
9228	Outback Steakhouse	280 Marsh Ave.	Outback Steakhouse	POINT (-74.1637 40.5775)	Sun, 25 Aug 2013 05:34:17 +0000	Edit Delete
9545	Outback Steakhouse	60 S. Broadway	Outback Steakhouse	POINT (-73.7611 41.0305)	Sun, 25 Aug 2013 05:34:58 +0000	Edit Delete
9588	Outback Steakhouse	1703 Central Park Ave.	Outback Steakhouse	POINT (-73.8419 40.9572)	Sun, 25 Aug 2013 05:35:12 +0000	Edit Delete
9160	Outback Steakhouse	1537 Riverdale St.	Outback Steakhouse	POINT (-72.629 42.1453)	Sun, 25 Aug 2013 05:34:15 +0000	Edit Delete
9587	Outback Steakhouse	30 Indian Rock Rte. 59	Outback Steakhouse	POINT (-74.1152 41.1123)	Sun, 25 Aug 2013 05:35:12 +0000	Edit Delete
9240	Outback Steakhouse	25 Crystal Run Crossing	Outback Steakhouse	POINT (-74.3648 41.4465)	Sun, 25 Aug 2013 05:34:17 +0000	Edit Delete

Figure 18. List of locations in PlayPin

## 5.4 Training and Generation of Weights

Overall, searches using multiple providers are not very precise. That is, the results for each location can vary substantially. On average, though, over many locations, certain geocode providers tend to be more accurate than others. Using the results of the geocoding of training data we can calculate a weight for each provider based upon its overall accuracy. The weighting function is this:

$$\omega = \frac{\#location\ hits}{\# locations}$$

These weights are calculated and stored in the database for each of the providers that we gathered geocoding data from.

### 5.4.1 Training Data Geocoding

To seed this data, we first take a small subset of location data and process it to generate weights. The first step is to geocode the subset with all the available providers. For this experiment, we will use the Outback dataset as the “training” set in order to generate these weights. It is important to note, that for all locations we have a corresponding GS geo-coordinate to measure distance against.

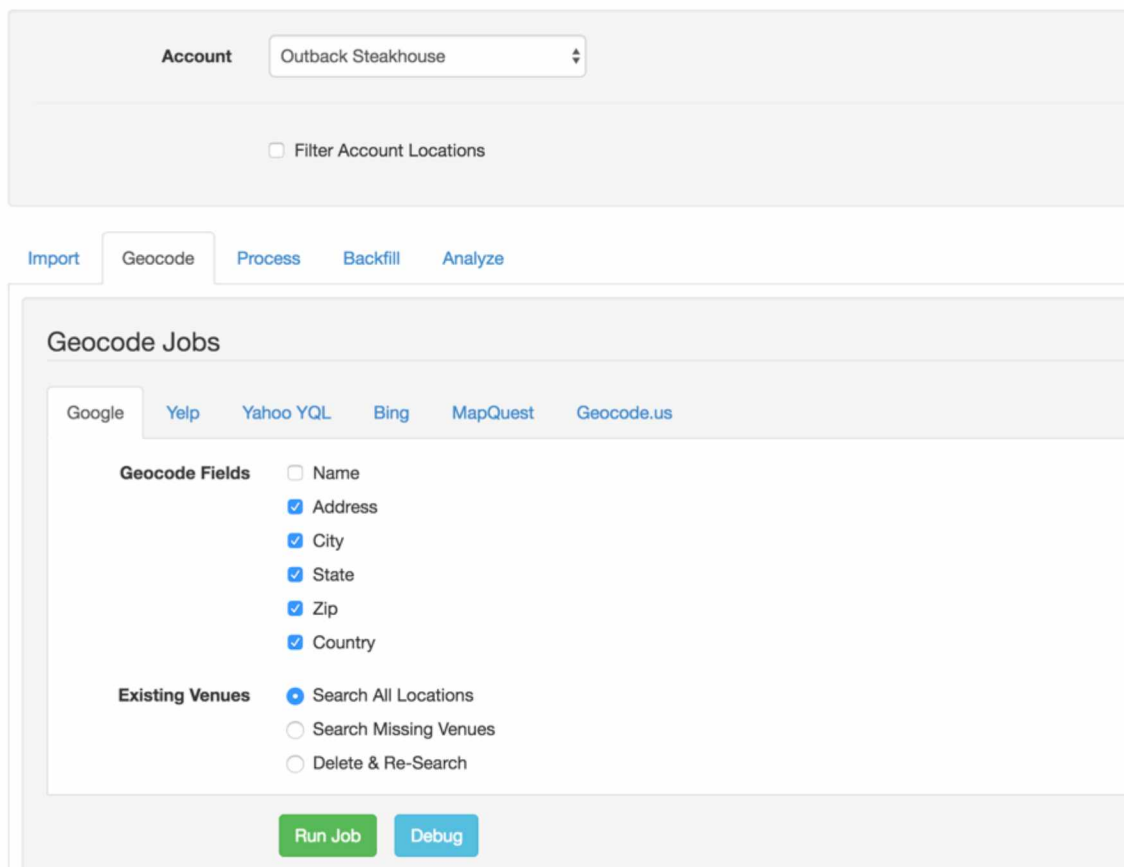


Figure 19. Google Geocode Job Interface

Figure 19 shows the interface to run the various geocoding jobs. Each geocoding provider expects different types of parameters in their API requests and this interface allows you to customize the parameters that are sent. For the purposes of training, we will run all the geocode provider jobs against the Outback Steakhouse location data.

### 5.4.2 Distance Calculations

Once we have a full set of geocoded points, we can process the weights for each geocode provider. This involves first calculating the distance from the GS coordinate (shown in the Figure 20 as “Placematched Center”) to the geocoded point.

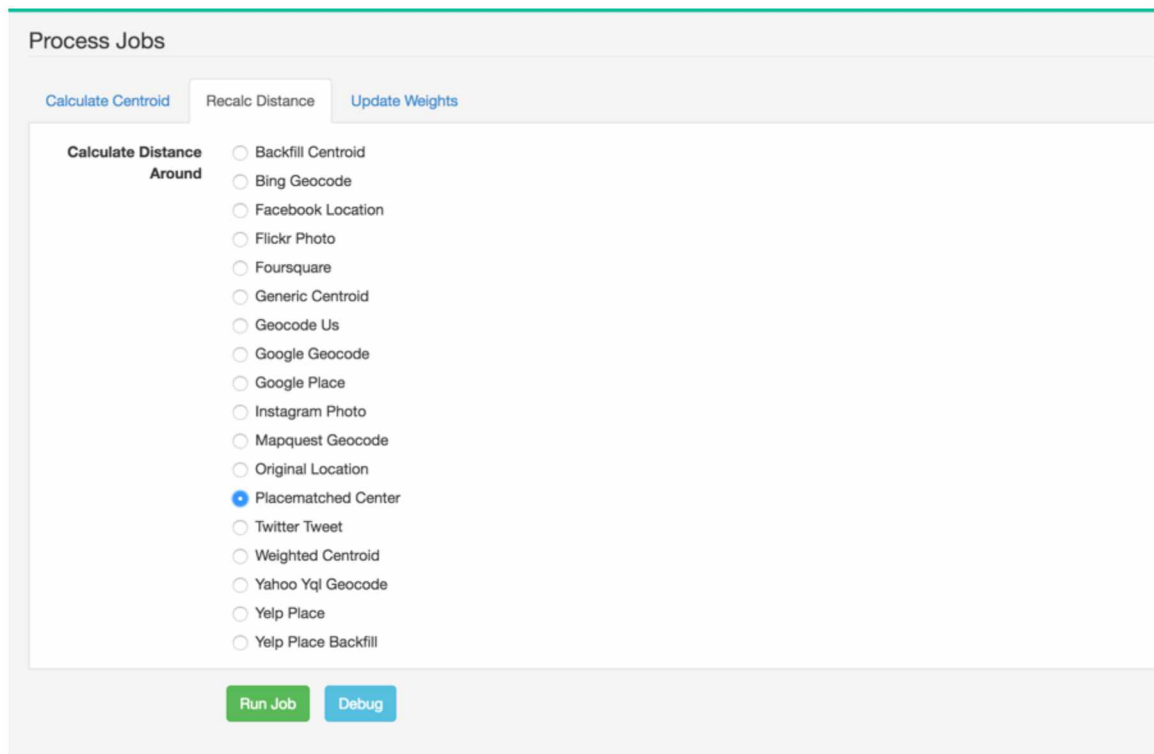


Figure 20. Distance Calculation Job Interface

There is a separate job to calculate and store the distance of all venues around a specified point. We will always use the GS coordinate when possible. When this job finishes, we will have a set of data that includes the distances of all the geocoded points to the GS coordinate.

### **5.4.3 Weight Processing**

We can now calculate the weights. The weight of a geocode provider is the ratio of location hits, to the total locations queried. For the purposes of this experiment we are considering the set of “hits” to be the coordinates that fall within 50m of the GS coordinate and “misses” to be the coordinates that fall outside 50m distance to the GS coordinate or coordinates that don’t have any resulting data after geocoding.

Account: Outback Steakhouse

Filter Account Locations

Import   Geocode   **Process**   Backfill   Analyze

**Process Jobs**

Calculate Centroid   Recalc Distance   **Update Weights**

Max. Distance Cutoff (m): 50

Run Job   Debug

© Adrian Mummey 2013

Figure 21. Weight Processing Interface

We use the Update Weights job in PlayPin to process these weights using a distance cutoff of 50m.

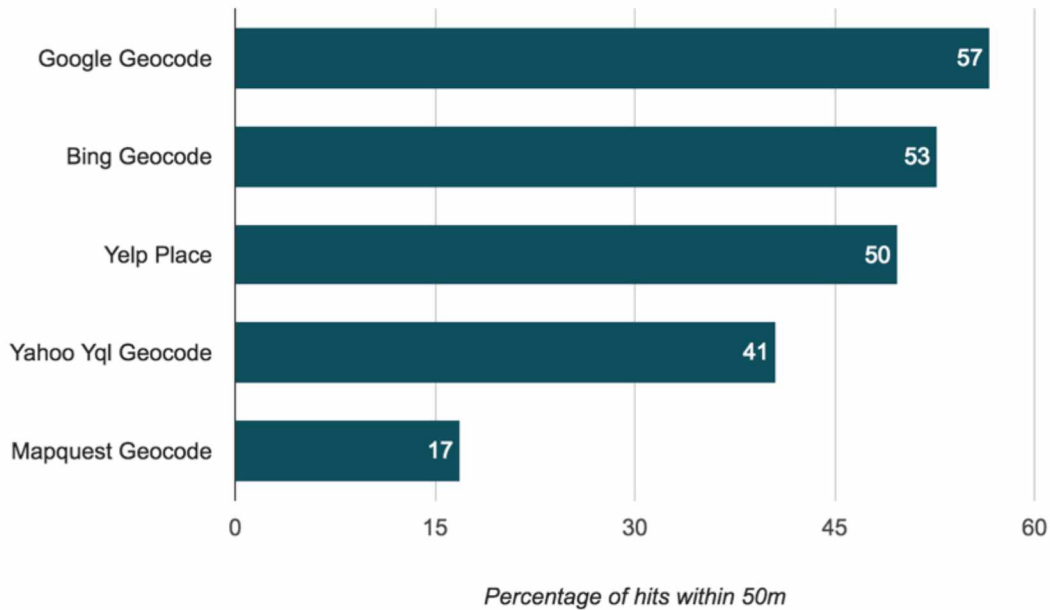


Figure 22. Training Results for Outback location data

Figure 22 shows us the percentage of location hits from each geocoder processed from the training data. These values directly correspond to the weight of each geocoder and will be stored in the database to be later used in the experiment.

## 5.5 Coordinate Estimation

Once the weights have been established on the training data, we can set up the accounts and locations in a similar matter for the experimental set of data, which will include an additional 3,518 Sonic Drive Through locations. The following diagrams detail the experimental steps by visualizing a single location's geocoding and LBE data as it flows through PlayPin:

1. Figure 23 illustrates the results of querying the geocoding providers for geocoordinates for each location. We use a similar process described in Section 5.4.1 for geocoding the locations. Figure 24 shows a zoomed in view of the cluster on the left. \*Note the dispersal of coordinates and the outliers:



Figure 23. A view of geocoding query results for a sample location



Figure 24. A zoomed in view of the bottom left cluster from Figure 23

*2. Use the Calculate Centroid job shown in Figure 25 to generate a weighted centroid. This process uses a clustering algorithm to exclude outliers and generate a weighted centroid from the resulting set. In Figure 26 we can see the clustered points that contribute to the centroid as the points of the green triangle and the centroid in the middle. The yellow star represents the GS coordinate.*



**Process Jobs**

Calculate Centroid    Recalc Distance    Update Weights

Venue Types to Include	Geocode Venues	Backfill Venues
	<input checked="" type="checkbox"/> Bing Geocode	<input type="checkbox"/> Google Place
	<input checked="" type="checkbox"/> Geocode Us	<input type="checkbox"/> Instagram Photo
	<input checked="" type="checkbox"/> Google Geocode	<input type="checkbox"/> Facebook Location
	<input checked="" type="checkbox"/> Yelp Place	<input type="checkbox"/> Foursquare
	<input checked="" type="checkbox"/> Original Location	<input type="checkbox"/> Flickr Photo
	<input checked="" type="checkbox"/> Mapquest Geocode	<input type="checkbox"/> Twitter Tweet
	<input checked="" type="checkbox"/> Yahoo Yql Geocode	<input type="checkbox"/> Yelp Place Backfill

**Centroid Type**    Generic Centroid

**Kth Nearest Neighbor**    2

**Min. Neighbor Distance**    25

**Recalc failed clusterings (Max distance to go m.)**     1000

**Run Job**    **Debug**

Figure 25. Centroid Calculation Interface

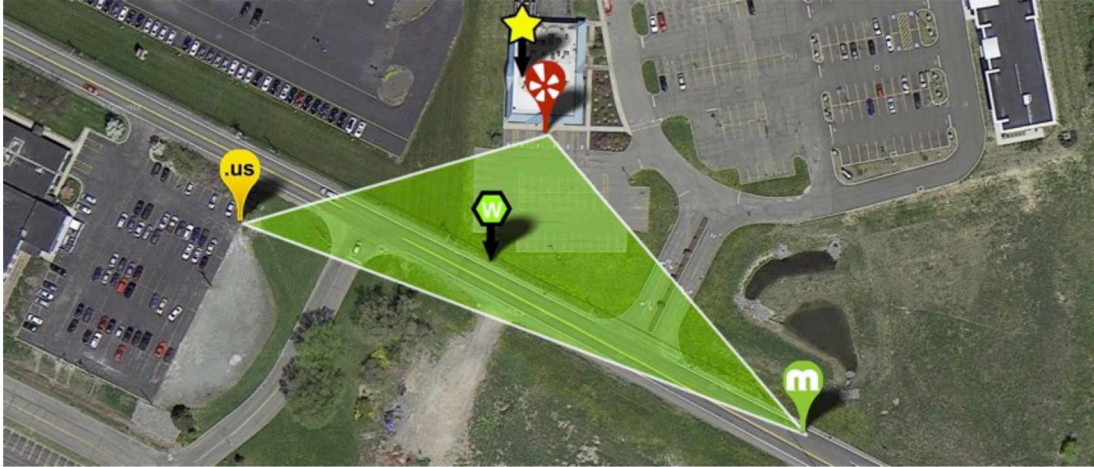


Figure 26. Weighted centroid in middle of cluster represented by green hexagon.

3. Use the Backfill Job shown in Figure 27 to backfill location based engagement (LBE) data around the weighted centroid. Note the spatial and contextual relevancy of the highlighted LBE data shown in Figure 28.

Import Geocode Process **Backfill** Analyze

### Backfill Jobs

Instagram Photos **Tweets** Facebook Places Foursquare Venues Google Places Flickr Photos

**Date Range**

**Radius (m)**

**Match by Keyword(s)**  Outback

Add Location Name to Keywords

**Search around point**

---

**Existing Venues**  Search All Locations  
 Search Missing Venues  
 Delete & Re-Search

Figure 27. Location Based Engagement Backfill Interface

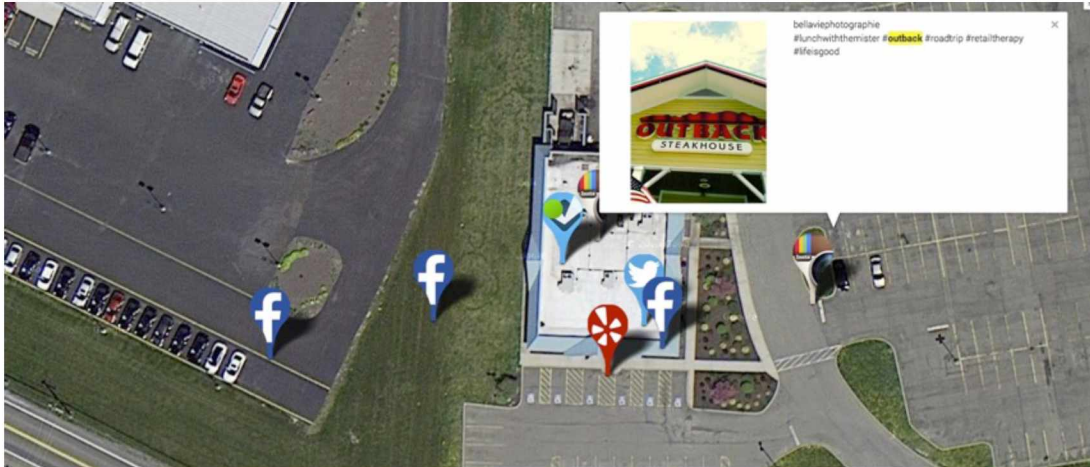


Figure 28. Contextually and proximally relevant LBE around a location

4. At this point we once again run the centroid calculation but this time using the LBE data. In Figure 29 the points on the pink polygon represent the clustered LBE points contributing to the new centroid.

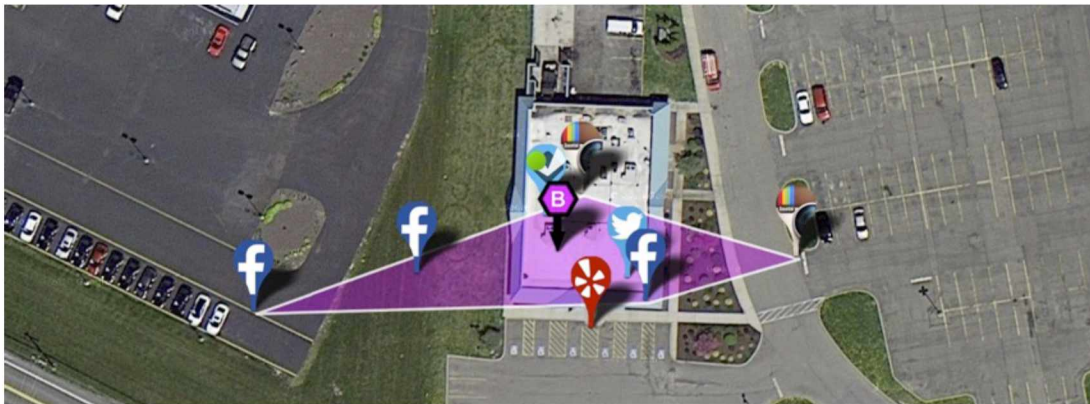


Figure 29. Backfill centroid and its cluster represented by pink hexagon.

5. In Figure 30 note the proximity of the backfill centroid versus the weighted centroid to the GS coordinate denoted by the yellow star



Figure 30. Comparison of backfill & weighted centroids

## **Chapter 6 : Experiment & Analysis of Results**

Now that the experimental steps have been outlined, we can now cover the actual experiment and analysis of the results.

### **6.1 Geocoding processing**

To recap, geocoding is the process of converting street address data (street, city, zip, state) into geo-coordinate data. The first step of the experiment is to query various geocoders with the address data we have imported. A sample Google Maps API query is shown in Figure 31, with the result shown in Figure 32.



```
http://maps.googleapis.com/maps/api/geocode/json?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=true_or_false
```

Figure 31. Sample Geocoding Query (Google Maps API)

```
    }
  },
  "formatted_address" : "1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA",
  "geometry" : {
    "location" : {
      "lat" : 37.42291810,
      "lng" : -122.08542120
    },
    "location_type" : "ROOFTOP",
    "viewport" : {
      "northeast" : {
        "lat" : 37.42426708029149,
        "lng" : -122.0840722197085
      },
      "southwest" : {
        "lat" : 37.42156911970850,
        "lng" : -122.0867701802915
      }
    }
  },
  "types" : [ "street_address" ]
},
"status" : "OK"
}
```

Figure 32. Sample Google Maps API response

In the case of the above API query to Google Maps, we make an HTTP request to the Google Maps API using the above URL, and include a parameter for the address we are searching. In this case, it is “1600 Amphitheatre Parkway, Mountain View, CA”. A successful request will yield a response encoded in JSON format. The response will include a field for “location” which will contain the geocoded latitude and longitude data.

Using the robust worker architecture of PlayPin we can run thousands of these queries asynchronously on many different geocoding providers. When rate limits occur on the various providers, PlayPin will automatically wait and try the API requests when the limits have been lifted.

For this experiment, we will run queries using the following providers:

- Google
- Yelp
- Yahoo
- Bing
- MapQuest
- Geocode.us

The geocoding process involves querying each of the different APIs with a request for an address, extracting the latitude and longitude data (if found), and then storing the resulting coordinates in a common data structure. Once the geocoding is finished we can plot the results on a map and explore them on a per-location basis:





Figure 33. Results from geocoding a single location

In Figure 33 we can see where the various geocoding providers including Yahoo, Bing, MapQuest, Yelp and Geocoder.us think the location is. The star represents the GS coordinate.

## **6.2 Geocoding Results**

For each location, we have made at least six requests to the remote APIs for data, resulting in about 25,000 total API calls. By comparing the gathered geocoding data with our Gold Standard data, we can calculate some initial results. The methodology here is to find a way of classifying a positive result versus a negative result. For the purposes of this project we assume that any coordinate within 50 meters of the actual location will be classified as a positive result and any coordinate outside of 50 meters, or a failure to return any result will be classified as a negative result. To recap, we will refer to the locations that fall within 50m as the set of “hits” and those that fall outside or fail to have a value as the set of “misses”. When we analyze the output from the geocoders in a histogram, we are presented with this:

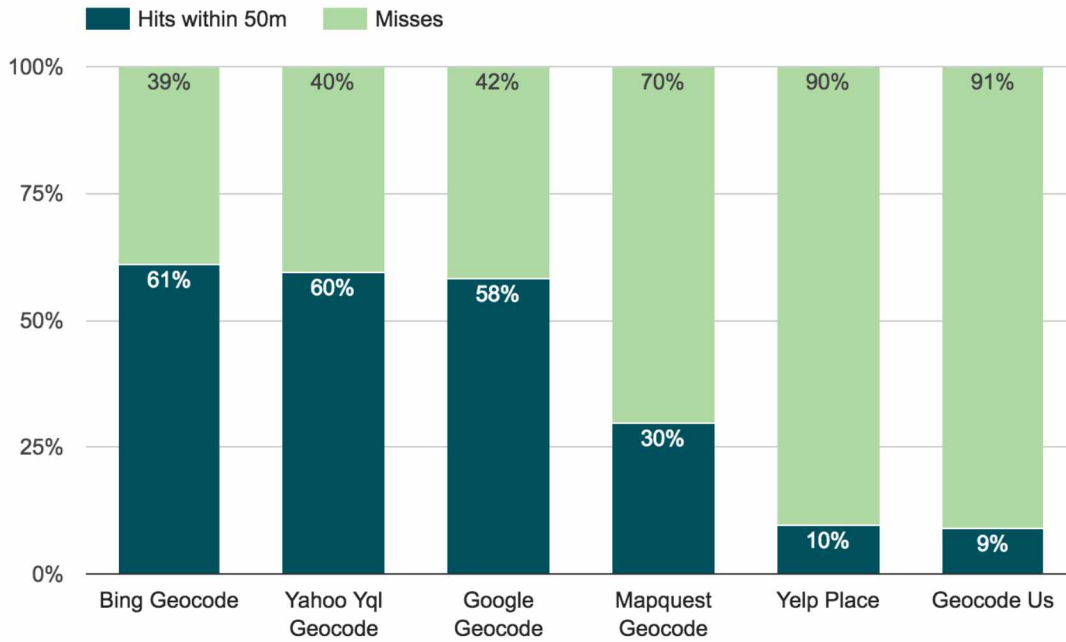


Figure 34. Histogram of hits vs. misses of geocoding results

As we can see above, for our experiment Bing was the most successful geocoder with 2,617 hits and Geocode.us as the worst performer with just under 400. This shows that Bing, as the top-performing geocoder, was only able to generate a “hit” about 60% of the time. In other words, if you used Bing to direct you to any of the locations tested, it would only get you to within 50m about 60% of the time.

### 6.3 Clustering and Centroid Estimation First Pass

To increase the accuracy of a point estimate based on the centroid of the geocoded points we can exclude outlier points from the set using clustering techniques. The technique used here is a modification of the **k-nearest neighbors** algorithm (See Appendix C for details). We use this to classify a grouping of points that are close to one another referred to as a cluster.

For the purposes of this project we wish to divide the points into two classes, points contributing to the cluster and outliers. The metric we use to compare the points is the linear distance between them.

To use the clustering algorithm, we must define two values, first is the class membership function and second is the value of  $k$ . The class membership function is defined as

$$C(X) = d(X) < Y$$

where  $d(X)$  is the linear distance from point  $X$  to its  $k^{th}$  nearest neighbor and  $Y$  is some distance in meters. Through repeated simulation, I have found  $k = 2$  and  $Y = 50m$  produce acceptable results. In the case where a cluster cannot be found, a heuristic ensures that a set is generated:

1. If a suitable cluster is not found, increase  $Y$  by a fixed amount and try again
2. If  $Y$  exceeds some value  $Z$ , then  $k = k - 1$ , reset  $Y$  to original value
3. If  $k = 1$  then add the top weighted geocoded point to the set and return

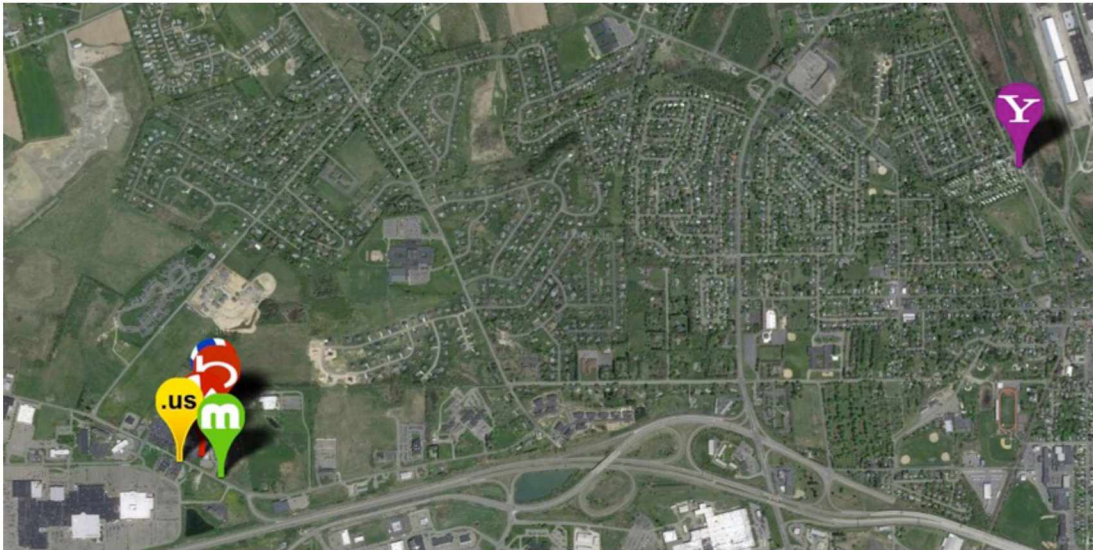


Figure 35. Cluster of geocoded points with outlier

The heuristic ensures a set because if a suitable cluster cannot be found, the top weighted point is always selected as the set. In Figure 35 we can see the results of a geocoding pass where there is a clear outlier in the set. After running the clustering algorithm, we will be left with a reduced set that will be used in centroid calculation.

### 6.3.1 Centroid Calculation

To estimate a point using our set we use a centroid  $C$ , which is the geometric center of the region formed with our points. It is defined as

$$x \in X \quad C = \frac{\sum_1^n x_i}{n}$$

Figure 36. Centroid Equation

The centroid calculation involves summing all the points in  $X$  and dividing by the number of points. In Section 2.1 we discuss the distance calculations between points on earth, Figure 36 defines a simple geometric mean of the points and will not work correctly when working with the spherical latitude and longitude points. To efficiently find the centroid of the geo-coordinates in close proximity, we can project the spherical coordinates onto a plane using the Mercator projection, find the centroid of the projected points, and finally project them back into spherical coordinates. We will refer to this calculated value as the Generic Centroid.

In addition to calculating the centroid we will also calculate a Weighted Centroid with the weights we generated in training. The weighted centroid is defined in Figure 37.

$$x \in X \quad C = \frac{\sum_1^n \omega_i x_i}{\sum_1^n \omega_i}$$

Figure 37. Weighted Centroid Equation

The weighted centroid is defined as the sum of the product of a provider weight and a provider point divided by the sum of all the provider weights.



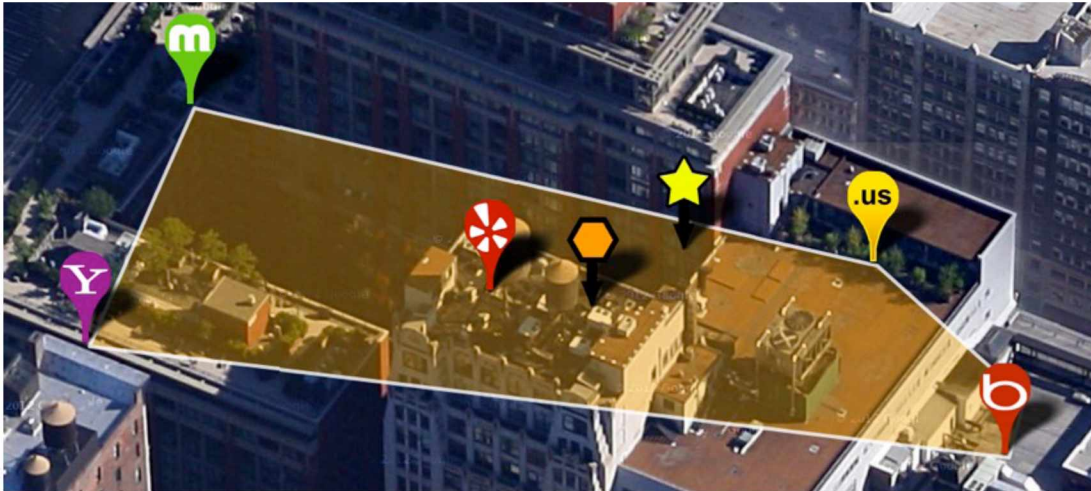


Figure 38. Centroid of Geocoded points represented by orange hexagon

Figure 38 shows a set of geocoded points and the convex hull that surrounds them. We can see the generic centroid represented by the orange hexagon and the GS location represented by the yellow star. From a visual standpoint, it appears that the generated centroid is closer than any of the individual provider geocoded points.

### 6.3.2 Centroid Results

For each location, we used the clustering algorithm (Appendix C) to reduce the point set thus excluding outliers. Using this constrained point set we calculated both the generic centroid and weighted centroid of each set of points.

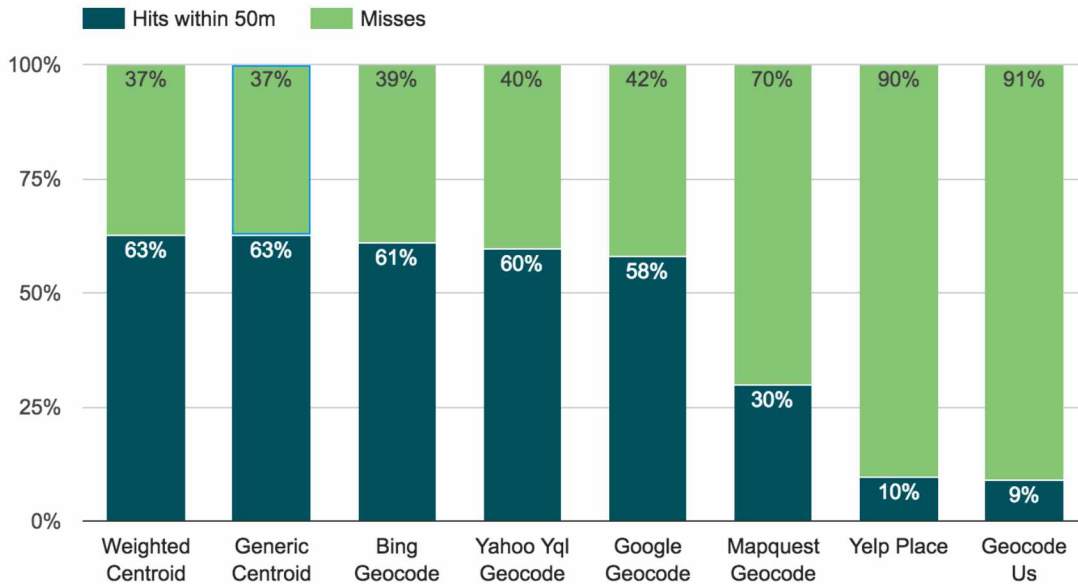


Figure 39. Histogram of hits vs. misses of weighted centroid results

We can see that the generated centroids overall fall within 50 meters of the GS location more often than any other provider, just slightly edging out Bing in terms of accuracy. The weighted centroid performs just slightly better than the generic centroid.

## 6.4 Backfilling

Backfilling is the process of gathering Location Based Engagement around a point. An example of a backfill would be gathering all the photos taken within 500 meters of a geo-coordinate. Using the search APIs from various social media providers, we can conduct these types of queries.



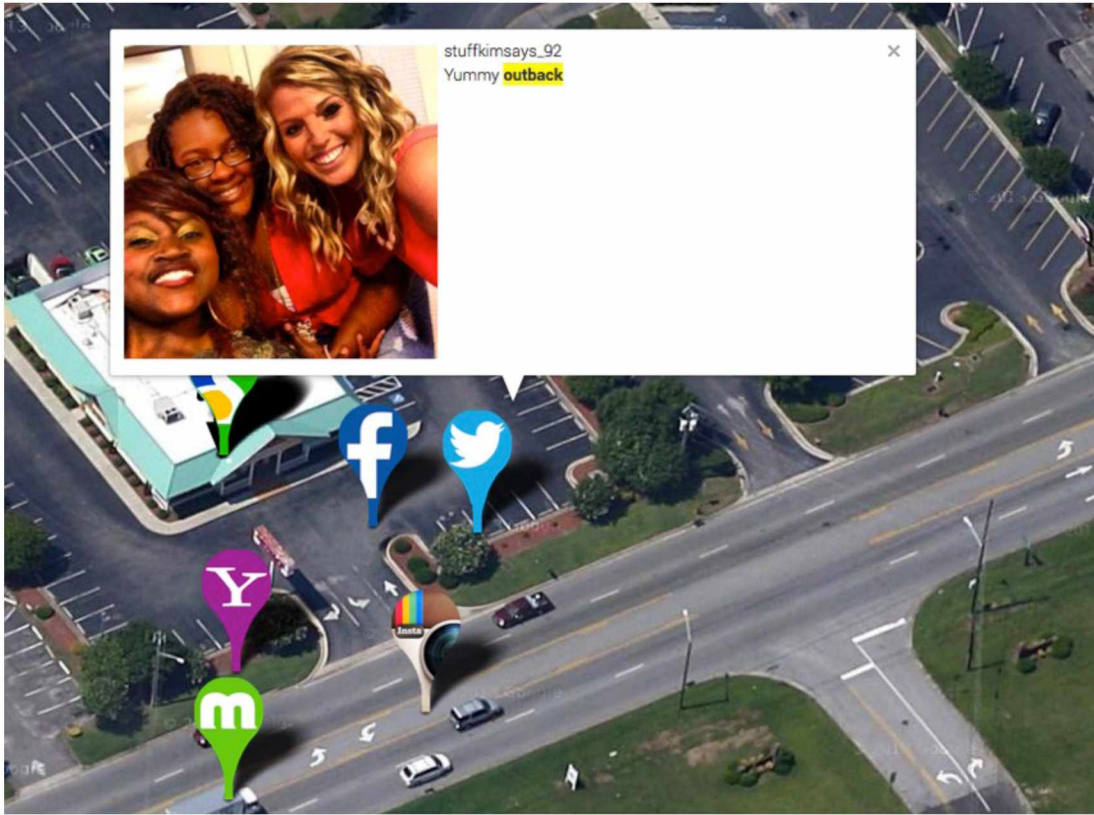


Figure 40. A Tweet that happened in proximity to an Outback Steakhouse.

The idea behind backfilling is that relevant social media data can be useful in determining location. There are two factors that contribute to the relevancy of a piece of social media. First is the proximity to the location and second is the inclusion of keywords associated with the social media metadata. To address the first factor, we only gather data that falls within a specified radius around the location. The second factor is addressed by analyzing keywords contained within the social media activity. For this experiment, I use a radius of 250 meters as the cutoff for proximity of a social media action. Classification for contextual relevancy of the keywords differs for each provider as follows:

- Twitter: tweet must contain one of the specified keywords
- Facebook Location: Location's name must match Facebook location name
- Flickr Photo: keywords are used as a search parameter
- Google Place: Location name is used as search parameter
- Instagram photo: Photo comments must contain at least one keyword

I have developed a heuristic to gather and process the backfill data for each location.

1. Use the weighted centroid as the point around which the search is conducted. If the weighted centroid is not available, use the next best point determined by the previously calculated weight. In other words, if the weighted centroid is not present, use the point returned by Bing, if Bing is not available use Google.
2. Query each of the social media providers for data around the point.
3. Classify each piece of social media data as spatially relevant. That is, filter out the LBE that is further than 250m from our search point.
4. Classify each piece of social media data as contextually relevant by using keyword matching.
5. Create a set of relevant data that meet both classifications

After each location is processed with the above heuristic we will have a set of relevant social media data, each with a geo-coordinate that will be used for a second pass estimation. Like the geocoded data, we can generate weights for each provider based on our weighting formula.

### 6.4.1 Backfill Results

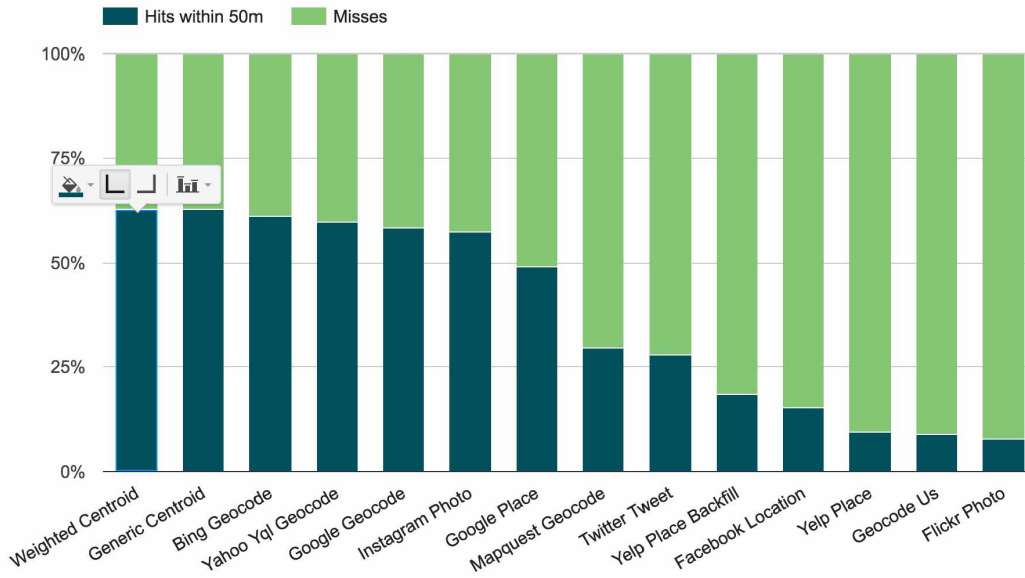


Figure 41. Histogram of hits vs. misses of backfill results

Figure 41 illustrates that the points produced by the backfills don't seem to perform very well. But it is notable that not every location has each type of LBE, some may have only tweets and some may have only photos.

### 6.5 Clustering and Centroid Estimation Second Pass

The clustering and centroid calculation happen in a similar manner as with the geocoded points, however, this pass will only include the points within the set of relevant LBE data and we will use a smaller  $Y$ ,  $Y = 10m$  for our clustering algorithm to provide a tighter spatial constraint for the resulting set.

To ensure a set generation for the second pass we use the following heuristic:

1. If a suitable cluster is not found, increase  $Y$  by a fixed amount and try again
2. If  $Y$  exceeds some value  $Z$ , then  $k = k - 1$ , reset  $Y$  to original value and try again
3. If  $k = 1$  then add the value of the geocoded centroid to the set

We will refer to this as the Backfill Centroid as the centroid generated from the LBE data.

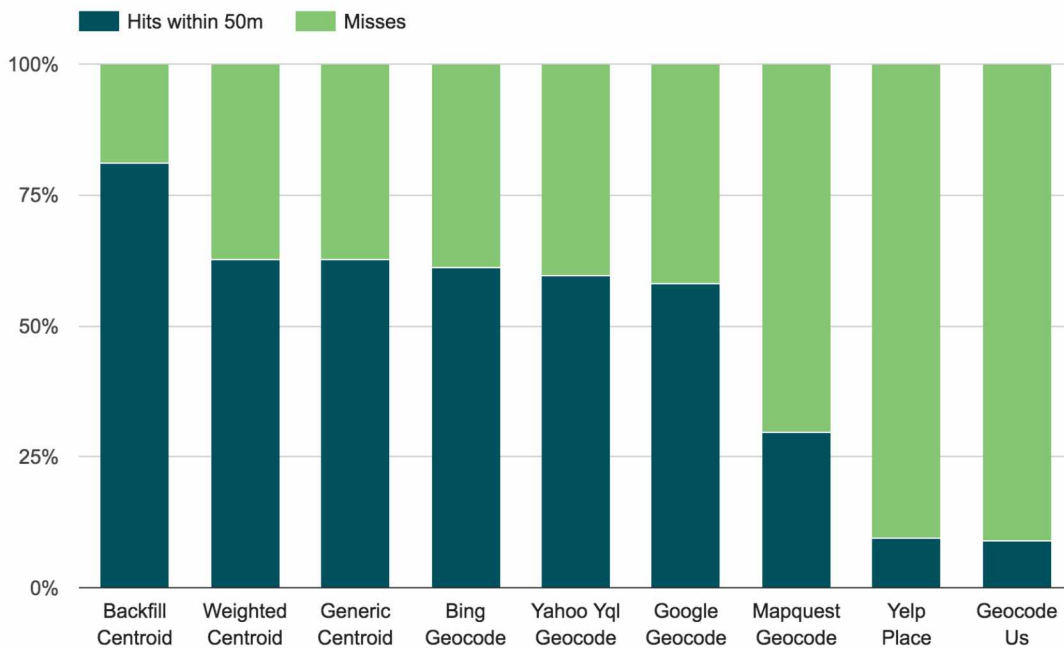


Figure 42. Histogram of hits vs. misses of backfill centroid results

We see that the backfill centroid has more hits than any other centroid or any other single geocoding provider. In fact, the backfill centroid includes approximately 50% more points within 50m than the next best geocoding provider Bing and represents a significant improvement over the weighted centroid.

Figure 42 gives a good validation of the second pass using LBE data and of the process of estimation. There is a significant improvement in accuracy when compared to all the other methods.

## **6.6 Experimental Discussion**

Figure 42 shows us that PlayPin has produced a hit rate of 80% within 50m and the best geocoder, Bing, achieves a 60% hit rate on the same data. The main reason that this system works so well is that it can collect a large amount of candidate coordinates that are within close proximity to the GS coordinate. By using multiple geocoders, and prior knowledge from the training set of data, we can generate an aggregate point on the first pass.

It is important to make a distinction in the types of locations that this process works for. Like all the geocoders tested, this process only works for locations with an address, and as some of the geocoders I used restrict those addresses to the United States. The second pass that integrates the LBE data would only work well for locations that have contextually relevant LBE data around them. That is, locations where people are engaging with social media and adding contextual clues to their content. A good example of this is restaurants where people sitting in the restaurant are taking pictures of their food or tweeting about where they are.

The PlayPin estimation process would work much better for say a Starbucks than someone's home because a private residence gets much less contextually relevant LBE than a Starbucks. Large brands could potentially use this process of estimation to validate or generate accurate geo-coordinates for all their locations in an automated fashion. Rather than manually determining the exact latitude and longitude for each of their locations, they could simply run PlayPin and get very good results.

Although the results of my experiments showed clear improvement over other geocoding providers, there are still a substantial percentage of locations that were not accurately estimated. From my investigation, it seems that the problem with some of these locations, wasn't due to the process, but rather it was due to bad data. Going back through the locations that were not geocoded I discovered that several were either improperly formatted or referenced locations didn't exist. Unfortunately, there is nothing my current process of estimation can do to improve the accuracy of these types of addresses. I have found services on the Internet that claim to "clean" address data, that is, they take improperly formatted address data and attempt to format it correctly [36]. Using a service like this prior to loading the structured location data in PlayPin could improve the results of the geocoding.

## **Chapter 7 : Conclusions**

As a result of my experimentation I have concluded that the geocoding providers that were tested have significant problems with inaccuracies. The best geocoder, Bing, was successful in producing a coordinate within 50m in only 60% of the locations tested. The worst geocoder, Geocode.us, had a hit rate of only 9%. The source of this problem can be attributed to bad data coming from the providers themselves. In the context of physical stores and brands that have retail locations, these geocoding inaccuracies can have disastrous consequences

For the locations used in this study, I have demonstrated that it is possible to generate more accurate latitude and longitude estimates by using the method I developed. I have shown that by taking the results from a variety of geocoding providers and using a weighted average of these results, overall we get more accurate results than any individual provider of data. The main reason this works is that the algorithm used separates any outlying geographic points from the cluster that is interrogated. This minimizes the error contribution from any individual provider of data. Furthermore, we can get even more accurate results if we consider the Location Based Engagement (LBE) data around the estimated point. LBE data works well because we can filter it by its context and proximity. In other words, we can get data that originates from or close to the location we are looking for. Overall, using the two-pass process for geocoding we can get very accurate results. On the experimental set of data, the system achieved approximately 80% of hits within 50m versus the best geocoder Bing with about 60% of hits.

Within the bounds of this experiment, PlayPin has achieved its goal of generating accurate geo-coordinates from structured location data. Although I haven't found any other studies that employ this exact methodology, I did come across a social media company called Foursquare that is using social media data to help define location boundaries. Their system, Quattroshapes, is based on social data called Foursquare checkins and Flickr Photos [37].



## **Chapter 8 : Future Work**

During the development of PlayPin and after running the experiment, I discovered quite a few areas of improvement. Improvements are divided into three main areas discussed in detail below.

### **8.1 Usability**

One of the major problems with PlayPin as useable software is that there is a very specific set of steps that needs to be followed to get optimal results. This process of loading the data and initiating all the steps required to generate coordinates is a bit cumbersome for new users. If the program itself were ever to be adopted as a tool for location estimation, it might be helpful to automate the entire process. Currently there are many options to be filled in and switches to be flipped before any meaningful data can be collected. If this point generation process could be automated into a simpler process for the user, it would make the system much easier to use overall. Currently a sample workflow as it stands would be this:

1. Create new account
2. Upload locations
3. Run Geocoding jobs for all providers
4. Run first-pass Centroid calculation
5. Run job to reset the center of each location to generated centroid
6. Run Location Based Engagement backfills
7. Run job to reset the center of each location to backfill centroid
8. Export data

And improved workflow could be boiled down to three core steps

1. Create new account with location upload
2. Set job parameters and run all jobs sequentially and automatically
3. Export Data

## **8.2 Foundational**

There are many pieces of PlayPin that could benefit from more research. For instance, one of main parts of the process is the clustering algorithm. A more advanced heuristic may increase the accuracy of the system. It may also be possible to determine optimal parameters for both the clustering algorithm and the weights of each provider. Currently, only the overall accuracy of each provider is used for weighting. However, there may be a more optimal approach. In addition, maybe the exclusion of the low weighted geocoding providers, like Geocode.us, or LBE providers would yield better results. One approach might be to run many simulations to generate better parameters for the clustering process. In Appendix C we discuss the algorithm and describe the two parameters  $k$  and  $Y$ . The value of these parameters for the experiment was chosen empirically and somewhat arbitrarily after a few runs to see what yielded the best results. However, if we were able to run many more simulations with different parameter values, it might be possible to ultimately derive new values that yield more accurate coordinates.

Outside of modifications to the clustering algorithm, improvements in the quality of structured address data could also improve the accuracy of the resulting coordinates. Available online services could “clean up” improperly formatted addresses and wrong addresses. It may also be possible to clean up the data with one of these services prior to loading it in PlayPin to get better results. I suspect that if the initial seed data of addresses were all formatted correctly and referenced actual places then we would see even better results.

### **8.3 Additions**

There are a few useful additions to PlayPin scheduled for future development. One is the ability to generate seed location data automatically. For instance, for large retail chains, rather than assemble a list of addresses of all locations and compile that data into a formatted set, it would be helpful just to have a service automatically do it. It may be possible to use online providers to search for all the addresses of a brand's stores and processes these automatically. This would save time and be much more efficient than having to compile and export a list of locations. Another possible addition would be the ability to use the resulting coordinate estimations to correct the inaccuracies of online providers. The thought being if PlayPin can generate accurate geo-coordinates, then we should notify Google, Bing and others of the new and more accurate points rather than allowing them to drive customers to wrong addresses. This could be accomplished through API calls that push the new location data to their servers.

# Appendix A

## The PlayPin Geocoding Process

This appendix describes the steps of the PlayPin geocoding process from start to finish.

### Training

Training the system needs to be done only once on a small set of addresses. The weights generated from the training data can then be used on any future processing of subsequent data sets.

#### 1. Load Training Data

A CSV (comma separated list) or MS Excel file is loaded into the system. This list must contain structured address data.

#### 2. Initial Training Geocoding

After the loading of address data, the generation of these coordinates will consist of two similar phases. The initial phase involves querying various geocoding providers for each address and acquiring the geo-coordinate. In other words, for each physical address we will gather a set of geo-coordinates. For each result of the training data, we determine its error, the distance from the geocoding result to the GS coordinate.

#### 3. Weight Generation

The errors measured in training are then averaged and used to generate a weight for each geocoding provider. This weight is a representation of the overall accuracy of each provider.

### Point Estimation

These steps involve geocoding the working set of data and generating the final point estimation.

### **1. Load Working Address Data**

A CSV (comma separated list) or MS Excel file is loaded into the system. This list must contain structured address data. This will be a different set of data than the training data.

### **2. Initial Geocoding**

After the loading of address data, the generation of these coordinates will consist of two similar phases. The initial phase involves querying various geocoding providers for each address and acquiring the geo-coordinate. In other words, for each physical address we will gather a set of geo-coordinates. We can assess the quality of this gathered data by measuring various properties such as the centroid and standard deviation of the coordinates.

### **3. Cluster Geocoded Points and Generate Weighted Centroid**

Each set of points will then be processed using a clustering algorithm. It evaluates the distance of the points to exclude outliers and find the most spatially relevant set of points. From this reduced set of relevant points, we will generate a new, estimated point based on a centroid of the set. During centroid generation, we use the weights of the training data to generate the centroid.

### **4. Backfill Location Based Engagement Around Centroid**

The secondary phase is much like the first, but different in that we will query location based engagement (LBE) around the generated point. This LBE will be filtered using spatial relevancy and keyword matching

### **5. Cluster LBE Points and Generate Weighted Centroid**

We will process the resulting LBE coordinates using a clustering algorithm. The resulting cluster is again filtered based on keyword relevancy if possible. Once again we will generate a final estimated geo-coordinate with the reduced point set using a weighted centroid function.

## **6. Analyze Results**

After we have an estimated geo-coordinate, we can analyze how effective the process is by comparing the accuracy of our generated point, to that of the various geo-coding providers.

## Appendix B

### Distance Calculations

Geographic coordinates represented in lat/long are not Cartesian coordinates; they are a measure of angular distance not linear distance from the origin on a plane [38].

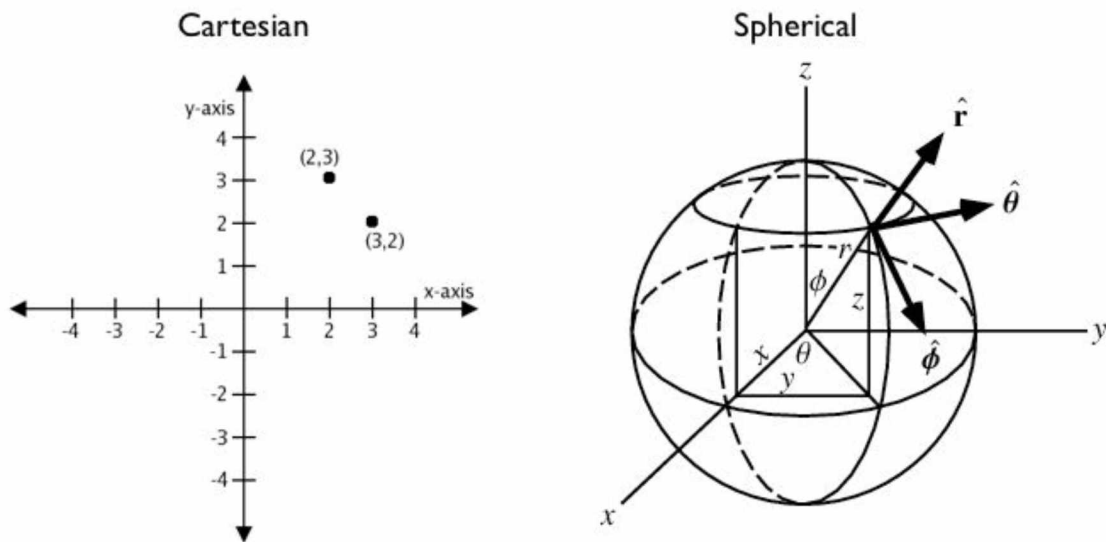


Figure 43. Cartesian vs. Spherical Coordinate Systems

For points in the Cartesian plane: if  $\mathbf{p} = (p_1, p_2)$  and  $\mathbf{q} = (q_1, q_2)$  then the distance is given by

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

Figure 44. Distance formula for two dimensional plane

If we try and use Cartesian based distance calculations on geographic coordinates, the results can be meaningless. We can only make meaningful calculations if we treat the coordinates as true spherical coordinates and measure their distance as paths around a large sphere. If this size of this large sphere is close to the size of the earth, then we can approximate the true distance between the two geographical coordinates [39].

One popular way to calculate the distance between two geographic coordinates is called the Haversine formula. It measures the shortest distance between two points on the surface of a sphere.

$$\Delta\sigma = 2 \arcsin \left( \sqrt{\sin^2 \left( \frac{\Delta\phi}{2} \right) + \cos \phi_1 \cos \phi_2 \sin^2 \left( \frac{\Delta\lambda}{2} \right)} \right).$$

Figure 45. Haversine Formula

The earth itself is not a true sphere, but rather an irregular shape approximating a biaxial ellipsoid [40], and using a spherical approximation can yield inaccuracies in certain instances. More accurate distance formulas that try to find the shortest path on an ellipsoid have been created to address this.



## **Appendix C**

### **Clustering Algorithm**

The clustering algorithm used is a modified k-nearest neighbor algorithm and is a straightforward classification and regression algorithm. In the context of this thesis we are using it to determine clustered coordinates and outlier coordinates for each location based on their distances from each other [41].

The steps taken in this process are:

1. Calculate distance matrix using all points
2. Determine cluster and outlier classification
3. Modify parameters and repeat if necessary
4. Choose best cluster

As an example, we will run through a sample location. After initial geocoding, we see the various provider venues in Figure 46:

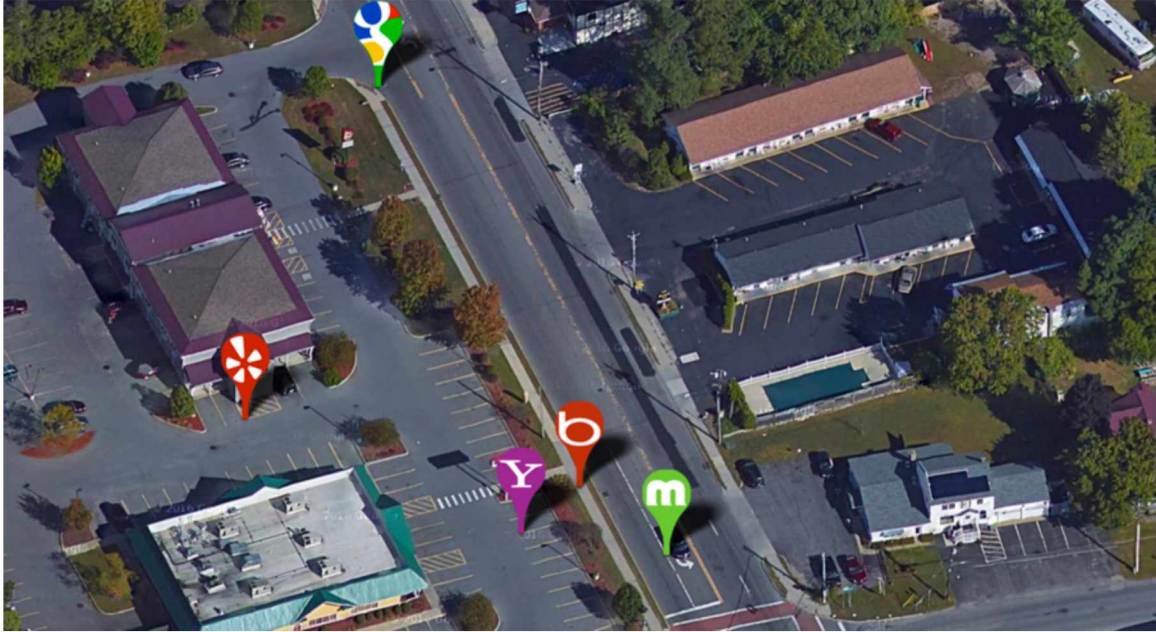


Figure 46. Geocoded venues

The first step in the process to determine classification of outliers is to generate a distance matrix. This takes all the provider venues and calculates the distance in meters to all other venues.

	Yahoo Yql Geocode	Yelp Place	Mapquest Geocode	Bing Geocode	Google Geocode
Yahoo Yql Geocode		39.47	18.37	10.89	80.04
Yelp Place	39.47		57.18	42.93	60.31
Mapquest Geocode	18.37	57.18		16.03	89.27
Bing Geocode	10.89	42.93	16.03		74.24
Google Geocode	80.04	60.31	89.27	74.24	

Figure 47. Distance matrix of geocoded venues

We process the data using two constants;  $k$  is the number of nearest neighbors to evaluate, and  $Y$  is the distance threshold for outliers. In this example, we will use  $k=3$  and  $Y=30$ . We can prune the table to only include the  $k=3$  nearest neighbors.

	Yahoo Yql Geocode	Yelp Place	Mapquest Geocode	Bing Geocode	Google Geocode
Yahoo Yql Geocode		39.47	18.37	10.89	
Yelp Place	39.47		57.18	42.93	
Mapquest Geocode	18.37	57.18		16.03	
Bing Geocode	10.89	42.93	16.03		
Google Geocode	80.04	60.31		74.24	

Figure 48. Pruned Distance Matrix

To determine the outliers, we will see if the  $k$ th nearest neighbor exceeds the threshold value  $Y$ . The outliers are highlighted in the following table.

	Yahoo Yql Geocode	Yelp Place	Mapquest Geocode	Bing Geocode	Google Geocode
Yahoo Yql Geocode		39.47	18.37	10.89	
Yelp Place	39.47		57.18	42.93	
Mapquest Geocode	18.37	57.18		16.03	
Bing Geocode	10.89	42.93	16.03		
Google Geocode	80.04	60.31		74.24	

Figure 49. Outliers at k=3

It is clear that with  $Y=30$  and  $k=3$ , all the points are classified as outliers. In this case, we use a heuristic and reprocess the data. To find a cluster, we will reduce “k” by one ( $k=2$ ) and see if there are any valid points.

	Yahoo Yql Geocode	Yelp Place	Mapquest Geocode	Bing Geocode	Google Geocode
Yahoo Yql Geocode			18.37	10.89	
Yelp Place	39.47			42.93	
Mapquest Geocode	18.37			16.03	
Bing Geocode	10.89		16.03		
Google Geocode		60.31		74.24	

Figure 50. Outliers at k=2

We now have a set of candidate points to look for the best cluster. For any point that is not an outlier, the cluster will be the set of points that include itself and its k-nearest neighbors. In this case the set of three candidate points exactly make up the cluster, so it is chosen as the best cluster.

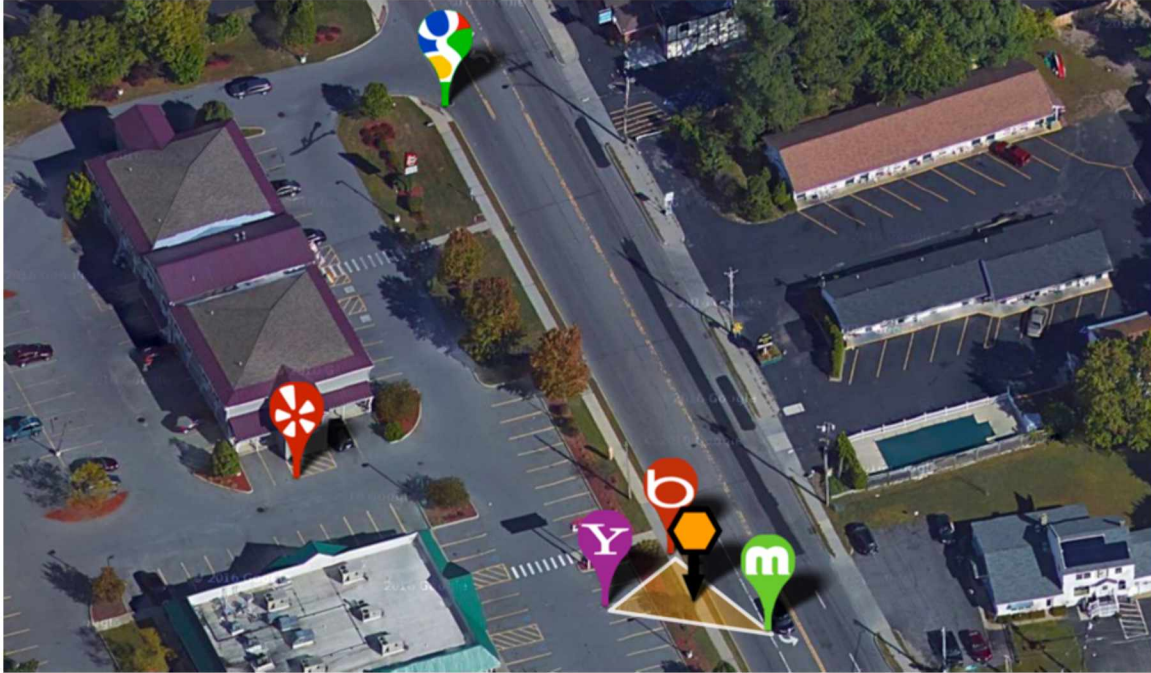


Figure 51. Final and Best Cluster

In some cases there may be more than one cluster to choose from. In the case that there is more than one cluster to choose from I have defined two rules to break the tie:

1. The coordinate with the most neighbors that fall within the distance threshold is chosen.
2. If the clusters have the same number of neighbors within the threshold, then the cluster with the highest sum of venue weights of neighbors within the threshold will be chosen. The venue weights are described in 6.2 and Appendix A.

The reasoning behind rule #1 is based on common sense logic that if there are more geocoders clustered together, it seems more likely that the actual point is somewhere near that cluster. Rule #2 is based on the common sense logic that the most accurate geocoders will generally produce the best results.

## Appendix D

## References

---

- [1] Cayo, Michael R, and Thomas O Talbot. "Positional Error in Automated Geocoding of Residential Addresses." *International Journal of Health Geographics* 2 (2003): 10. PMC. Web. 10 May 2015.
- [2] Stover, Jeffrey A. et al. "Improving Surveillance of Sexually Transmitted Diseases through Geocoded Morbidity Assignment." *Public Health Reports* 124.Suppl 2 (2009): 65–71. Print.
- [3] Gilboa, Suzanne M.; Mendola, Pauline; Olshan, Andrew F.; Harness, Catherine; Loomis, Dana P.; Langlois, Peter H.; Savitz, David A.; & Herring, Amy H. (2006). [Comparison of Residential Geocoding Methods in Population-Based Study of Air Quality and Birth Defects](#). *Environmental Research*, 101(2), 256-62.
- [4] Zoe Fox. "The 10 Most Frequently Used Smartphone Apps - Mashable." 2013. 3 Nov. 2013 <<http://mashable.com/2013/08/05/most-used-smartphone-apps/>>
- [5] Apple Maps Is So Bad It Will Tell You To Drive Across An Airport ..." 2013. 3 Nov. 2013 <[http://www.huffingtonpost.com/2013/09/25/apple-maps-bad\\_n\\_3990340.html](http://www.huffingtonpost.com/2013/09/25/apple-maps-bad_n_3990340.html)>
- [6] "So Wrong Like a Bad Lat/Long - MomentFeed." 2013. 3 Nov. 2013 <<http://momentfeed.com/location-blog/so-wrong-like-a-bad-latlong/>>
- [7] "The SoLoMo Manifesto - MomentFeed." 2010. 4 Nov. 2013 <<http://momentfeed.com/whitepaper/>>



- [8] "Geographic coordinate system - Wikipedia, the free encyclopedia." 2003. 2 Nov. 2013 <[http://en.wikipedia.org/wiki/Geographic\\_coordinate\\_system](http://en.wikipedia.org/wiki/Geographic_coordinate_system)>
- [9] "GPS.gov: GPS Accuracy." 2011. 2 Nov. 2013 <<http://www.gps.gov/systems/gps/performance/accuracy/>>
- [10] North, Matthew A. "An Empirical Analysis Of Mobile Smart Device Accuracy And Efficiency In Gps-Enabled Field Data Collection." *Issues in Information Systems* 12.1 (2011): 318-327.
- [11] Luo, J., Joshi, D., Yu, J., & Gallagher, A. (2010). Geotagging in multimedia and computer vision—a survey. *Multimed Tools Appl*, 51(1), 187-211. doi:10.1007/s11042-010-0623-y
- [12] "The SoLoMo Manifesto - MomentFeed." 2010. 4 Nov. 2013 <<http://momentfeed.com/whitepaper/>>
- [13] Facebook,. "The Complete Guide To Facebook Analytics". *facebook.com*. N.p., 2016. Web. 7 Oct. 2016.
- [14] "Facebook shares stats about businesses using pages and promoted ..." 2012. 4 Nov. 2013 <<http://www.insidefacebook.com/2012/12/06/facebook-shares-stats-about-businesses-using-pages-and-promoted-posts/>>
- [15] "Is Your API Naked? - Roadmap Considerations For API Product & Engineering Managers". *apigee.com*. N.p., 2016. Web. 7 Oct. 2016.
- [16] *Mercator projection*. (2016).*En.wikipedia.org*. Retrieved 15 October 2016, from [https://en.wikipedia.org/wiki/Mercator\\_projection](https://en.wikipedia.org/wiki/Mercator_projection)
- [17] *Google Maps Geocoding API Usage Limits | Google Maps Geocoding API | Google Developers*. (2016). *Google Developers*. Retrieved 26 December 2016, from <https://developers.google.com/maps/documentation/geocoding/usage-limits>
- [18] *Bing Maps API Best Practices*. (2016). *Msdn.microsoft.com*. Retrieved 26 December 2016, from <https://msdn.microsoft.com/en-us/library/dn894107.aspx>

- [19] *Geocoding API ~ Location precision and confidence*. (2015). *MapQuest Developer Network*. Retrieved 26 December 2016, from <https://developer.mapquest.com/products/geocoding>
- [20] *Yahoo BOSS Geo Services*. (2016). *Developer.yahoo.com*. Retrieved 26 December 2016, from <https://developer.yahoo.com/boss/geo/>
- [21] *Business Search API - Yelp Fusion*. (2016). *Yelp.com*. Retrieved 26 December 2016, from [https://www.yelp.com/developers/documentation/v3/business\\_search](https://www.yelp.com/developers/documentation/v3/business_search)
- [22] "Using the Graph API - Facebook Developers." 2013. 22 Sep. 2014  
<<https://developers.facebook.com/docs/graph-api/using-graph-api/>>
- [23] "Flickr Services: Flickr API: flickr.photos.search." 2012. 22 Sep. 2014  
<<https://www.flickr.com/services/api/flickr.photos.search.html>>
- [24] *Google Places API | Google Developers*. (2016). *Google Developers*. Retrieved 26 December 2016, from <https://developers.google.com/places/>
- [25] *Instagram Developer Documentation*. (2016). *Instagram.com*. Retrieved 26 December 2016, from <https://www.instagram.com/developer/>
- [26] Twitter API Overview (2016). Retrieved 26 December 2016, from <https://dev.twitter.com/overview/api>
- [27] Shahabi, Cyrus, Mohammad R. Kolahdouzan, and Mehdi Sharifzadeh. "A road network embedding technique for k-nearest neighbor search in moving object databases." *GeoInformatica* 7.3 (2003): 255-273.
- [28] "PostgreSQL: The world's most advanced open source database." 3 Nov. 2013  
<<http://www.postgresql.org/>>
- [29] "PostGIS — Spatial and Geographic Objects for PostgreSQL." 2007. 3 Nov. 2013 <<http://postgis.net/>>
- [30] "Puma: A Modern, Concurrent Web Server for Ruby." 2011. 7 May. 2014 <<http://puma.io/>>
- [31] "Ruby on Rails." 2004. 3 Nov. 2013 <<http://rubyonrails.org/>>
- [32] jquery.org, j. (2016). *jQuery. Jquery.com*. Retrieved 26 December 2016, from <http://jquery.com/>

- [33] *Backbone.js*. (2016). *Backbonejs.org*. Retrieved 26 December 2016, from <http://backbonejs.org/>
- [34] *Interactive JavaScript charts for your webpage* | *Highcharts*. (2016). *Highcharts.com*. Retrieved 26 December 2016, from <http://www.highcharts.com/>
- [35] *Bootstrap*. (2016). *Getbootstrap.com*. Retrieved 26 December 2016, from <http://getbootstrap.com/2.3.2/>
- [36] *Bulk Address Cleaning* | *Experian Data Quality*. *Experian Data Quality*. Retrieved 15 October 2016, from <https://www.edq.com/uk/solutions/contact-data-management/address-validation/bulk-cleansing/>
- [37] *Quattroshapes by foursquare*. (2016). *Quattroshapes.com*. Retrieved 26 December 2016, from <http://quattroshapes.com/>
- [38] *Spherical Coordinates -- from Wolfram MathWorld*. (2016). *Mathworld.wolfram.com*. Retrieved 8 October 2016, from <http://mathworld.wolfram.com/SphericalCoordinates.html>
- [39] *Great-circle distance*. (2016). *Wikipedia*. Retrieved 8 October 2016, from [https://en.wikipedia.org/wiki/Great-circle\\_distance](https://en.wikipedia.org/wiki/Great-circle_distance)
- [40] "Geographic coordinate system - Wikipedia, the free encyclopedia." 2003. 17 Nov. 2013  
<[http://en.wikipedia.org/wiki/Geographic\\_coordinate\\_system](http://en.wikipedia.org/wiki/Geographic_coordinate_system)>
- [41] Aggarwal, Shruti, and Janpreet Singh. "Outlier detection using K-mean and hybrid distance technique on multi-dimensional data set." *International Journal of Advanced Research in Computer Engineering and Technology* 2.9 (2013): 2626-31.