Learning Locality Maps from Noisy Geospatial Labels

Manjeet Dahiya
Delhivery, India
manjeet.dahiya@delhivery.com

Devendra Samatia
Delhivery, India
devendra.samatia2@delhivery.com

Kabir Rustogi
Delhivery, India
kabir.rustogi@delhivery.com

ABSTRACT

E-commerce and logistics operations produce a vast amount of geospatial data labelled with postal addresses. The data has great potential to mine geospatial knowledge, and we demonstrate that regional maps can be automatically built using the same. We propose an algorithm to construct non-overlapping polygons of the localities at a city level. The algorithm involves non-parametric spatial probability modelling of the localities followed by locality classification of the cells in a hexagonal grid. We show that our algorithm is capable of handling noise, which is significantly high in our setting due to the small scale of localities. A property about the noise and the correct information is presented such that our algorithm infers a correct locality polygon. We quantitatively measure the accuracy of our system by comparing its output with the available ground truth. We also discuss multiple applications of the generated maps in the context of e-commerce and logistics operations.

CCS CONCEPTS

• **Information systems → Geographic information systems**; **Data mining**; • **Computing methodologies → Machine learning**;

KEYWORDS

geospatial mining, labelled geospatial data, postal addresses, maps, polygons, localities

ACM Reference Format:
Manjeet Dahiya, Devendra Samatia, and Kabir Rustogi. 2020. Learning Locality Maps from Noisy Geospatial Labels. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3373933

1 INTRODUCTION

E-commerce and logistics organizations produce a huge amount of geospatial data from their daily operations. Their custom made mobile handheld devices are used to capture the geo-coordinates (using GPS) in an automated way for different events such as pickup and delivery. The amount of data produced is vast, e.g., as a logistics company supporting the e-commerce operations in multiple Asian countries, we produce over 50 million geo-coordinates daily. A part of these geo-coordinates is tagged with additional information such as postal address, type of address (residential/commercial), validity of an address, delivery refusal status, number of attempts for a successful delivery and more. This associated information is either provided by the end customer or by the operations team of these organizations who physically visit each address for deliveries.

The postal addresses contain details such as door information, locality name, city, state, country and PIN code[1]. Given a huge set of geo-coordinates labelled with such rich information, it presents an interesting opportunity of mining geospatial knowledge out of it. Intuitively, one can think of automatically building regional maps from the respective data. Mining such information can be used to enrich or create the gazetteers automatically.

Unlike in developed countries, the postal addresses of developing countries, especially for unplanned cities, lack adequate addressing systems [4]. The respective gazetteers are not available or not complete with the information required for decoding the addresses. Moreover, it is also common in developing countries that the population uses a different de-facto addressing convention instead of the official one. For example, unplanned localities, which started without the government's authorization, do not have a consistent addressing system and are not available in the gazetteers. The best way to learn about such localities is by exploring the convention followed by its population, i.e., from the addresses written by them. Furthermore, for the purposes of e-commerce and logistics operations, it is more important that we have access to these de-facto gazetteers and maps (better representative of reality) instead of the official ones, which could be incomplete or unavailable.

Researchers have previously worked towards building maps of continents, countries and states using geospatial data annotated with textual tags [3, 9, 10, 13, 17]. Our work takes it a step forward, which is to learn the maps of *localities*[2] from the available geo-coordinates labelled with *postal addresses*. Producing maps of localities involve learning the polygons of localities. The abstraction of polygons subsumes a set of scattered geo-coordinates into a single object, and it represents how humans naturally perceive the boundary of a region. Importantly, polygons enable efficient and effective analytical reasoning. Consequently, polygons can have applications such as predicting geo-coordinates of a new postal address, geofencing the new geo-coordinates with respect to a locality, locality level analytics, geospatial database enrichment, and mining relations between the concepts [10, 18, 19]. More details of the applications are provided in Section 3.1.

Previous work has extensively used labelled geospatial data to harvest maps, places and events. Publicly available geotagged photographs of the photo sharing app Flickr provide a huge amount of geospatial data annotated with textual tags. Researchers have used

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373933

[1] PIN refers to a six digit postal index number. It is the postal code used in India.
[2] In this paper, the term locality refers to an area or a neighborhood that people typically write in their postal addresses.

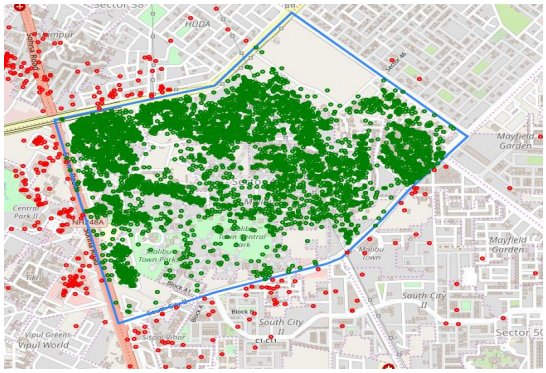

Figure 1: Scatter plot of the observations of a single locality. Points lying outside the polygon (red) are noisy. Note that observation points are not available for green zones and open areas.

it to mine places [10, 17], events [17], maps [3, 9]. Another related work has also demonstrated mining vague places and their boundaries to enrich gazetteers by extracting geospatial data from the Web pages [13]. The work on mapping world's photos [3] produced maps of a few continents by way of simple scattered plots, and the output resembled the actual boundaries. Intagorn [9] determined places such as states and cities, and their boundaries (polygons) using Flickr's data. They also pointed a challenge of dealing with noise in determining boundaries and also presented a *crowd-based noise filtering* solution to handle it.

Albeit the geo-coordinates data labelled with postal addresses is usually of good quality but it also contains noise. The labels, as well as the geo-coordinates, could be noisy. The noise manifests due to human errors and GPS connectivity issues. The errors that lead to noisy data are: 1) Often, the end customer provides an ambiguous address containing a nearby locality or a landmark. 2) The field executive (delivery person) delivers to the correct location, however, they mark the delivery at a different location — resulting in an incorrect geo-coordinate for that address. 3) The address is correct and even the marking of the delivery is also correct, however, GPS connectivity issues such as weak signals and poor accuracy resolution also lead to noise in the data, and this is especially severe while working at the granularity of localities. Figure 1 demonstrates the noisy points for a single locality.

Previous work has focused on determining the maps of either point of interests (a point) [17] or very big places such as states and cities [3, 9, 10]. In contrast, the focus of this paper is to determine the maps of localities, which are not points but smaller than cities and states. The average radius size of a locality is the order of 500 m, whereas, that of a city is 50 km. We find that the challenge of handling noise and ambiguities at the locality scale is significantly higher than that faced by previous work. The reason of high noise is that the localities are smaller in scale and closer to each other in space — they could even be next to each other. In this setting, the noise due to GPS accuracy resolution is highly prominent, and the human errors such as ambiguities in address tend to occur more.

In this paper, we present an algorithm to construct the maps of *localities* from geospatial data labelled with postal addresses. Specifically, the algorithm is robust in dealing with noise and ambiguities in the available dataset. The algorithm is generative in that we learn the spatial probability distributions of the localities using the technique of kernel density estimation [20]. Unlike previous related work [9, 10], our approach is non-parametric — it is flexible and allows us to produce the probability distributions and the polygon boundaries of any shape without making assumptions. Note that in general, the boundaries of localities can take any kind of shape and we do not want to limit that. Once we have the probability distributions, we use them to eliminate noise and create non-overlapping regions that have the least misclassification error. Finally, we create the polygons for the different localities resulting in the maps. We evaluate and test our algorithm on publicly available maps, and we find that resulting precision and recall is 0.86 and 0.63 respectively.

The rest of the paper is organized as follows: Section 2 states the problem and presents our approach, which is divided into three steps. Our approach is evaluated in Section 3, and it also presents the applications and use cases around locality polygons. We compare our work with previous work in Section 4, and finally we conclude in Section 5.

2 PROBLEM STATEMENT AND APPROACH

2.1 Problem statement

We are given the localities $\mathscr{L}$ of a city and corresponding to each locality $L \in \mathscr{L}$, we are also given a set of geo-coordinates L^X (observations). The set of geo-coordinates may contain noisy points, i.e., there may be points in the set that do not actually belong to the true region of the locality and lie outside of it. With the given data, our objective is to determine the polygons of these localities $\mathscr{L}$ such that the polygons are non-overlapping and the misclassification error is minimized.

Note that we assume that there exists a system that can map postal addresses to the respective localities. Parsing of postal addresses is a challenging problem in its own right and most of the e-commerce and logistics organizations need such a system for running their operations in an automated manner. For our purposes, we have used a named entity recognition system, which tells the complete hierarchy, i.e., country, state, city and locality, corresponding to an address. This system works on top of a tree-based database, which represents the states, cities and localities of a country in a hierarchical topology. We do not discuss the details of this system as it is orthogonal to the problem discussed.

2.2 Learning locality polygons

Before presenting the approach, we discuss the representation of the space that we have used in our system. Our representation of the space is a 2-dimensional grid, which is divided into hexagonal cells of equal size and shape (Figure 2-a). The input geo-coordinates are projected onto the grid. A point falling anywhere in a cell is approximated to be present at the center of that cell. Once the projection is done, the rest of the processing takes place in terms of the cells of the grid. For example, various probability estimations for all the points in a cell are approximated by computing these at the center of the cell. This approximation is important as it leads

to efficient processing of our tasks, and given a sufficiently small size of the cells, the approximation is practical as well. We discuss more about the hexagonal grid and our motivation for choosing it in Section 2.2.3.

Our approach of determining locality polygons from noisy scattered data has three steps. In the first step (Section 2.2.1), we determine the spatial probability distribution of each locality. The spatial probability distribution of a locality represents the conditional probability at a point X, given the locality, i.e., $P(X|L_i)$, where X is typically the center of a cell ($X \in \mathbb{R}^2$, i.e., lying in 2D space) and L_i is the locality. We use the technique of kernel density estimation [20] to determine these spatial probability distributions. Second (Section 2.2.2), we determine the locality for each cell. It involves estimating the prior probability of each locality $P(L_i)$ and the conditional probability for each locality given a cell $P(L_i|X)$. The probability $P(L_i|X)$ is computed by applying Bayes theorem on the spatial probability and the prior probability. The locality of each cell is then decided by finding the locality that has the maximum value of $P(L_i|X)$. This step handles the issues due to noise automatically. In the last step (Section 2.2.3), we build polygons from the cells already classified with localities. We model this as a graph and determine the connected components by using a graph traversal algorithm, and finally, we generate the boundaries of these connected components.

Our approach is non-parametric, i.e., there are no constraints on the functional representations of the probability distributions, and it can take any shape. Consequently, it does not put any constraints on the shape of the resulting boundaries. We deliberately chose the path of generative modelling, i.e., learning the exhaustive spatial probability distribution $P(X|L)$, as it allows us to eliminate noise in a formal manner. Moreover, in our setting, where the data keeps increasing, generative approaches like KDE keep improving their estimations.

In the following sections, we describe each step in detail:

2.2.1 Probability distribution estimation. Kernel density estimation (KDE) is an old technique to determine the probability density from discrete observations. We use it to determine the spatial probability distributions of the localities. For each locality $L \in \mathscr{L}$, we want to determine $P(X|L)$ for $X \in \mathbb{R}^2$. In our case, X is usually the center of a cell, and it lies in 2-dimensional space.

Intuitively, KDE involves determining the probability density at a point using the observations in its neighborhood (subset of L^X). In our setting, the neighborhood is a circle of radius λ and the probability density at the center is calculated using the points lying in the circle. The observations contribute to the density basis their distance from the center, a closer observation has more contribution than a distant one. An equivalent interpretation is that an observation being at the center of a circle of radius λ has an additive contribution to the probability distribution within the circle such that the contribution decreases with increasing distance from the center. The amount of contribution of each point is governed by a function called *kernel*.

Following is the formal definition of the computation of probability distribution $P(X|L)$:

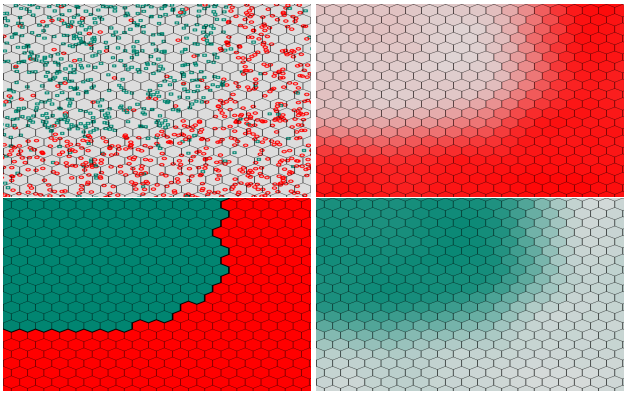

Figure 2: The status of a small portion of the grid at different steps of the algorithm. In clockwise order, a) Observations of two localities shown as red and green (top-left) on the grid. Notice the noisy observations in a locality due to the other locality. b & c) Spatial distributions $P(X|L)$ of the two localities. Higher color intensity reflects higher value of probability. d) Result of classification after the decision rule and the boundary is also shown in black.

$$P(X|L) = \frac{\sum_{O \in L^X} K_\lambda(X, O)}{\pi\lambda^2 N_L}$$

Here, $N_L = |L^X|$ is the number of observations for locality L, the variable $O \in L^X$ indicates an observation, and K_λ is the kernel function, which in our case is *Epanechnikov kernel* [5] with an appropriate change in the constant factor to make its integral equal to $\pi\lambda^2$. As a result, $P(X|L)$ becomes a valid probability measure.

$$K_\lambda(X, O) = 2(1 - \frac{d(X, O)^2}{\lambda^2}) I(\frac{d(X, O)}{\lambda} \leq 1)$$

K_λ is defined in terms of the distance $d(X, O)$ between X and O, $d(X, O) = ||X - O||$, λ is the radius of the circle, and $I(y)$ is an indicator function, it is equal to 1 for $y = True$, otherwise, 0. Notice that the value of the kernel function is highest at the distance zero and decreases with increasing distance. Also, the value is zero if the distance is greater than λ, i.e., when O lies outside the circle centered at X. Our choice of kernel comes from two ideas: First, it should give more weight to the closer points over the distant ones so that the closer points influence the probability density more. Second, after a cutoff (λ), the contribution should be zero as it leads to fewer computations and making it efficient. We tested other similar kernels like *Quartic* and the results were similar.

The radius of the circle λ is a hyperparameter of our algorithm. A high value of λ can lead to a very smooth probability distribution resulting in *underfitting*. On the other hand, its small value can lead to a noisy curve exhibiting the phenomenon of *overfitting*. One needs to determine a suitable value of λ experimentally. Section 3 discusses the value chosen in our experiments and its impact on our results. For our example, Figure 2-b and Figure 2-c visualize the spatial distributions of the two localities estimated from the observations of Figure 2-a using the above method.

2.2.2 Locality classification for cells. The goal of this section is to assign a locality to each cell using the locality probability distributions we have estimated in the last section. Note that these distributions cannot be used directly. For example, for the cell centered at X and the localities L_1 and L_2, the values $P(X|L_1)$ and $P(X|L_2)$ are not comparable, these are spatial distributions of individual localities, and cannot be compared across the localities. The quantities that we instead require are $P(L_1|X)$ and $P(L_2|X)$, i.e., the probability of being a particular locality for a given cell.

We can compute $P(L|X)$ from $P(X|L)$ by using Bayes theorem as follows:

$$P(L|X) = \frac{P(X|L)P(L)}{P(X)} \propto P(X|L)P(L)$$

Here, $P(L)$ is the prior probability of the locality and $P(L|X)$ is the posterior probability. We can ignore $P(X)$ as it remains constant across the localities for the given cell centered at X. We have already determined $P(X|L)$ in Section 2.2.1, and the estimation of $P(L)$ is discussed after the following discussion on decision rule.

Decision rule: Once we have estimated the posterior probability of each locality given a cell, we are set to classify the cell. For each cell, we compare the posterior probability of all localities and assign the locality with the maximum value. That is, the locality with the maximum value of $P(L|X)$ is selected. Formally, the decision rule to assign the locality X_L to the cell centered at X is:

$$X_L = \arg\max_{L_i} P(L_i|X)$$

One can prove that this decision rule leads to the least expected misclassification error, given that the probability estimations are correct [2].

Locality prior: The prior probability of a locality $P(L)$ represents how much spread out the observations of the locality in space are. We approximate the spread out with the number of cells required to hold the observations. Higher the spread out, then higher is the prior probability. We can easily compute the number of cells a locality belongs to ($Cells_{L_i}$), i.e., the spread out, by counting the number of unique cells the observations of that locality are present in.

Formally, we estimate the prior of a locality as follows:

$$P(L_i) = \frac{|Cells_{L_i}|}{\sum_{L_j \in \mathscr{L}} |Cells_{L_j}|}$$

$$Cells_{L_i} = unique\{X_{Cell} | X \in L_i^X\}$$

Here, $|y|$ represents the cardinality of the set y, X_{Cell} represents the cell corresponding to the observation X, and the operation $unique(y)$ returns the elements of the set y only once.

We find that this construction of prior helps in dealing with the case of a vertically-spread locality L_{VS} in a small area next to a horizontally-spread locality L_{HS} in a larger area such that the number of observations are the same. That is, a small locality with high density of observations next to a large locality with a lower density. The spatial probabilities of the cells at the boundary of the two localities would be such that $P(X|L_{HS})$ is considerably lower than $P(X|L_{VS})$. However, the prior of the latter $P(L_{HS})$ would be significantly greater than that of the former $P(L_{VS})$, and it would end up correctly estimating the probabilities $P(L_{VS}|X)$ and $P(L_{HS}|X)$. As a result, the cells near the boundary would be classified correctly, and subsequently, would result in generating more precise polygons. Notice that a common estimator of prior probabilities using the number of points would clearly fail here.

Note that this step of locality classification inherently handles noisy data. The noise in the data would get reflected in the probability distributions $P(X|L)$ that we compute in the first step and the second step would ignore those. Recall that for computing $P(X|L)$, we find the number of points lying in a circle of radius λ centered at X. If we assume that the number of points in the circle due to noise is less than the number of points of the correct locality, then the probability due to noise would be less than that of the correct locality (under certain assumptions). Consequently, at the time of predicting the locality of a cell, the locality with the highest probability (assuming our estimate of prior probabilities is correct) is selected — resulting in the correct prediction. Thus, avoiding the noise in the data. In simpler words, for each cell, the neighboring points of different localities compete for the claim of the cell with each other, and the greater points of the correct locality end up winning over the fewer noisy points of an incorrect locality. However, in case the noise is systematic and significantly high, then it is quite possible that a wrong locality gets identified. Section 2.2.4 formally discusses more about noise. Figure 2-d illustrates the final cell classification for our running example.

2.2.3 Determining polygons. We now construct locality polygons from the cells already classified with localities. Our approach is based on the idea that two adjacent cells with the same locality can be merged into one. This idea is applied recursively, to keep forming bigger cells in each step. Finally, the algorithm terminates when no more cells can be combined. The output of the algorithm would be a set of collection of cells such that each collection of cells is connected.

Precisely, we model the grid as an undirected graph. The nodes of the graph are the cells and two nodes are connected with an edge if they are adjacent to each other and they belong to the same locality. A depth-first search traversal in this graph can determine all the connected components, which is exactly what we want to find. One connected component would represent a collection of cells adjoining each other.

A polygon boundary can then be created for each collection of cells, representing the outer boundary of the collection. The outer boundary of a collection is determined by identifying the edges of the cells that are shared by two different localities. Connecting these edges would result in the boundary of the collection, i.e., polygon. The decision boundary between the two localities of our running example is shown in Figure 2-d.

Note that this step of creating boundaries may generate multiple such boundaries because of the presence of holes in the locality. A hole gets generated when within a locality a sufficiently big region exists such that no observation is present in it. This leads our algorithm to assign no localities to such cells — resulting in possibly multiple inner boundaries. The outer boundary among these is identified by finding the *extreme point* [12, 14] among the points of

vertices of these boundaries. The rest of the boundaries are considered holes. A hole with the number of cells below a certain threshold (i.e., very small hole) can be deleted in the post-processing steps. Depending upon the use cases, applications using the polygons can tailor the processing of holes as per their usage, some use cases might want to completely remove the holes and the others would want to keep them. Finally, for each locality, the above process can possibly generate multiple disjoint polygons. We combine multiple such polygons into a single entity in the form of a multi-polygon.

Hexagonal grid: A hexagonal grid has many nice properties that makes it suitable for our work in comparison with a square grid [1, 21]. The properties relevant to our work are: 1) The definition of neighbours of a cell in a hexagonal grid is well defined. It comes from the fact that in a hexagonal grid, two neighbouring cells always share an edge, leading to a well-defined neighbours list. Whereas in a square grid, two cells can have contact via vertices with each other such that they do not share an edge between each other, which leads to ambiguities on how to define neighbours. Another positive side effect of this property of a hexagonal grid is that invalid polygons are never created in it. For example, in a square grid, two cells connected by their vertices (without sharing an edge) can be considered a single polygon as well as two different polygons, however, such a case is not possible in a hexagonal grid. 2) A hexagon approximates a circle better than a square. In addition, the centers of the hexagons at a fixed distance from some hexagon also form a circle. This property allows us to efficiently compute the probability density of Section 2.2.1 without constructing the actual circles. 3) Finally, a hexagonal grid approximates curves better than a square grid at the same level of grid resolution. Altogether, these properties of a hexagonal grid make our algorithms simple and efficient.

2.2.4 Noise vs information. In this section, we discuss the bounds on the noise with respect to the correct information such that our algorithm infers the correct locality.

Consider a cell centered at X in the grid, whose actual locality is L_a, and L_n is a noisy locality that could potentially be assigned to the cell. The condition for our algorithm to identify the correct locality is: $P(L_a|X) > P(L_n|X)$. On expanding the same we get:

$$P(X|L_a)P(L_a) > P(X|L_n)P(L_n)$$

The value $P(X|L)$ is computed by using the kernel on the points falling in the circle of radius λ centered at X. Let $N^{\lambda}_{L_a}$ and $N^{\lambda}_{L_n}$ be the number of points lying in the circle and N_{L_a} and N_{L_n} be the total number of points of the localities L_a and L_n respectively. Expanding the inequality further:

$$\frac{\sum_{O \in L^X_a} K_\lambda(X,O)}{\pi\lambda^2 N_{L_a}} P(L_a) > \frac{\sum_{O \in L^X_n} K_\lambda(X,O)}{\pi\lambda^2 N_{L_n}} P(L_n)$$

If we assume that the points of L_a and L_n in the circle are drawn from the same distribution, then $\sum_{O \in L^X_{N_a}} K_\lambda(X,O) / \sum_{O \in L^X_{N_n}} K_\lambda(X,O)$ can be approximated by $N^{\lambda}_{L_a}/N^{\lambda}_{L_n}$ when $N^{\lambda}_{L_a}$ and $N^{\lambda}_{L_n}$ are sufficiently large. Note that the approximation is valid when $N^{\lambda}_{L_a} \to \infty$ and $N^{\lambda}_{L_n} \to \infty$. Using this result, we can now simplify the inequality to:

$$\frac{N^{\lambda}_{L_a}}{N_{L_a}} P(L_a) > \frac{N^{\lambda}_{L_n}}{N_{L_n}} P(L_n)$$

Defining $\chi^{\lambda}_L = \frac{N^{\lambda}_L}{N_L}$ and further simplifications yield:

$$\chi^{\lambda}_{L_a} P(L_a) > \chi^{\lambda}_{L_n} P(L_n)$$

A simple implication of the result, under the assumptions, is that the algorithm correctly predicts the locality of a cell if the product of the fraction of the points of the correct locality lying in the circular neighborhood with the locality prior is more than that of the noisy locality. The exact assumptions might not always hold, however, this analysis can be performed upfront to get an approximate idea about the quality of the data and the feasibility of our algorithm in predicting the locality boundaries.

3 RESULTS

We worked on a dataset spanning over 1022 cities/towns of India consisting of 29089 localities. The number of geo-coordinates and address pairs considered in our experiments are 25 million. We populated the locality for each address using our named entity recognition system for postal addresses (Section 2.1).

To test the accuracy of our approach, we required locality polygons for a set of cities representing the test set. We created this test set manually from multiple sources like OpenStreetMap (OSM) database [8] and unstructured local maps. OSM is an open source mapping service; it is free and its data is contributed by users voluntarily. However, OSM is not rich in data for developing countries, especially in non-metropolitan and rural areas. We also used maps from local government bodies, wherever possible. But this data was unstructured, inconsistent and incomplete. Finally, we created a test set of 21 cities consisting of 1030 localities with a polygon for each locality. We ensured that these cities are distributed throughout the country and are varied across the tiers. The count of cities from the northern-central, western, southern, and eastern zones is 9, 6, 4, and 2 respectively. With respect to the city types, the count of metropolitan, tier-1 and tier-2 cities is 7, 6 and 8 respectively.

Using the test data, we measured the percentage of noise present in our dataset. For each locality in the test data, we counted the number of observations lying outside the test set polygon of the locality and computed the percentage with respect to the total observations of the locality. The 25th percentile, median and 75th percentile noise percentages are 6.1%, 13.4% and 23.9% respectively. The overall noise percentage in our data is 16.4 %.

We ran our algorithm for all the cities of the dataset and constructed polygons of the respective localities. We compared the generated polygons with the polygons in the test data. The comparison of two polygons was performed by computing the precision and recall. Given a true polygon (P_T) and a generated polygon (P), precision is computed as the area of the true positive region with respect to the area of the predicted polygon ($\frac{area(P \cap P_T)}{area(P)}$). Recall is computed as the area of the true positive region with respect to the area of the true polygon ($\frac{area(P \cap P_T)}{area(P_T)}$). Here, $area(x)$ operator provides the geometric area of the polygon x and $\cap$ represents polygon intersection.

Table 1: Precision, recall and F_1 score for different values of λ.

$\lambda(m)$	Precision (%)	Recall (%)	F_1 score
43.3	0.89	0.54	0.62
86.6	0.86	0.63	0.68
129.9	0.84	0.68	0.70
173.2	0.82	0.70	0.71
216.5	0.80	0.72	0.71

The precision, recall and F_1 score of our algorithm basis the test data is provided in Table 1. It also compares these metrics for different values of the hyperparameter λ. Notice that with increasing λ, precision drops and recall increases. The correctness (i.e., not spreading outside the test set polygon) of a polygon comes at the cost of coverage (i.e., how much of the test set polygon is covered). A polygon with higher precision tend to be smaller than the one with lower precision. Depending upon the use case, one can select a balance between precision and recall by choosing a suitable value of λ.

The recall values in the table are relatively lower than the precision values. The reason for the same is that we tend to generate smaller polygons for some of the localities. This reflects a limitation of our algorithm, when few observations are available or when the observations are not spread throughout a locality, then we end up generating multiple smaller polygons — resulting in a relatively smaller recall value. However, with time, when more data gets captured, we will end up correcting those polygons too.

The cell size of the grid in our experiments was 5 m, which is the length of the edge of the hexagonal cells. A larger cell size leads to efficient computations at the cost of accuracy. One needs to fix it depending upon the scale of the problem. For example, we can set it to 500 m for building the polygons of the states of a country. Note that a smaller value of cell size would always improve or maintain the accuracy but it would require more computation. The ideal cell size is as small as the computational power allows.

Figure 3 shows the constructed locality polygons of the city Noida, with OpenStreetMap in the background. Notice that the boundaries are close to natural separators such as roads and green zones, however, there are crossovers too, perhaps, because of some systematic excessive noise. As expected, some of the areas, for instance, green zones and open facilities have not been created because of the unavailability of the respective observation points. Also, notice that the boundaries are rough (zig zag) and some smoothing or simplification can be performed to make them look less noisy.

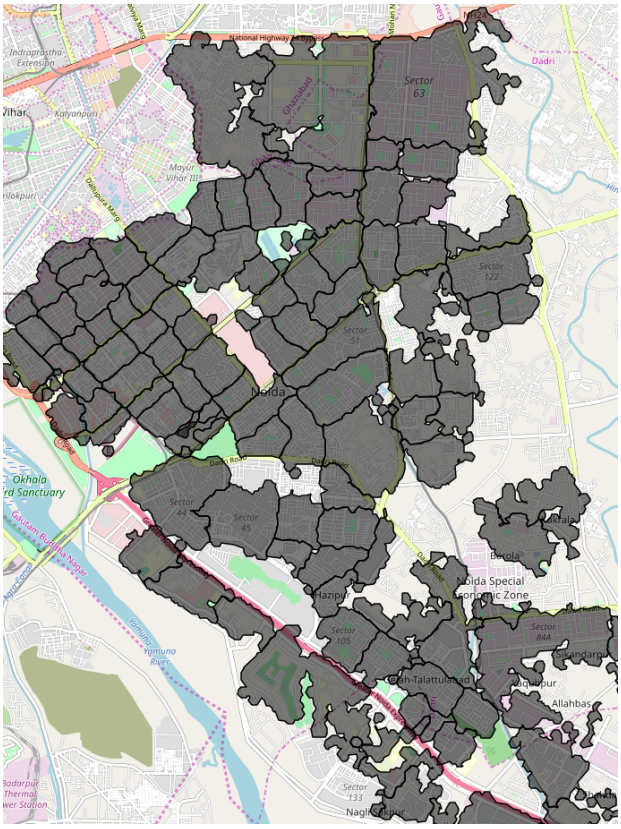

Figure 3: Generated locality polygons of the city Noida, with OpenStreetMap in the background.

3.1 Applications

Apart from the general applications, namely enriching geospatial databases (e.g., gazetteers) and learning relationships between the localities, locality polygons have important use cases in the context of logistics industry. We discuss some of these applications and their implications to our business.

Geofencing: Locality polygons enable us to perform virtual fencing of the future deliveries. For a package delivery marked for a particular locality, we want to ensure that it is being delivered in the respective locality. At the time of delivery, when the delivery event is triggered by the field executive, we check in real-time if the delivery is being performed within the locality polygon. In case the delivery point is far out of the locality polygon, then we trigger a soft warning to the field executive, and an explanation is expected.

We have over 15000 field executives and such checks allow us to understand their behaviour better. In addition, if we only consider the observations that have been geofenced, then it results in data that is free of noise (to some extent). This makes our other analyses, which deal with the scattered geospatial data, very simple as they do not have to worry about the noise in the data.

Locality intelligence: Our systems record the trace data of the field executives when they are on the ground, travelling and delivering the packages. We record the movements of the field executives at the granularity of 10 sec. For every locality, we figure out the amount of time on average a field executive spends delivering a package, we call this statistic of a locality *time per shipment* (TPS). We compute this by dividing the total time spent by the field executives by the total number of deliveries. The time spent in a locality by a field executive is determined using the trace data time stamps and the polygon of the locality.

It is not surprising that the time per shipment varies across localities. There are multiple reasons for this observation: 1) In different localities, a field executive has to clear different levels of security before reaching to the door of a customer. It leads to differences in time per shipment across localities. 2) Another reason of the difference is due to poor addressing of unplanned (or unauthorized) localities. A planned locality has a systematic addressing, where it is easier to locate an address. Whereas in unplanned localities, the addressing is usually poor. In fact, unavailability of door information is also not uncommon and it leads to a significantly higher time per shipment.

Time per shipment of a locality has turned out to be a very important statistic in characterizing the localities. It helps in assigning the daily load to the field executives. TPS also gets consumed in predicting the estimated time of arrival of packages. Furthermore, it allows us to determine the cost per shipment of packages for each locality. Cost per shipment statistic helps us determine the variable part of the salaries of the field executives consistently across the localities.

Geocoding of addresses: Predicting geo-coordinates from an address is referred to as geocoding. We have been working on a geocoding system using the polygon algorithm. The idea is very simple, from the historical geo-coordinates labelled with addresses, we build the polygons of various door level information present in the address for each locality; at the time of prediction, we fetch the polygons from this prebuilt database of polygons and take an intersection. The intersection results in a polygon, and we return the centroid and the radius of this polygon as the geo-coordinate and the confidence of our prediction respectively. The work is still under progress. In our limited and small set of initial experiments, the results have been extremely good. We have been able to predict the geo-coordinates of over 70% of the addresses within 200 m confidence.

There are challenges in identifying the door level information and handing commonly written poor address, leading to difficulties for the field executives. Geocoding will have a huge impact on our last-mile delivery operations. We would be able to accurately route our field executive, reducing the time required in locating an address. Moreover, it would also allow us to make a decision on the set of packages to give to a field executive such that our objectives like time per shipment and cost per shipments are optimized.

4 RELATED WORK

Work on learning boundaries [9] from Flickr's geospatial data annotated with tags proposed a crowd-based noise filtering approach to handle the noise in data. The idea of crowd-based noise filtering approach is that if we measure the number of distinct people who have annotated with the same tag, then it results in a better measure over the number of tags. Note that this approach is inherently making an assumption and asking for additional information about distinct users. On the other hand, in our case, we are neither making this assumption, nor we are asking for additional data. Also, they build polygons of a place one at a time, without considering the points of the other classes (localities) like us. It would end up generating larger polygons for our setting with no mechanism to meet the non-overlapping requirement. They showed it to work for learning boundaries of very large places, namely the states of the USA and a set of countries. For the USA states, it resulted in polygons with a precision of 0.9 and a recall of 0.8. For the country set, the precision and recall metrics were 0.6 and 0.4 respectively. However, note that building polygons of places such as states is easier as the amount of noise is sparse at the scale of 50 km radius. Whereas, in our case, the localities are small (radius of the order of 500 m) and the noise is considerably high. For example, in case of a state or country, the proportion of noise with respect to the correct information would be significantly lower in comparison to localities.

Work on mining places [11] from the same Flickr data used a probabilistic approach and modelled the probability distributions of different annotation tags. They used Gaussian mixture models (GMM) to model the probability distributions and employed expectation-maximization to learn the parameters. Note that the approach is parametric and given that the boundaries can take any shape, one would have to require a high number of components in GMM to produce the desired shape. Kernel density estimation (KDE) on the other hand is non-parametric and does not make any assumption about the density function, consequently, the boundaries are not constrained. Moreover, in our setting, where the data keeps increasing, it is logical to use KDE because the probability density estimation using KDE usually improves with more data.

Using KDE for probability estimation is a common method in geospatial probability estimation [15]. The work on mining vague large places from the Web [13] generated boundaries of vague places like like "Midwest towns" and "The south of France" using KDE based probability estimation. They first found the known places that co-occur with these vague places on the Web, and then estimated the probability distribution using the geo-coordinates of the known places. Unlike our work, they worked on spatial distributions $P(X|L)$ without the prior probability correction. Which is suitable for them as they are working on one place at a time, and they are not comparing across the different places. In comparison with our work, they work on a larger scale than ours and they do not point out the challenge of noise. Moreover, they do not have the constraint of producing non-overlapping boundaries.

Density-based spatial clustering of applications with noise (DBSCAN) [6] is a clustering algorithm that works really well in the presence of noise. The output of this algorithm is a set of valid clusters and a set of noisy points. One can think of running this algorithm for each locality and then building concave hulls for the valid points of each locality. However, this scheme does not work in our setting. This mechanism can generate polygons of two different localities that are overlapping with each other. But our setting requires that the polygons of different localities must be non-overlapping. Unlike our work, DBSCAN has no notion of multiple classes, i.e., multiple localities. It is an algorithm to generate clusters by considering the points of only one class.

Our work has similarities with the goal of various concave hull algorithms [7, 14, 16]. Concave hull algorithms try to build the polygons that represent the geometrical characteristics of a given set of scattered points. In other words, it tries to build a polygon that represents the "footprint" [7] of the given points. It is similar to what we want. However, these algorithms do not work at all with noisy

data. In the presence of noise, these algorithms end up creating very large and spiky polygons. To our knowledge, there does not exist a concave hull algorithm that can handle noise. Perhaps, one can use our system to build concave hulls in the presence of noise. We would have to simply determine a suitable threshold for $P(L|X)$, below which the cells should not be considered.

5 CONCLUSIONS

E-commerce and logistics operations produce vast geospatial data labelled with postal addresses, and it presents an interesting opportunity to mine geospatial knowledge out of it. We demonstrated that this data can be used to automatically build the regional maps. We proposed and implemented a system to construct non-overlapping polygons of the localities of the cities. The system is capable of handling noise, which is significantly higher at the scale of localities. A property about noise with respect to the correct information is presented such that our algorithm infers a correct boundary. We quantitatively measured the accuracy of our system by comparing its output with the available polygons from multiple sources. We also discussed multiple applications of the generated maps, which we have developed, in the context of e-commerce and logistics operations.

In the future, we would like to exploit the information about the natural separators such as roads to create better boundaries. Importantly, we envision many applications of the constructed polygons, e.g., geocoding and would want to work on those. We hope that our work on geospatial data labelled with postal addresses brings attention to this data and triggers more related work in the future.

REFERENCES

[1] Colin P.D. Birch, Sander P. Oom, and Jonathan A. Beecham. 2007. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecological Modelling* 206, 3 (2007), 347 – 359. https://doi.org/10.1016/j.ecolmodel.2007.03.041

[2] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[3] David J. Crandall, Lars Backstrom, Daniel Huttenlocher, and Jon Kleinberg. 2009. Mapping the World's Photos. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, New York, NY, USA, 761–770. https://doi.org/10.1145/1526709.1526812

[4] Ilke Demir and Ramesh Raskar. 2018. Addressing the Invisible: Street Address Generation for Developing Countries with Deep Learning. *CoRR* abs/1811.07769 (2018). arXiv:1811.07769 http://arxiv.org/abs/1811.07769

[5] V. A. Epanechnikov. 1969. Non-Parametric Estimation of a Multivariate Probability Density. *Theory of Probability & Its Applications* 14, 1 (1969), 153–158. https://doi.org/10.1137/1114019

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507

[7] Antony Galton and Matt Duckham. 2006. What Is the Region Occupied by a Set of Points?. In *Geographic Information Science*, Martin Raubal, Harvey J. Miller, Andrew U. Frank, and Michael F. Goodchild (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 81–98.

[8] Mordechai (Muki) Haklay and Patrick Weber. 2008. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing* 7, 4 (Oct. 2008), 12–18. https://doi.org/10.1109/MPRV.2008.80

[9] Suradej Intagorn and Kristina Lerman. 2011. Learning Boundaries of Vague Places from Noisy Annotations. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '11)*. ACM, New York, NY, USA, 425–428. https://doi.org/10.1145/2093973.2094039

[10] Suradej Intagorn and Kristina Lerman. 2012. A Probabilistic Approach to Mining Geospatial Knowledge from Social Annotations. *SIGSPATIAL Special* 4, 3 (Nov. 2012), 2–7. https://doi.org/10.1145/2429177.2429178

[11] Suradej Intagorn and Kristina Lerman. 2012. A Probabilistic Approach to Mining Geospatial Knowledge from Social Annotations. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 1717–1721. https://doi.org/10.1145/2396761.2398504

[12] Ray A Jarvis. 1973. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters* 2, 1 (1973), 18–21.

[13] C. B. Jones, R. S. Purves, P. D. Clough, and H. Joho. 2008. Modelling vague places with knowledge from the Web. *International Journal of Geographical Information Science* 22, 10 (2008), 1045–1065. https://doi.org/10.1080/13658810701850547

[14] Adriano Moreira and Maribel Yasmina Santos. 2007. Concave hull: a k-nearest neighbours approach for the computation of the region occupied by a set of points. In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - Volume 1: GRAPP,*. INSTICC, SciTePress, 61–68. https://doi.org/10.5220/0002080800610068

[15] David O'sullivan and David Unwin. 2014. *Geographic information analysis*. John Wiley & Sons.

[16] Jin-Seo Park and Se-Jong Oh. 2012. A new concave hull algorithm and concaveness measure for n-dimensional datasets. *Journal of Information science and engineering* 28, 3 (2012), 587–600.

[17] Tye Rattenbury, Nathaniel Good, and Mor Naaman. 2007. Towards Automatic Extraction of Event and Place Semantics from Flickr Tags. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '07)*. ACM, New York, NY, USA, 103–110. https://doi.org/10.1145/1277741.1277762

[18] Mark Sanderson and Bruce Croft. 1999. Deriving Concept Hierarchies from Text. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '99)*. ACM, New York, NY, USA, 206–213. https://doi.org/10.1145/312624.312679

[19] Patrick Schmitz. 2006. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, Vol. 50. 39.

[20] David W Scott. 2015. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons.

[21] Wesley E Snyder, Hairong Qi, and William A Sander. 1999. Coordinate system for hexagonal pixels. In *Medical Imaging 1999: Image Processing*, Vol. 3661. International Society for Optics and Photonics, 716–727.