Fast semi-supervised regression: a geodesic nearest neighbor approach

Amit Moscovich, Ariel Jaffe, Boaz Nadler
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel
{amit.moscovich, ariel.jaffe, boaz.nadler}@weizmann.ac.il

Abstract

We consider semi-supervised learning methods based on a combination of geodesic graph distances and classical nonparametric statistics. The idea is to represent the geometry and connectivity of the unlabeled data by a single metric, which measures shortest path distances to the labeled points. Learning is then performed by applying fully-supervised methods, such as Nadaraya-Watson kernel smoothing or k-nearest-neighbors, but with weights determined by the geodesic graph distances. Our key contribution is a novel fast algorithm for finding the k geodesic nearest labeled neighbors of all unlabeled data points. In typical semi-supervised settings with a large number of unlabeled points, our algorithm is significantly faster than standard methods to compute geodesic nearest neighbors. It is also much faster than common spectral methods. We demonstrate the competitive accuracy and runtime of our approach on a problem of indoor positioning based on WiFi fingerprints.

1 Introduction

Consider the following learning setup: let $\mathcal{X}$ be an instance space and let $\mathcal{Y}$ be the output space which may be discrete or continuous. We assume that there is some unknown distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ over the product space $\mathcal{X}\times\mathcal{Y}$ with data drawn independently from it. In semi-supervised learning (SSL) we are given both a labeled set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$ sampled from $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$, and an unlabeled set $\mathcal{U} = \{\mathbf{x}_i\}_{i=L+1}^{n}$ where $n = L + U$, sampled from the marginal distribution $\mathcal{D}_{\mathcal{X}}$. The goal is to construct an accurate classifier or regression function using both the labeled and unlabeled data. Two common variants of this problem are *transductive* and *inductive* learning. In inductive learning, the constructed classifier or regression function must be able to output a predicted value for *any* point in $\mathcal{X}$. In the transductive setting, on which we focus in this work, the response y needs to be predicted only at the U unlabeled points.

With the increasing ease and feasibility of collecting many unlabeled samples, SSL has gained importance and popularity. For various reviews, see Chapelle et al. (2006); Zhu and Goldberg (2009); Subramanya and Talukdar (2014). One approach that has received much attention in recent years is *graph-based* semi-supervised learning. The underlying idea is to construct a weighted graph with vertices that correspond to all the labeled and unlabeled points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and edges that connect close pairs of points. A partial list includes the Laplacian-based approach of Belkin and Niyogi (2004), diffusion based methods (Zhu et al., 2003), methods based on minimum graph cuts (Blum and Chawla, 2001) and multiscale wavelet regression methods (Gavish et al., 2010). These methods can be challenging to apply on large data sets due to their runtime complexity. For example, the method of Belkin and Niyogi (2004) computes $O(L)$ eigenvectors of a large $n \times n$ matrix, Zhu et al. (2003) inverts a $U \times U$ matrix and the algorithm of Gavish et al. (2010) performs multiple runs of hierarchical clustering over the entire data set.

Several methods for efficient large scale semi-supervised learning on graphs have been proposed in recent years. One class of methods reduces the dataset to a small set of prototypes (Delalleau et al., 2005; Liu et al., 2010). A different class subsamples the edges of the graph (Herbster et al., 2009; Vitale et al., 2013; Zhang et al., 2015).

In this paper we present a simple computationally-efficient approach to SSL on a given (and potentially large) graph, that does not require its approximation. The main idea is to apply standard learning algorithms, but base them on geodesic graph distances. As described in Section 2, this is particularly suitable when the data lies on a manifold and the response is a smooth function along it. To apply this approach, one needs to efficiently compute, for all the unlabeled samples, their first few geodesic nearest labeled neighbors. Our key contribution, described in Section 3 and theoretically analyzed in Section 4, is a novel fast algorithm for this task. In typical semi-supervised settings with a large number of unlabeled points our algorithm is significantly faster than standard shortest path methods. Finally, in Section 5 we demonstrate its competitive accuracy and runtime on a problem of indoor positioning based on WiFi fingerprints. A second illustration on a dataset of facial poses appears in the appendix.

While our focus is on semi-supervised learning, our algorithm may be relevant to other applications as well, for example: (i) in the field of computer networking it may be used to find the k nearest cache servers to all clients in a content delivery network; (ii) in transportation, to find shortest paths from multiple locations to nearby points of interest; and (iii) in computer graphics, to efficiently locate the nearest landmark points for all points on a polygon mesh.

2 Semi-Supervised Learning with geodesic distances

For the unlabeled data to aid in learning, there must be some relation between the response and the marginal density of the unlabeled data, see for example Singh et al. (2009); Liang et al. (2007). One common relation is the *cluster assumption*. Here, instances with the same label tend to concentrate in well-defined clusters separated by low density regions (Rigollet, 2007; Chapelle and Zien, 2005). A second model, at the focus of this work, is *the manifold assumption.* Here, the data points are contained in one or several unknown low dimensional manifolds, with nearby instances on the manifold having similar response values. In this case, the learning problem has a lower intrinsic dimension than that of the ambient space. The unlabeled instances can be used to learn the manifold, thus overcoming the curse of dimensionality.

2.1 Geodesic distances in the manifold setting

Under the manifold assumption, one approach to use the unlabeled instances is to apply some nonlinear dimensionality reduction procedure that maps the data to a lower dimensional space (Tenenbaum et al., 2000; Roweis and Saul, 2000; Belkin and Niyogi, 2003), followed by a standard learning algorithm. In this approach, one needs to know or otherwise estimate the dimension of the underlying manifold. In addition, these methods may not work well when the data lives on a union of several possibly intersecting manifolds.

A second approach, at the focus of this paper, is to *directly* learn via graph geodesic distances. Let $\mathcal{M}$ be a compact submanifold of $\mathcal{R}^D$ with geodesic distance metric $d_{\mathcal{M}}$ and let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be points independently sampled from $\mathcal{M}$ according to some probability that is bounded away from zero. Let G be the symmetric kNN graph corresponding to the data, where i and j are connected if $\mathbf{x}_i$ is one of the k closest neighbors of $\mathbf{x}_j$ or vice versa. As the number of samples $n \to \infty$ and with a suitable choice of k, the shortest-path distance d_G converges to the geodesic distance on the manifold (see the supplementary to Tenenbaum et al. (2000)),

$$d_G(\mathbf{x}, \mathbf{x}') \xrightarrow{n\to\infty} d_{\mathcal{M}}(\mathbf{x}, \mathbf{x}'). \tag{1}$$

One practical consequence of Eq. (1) is that if the data lies on a submanifold $\mathcal{M} \subset \mathcal{R}^D$ of intrinsic dimension $d < D$ then the errors of learning procedures based on the manifold distance such as geodesic nearest neighbors and local regression are governed by the *intrinsic dimension* of the data d rather than the extrinsic dimension D.

2.2 Learning with geodesic distances

Motivated by the manifold assumption, we consider the following framework for transductive semi-supervised learning. Given samples $\mathbf{x}_i$ from an instance space $\mathcal{X}$ with a distance function $d(\mathbf{x}, \mathbf{x}')$:

1. Construct an undirected (sparse) graph G whose vertices are the set of all points $\mathcal{L} \cup \mathcal{U} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Pairs of distant points are assigned an edge with a large weight or no edge at all, whereas close pairs of points are assigned edges with *small* weights, by some rule.
2. Compute the shortest-path (geodesic) graph distance $d_G(\mathbf{x}_i, \mathbf{x}_j)$ for all $\mathbf{x}_i \in \mathcal{L}$ and $\mathbf{x}_j \in \mathcal{U}$.
3. Apply standard supervised learning algorithms using the labeled points and their geodesic distances d_G to the unlabeled ones.

This framework generalizes Bijral et al. (2011), which assumed that $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are vectors in $\mathbb{R}^D$ and the distance function is a power of a p-norm $\|\mathbf{x}_i - \mathbf{x}_j\|_p^q$. Specific graph construction rules include the ϵ-neighborhood cutoff and the symmetric k-NN rule, see for example Alamgir and von Luxburg (2012). The elegance of this framework is that it *decouples* the unsupervised and supervised parts of the learning process. It represents the geometry of the labeled and unlabeled instances by a single metric d_G, subsequently enabling one to directly apply any supervised learning algorithm based on a metric. For classification, a natural choice is the k nearest neighbors algorithm. To classify a point $\mathbf{x}$, we find its k nearest labeled points using the geodesic metric, and compute their majority label. For regression problems, in analogy to the Euclidean Nadaraya-Watson estimator, one may apply kernel smoothing, but base it instead on the geodesic distance. Here, given a kernel $K : \mathbb{R} \to \mathbb{R}$ and a bandwidth h, for any point $\mathbf{x} \in \mathcal{U}$ in the graph G we estimate its response y as an average of labeled samples, weighted by their *geodesic* distance,

$$\hat{y}(\mathbf{x}) = \frac{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{L}} K(d_G(\mathbf{x}_i, \mathbf{x})/h) y_i}{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{L}} K(d_G(\mathbf{x}_i, \mathbf{x})/h)}. \tag{2}$$

2.3 Main contributions and outline

There are two main computational challenges in applying the geodesic nearest neighbor framework described above: (i) constructing the graph, and in particular efficiently finding the nearest neighbors of all the given instances; and (ii) computing the nearest geodesic labeled instances for all unlabeled points. The first challenge is relevant to many graph-based learning methods, and may be addressed by various exact or approximate NN methods. In this paper we focus on the second challenge. Our main contribution, detailed in Section 3 is a novel fast algorithm for computing the k nearest labeled vertices for all the nodes in a graph. As we show both theoretically in Section 4 and empirically in Section 5.3, in typical semi-supervised settings with a large number of unlabeled samples, this algorithm is significantly faster than standard methods.

Our work is motivated in part by the problem of indoor localization using WiFi signals. Applying semi-supervised methods for this problem is a natural choice since (i) the amount of unlabeled data is almost limitless, whereas labeled data is costly to acquire; and (ii) from physical principles, the collected signals lie in a 2 or 3 dimensional manifold non-linearly embedded in a much higher dimensional space. In Section 5 we illustrate the benefits of using unlabeled data for this problem, and the competitive performance of our algorithm in both runtime and statistical accuracy.

3 Fast geodesic nearest-neighbors

Let $G = (V, E)$ be an undirected graph with non-negative edge weights, whose vertex set $V = \mathcal{L} \cup \mathcal{U}$ is the union of the labeled vertices $\mathcal{L}$ and the unlabeled vertices $\mathcal{U}$. How fast can we find, for every vertex in V, its set of k nearest *labeled* vertices?

A straightforward approach to this problem is to first apply the well-known Dijkstra algorithm from each of the $L = |\mathcal{L}|$ labeled points. This yields a matrix of size $L \times |V|$ of all pairwise shortest graph distances $d_G(s, v)$, where $s \in \mathcal{L}$ and $v \in V$. Then, for every vertex $v \in V$, we obtain its k nearest labeled neighbors by finding the smallest k elements of the corresponding column. Recall that the runtime of Dijkstra's algorithm, implemented with a Fibonacci heap is $O(|V| \log |V| + |E|)$, see for example Dasgupta et al. (2006). For small values of k, the second step of finding

the k nearest neighbors is negligible in terms of the overall run time, which is thus dominated by $O\left(L|V|\log|V|+L|E|\right)$.

In the special case $k=1$, where one computes the single nearest labeled vertex to every vertex in a graph, the result is known as the Voronoi diagram of the graph, with the labeled nodes acting as the centers of the Voronoi cells. A fast algorithm for this problem was developed by Erwig (2000).

Algorithm 1 Geodesic k nearest labeled neighbors

Input: An undirected weighted graph $G=(V,E,w)$ and a set of labeled vertices $\mathcal{L}\subseteq V$.
Output: For every $v\in V$ a list $kNN[v]$ with the k nearest labeled vertices to v and their distances.

```
Q ←PriorityQueue(), visited ← φ
for v ∈ V do
  kNN[v] ← Empty-List()
  if v ∈ L then
    insert(Q, key = 0, seed = v, vertex = v)
  end if
end for
while Q ≠ φ do
  (dist, seed, v_0) ← pop-minimum(Q)
  visited ← visited ∪ {(seed, v_0)}
  if length(kNN[v_0]) < k then
    append (dist, seed) to kNN[v_0]
    for all v ∈ neighbors(v_0) do
      if (seed, v) ∉ visited then
        decrease-or-insert(Q, key = dist + w(v_0, v), seed, v)
      end if
    end for
  end if
end while
```

In this section we present Algorithm 1, which for any k, efficiently finds the k geodesic nearest labeled neighbors. As analyzed below, its runtime is bounded by $O\left(\log|V|\cdot\min\{L|V|,k|E|\}+k|E|\right)$, which can be much faster than the naïve approach.

To motivate our algorithm, it is instructive to first briefly recall Dijkstra's shortest path algorithm. Given a graph $G=(V,E)$, and a seed vertex $s\in V$, Dijkstra's algorithm keeps, for every vertex $v\in V$, an upper bound on $d_G(s,v)$, denoted $u[v]$, initialized to 0 if $v=s$ and to ∞ otherwise. At every iteration, the vertex v_0 with the lowest upper bound is visited: For every neighbor v of v_0, if $u[v_0]+w(v_0,v)<u[v]$, then the current upper bound $u[v]$ is lowered. An important invariant of Dijkstra's algorithm is that every time the vertex v_0 with the lowest upper bound is visited, it can be proved that $u[v_0]=d_G(s,v_0)$. The correctness of Dijkstra's algorithm follows easily from this fact.

We now give an informal description of Algorithm 1. The basic idea behind it can be described as running L instances of Dijkstra's algorithm "simultaneously" from all labeled vertices, combined with early stopping whenever k nearest labeled neighbors have been found. In the classic algorithm of Dijkstra, every vertex is visited at most once. Running L independent copies of Dijkstra's algorithm from the L labeled vertices would lead to every vertex in V being visited L times (assuming G is connected). In typical semi-supervised settings, where $L\gg k$, this is unnecessarily slow. In our method, for every vertex we store a list of the first k visits to it from different labeled nodes and their distances. Whenever a vertex is visited k times, we stop any further processing of its neighbors. As we show in Section 4, due to this early stopping, Algorithm 1 is significantly more efficient than the naïve approach of running Dijkstra's algorithm separately from every labeled vertex. Instead of storing pairs $(dist,v)$ as in Dijkstra algorithm, we store triplets $(dist,seed,v)$ in a priority queue Q keyed by $dist$, where $dist$ is an upper bound on $d_G(seed,v)$. Moreover, in the proof of correctness of Algorithm 1, we show that when $(dist,seed,v)$ is popped from the queue, $dist=d_G(seed,v)$.

The proposed algorithm can be implemented using a priority queue based on a Fibonacci heap with the 3 standard operations: insert, pop-minimum and decrease-key. In the code of Algorithm 1, decrease-or-insert stands for decreasing the key of a pair $(seed,v)$ if it already exists in the queue and inserting it if not. The following theorem proves the correctness of Algorithm 1, namely that its

output for every vertex $v \in V$ is indeed the set of its k nearest labeled points, as measured by the geodesic graph distance.

Theorem 3.1. *Let $G = (V, E, w)$ be a graph with non-negative weights. For every vertex $v \in V$ let $\mathcal{L}_v$ denote the set of labeled vertices in the connected component of v and let $\ell_v = \min\{k, |\mathcal{L}_v|\}$. Algorithm 1 stops after a finite number of steps, such that for every $v \in V$ the output list $kNN[v]$ is of the form $kNN[v] = [(d_G(s_1, v), s_1), \ldots, (d_G(s_\ell, v), s_{\ell_v})]$ where $s_1, \ldots, s_{\ell_v}$ are the nearest labeled vertices, sorted by their distance to v.*

The proof appears in the Supplementary. It critically relies on the following property of shortest paths to labeled vertices.

Lemma 3.1. *Let $v \in V$ be a vertex and let s be its j-th nearest labeled vertex. If $s \rightsquigarrow u \rightsquigarrow v$ is a shortest path then $s \in NLV(u, j)$, where $NLV(u, j)$ is the set of j nearest labeled vertices to u.*

We note that Bijral et al. (2011) also proposed a variant of Dijkstra's algorithm. However, their method is an improvement of *single-source* Dijkstra in the setting of a dense graph constructed from points in R^D, whereas our method computes paths from *multiple sources* and applies to any graph.

4 Transductive runtime analysis

Lemma 3.1 explains why Algorithm 1 can stop exploring the neighbors of vertices whose k nearest labeled neighbors were found. As Theorem 4.1 shows, this can lead to dramatic runtime savings.

Theorem 4.1. *Given a graph $G = (V, E)$ with L labeled vertices, the runtime of Algorithm 1 is bounded by $O\left(k|E| + n_p \log |V|\right)$ where n_p is the total number of pop-minimum operations, which satisfies $n_p \leq \min\{L|V|, k|E|\}$.*

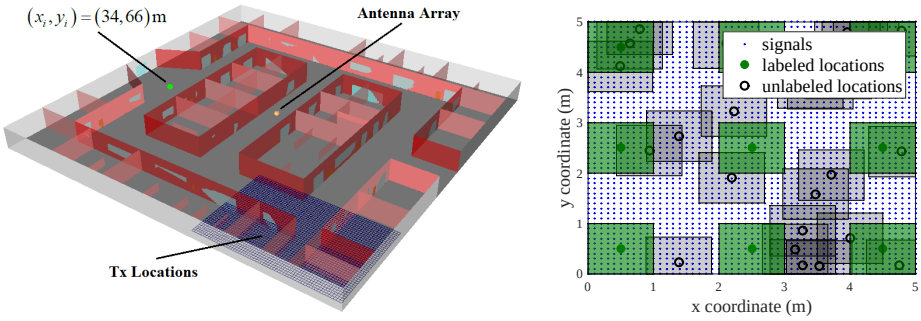
Proof. Recall that in a priority queue based on a Fibonacci heap all operations cost $O(1)$ amortized time except pop-minimum which costs $\log |Q|$. The runtime is dominated by the cost of all pop-minimum operations, plus the total cost of traversing the neighbors of the examined vertices. The latter takes $O(k|E|)$ time. Denote the total number of pop-minimum operations by n_p. We derive two different bounds on n_p. Every time a $(seed, v)$ pair is popped from Q, it is added to the $visited$ set, which prevents future insertions of that pair into Q. Hence, each pair $(seed, v) \in \mathcal{L} \times V$ may be popped at most once from Q, which implies that $n_p \leq L|V|$. In addition, n_p is bounded by the number of insertions into Q. First, there are L insertions during the initialization phase. Then, for each vertex $v_0 \in V$, the "if $length(kNN[v_0]) < k$" clause can hold true at most k times for that vertex. Each time, the neighbors of v_0 are examined and up to $deg(v_0)$ neighbors are inserted into Q. This yields the second bound, $n_p \leq L + k|E| = O(k|E|)$. □

An immediate corollary of Theorem 4.1 is that for a graph $G = (V, E)$ of bounded degree d, the runtime of Algorithm 1 is $O(kd|V| \log |V|)$. In contrast, the runtime of the naïve approach based on multiple Dijkstra runs is $O\left(L|V| \log |V| + Ld|V|\right)$. Comparing these two formulas, Algorithm 1 can provide significant speedups over the naïve approach in typical cases where $L \gg kd$. As we illustrate empirically in Section 5.3, our method is faster by a factor of 100-1000 even on graphs of moderate size.

5 A geodesic SSL approach to indoor localization

One motivation for our work is the problem of estimating the location of a mobile device in a closed environment using its Wi-Fi signature as received by a wireless router. This problem is gaining considerable interest in recent years due to its many potential applications, such as indoor navigation inside large commercial spaces (Liu et al., 2007). In indoor settings, the signal received by the router is a superposition of multiple reflections of the same source. These reflections usually differ in their arrival time and direction. This prevents the use of classic outdoor positioning methods such as triangulation, which require a direct line-of-sight between the transmitting device and the receiver.

One approach to tackle this problem, known as *fingerprinting* in the signal processing community, is based on nearest-neighbor search. First, a labeled set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$ is collected, where the vector $\mathbf{x}_i$ depends on features of a signal transmitted from a known location $y_i \in \mathbb{R}^2$. These features may



(a) (b)

Figure 1: (a) 3D model of a $80 \times 80m \times 5m$ floor. (b) Labeled locations (green circles) were placed on a $4m$ grid, whereas the unlabeled locations (grey circles) were placed at random.

include the signal strength, directions of arrival, time of arrival, etc. The location of a new instance is then estimated via non-parametric regression methods such as nearest neighbor averaging. The resulting localization errors are typically comparable to the distance between nearby labeled points. Therefore, for applications requiring high accuracy, recording and maintaining a suitable labeled data set may be prohibitively expensive. On the other hand, collecting vast amounts of unlabeled data is relatively easy, simply by recording the Wi-Fi signals of devices in the possession of individuals wandering around the venue. Hence, indoor localization seems to be a natural application for semi-supervised methods. Moreover, due to physical principles, the space of feature vectors is parameterized by the 2-dimensional (or 3-dimensional) location. Thus, we expect manifold based methods to perform well in this task.

In this section we extend the supervised Multipath Fingerprinting method of Kupershtein et al. (2013) and Jaffe and Wax (2014) to a semi-supervised method using our proposed geodesic regression approach. As shown below on both simulated and real-data, our geodesic-based SSL method yields a marked improvement in localization accuracy.

5.1 Dataset description and Graph construction

Simulated data: This data consists of 802.11 Wi-Fi signals in an artificial yet realistic environment generated by Kupershtein et al. (2013) using a 3D radio wave propagation software. The environment is an $80m \times 80m$ floor. In its center is a Wi-Fi router with $p = 6$ antennas. See Figure 1a. At various locations $(x, y) \in \mathbb{R}^2$, $N = 8$ consecutive samples of a Wi-Fi packet's (constant) preamble are recorded, at equally spaced time intervals of $50\mu s$. The samples are stored in a complex-valued vector $s_{x,y} \in \mathbb{C}^{pN}$ which we refer to as the *signal* received from location (x, y). The simulated signals were generated on a dense $0.1m$ grid covering the entire area of the floor.

Real data: This data consists of actual 802.11 signals, recorded by a Wi-Fi router with $p = 6$ antennas placed approximately in the middle of a $27m \times 33m$ office, see the supplementary for a schematic. The transmitter was a tablet connected to the router via Wi-Fi. The signal vector of each location (x, y) was sampled $N = 8$ times from every antenna. The transmitter locations were entered manually by the operator.

Graph construction for the simulated and real datasets: The Signal Subspace Projection (SSP) of Kupershtein et al. (2013) and Jaffe and Wax (2014) is used as the fingerprint for localization. It is based on the assumption that signals received from close locations maintain similar properties of differential delays and directions of arrival. Hence, signals originating from nearby locations are contained in a low dimensional subspace. The subspace around each location is used as its signature. Specifically, the signature P_ℓ of a location $\ell = (x, y) \in \mathbb{R}^2$ is computed as follows. First, the covariance matrix of the signals in the proximity of ℓ is computed, using all the signals in the dataset that are inside a $1m$ square centered around ℓ,

$$R_\ell := \sum_{\ell' \in \mathbb{R}^2 : \|\ell - \ell'\|_\infty < 0.5m} s_{\ell'} s_{\ell'}^*$$

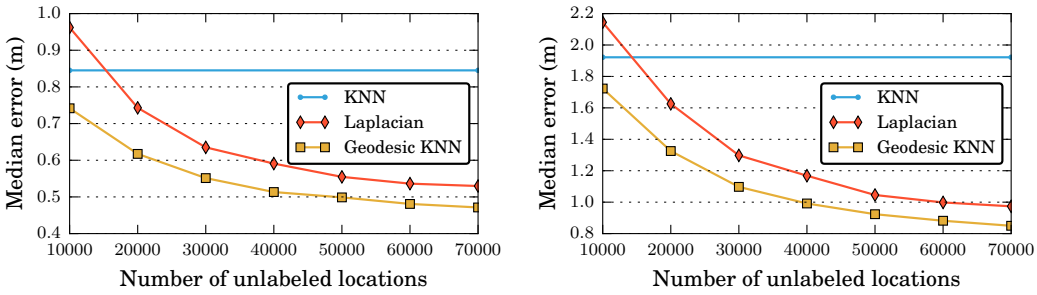


Figure 2: Median localization error vs. number of unlabeled points. The labeled points are placed on a regular grid, with an inter-distance of 2 meters (left panel) or 4 meters (right panel).

where $s_{\ell'}^*$ denotes the Hermitian transpose of the column vector $s_{\ell'} \in \mathbb{C}^{pN}$. Next, we compute the n_{pc} leading eigenvectors of R_ℓ, forming a matrix $V_\ell \in \mathbb{C}^{pN \times n_{pc}}$. The SSP signature is the projection matrix onto the space spanned by these eigenvectors, $P_\ell := V_\ell V_\ell^*$. In our experiments, we picked $n_{pc} = 10$, though other choices in the range $\{8, \ldots, 12\}$ gave results that are almost as good. The distance between pairs of locations is defined as the Frobenius norm of the difference of their projection matrices, $d_{i,j} := \|P_{\ell_\mathrm{i}} - P_{\ell_\mathrm{j}}\|_F$.

For the simulated dataset, to mimic how a real-world semi-supervised localization system might work, we generated the labeled locations on a regular square grid, whereas the unlabeled locations were randomly distributed over the entire area of the floor (see Fig. 1b). For the real dataset, due to phyiscal constraints, labeled and unlabeled points were not on a regular grid (see appendix for details). Then, we created a symmetric kNN graph by connecting every point x_i with its k_G closest neighbors, with corresponding weights given by $w_{i,j} = 1 + \epsilon d_{i,j}$. Here $\epsilon > 0$ is a small constant which gives preference to paths with smaller $d_{i,j}$. We took $k_G = 4$ neighbors for each vertex, which was experimentally found to be the best choice, both for the geodesic k-NN and for the Laplacian eigenvector regression. However, choosing $k_G \in \{3, 5, 6\}$ gave results that were almost as good.

To estimate the location of an unlabeled point, we considered Geodesic k-NN with exponential decaying weights such that the weight of the i-th neighbor is proportional to $1/2^i$. Experimentally, these exponential weights performed much better than uniform. We set $k = 7$ to obtain reasonable runtimes. Note that due to the fast exponential decay of weights, larger choices of k are expected to yield similar results. For the Laplacian eigenvector regression, we optimized over the number of eigenvectors by taking the best outcome after repeating the experiment with $10\%, 20\%, 30\%, 40\%$ and 50% of the labeled points.

5.2 Results

First, we compare our semi-supervised geodesic k-NN regressor to the Laplacian eigenvector regression method of Belkin and Niyogi (2004) on the artificial data set. As a baseline we applied standard k-NN, using only the labeled samples. For each number of unlabeled samples, we chose the value $k \in \{1, 2, 3\}$ that gave the best accuracy. Figure 2 shows the median localization error as a function of the number of unlabeled locations, where the labeled locations are on a fixed grid.

These results showcase the advantage of incorporating the unlabeled data in the solution to the localization problem. As expected, the improvement achieved by the geodesic k-NN increases with the number of unlabeled locations, attaining a reduction of roughly 50% in the localization error. In addition, not only is the geodesic k-NN regression more accurate than the Laplacian eigenbasis regressor, in terms of runtime it is also much faster, see Table 2.

Next, Table 1 shows the mean localization error on the real Wi-Fi localization data set for different densities of labeled points. In this benchmarks we have tested symmetric k-NN graphs with $k_G \in \{4, \ldots, 19\}$ constructed as described in the previous section and used the best result obtained for each algorithm. For the geodesic k-NN, the graph distances were set to the Frobenius distances of projection matrices, as in the previous section. We tested $k \in \{1, 2, 3\}$, however $k = 1$ gave the best results in all cases. This may be due to the low number of labeled samples, as we have

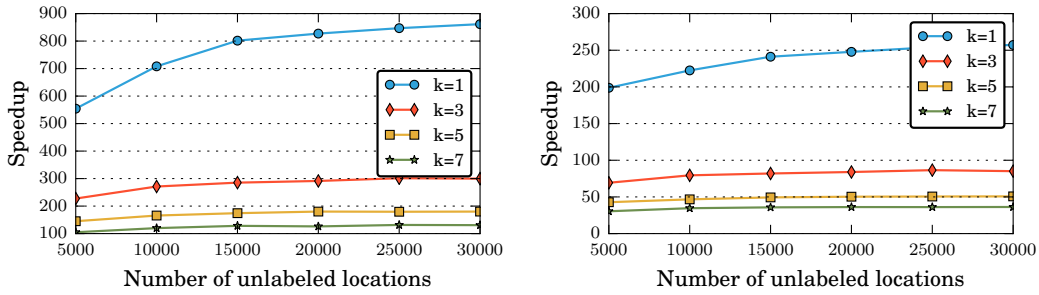


Figure 3: Relative speedup of Algorithm 1 compared to the naïve approach of running Dijkstra's algorithm from each labeled point. (left panel) A total of 1600 labeled locations are placed on a 2m square grid. (right panel) A total of 400 labeled locations are on a 4m grid.

observed similar behavior on the simulated data set when the number of labeled samples is low. For the Laplacian eigenvector regression algorithm, we used binary weights and chose the number of eigenvectors by testing the entire range of $1, \ldots, \#labeled$. In all cases, our geodesic approach yielded more accurate results.

Table 1: Mean accuracy of semi-supervised localization on the real data set

Labeled point grid size	Labeled points	NN	Geodesic NN	Laplacian
1.5 meters	73	1.49 meters	**1.11** meters	1.36 meters
2.0 meters	48	2.27 meters	**1.49** meters	1.65 meters
3 meters	23	3.41 meters	**2.41** meters	2.79 meters

5.3 Runtime comparison

We empirically compare the runtime of our approach to the naïve method of running Dijkstra's algorithm from each of the labeled points. To make the comparison meaningful we implemented both algorithms in Python, using a similar programming style and the same heap data structure. Figure 3 shows the relative speedup of Algorithm 1 compared to multiple Dijkstra runs. In accordance to the theoretical analysis of Section 4, the speedup is roughly proportional to $1/k$.

Table 2: Runtime of Geodesic 7-NN vs. time to compute Laplacian eigenvectors

#unlabeled	Geodesic 7-NN	Laplacian
1000	2.3 seconds	7.6 seconds
10000	7 seconds	195 seconds
100000	56 seconds	114 minutes

Table 2 compares the runtime of the geodesic k-NN method to computing eigenvectors of the Laplacian matrix, using the simulated indoor localization data set with labeled locations every $2m$. The number of eigenvectors was chosen to be 320, which is equal to 20% of the number of labeled points. Computing the geodesic nearest neighbors via Algorithm 1 is several orders of magnitude faster than computing the eigenvectors. This is despite the fact that the eigenvector computation is performed using the highly optimized Intel® Math Kernel Library whereas the geodesic nearest neighbor computation uses a simple Python implementation. We expect an efficient implementation of geodesic k-NN to be at least 10 times faster.

References

Alamgir, M. and von Luxburg, U. (2012). Shortest path distance in random k-nn graphs. *ICML 2012*.

Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396.

Belkin, M. and Niyogi, P. (2004). Semi-Supervised Learning on Riemannian Manifolds. *Machine Learning*, 56:209–239.

Bijral, A. S., Ratliff, N., and Srebro, N. (2011). Semi-supervised Learning with Density Based Distances. In *27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*.

Blum, A. and Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts. In *ICML*.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT press.

Chapelle, O. and Zien, A. (2005). Semi-Supervised Classification by Low Density Separation. In *AISTATS*.

Dasgupta, S., Papadimitriou, C. H., and Vazirani, U. (2006). *Algorithms*. McGraw-Hill, Inc.

Delalleau, O., Bengio, Y., and Le Roux, N. (2005). Efficient non-parametric function induction in semi-supervised learning. In *AISTATS*.

Erwig, M. (2000). The graph Voronoi diagram with applications. *Networks*, 36:156–163.

Gavish, M., Nadler, B., and Coifman, R. R. (2010). Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *ICML*.

Herbster, M., Pontil, M., and Rojas-Galeano, S. (2009). Fast Prediction on a Tree. In *NIPS*.

Jaffe, A. and Wax, M. (2014). Single-Site Localization via Maximum Discrimination Multipath Fingerprinting. *IEEE Transactions on Signal Processing*, 62(7):1718–1728.

Kupershtein, E., Wax, M., and Cohen, I. (2013). Single-site emitter localization via multipath fingerprinting. *IEEE Transactions on Signal Processing*, 61(1):10–21.

Liang, F., Mukherjee, S., and West, M. (2007). The Use of Unlabeled Data in Predictive Modeling. *Statistical Science*, 22(2):189–205.

Liu, H., Darabi, H., Banerjee, P., and Liu, J. (2007). Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man and Cybernetics*, 37(6):1067–1080.

Liu, W., He, J., and Chang, S.-F. (2010). Large graph construction for scalable semi-supervised learning. In *ICML*.

Rigollet, P. (2007). Generalization error bounds in semi-supervised classification under the cluster assumption. *JMLR*, 8:1369–1392.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–6.

Singh, A., Nowak, R., and Zhu, X. (2009). Unlabeled data: Now it helps, now it doesn't. *NIPS*.

Subramanya, A. and Talukdar, P. P. (2014). Graph-Based Semi-Supervised Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(4):1–125.

Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–23.

Vitale, F., Cesa-Bianchi, N., Gentile, C., and Zappella, G. (2013). Random spanning trees and the prediction of weighted graphs. *JMLR*, 14(1):1251–1284.

Zhang, Y.-M., Huang, K., Geng, G.-G., and Liu, C.-L. (2015). MTC: A Fast and Robust Graph-Based Transductive Learning Method. *IEEE Trans. Neural Net. Learning Sys.*, 26(9):1979–1991.

Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. *ICML*.

Zhu, X. and Goldberg, A. B. (2009). *Introduction to semi-supervised learning*. Morgan & Claypool Publishers.