Getting Your Package to the Right Place: Supervised Machine Learning for Geolocation

George Forman

Amazon
ghforman@amazon.com

Abstract. Amazon Last Mile strives to learn an accurate delivery point for each address by using the noisy GPS locations reported from past deliveries. Centroids and other center-finding methods do not serve well, because the noise is consistently biased. The problem calls for supervised machine learning, but how? We addressed it with a novel adaptation of *learning to rank* from the information retrieval domain. This also enabled information fusion from map layers. Offline experiments show outstanding reduction in error distance, and online experiments estimated millions in annualized savings.

Keywords: Learning to rank · Geospatial supervised learning

1 Introduction

Amazon Last Mile delivers millions of packages daily to homes and businesses all over the world. To do this efficiently requires myriad optimization technologies, most of which rely on accurate geocoding. The geolocation (latitude, longitude) of each address is needed for partitioning and optimizing routes among vehicles, and guiding drivers on the road and to the door. Here we focus on the geolocation

Fig. 1. (Left) The centroid (cyan) of the GPS fixes from past deliveries (blue) lies in the middle of the street, whereas GeoRank correctly identified the doorstep (red), which is neither a centroid nor a high density point. (Right) The centroid (cyan) lies at an unrelated building between GPS clusters near the apartment (east) and the leasing office (west). The doorstep (black) was correctly identified by the machine learning ranking model (red).

problem of determining a precise *delivery point* (DP) geocode for each address, i.e. the location of the customer's doorstep, building entrance, or loading dock, as appropriate. Marking an accurate DP on the driver's map is especially useful and time-saving wherever house number signage is weak, missing, or obscured by trees. Inacdurate DPs lead to a bad driver experience, which may require walking around to find neighboring house numbers to deduce the correct delivery location. And this effort may get repeated each day by different drivers. Further, badly mislocated DPs can lead to inefficient route planning, sending packages on the wrong vehicles, and even misdeliveries or missed delivery time promises.

A strawman method for the DP would be to compute the *centroid* of GPS fixes from past deliveries.[1] Unfortunately, this can point drivers to the middle of the street (see the hypothetical example in Fig. 1 left) or to an unrelated building, depending on GPS noise and exactly when the driver marks the delivery complete. Centroids and mediods are prone to outliers (frequently miles away) and also make poor choices with multi-modal distributions, where they can fall on an unrelated building between popular delivery locations (Fig. 1 right). These examples also illustrate that selecting dense locations via clustering or Kernel Density Estimation (KDE) [18] does not quite suit, as the best point may have low density at the edge of the cloud.

The complexity of the geospatial inference problem clearly calls for some kind of supervised machine learning (ML) rather than a relatively simple geometric computation, but the research literature does not provide guidance. We need to train from a collection of situations where we have labeled the best point, and expect the system to learn to estimate the best point in new situations.

To do this, we have adapted *learning to rank* from the information retrieval domain, where, for example, if a user clicks on only the #3 search result, it implies a preference for #3 over #1 and over #2; these two implicitly labeled preference pairs can then be added to a training set to learn to rank more effectively for future searches. In our geospatial domain, when we manually label the best building entrance as the DP for an address, it can yield hundreds of labeled preference pairs between candidate locations based on distance. After training the ML ranking model on many labeled addresses, it can be applied to new situations to estimate the best DP.

This approach, which we call *GeoRank*, has several advantages: It is highly accurate and dominates other methods we have explored. It can deal with a variety of different situations simply by additional supervision. It is much easier to evaluate or label individual cases in this domain (with satellite map backgrounds) and, it is more objective than traditional information retrieval or ads ranking, where it is difficult to know that any given document or ad is truly not of interest to an anonymous user. Finally, it can leverage information from the underlying map while not requiring the map to be complete or even accurate.

[1] Before there is any delivery history for an address, the process is bootstrapped by other approximate geocoding methods to guide the driver. Third-party geocodes are not used in our geocode computations.

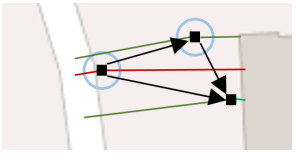

Fig. 2. Diagram representing rank comparisons between 3 candidate locations (black squares), the arrows indicating the winner of each pairwise comparison using feature vectors involving historical GPS point density (blue) and distances to roads and buildings (red or green lines), if present in the map. Given the three pairwise comparisons, it chooses the eastern point as the delivery point.

The contributions of this paper include (1) a description of a valuable geospatial task not previously in the machine learning research literature, (2) two very large ranking datasets (387K;445K cases with 43M;49M candidates with precomputed feature vectors and loss labels) released publicly to enable others to research effective, scalable ranking methods in this domain, and (3) our novel, geospatial supervised learning method, and evaluations thereof. The following section describes the GeoRank method in more detail, including candidate generation, feature extraction, and other algorithmic issues. Section 3 shows the method is highly effective in experiments on the New York and Washington datasets. Section 4 discusses limitations of the model and a brief description of additional offline and online evaluations we conducted before deploying the method in Amazon Last Mile. Section 5 describes related work and Section 6 concludes with future work.

2 Supervised Geolocation by Ranking

The inference task for each address is to estimate the best delivery point (DP) given as input a set of noisy GPS fixes of past deliveries. We adapt supervised ranking for our geospatial task as follows: For each address, we determine a set of candidate DP points and make pairs thereof (Section 2.1). For each pair, we extract a feature vector (Section 2.2), which includes information drawn from the map, where available. During training, we use the measured loss from the ground-truth labeled DP point to establish the preference order of each pair. We frame the base learning task as binary classification of the preference order, as in *pairwise* ranking [14, ch. 3]. During inference, we simply apply the pairwise binary classifier to all pairs to predict the candidate point of minimum loss.

The loss for a candidate location is its distance to the ground-truth labeled DP point. Additionally, we add a +20 meter penalty if that candidate lies atop a different building outline than the one attached to the label, if available. This is to train the system to prefer ambiguous locations outside buildings rather than show a pin on the wrong building. Note that some areas of the map have no building outlines available, and some may be missing or incorrect.

2.1 Candidate filtering and generation

Simply considering each past delivery location as a candidate naturally results in long $O(N^2)$ run times when there are thousands of past deliveries. We greatly

improved scalability by filtering near-duplicate candidates on a virtual hash grid; this filter and other scalability improvements reduced several days of computing for the US to less than half a day.

When there are very few past deliveries, the best point available may not be particularly close to the building. Where building outlines are available, we also generate candidate points along the faces of nearby buildings (see the diagram in Fig. 2). This helps solve a troublesome problem with occasional tall buildings with multipath GPS reflections that exhibit a *consistent GPS bias* away from the building, not generally noted in the GPS literature. Even where all the past GPS fixes are on the wrong side of the street, the system can recover by considering candidates on building faces.

We are able to generate a class-balanced training set by randomly choosing whether to put the better candidate in the first or second position of each pair. For training, rather than generate all $\frac{N(N-1)}{2}$ pairs of candidates—which produces vastly more pairs for those addresses having more delivery history (or having more scattered points that get past deduplication)—we generate just $O(N)$ pairs, selecting the best vs. each of the others. Not only did training time improve, but it also led to improved model accuracy. This is akin to research in information retrieval ranking that shows selecting more pairs with the most relevant documents can lead to improved ranking, e.g. [4]. (We also limit the number of training pairs from any single address to 100, so that addresses with many deliveries do not dominate the learning.)

2.2 Feature vectors

We compute three kinds of features from which the GeoRank model learns to do its ranking. Each feature is designed to be invariant to rotation and resilient to outliers or missing information in the map.

Features based on the GPS fixes from past deliveries:

Our app gathers a GPS fix at various driver events, e.g. photo-on-delivery or when the driver marks a package as delivered, which may or may not be close to where the package was actually dropped off. Given a history of G GPS fixes for a particular address, we compute the following features for each candidate DP: the density of past GPS fixes nearby (e.g. using Gaussian KDE with a bandwidth of 25m), the distance to the GPS fix having maximum KDE, the mean distance to the K-nearest neighbors ($K = \sqrt{G}$), and the percentage of those K neighbors that were delivered to the office (mail room, receptionist, etc., rather than the customer or customer's doorstep, as noted by the driver).

The intuition behind this last feature is to discriminate between the leasing office and separate apartment buildings. We intend the DP to identify the individual apartment building entrance, as it is comparatively trivial for the driver to find the shared leasing office. (For scalability we limit G to a random sample of $\leq$500 past deliveries. Here we cannot use the near-duplicate filter described previously, else we lose fine-grained information about density.)

Features drawn from the underlying map:

The local map can provide useful geospatial context. For example, drivers often mark packages as delivered when they get back to their vehicle, but GPS fixes located on the street or in a parking lot are unlikely to be good customer DP locations compared to GPS fixes nearer building outlines. Thus, for each candidate DP, we compute: the distance to the nearest street, the distance to the nearest parking lot (zero inside the parking lot), the distance to the nearest building (zero inside), the distance to the building closest to most GPS fixes, etc. To compute these features efficiently, we build in-memory geospatial indices: a KD-tree for map points marked with an address, and Sort-Tile-Recursive R-trees [16] for streets, buildings, and parking lots.

We use a maximum feature distance of 1000m to cover the common situation where the map is incomplete, e.g. areas lacking parking lots or building outlines. This maximum value serves as an indicator to the ML model that the feature is locally missing.

Occasionally the map contains an explicitly marked address point. Where available, one can imagine using such information to bypass the ranking computation altogether, but any map error would cause DP errors. Instead, we include this information as an additional feature: the distance to the nearest building or point that is marked with the *sought* house number and street name, if available (else 1000m). This feature, which includes text address matching logic, enables the ranker to leverage the information but also to override it where there is sufficient evidence from past GPS data that disagree with the map. The discrepancy may be because of errors in the map (see Fig. 3 where house number 18 is recorded incorrectly) or because deliveries are actually around back or are redirected to another building.

Context features:

For all candidate pairs, we compute these context features: the number of past deliveries, the number of building outlines nearby (some areas have none), the median and P10 distance between pairs of GPS fixes (indicating how dispersed the points are; we found the average or P75 very sensitive to a single outlier), the point density (number of GPS points ÷ median distance), the median GPS-reported accuracy of the fixes (worse in city canyons), and whether there is any nearby address in the map with an exact match. These context features can be leveraged across the whole training set to learn preferences in different situations. For example, the ranker could potentially learn different behavior where GPS accuracy tends to be weaker.

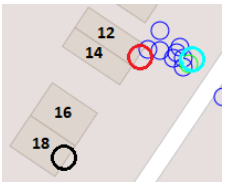

Fig. 3. Though the map marks house 18 at the south (black circle), the GPS evidence (blue) indicates it is actually at the north, which the GeoRank model correctly selected (red). Meanwhile the centroid (cyan; some data not shown) is near the street, which would leave ambiguity between the two northerly buildings for the driver.

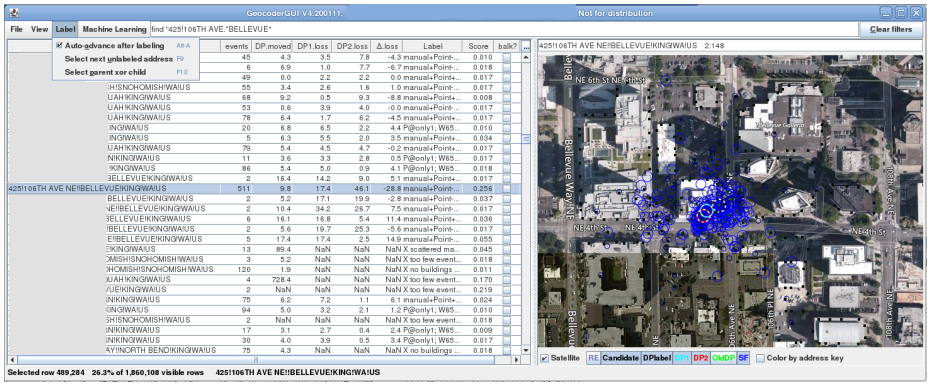

Fig. 4. Research GUI showing our Amazon building in Bellevue. Blue circles show past GPS fixes; square dots show candidate points to rank; dashed circles show DP choices by different methods.

Pairwise ranking for a linear model f often uses the difference-vector $\mathbf{u} - \mathbf{v}$, where $\mathbf{u}$ and $\mathbf{v}$ are the feature vectors of the two candidates in the pair. (Proof sketch: the pairwise binary classifier $f(\mathbf{u}) \gtrless f(\mathbf{v}) \equiv \mathbf{w}\cdot\mathbf{u} \gtrless \mathbf{w}\cdot\mathbf{v} \equiv \mathbf{w}\cdot(\mathbf{u}-\mathbf{v}) \gtrless 0$, where $\mathbf{w}$ is the weight vector to be learned by the model.) Since pairs stem from the same address, context features have the same value for both candidates, rendering their difference useless to the classifier. Thus, we compute $\mathbf{u}$ and $\mathbf{v}$ from only the first two kinds of features above, and generate the composite feature vector of the pair as $(\mathbf{u} - \mathbf{v}, \mathbf{u}, \mathbf{v}, \mathbf{c})$, where $\mathbf{c}$ are the context features appended at the end. Evaluations of feature importance show that all four portions of the feature vector are used by the base classifiers.

2.3 Base classifiers and implementation

Although the difference-vector representation is motivated by linear classifiers, we found better performance with non-linear models. We experimented with a variety of base classifiers, such as logistic regression, SVM, decision trees, Random Forests and Gradient Boosted Decision Trees. The number of training pairs can be in the millions for a region, but with an efficient, thread-parallel implementation of decision tree learning, such as in the SMILE library [12], the base classifier usually trains in 1–10 minutes and exhibits 98–99% pairwise accuracy on a holdout set. Once the data is staged, separate regions are computed as *embarrassingly parallel* jobs on a fleet of AWS EC2 instances.

Fig. 4 shows a screenshot of our custom Java Swing GUI tool, which was essential to research, develop, and debug the details of our GeoRank method. We used it to inspect results, label DP locations, and sort & search a table with millions of addresses for informative cases to label, e.g. via active learning where multiple models disagree widely.

Table 1. Datasets

	New York (NY)	Washington (WA)
cases	387,054	445,073
candidates	43,436,526	48,992,148
avg candidates/case	112.2 (2–500)	110.1 (2–500)
file size	6.1 GB	6.8 GB

3 Experiments

We have done extensive offline and online experiments to develop, refine, and validate our geospatial supervised learning method. Here we describe a limited set of experiments to answer fundamental questions about its performance, which illustrate its effectiveness.

3.1 Datasets

The experiments are conducted on two labeled ranking datasets that we generated for New York state and Washington state, each containing many millions of candidate points (see Table 1). Although the ranking problem is fundamentally geospatial, in order to release the data for research and reproducibility, we have constructed it so that it does not contain any information about actual addresses or physical locations, and it is not a random sample of our actual cases. Nonetheless, it is a valuable resource for research: this domain has fundamentally different qualities than ranking datasets in information retrieval and ad ranking, where query sets are sparse, subjective, and discretized into a few levels. But for these datasets the average number of candidates to be ranked per case is over 100, and each is labeled with a non-negative scalar loss. When we split the dataset into folds for train and test sets, we partition based on a hash of the normalized address fields (excluding the apartment number, so an apartment complex will not be split across train and test). Only the randomized fold number is included in the dataset, which has 100 distinct folds, should others wish to perform consistent cross-validation splits and/or consider learning curves with fine granularity. These datasets have 18 features per candidate, plus 16 context features per case. Together they generate 70 features per pair, when expanded as described at the end of Section 2.2.

3.2 Loss vs. business objective

Recall that the loss is a measure of distance to the ground truth label, which represents a judgment of the best delivery point, penalized by +20m if the point falls on the wrong building. In this domain there are very many addresses that are relatively straightforward to process, e.g. where the GPS fixes are fairly close together and are near the best delivery point, such as the front door of a lone building. But it is the occasional large geolocation errors that cause the greatest expense: because of a stray geocode far away, we may place packages on the

wrong delivery vehicle, and the driver may either have to drive far outside their planned route and time, or else return the package to the delivery station at the end of their route for a later redelivery attempt—running the risk of missing the delivery date promised to the customer. Thus, rather than focus on the mean loss, as most machine learning work, we focus on the P95 loss.

This objective is fundamentally different than optimizing the average loss or the mean squared error. Consider two Gedanken experiments to make this plain. First, suppose that 90% of the cases had 1m loss, and with a variant model we could reduce loss to exactly zero for only these cases. It would bring the average down, but the last meter of perfection would hardly make a difference to delivery drivers in the real world. If this variant model were also slightly worse in the tail, the performance objective should reflect this problem. Second, suppose that the P99.99 were either in the next state or else in another country. The difference between these two can have a large effect on the average or the mean squared error, but the business cost of these two problems would be about the same. Hence, we focus on the P95 loss as the objective to optimize in this paper (in practice we also use higher tails). Another advantage of using a percentile instead of a mean is that the metric is unaffected by small errors in labeling precision, such as when an auditor clicks a few pixels off from the ground truth point; this also speeds up labeling and greatly focuses label validation efforts on a small minority of points.

3.3 How does it perform against baselines?

We begin by framing the performance against several baselines: random selection, centroid/mediod, maximum KDE, and oracle selection. These illustrate the feasible range of performance, and the latter represents an unrealizable lower bound. Even the oracle cannot achieve zero loss in many cases, because no candidate has been generated that is exactly zero distance from the DP label, which may itself be imperfect.

Fig. 5 shows, for each dataset, the loss of each method across the whole distribution from P01 to P99 (lower is better). Each point was determined by 20-fold cross-validation (CV), and the vertical width represents the 95% confidence interval of the value (using T-distribution with 19 degrees of freedom). Zoom to see the width, which is barely discernable in many cases.

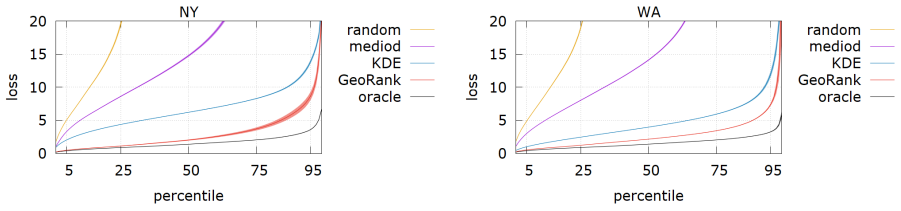

Fig. 5. Comparing methods across whole CDF loss distribution; each point shows 95% confidence interval from 20-fold CV.

Random: selects one of the candidate DP points at random. Its median loss (54 NY; 55 WA) and P95 loss (157; 138) are off the chart, literally. This oblivious method does nothing to avoid outlier points, and represents a kind of upper-bound just as a majority-voting baseline does in classification experiments.

Mediod: selects the candidate nearest the centroid. Its P95 loss is also off the chart (84; 78). Whenever there are multi-modal point distributions, e.g. between the leasing office and the customer's apartment building in a large multi-building community, it will tend to select a point between the two, usually an unrelated building. This would point drivers to the wrong building for delivery (as in Fig. 1 right). In situations such as large warehouses or malls with multiple entry points, the centroid will tend to point to the middle of the building. This is sufficient for most drivers, assuming the entrances are plain, but may not enable an optimized route plan.

Maximum KDE: selects the candidate in the densest cloud of GPS fixes. This can be quite competitive. If there are multiple clusters at different building entrances, it usually picks the more popular entry. These points tend to be somewhat away from the face of the building, however. Often the densest locations tend toward the parking location of the vehicle. All of these methods so far are oblivious to the real-world constraints, e.g. that building entrances do not lie inside parking lots (actually, there can be underground buildings on hillsides with rooftop parking—everything happens in this domain).

Oracle: selects the best available candidate point. It does this, of course, by cheating: it uses the hidden loss of the test case to make its choice, and as such is not a practical algorithm. It is useful here to understand the lower bound on loss, given the candidate DPs that are available from the practical generation algorithm described in Section 2.1. Its loss at P99 suggests future work in candidate generation.

GeoRank: supervised geolocation learning Overall, this approach solves this geospatial problem very nicely: its loss distribution tightly hugs the oracle performance up until the tail of the distribution, and at P99 it rejoins the loss of the KDE method—but does not exceed it, as we shall see. If we subtract the oracle loss as the lower bound, GeoRank at P95 reduced the delta loss by 50–53% of the KDE algorithm.

To complete the 20-fold CV in a timely manner, rather than train on all 95% of the dataset available for training, we trained each fold on only 20% of the dataset. (Thus, the confidence intervals are more accurate than if the models each trained on highly overlapped training sets.) This yielded 6–7M labeled pairs to train each model—plenty for an accurate pairwise classifier. The decision tree learning algorithm simply splits nodes best-first based on the Gini index, up to a maximum of 1024 leaf nodes, which it always used. Each fold trains in $\sim$5 minutes in parallel on 16 Xeon 2.3GHz CPUs; together they inference 13K addresses/second, even with the $O(N^2)$ pairwise comparisons used to determine which candidate has the most wins.

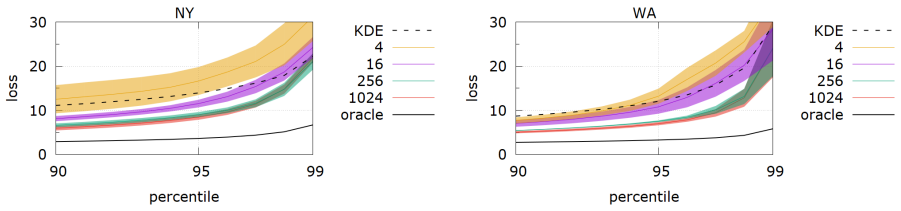

Fig. 6. Zooming on the tail of the loss distribution, as we vary model capacity: the number of leaves in the decision tree.

3.4 How is the tail affected by model capacity?

In Fig. 6 we zoom in on the tail of the loss curve (P90–P99), as we vary the pairwise classifier's model capacity: the number of leaf nodes in the decision tree. When just 4 leaves are permitted, its performance was worse than KDE. With 16 leaves, it beat KDE except for the extreme tail of the distribution. With more model capacity, it always beat KDE. The 1024 curve represents our default for GeoRank.

3.5 Lesion studies and RankNet comparison

Finally, we perform a series of independent lesion studies on the method, in order to measure the relative importance of various component ideas. For these experiments, we want to have confidence intervals, so we again use 20-fold CV, but to reduce the computation load we train on just 5% of the data (which the previous experiment showed was sufficient, and it saves a great deal of time, esp. for the RankNet comparison, later).

Fig. 7 shows the P95 loss of each variant under 20-fold cross-validation, and the whiskers extend to cover the 95% confidence interval. Just for reference, the vertical lines indicate the performance of the oracle and the KDE method. The baseline performance is the basic GeoRank method.

The second data point indicates the performance of using the classifier trained from the opposite state. In both cases the result was statistically significantly worse. This endorses the idea of having separate models trained for each region, as regional geometric patterns may vary. For example, we observed the setbacks from the road were often larger in Arizona.

The next comparison simply removes two components from the four-part composite feature vector; $\mathbf{u}$ and $\mathbf{v}$ are removed, reducing the total number of features from 70 to 34 (18 difference features $\mathbf{u}-\mathbf{v}$ and the 16 context features). Removing these features had a relatively smaller effect on performance, which was not statistically significant for NY.

The next two bars represent changes in how we generate training pairs. Recall that we pair the best candidate against each of the other candidates. A common approach is simply to generate random pairs, rather than all $O(N^2)$ pairs, which can overwhelm computing resources. For each candidate, we pick another at

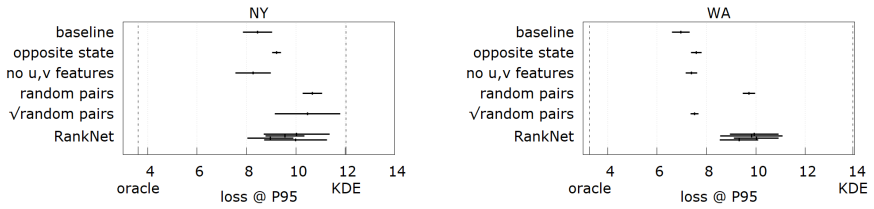

Fig. 7. Lesion studies on the GeoRank method and RankNet comparison.

random to pair with it, producing $O(N)$ pairs, but the result is significantly and substantially worse.

The other variant, marked with the $\surd$ symbol, generates training pairs as follows: for each candidate $i = 2..n$ in the training list sorted by loss, we randomly select one of the first $\lfloor\sqrt{i}\rfloor$ candidates. The idea is to focus more attention on the better candidates, but not overly focus on the single best candidate. It worked better than random pairs on average for both states (though not significantly so for NY); it is nonetheless significantly worse than the 'best vs. rest' strategy employed by GeoRank.

Finally, we replaced our pairwise ranker with RankNet [7] from the RankLib research implementation [9]. The four variants represent efforts at parameter tuning, varying the number of training epochs and whether or not the candidate features were included (oddly, they often hurt RankNet's performance). In all cases and for both states it performed worse on average than the baseline, though not always statistically significantly. We considered running similar comparisons with other well-known information retrieval ranking algorithms such as LambdaMART, but adapting them to use our loss scale rather than discretized document relevance is beyond the scope of this paper—potential future work.

4 Discussion

We have done additional lesion studies that are beyond the scope of this paper. For example, we have demonstrated that if all building outlines are removed from the map when generating the features for the test cases, the quality of the geolocation degrades only a little. This is an important property, for there are whole regions of our world map that contain no building outlines yet. It would be complicated to have to train separate models for situations with and without building outlines, and then to correctly classify when to use each model. Better to have a single robust model that can cope with their absence.

4.1 Real-world offline evaluations

When we began to apply machine learning to this business problem, we ran into a hitch: For years our established way to measure the quality of our DPs was by their average distance to deliveries, as reported by the GPS fixes. And yet, by this

objective metric, there would be no way to outperform the centroid, by definition. Thus, we had to drive the adoption of a different goal metric, which would rightly show improvement when we selected a good building entrance, even if it were away from the dense center of the point cloud. But we had no automated way to determine the true quality of a DP, otherwise we would already have solved the problem. In view of this, our internal Curated Ground Truth (CGT) program was born: a worldwide manual labeling effort, with various quality controls to ensure that we are grading ourselves against accurate points. Given this labeled dataset of thousands of randomly selected addresses, we were able to determine that the GeoRank method reduced the P95 error distance vs. the existing legacy system by ∼18%. These CGT points were only used for evaluation of the final model, not for model training or hyper-parameter optimization. In fact, we need to maintain separation of the points used for training and model selection vs. those held-out points that are secreted away only for use in monitoring the health and quality of our systems. Because the majority of cases are straightforward and provide little incremental insight, the sampling of points for training and model selection are biased toward the tail by various means, including query-by-committee active learning. Given that the CGT points are randomly sampled in order to correctly identify the percentiles, we sought additional validation of the GeoRank model before risking its live deployment. We explored the tail in further offline testing. In some states we identified thousands of cases (excluding training cases) where GeoRank moved the DP by over 100m from what was previously vended. Though this was a only a small fraction of the whole dataset, they could prove to be costly if the software threw some points far astray, which would break customer delivery date promises and erode trust. We obtained a random sample of such points using the research GUI and were able to quickly label a good fraction of them, though some tail cases are quite difficult to understand. In this skewed sample, we determined that the median loss of the old points was ∼16x the median loss of the GeoRank model. This was a strong endorsement indeed, though only for a small sample. We also performed some automated offline testing against a large number of OpenStreetMap addresses [1], which again showed strongly favorable results.

4.2 Real-world online evaluations

Once we believed that the GeoRank delivery points were superior in offline testing, we moved to online A/B testing, dialing up to 10% treatment across the US. The assignment of treatment and control groups was determined by a hash of the address, excluding any apartment number so that a complex would not fall into both groups, for greater statistical power. Analysis of the service time per package determined a substantial and statistically significant improvement, projecting millions of dollars saved annually. One aspect of owning the data quality of an upstream system is that your errors impact many downstream processes. Besides service time improvements, there were also savings because of improved planning, since the DPs were now more truthful.

4.3 Limitations

Several things have become clear after substantial labeling and domain experience. First, there is no way to do the job accurately without seeing the historical delivery data. Although you may identify the front door of a building address by three methods and even walk there yourself, it might be that deliveries are actually redirected around back to a loading dock or even to a distant campus-wide mail hub.

Second, having more delivery history helps both in labeling as well as in GeoRanking. However, a good fraction of customer addresses are new and have very little history to work from, especially in areas of rapidly growing demand, such as India. Thus, there is a natural labeling bias and research bias toward addresses with more delivery history. How can one accurately geocode given a single, possibly noisy delivery event in an area of the map that is yet uncharted?

Third, there can be more than one delivery location for a single address, and in different ways. From the event history, we may see multiple clusters, but only by deeper investigation (or machine learning) can we determining things like 'small packages are accepted at the office, but large packages must go to the loading dock.' Or that 'packages are accepted at the leasing office on Tuesday and Saturday mornings, but otherwise all packages must be driven to the individual apartments.' Though manual labeling for multiple delivery locations is harder, fortunately they can easily be represented in a ranking dataset: for each candidate, simply take the minimum loss compared to the alternative DP labels. If we wish to compute the alternate DP locations, we can re-run the ranking algorithm after removing candidates near the DP that was chosen on the first iteration. Though ranking is fast, we can avoid the $O(N^2)$ pairwise comparisons if we keep track of the matrix of pairwise wins. If the second run chooses a sufficiently distant point, then it identified an alternative; but sometimes it would simply pick the next closest candidate to the removed area. This is another area for future work. It can be difficult to decide whether multiple alternatives are warranted or not, given scattered data.

Fourth, labeling or locating the best building entrance or loading dock in the real world can be arbitrarily difficult. There are addresses that have large amounts of delivery history, and yet remain impossible to nail down, such as a complex hospital with multiple entrances, buildings, and delivery procedure complexity as well. Because of these biases and others, none of our labeled datasets fairly represent the full difficulty of the general problem. And regardless of labeling difficulty, it is unclear what the best distributed sample should be: Should

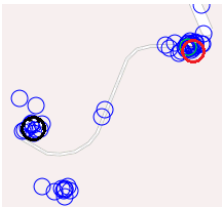

Fig. 8. Long driveway with locked gate. The model experienced high distance loss because it picked the gate (red) instead of the southern garage or the house (black label).

it be weighted by package volume? Or by service time, so difficult addresses are worth more? Or by the frequency that *inexperienced* drivers visit an unfamiliar, complex location. We know that the cases with high loss do not always represent high business cost. For example, in the tail we see a plethora of cases where a customer has a locked gate across a long driveway (see Fig. 8). Though the best DP is at the house, it may be that nearly all packages are delivered near the locked gate. Though this results in high measured loss, it does not result in any real driver confusion or delays. And though we would like to label the apartment door as the correct DP, what about situations where 99% of deliveries go to the leasing office? Should we consider it a high loss case if the GeoRank model chooses the office?

5 Related work

The task of *geocoding* is the process of mapping address text or an IP address to an (often coarse) geographic location, which requires geographic databases. Our *geolocation* problem is substantially different: given multiple noisy samples of a location, determine a (precise) geographic location for future deliveries; this can be done with the points alone by computing the centroid, or the argmax KDE point—or by the supervised GeoRank method we developed, which has the advantage that it can also leverage features from maps.

Note that our geolocation problem is not to be confused with research on 'geographic information retrieval,' which are traditional information retrieval (IR) applications that aim to improve the ranking of their search results by leveraging the user's location in order to adapt to regional preferences [2]. ML ranking has also been applied to Point-Of-Interest (POI) information retrieval, which leverages the user's noisy GPS position as well as features such as user history, popularity of venue, and nearby social network friends. The learned ranking model optimizes the short list of potential named locations (such as tourist sites or restaurants) from which the user may select to 'check-in,' i.e. confirm their location [20, 13, 22]. These confirmations can be used as a ground-truth POI label for testing or retraining the ranking model. Such IR applications measure their success by metrics on the ranked position of the correct answer(s) in the search results list; whereas our task measures loss in terms of meters away from the ground-truth location. Other related work includes the destination prediction problem popularized by the ECML/PKDD taxi trajectory prediction competition [6, 3], although some have approached destination prediction by building models to score or rank a few candidates, such as home, work, or school (which is akin to classification models with a few classes) or a city's popular tourist destinations [11, 5]. Finally, there is general geospatial work to re-identify delivery locations, such as by [17] which used centroids of past 'stay point' locations.

There has been a tremendous amount of research in learning to rank [14], esp. for information retrieval (IR) and, more profitably, for advertising. The field has benefited greatly from having research datasets available, such as the LETOR collection [15]. Similarly, we hope to launch research in supervised geolo-

cation with the two datasets we provide; being orders of magnitude larger than LETOR, they are useful for more general ranking research, e.g. selective sampling or scalability research [19]. For confidentiality reasons, neither the address nor the actual locations can be reconstructed from the data; thus, we have also provided location-invariant feature vectors based on attributes from the map, etc. Should some future researcher be in a position to release data including latitude/longitude locations, many creative research avenues could be pursued involving features from maps.

As mentioned in Section 3.1, geo-ranking data is substantially different. First, each case has many labeled candidates, whereas traditional ranking datasets are much sparser. Second, relevance judgments for traditional datasets, which are hard to obtain objective truth for, often have a small ordinal ranking for each query, indicating the rough degree of document relevance, with a default of 'irrelevant' for most ungraded documents. Geo-ranking datasets have a continuous real-number for the loss, which provides a total ranking of all candidate points. Third, traditional datasets often have shared pool candidates, e.g. documents in a fixed collection searched by many different queries. In our datasets, candidate points are not shared across addresses; each has a distinct loss and feature vector dependent on the target address.

There have been a variety of algorithmic improvements in learning to rank, following a progression: (1) point-wise learns a regression for each point independently; (2) pair-wise learns to select the better of a pair of points; and (3) list-wise learns to return a list of the top-K points optimized by any list-scoring method. List-wise ranking is not necessary for our domain, as we only seek a single best location. A common methodology for low-latency situations is to train a model on pairwise examples with known or implicitly inferred user preferences, in order to learn a pointwise scoring function that can be run independently in $O(N)$ time (embarrassingly parallel) for thousands of candidate documents or ads selected by a traditional information retrieval system [8]. In our domain, we only have hundreds not thousands of candidates per address, so an $O(N^2)$ pairwise comparison is plenty fast and appears to be more accurate generally. There is algorithmic research to reduce the $O(N^2)$ pairwise comparisons to $O(N \log N)$, which may help scale other applications [10, 21].

6 Conclusion and future work

We hypothesized that the learning to rank paradigm would adapt well for our geolocation problem. It proved able to efficiently and accurately place delivery points near building entrances. By showing an accurate point on the map in our app, we help drivers more quickly and accurately identify the correct delivery address. Online A/B tests showed that it resulted in time savings worth millions of dollars annualized.

It is fortunate that the method was effective with the existing data quality. This is faster and more frugal than efforts to try to obtain better input data, e.g. expensive GPS devices. The GeoRank method makes up for relatively poor

GPS precision in big cities by leveraging map information, which also tends to be more complete in cities. And yet the method is not dependent on the map, for there are large regions with few building outlines and many missing roads. We have since applied ranking to other geolocation problems successfully, such as the parking location. It's always 'Day 1' at Amazon.

References

1. Open Street Map (OSM), www.openstreetmap.org
2. GIR'10: 6th Workshop on Geographic Information Retrieval. ACM (2010)
3. ECML/PKDD Competition: Taxi Trajectory Prediction (2015), kaggle.com
4. Aslam, J.A., Kanoulas, E., Pavlu, V., Savev, S., Yilmaz, E.: Document selection methodologies for efficient and effective learning-to-rank. In: SIGIR'09 (2009)
5. Baraglia, R., Muntean, C.I., Nardini, F.M., Silvestri, F.: Learnext: Learning to predict tourists movements. In: CIKM '13. pp. 751–756 (2013)
6. de Brébisson, A., Simon, É., Auvolat, A., Vincent, P., Bengio, Y.: Artificial neural networks applied to taxi destination prediction. arXiv CoRR (2015)
7. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: ICML '05. pp. 89–96 (2005)
8. Burges, C.J.: From RankNet to LambdaRank to LambdaMart: An overview. Tech. Rep. MSR-TR-2010-82, Microsoft Research (2010)
9. Dang, V.: The Lemur Project-RankLib, sourceforge.net/p/lemur/wiki/RankLib/
10. Hong, L.J., Luo, J., Zhong, Y.: Speeding up pairwise comparisons for large scale ranking and selection. In: IEEE WSC '16. pp. 749–757 (2016)
11. Lassoued, Y., Monteil, J., Gu, Y., Russo, G., Shorten, R., Mevissen, M.: A hidden Markov model for route and destination prediction. In: ITSC'17. pp. 1–6 (2017)
12. Li, H.: SMILE: Statistical Machine Intelligence and Learning Engine, github.io
13. Lian, D., Xie, X.: Mining check-in history for personalized location naming. ACM Trans. Intell. Syst. Technol. **5**(2) (2014)
14. Liu, T.Y.: Learning to Rank for Information Retrieval. Springer-Verlag (2011)
15. Qin, T., Liu, T.Y., Xu, J., Li, H.: LETOR: A benchmark collection for research on learning to rank for information retrieval. Information Retrieval **13**(4) (2010)
16. Rigaux, P., Scholl, M., Voisard, A.: Spatial Databases with Application to GIS. Morgan Kaufmann (2002)
17. Ruan, S., Xiong, Z., Long, C., Chen, Y., Bao, J., He, T., Li, R., Wu, S., Jiang, Z., Zheng, Y.: Doing in one go: Delivery time inference based on couriers' trajectories. pp. 2813–2821. KDD'20 (2020)
18. Scott, D.: Multivariate Density Estimation: Theory, Practice, and Visualization. John Wiley & Sons (1992)
19. Sculley, D.: Large scale learning to rank. In: NIPS 2009 Workshop on Advances in Ranking (2009)
20. Shaw, B., Shea, J., Sinha, S., Hogue, A.: Learning to rank for spatiotemporal search. pp. 717–726. WSDM'13 (2013)
21. Wauthier, F., Jordan, M., Jojic, N.: Efficient ranking from pairwise comparisons. In: ICML'13. vol. 28, pp. 109–117. Atlanta, Georgia, USA (17–19 Jun 2013)
22. Ying, J.J.C., Lu, E.H.C., Kuo, W.N., Tseng, V.S.: Urban point-of-interest recommendation by mining user check-in behaviors. pp. 63–70. UrbComp'12 (2012)