Semi-Supervised Learning

EN.553.738 High Dimensional Approximation, Probability, and Statistical Learning

Ashwin De Silva, Jeremy Welland, Sai Koukuntla, Zhenghan Fang

Department of Biomedical Engineering
Johns Hopkins University

May 17, 2023

Problem statement

Let $x_1, \ldots, x_n \in \mathcal{X} \subseteq \mathbb{R}^d$ be a dataset where only the first $m << n$ samples have labels $y_i = f(x_i) \in \mathcal{Y} = \{1, \ldots, C\}$. The function $f : \mathcal{X} \mapsto \mathcal{Y}$ assigns class labels to $x_i, i = 1, \ldots, m$.
In semi-supervised learning (SSL), we wish to learn a function $\hat{f} : \mathcal{X} \mapsto \mathcal{Y}$ using all the n samples such that $\hat{f}$ may achieve a low misclassification error on the unlabeled samples:

$$err(\hat{f}) = \frac{1}{n-m} \sum_{i=m+1}^{n} \mathbb{1}\{\hat{f}(x_i) \neq f(x_i)\}.$$

The objective is to exploit all the samples to understand the underlying geometry of the data, and leverage this information to learn an appropriate classifier $\hat{f}$ using a few labeled samples.

Laplacian Eigenmaps-based SSL

Overview

- The Laplacian eigenmaps SSL algorithm[1] asssumes that classification functions f are defined over a l-dimensional submanifold $\mathcal{M}$ instead of the total ambient space $\mathbb{R}^d$.
- f is assumed to be smooth over $\mathcal{M}$.
- First, the unlabeled data is used to recover a suitable basis for the function space.
- Then, a classifier is constructed using labeled samples over space spanned by this basis.

[1] Belkin and Niyogi. "Using manifold stucture for partially labeled classification."

Laplacian Eigenmaps-based SSL

Algorithm

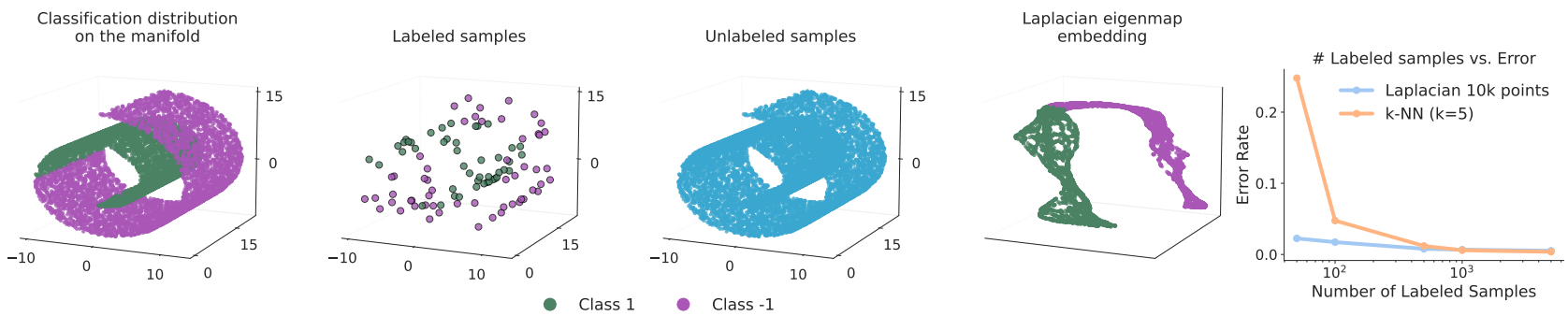
1. Construct the adjacency matrix $W_{n\times n}$ of graph G

$$W_{ij} = \begin{cases} 1, & \text{if } x_i \in k\text{-nn}(x_j) \text{ or } x_j \in k\text{-nn}(x_i), \\ 0, & \text{otherwise.} \end{cases}$$

2. Compute the first p eigenvectors $e_1, \ldots, e_p$ of the graph Laplacian L s.t. $\lambda_1 = 0 \leq \cdots \leq \lambda_p$ (Note: $L = D - W$ where $D_{ii} = \sum_j W_{ij}$).

3. Build the linear classifier by solving the following ordinary least squares problem:

$$a^* = \underset{a=(a_1,\ldots,a_p)}{\operatorname{argmin}} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{p} a_j e_j(i) \right)^2 = U^\dagger y; \;\; U_{ij} = e_j(i)$$



Laplacian Eigenmaps-based SSL

Theoretical Foundations

- $\mathcal{L}^2(\mathcal{M})$ is a Hilbert space with $\langle f, g \rangle = \int_{\mathcal{M}} fg d\mu$ and $\|f\|^2 = \int_{\mathcal{M}} |f|^2 d\mu$
- Eigenfunctions $\{e_j\}_{j=1}^{\infty}$ of Laplace operator Δ on $\mathcal{M}$ form a natural basis for $\mathcal{L}^2(\mathcal{M})$ i.e. $f(x) = \sum_{j=1}^{\infty} a_j e_j(x)$ for any $f \in \mathcal{L}^2(\mathcal{M})$
- A smoothness measure on $\mathcal{M}$:

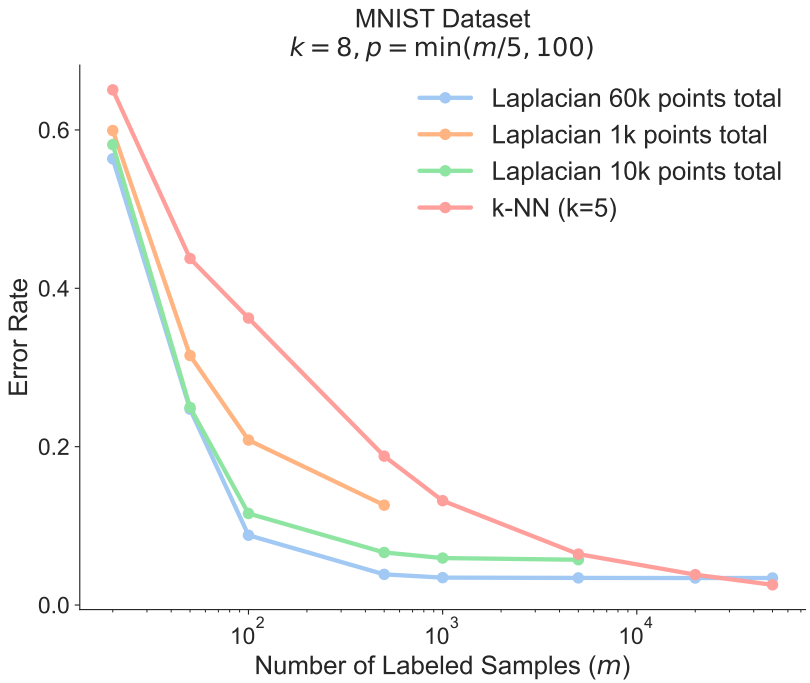
$$S(f) := \int_{\mathcal{M}} \|\nabla f\|^2 d\mu = \langle \Delta f, f \rangle$$

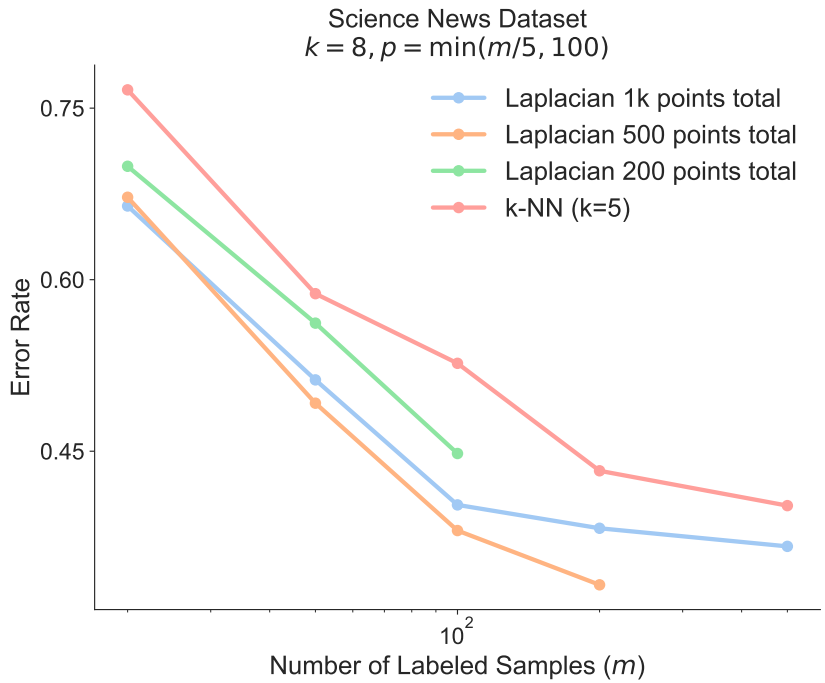
- $f = \sum_{j=1}^{\infty} a_j e_j \implies S(f) = \sum_j \lambda_j a_j^2$ i.e. smoothness is controlled by eigenvalues of Δ.
- Thus, a smooth approximation $f(x) \approx \sum_{j=1}^{p} a_j e_j(x)$ is built by minimizing $\|f(x) - \sum_{j=1}^{p} a_j e_j(x)\|^2$ w.r.t $(a_1, \ldots, a_p)$.
- Graph counterpart: G models the manifold $\mathcal{M}$, L approximates Δ
- For a function f on a graph G,

$$S_G(f) := \sum_{i,j} W_{ij}(f_i - f_j)^2 = f^\top L f$$

Laplacian Eigenmaps-based SSL

Experimental Results (Real Data)

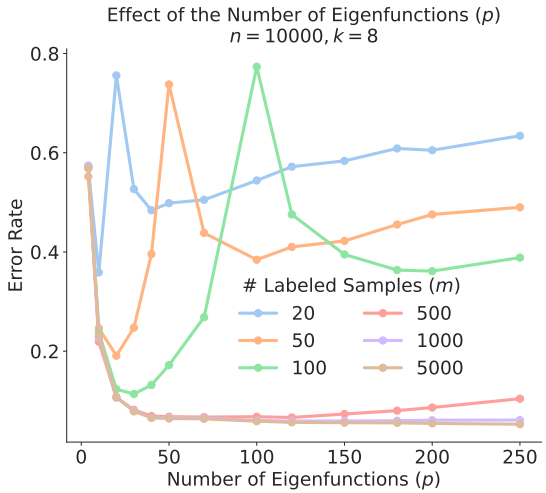


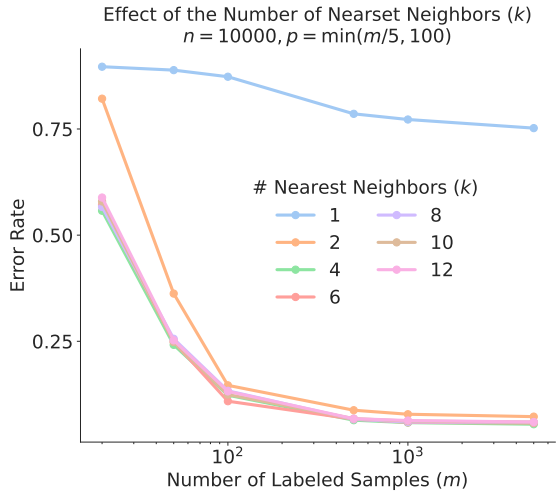


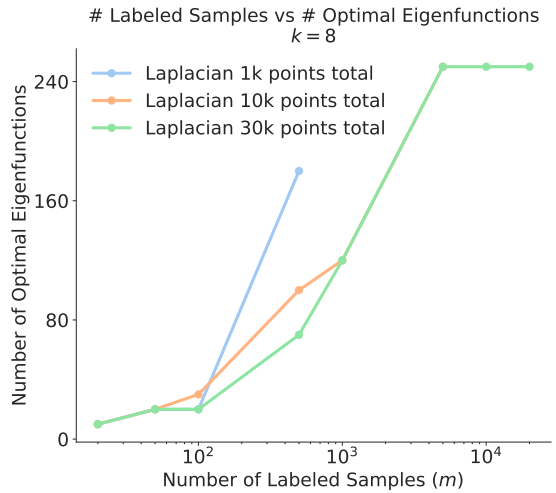
- The algorithm significantly does better than k-NN with at low amounts of labeled samples.
- Performance improves when we have a larger amount of unlabeled samples.

Laplacian Eigenmaps-based SSL

Experimental Results (Algorithmic Analysis)







- Nature of p vs. error rate curves depends a lot on the number of labeled samples (can be explained by the bias-variance tradeoff and the double descent phenomenon).
- Error bound[2] for the algorithm in the infinite limit of unlabeled data:

$$\mathbb{E}_{x\sim\mu}\left(f(x)-\hat{f}(x)\right)^2 \leq \frac{S(f)}{\lambda_{p+1}} + \frac{Cp\log m}{m},$$

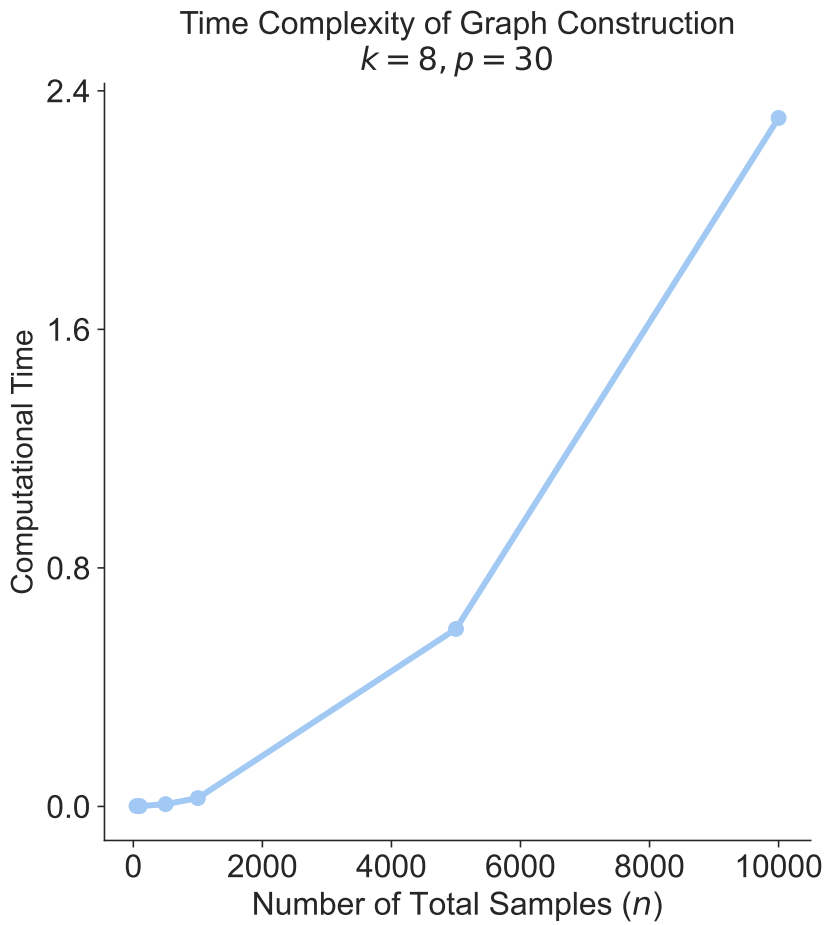
- Based on this bound, we have

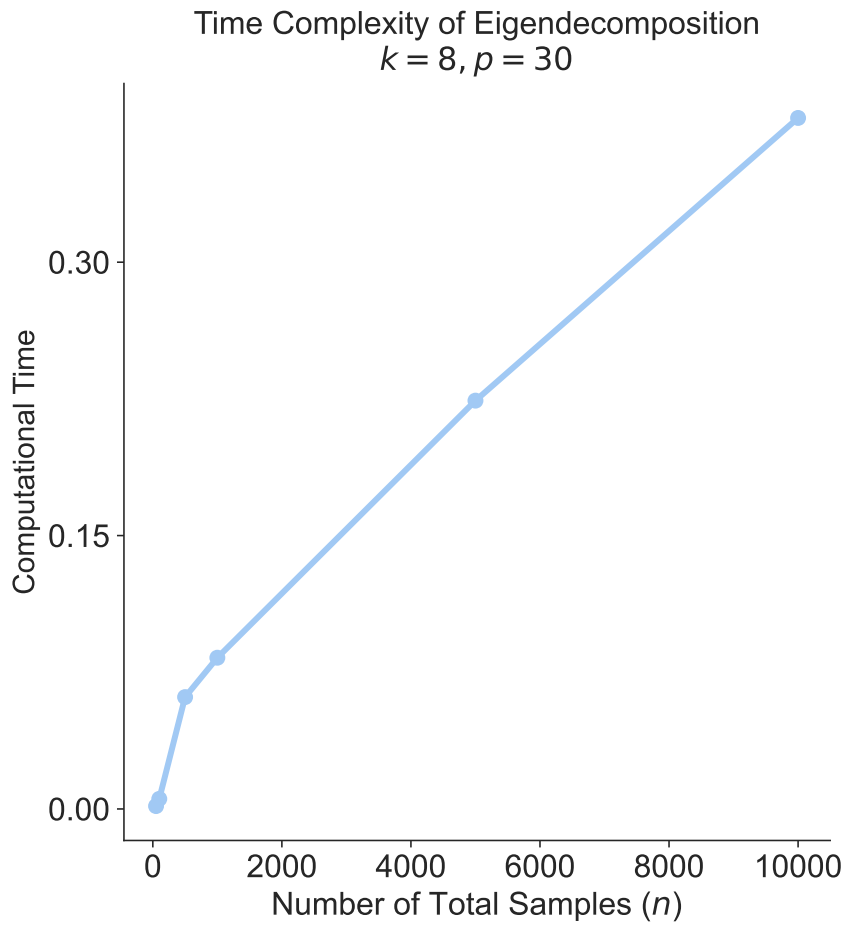
$$p^* = O\left((m/\log m)^{\frac{l}{l+2}}\right) \text{ at } err^*(\hat{f}) = O\left((m/\log m)^{-\frac{2}{l+2}}\right)$$

[2] Zhou, Srebro, "Error Analysis of Laplacian Eigenmaps for Semi-supervised Learning"

Laplacian Eigenmaps-based SSL

Experimental Results (Time Complexity)





It is evident that the time-complexity of the algorithm is $O(n)$.

Laplacian Eigenmaps-based SSL

Limitations

- Manifold underlying the data could be non-smooth or contain a boundary
- Data sampled non-uniformly from the manifold (the algorithm implicitly assumes that the data is uniformly sampled)
- Label noise and inconsistencies present in the datasets
- Large scale datasets (difficult to compute eigenvectors)
- Challenging datasets ($n << d$, data contaminations, manifold not informing the downstream task well)

Regularization on Graph with Function-Adapted Kernels[3] - Eigenfunction-based SSL

Overview

- A graph-based semi-supervised learning method
- View the SSL problem as function approximation/smoothing on graph -> can take advantage of harmonic analysis and diffusion tools for non-Euclidean geometry
- Use non-binary weight matrix with self-tuning normalization instead of binary edges from kNN
- Modify the original geometry of data with function-adapted kernel to facilitate function smoothing/approximation

[3] Szlam, A.D., Maggioni, M. and Coifman, R.R., 2008. Regularization on graphs with function-adapted diffusion processes. Journal of Machine Learning Research, 9(8).

Weighted Graph with Self-tuning Normalization

- Local similarity between two data points x and y:

$$W_\sigma(x, y) = h\left(\frac{\rho(x, y)^2}{\sigma}\right) \tag{1}$$

- Self-tuning normalization: scale the distances at each point so that the j-th nearest neighbor has distance 1:

$$\rho_x(z, z') = \rho(z, z')/\rho(x, x_m)$$

- The Self-tuning Weight matrix:

$$W_\sigma(x, y) = h\left(\frac{\rho_x(x, y)\rho_y(x, y)}{\sigma}\right) \tag{2}$$

Function-adapted Kernel

Motivations:

- Function f may be not smooth with respect to the given geometry of data (e.g., from kNN based on $\ell 2$ distance)
- Geometry can be modified to make f more smooth on the new geometry, and more amenable to harmonic analysis
- The structure of f can be exploited when constructing the graph (e.g., points on the same level set should have high similarity)

Definition of function-adapted kernel:

$$W^f(x,y) = \exp\left(-\frac{\|x-y\|^2}{\sigma_1} - \frac{\|\tilde{f}(x)-\tilde{f}(y)\|^2}{\sigma_2}\right) \tag{3}$$

- $\tilde{f}$: some estimate for the true function to be approximated f
- Incorporate the geometry of f in the construction of weight matrix

Graph-based SSL

- Eigenfunction-based: least square regression
- Diffusion-based

MNIST Results: Effect of Number of Eigenfunctions, Non-adapted

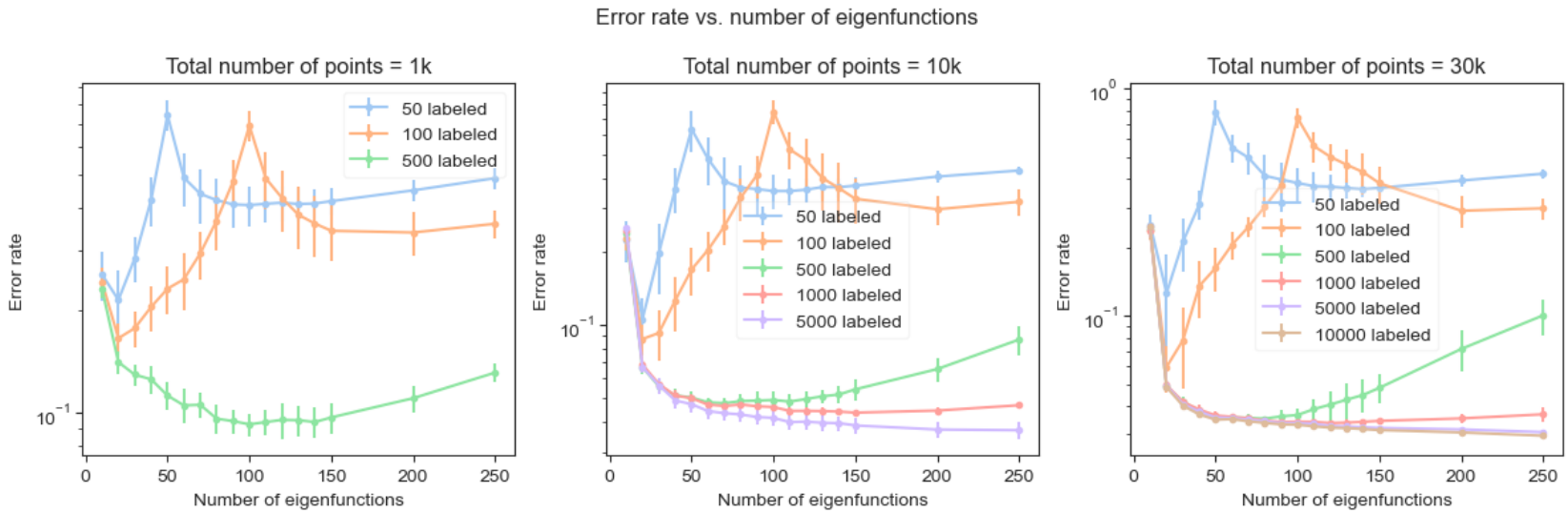


Figure: Error rate vs. number of eigenfunctions.

- An appropriate number of eigenfunctions is needed for best performance.
- The best number of eigenfunctions differs for different numbers of labeled points.

MNIST Results: Best Number of Eigenfunctions, Non-adapted

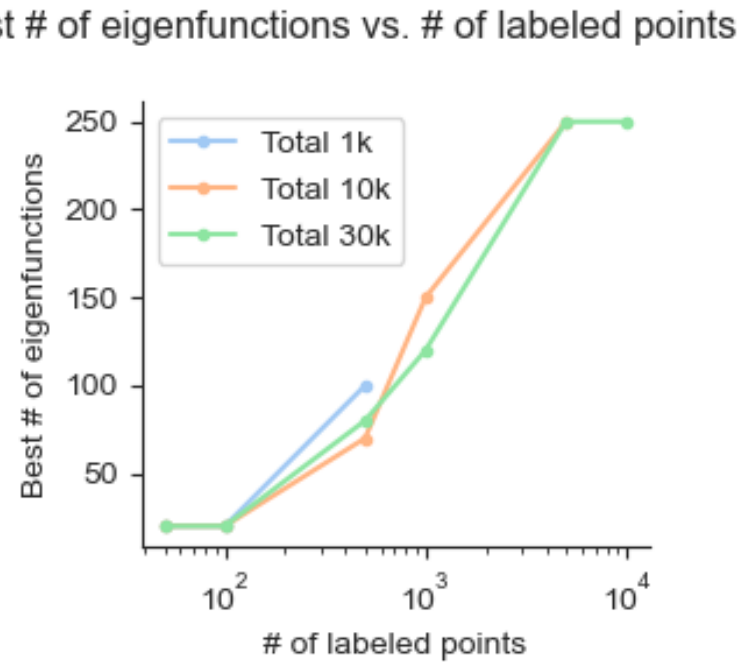


Figure: Best number of eigenfunctions as the number of labeled points scales up.

- The best number of eigenfunctions scales up with number of labeled points
- The best number of eigenfunctions does not differ by total data size.

Science News Results: Effect of Number of Eigenfunctions, Non-adapted

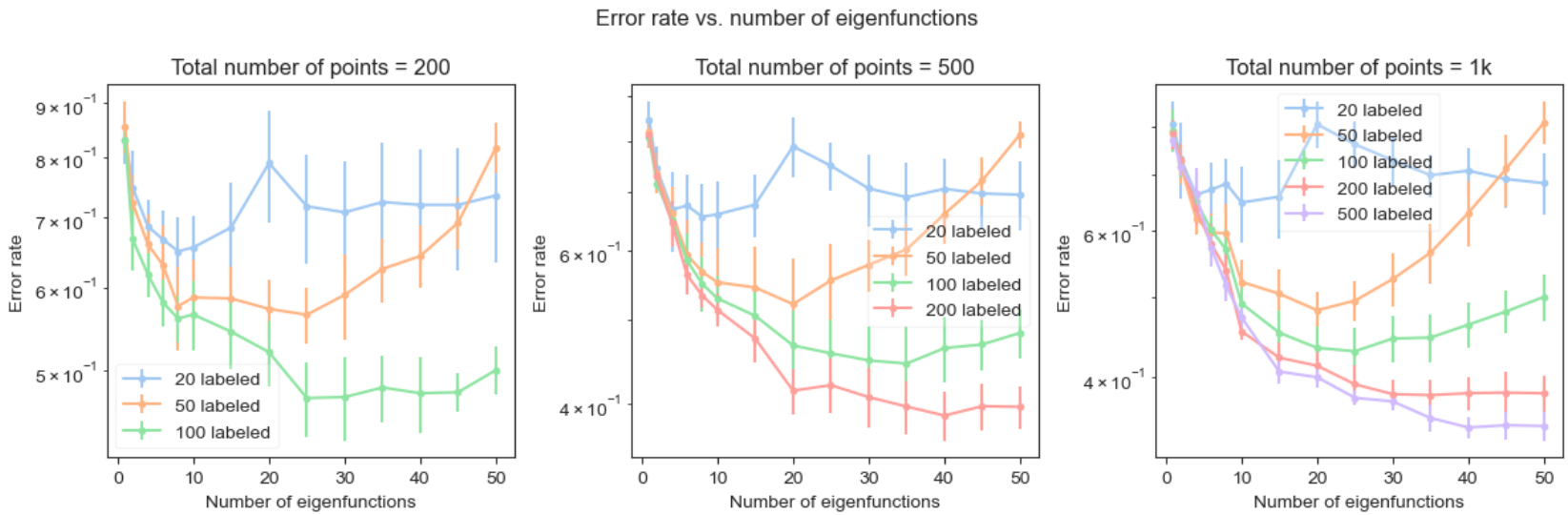


Figure: Error rate vs. number of eigenfunctions.

Similar observations to MNIST dataset.

Science News Results: Best Number of Eigenfunctions, Non-adapted

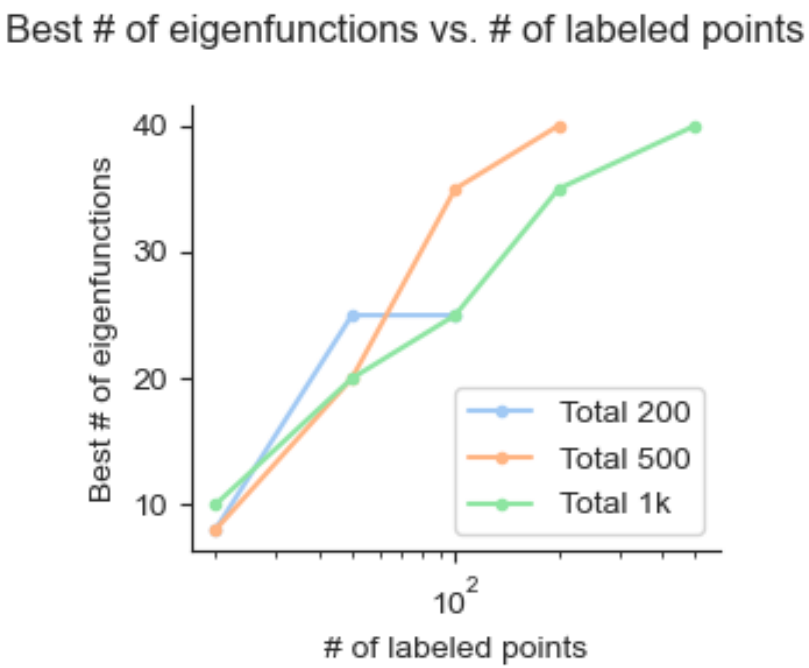
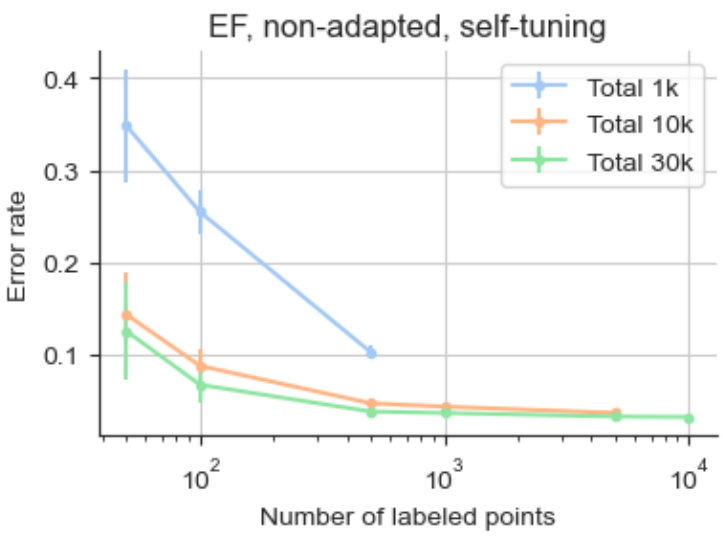
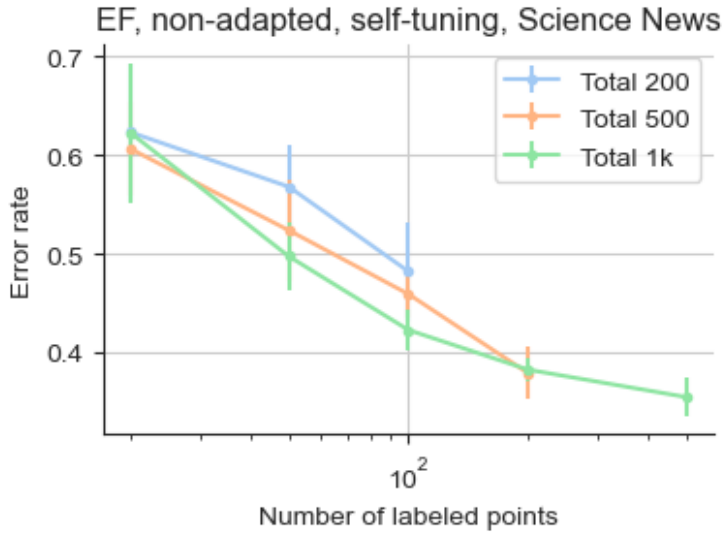


Figure: Best number of eigenfunctions as the number of labeled points scales up.

Error rate vs. Number of Labeled Points, Non-adapted



(a) MNIST.



(b) Science News.

Figure: The classification error rate of eigenfunction-based SSL using self-tuning weight matrix with non-adapted kernel on two datasets.

- Error rates decrease with more labeled samples.
- Performance improves with larger amounts of unlabeled samples.
- Performance is slightly better than the binary graph counterpart.

Results of Function-Adapted Kernel

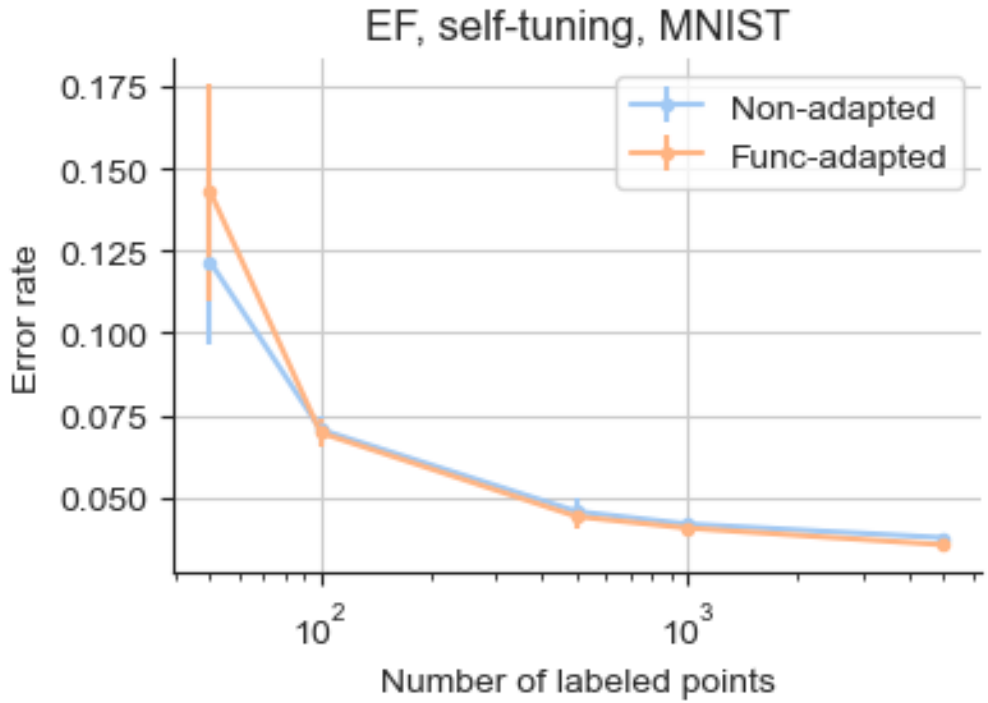


Figure: Results of function-adapted vs. non-adapted kernel with eigenfunction-based SSL using 10k total points in MNIST.

- Function-adapted kernel is helpful given a sufficient amount of labeled data.
- Improvement by FA is greater with more labeled data.
- With extremely small labeled data, FA performs worse.

Regularization on Graph with Function-Adapted Kernels[4] - Diffusion-based SSL

Overview

- Data lies on a low-D manifold that can be approximated by a graph W
- Assume that class labels at nodes constitute a (relatively) smooth function f on the graph
- Can estimate unknown labels by using mollifiers to propagate known labels through the graph
- If f is not smooth on W, function adaptation can modify W so that f is as smooth as possible.

[4] Szlam, A.D., Maggioni, M. and Coifman, R.R., 2008. Regularization on graphs with function-adapted diffusion processes. Journal of Machine Learning Research, 9(8).

Diffusion through Heat Equation

- Heat equation is a mollifier with a simple analog on graphs:

$$f_{t+1} = Kf_t, \tag{4}$$

  where f_t are the estimated labels at step t and K is a normalized version of the weight matrix W
- Can also be interpreted as random walks on the graph, K specifies how nodes are connected
- We take K to be the self-tuning weight matrix defined before

Diffusion-based Classifiers

Two similar classifiers:

- Kernel smoothing (KS): known labels specify initial condition f_0, allow free heat diffusion for N iterations
- Harmonic Classifier (HC): known labels specify initial condition f_0 and are reintroduced at each of N iterations of the heat equation.

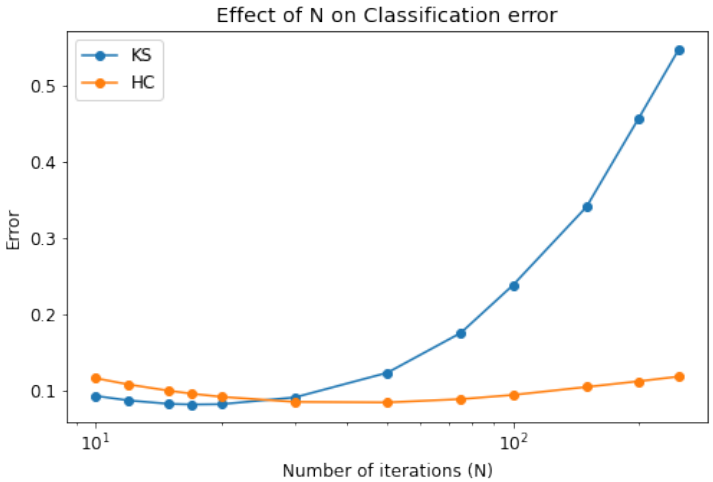
Function-adapted versions (FAKS and FAHC) :

- Calculate steady-state predictions using HC with $N = 250$:

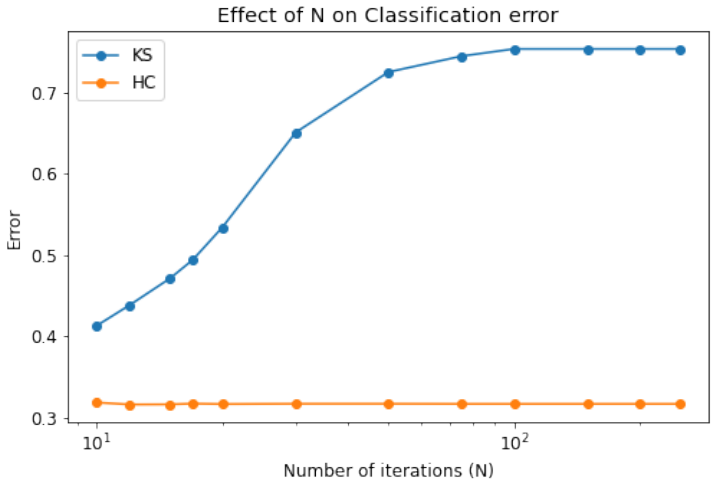
$$\tilde{f} = \frac{f^i_{250}(x)}{\sum_i f^i_{250}(x)}, \tag{5}$$

  where i indexes classes and $f^i_{250}(x)$ is the function value at node x after 250 iterations
- Use estimated class probability as additional features to construct a new graph

Effect of Number of Iterations



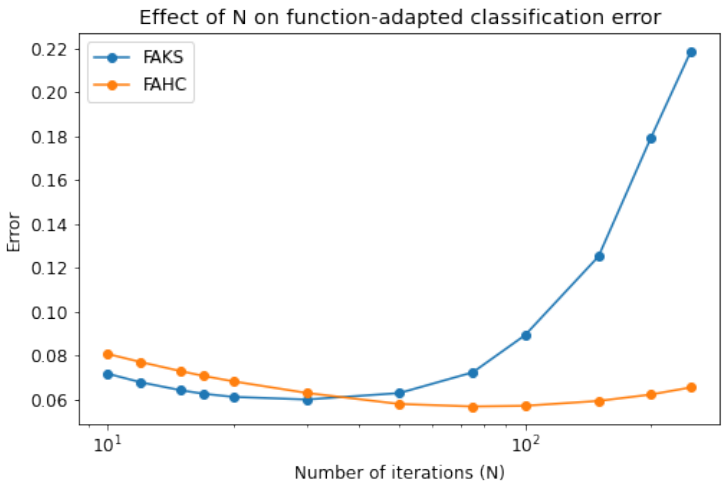
(a) MNIST.



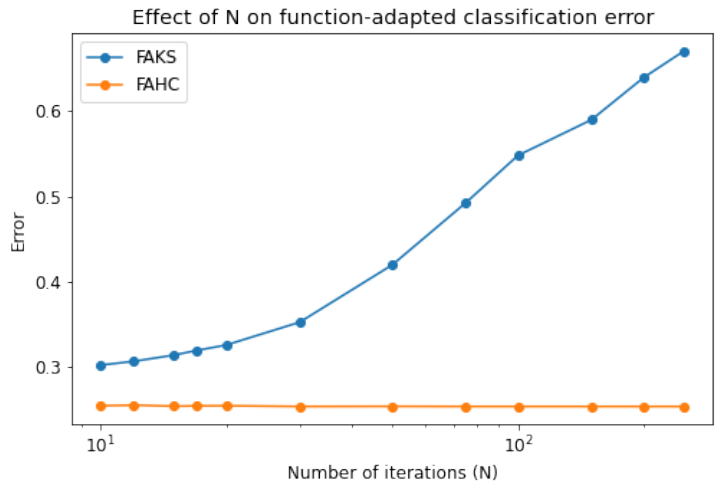
(b) Science News.

Figure: The effect of number of diffusion iterations N on the classification error rates of kernel smoothing and harmonic classifiers.

Effect of Number of Iterations (FA)



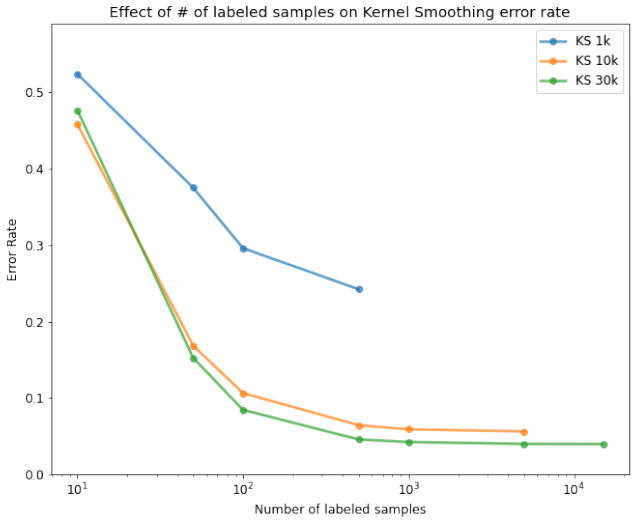
(a) MNIST.



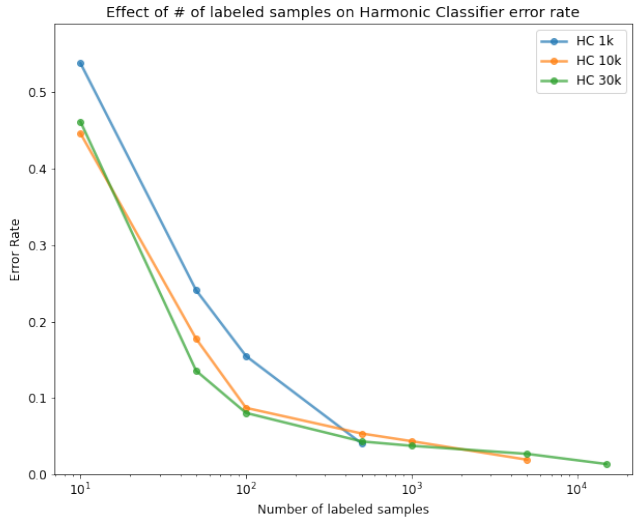
(b) Science News.

Figure: The effect of number of diffusion iterations N on the classification error rates of function-adapted kernel smoothing and harmonic classifiers.

Error rate vs. Number of Labeled Points, Non-adapted, MNIST



(a) KS.

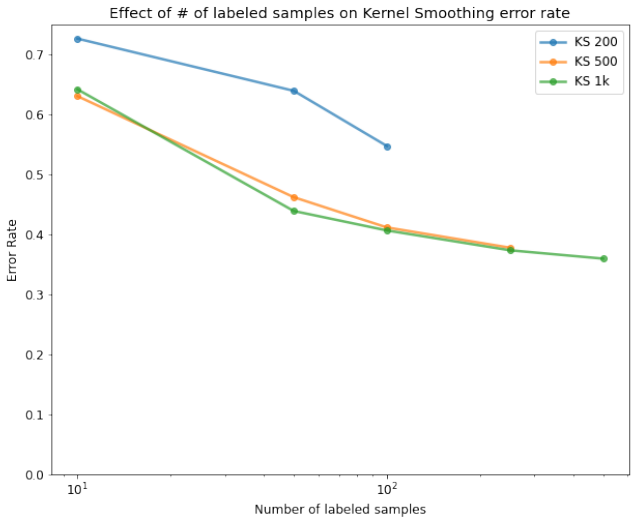


(b) HC.

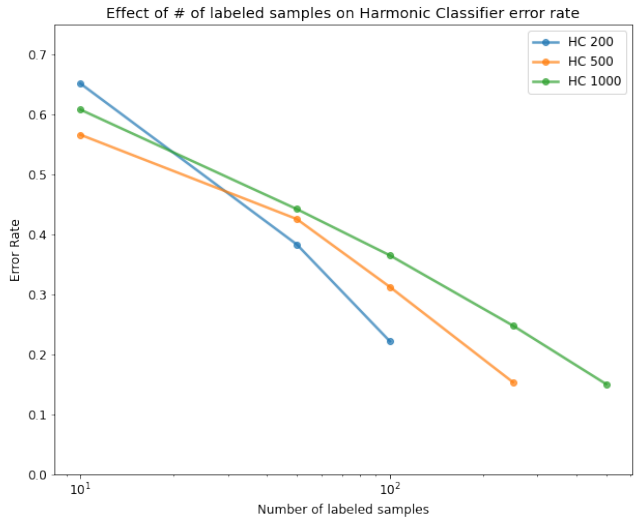
Figure: The classification error rate of non-adapted diffusion methods on MNIST.

- Similar trends as eigenfunction SSL.
- Performance is better than eigenfunction-based methods.
- HC is slightly better than KS.

Error rate vs. Number of Labeled Points, Non-adapted, Science News



(a) KS.

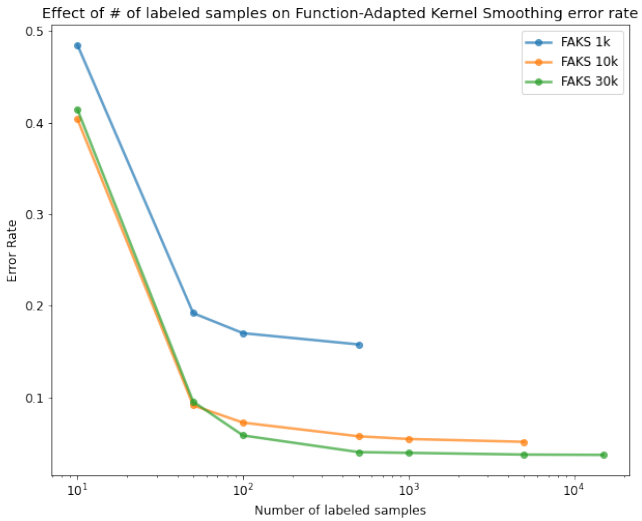


(b) HC.

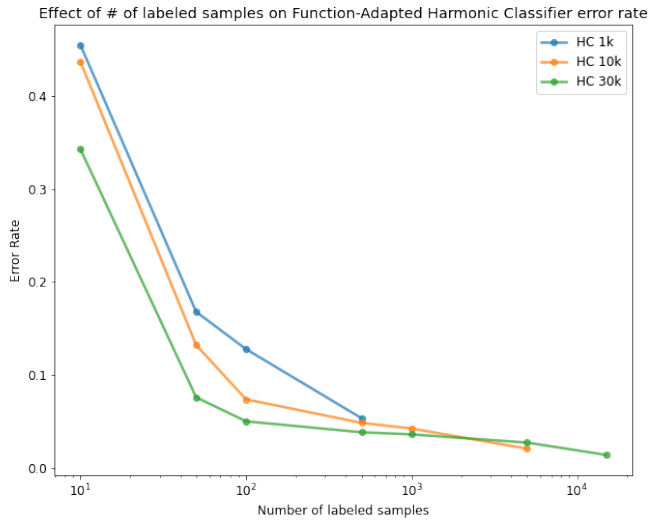
Figure: The classification error rate of non-adapted diffusion methods on Science News.

- Performance is worse than on MNIST – not enough samples?

Error rate vs. Number of Labeled Points, Function-adapted, MNIST



(a) KS.

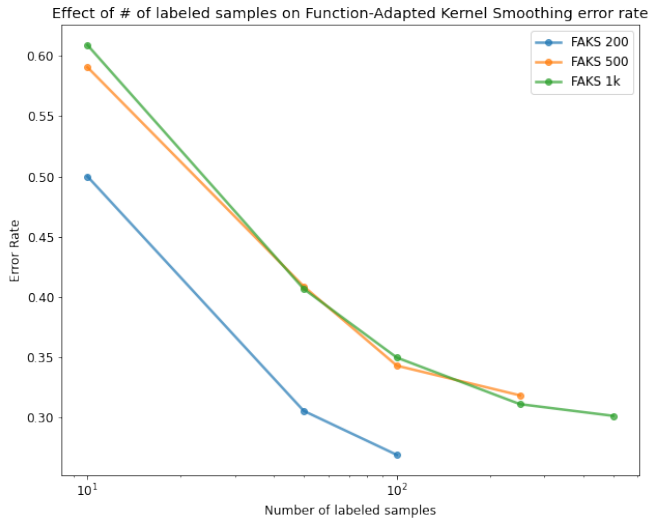


(b) HC.

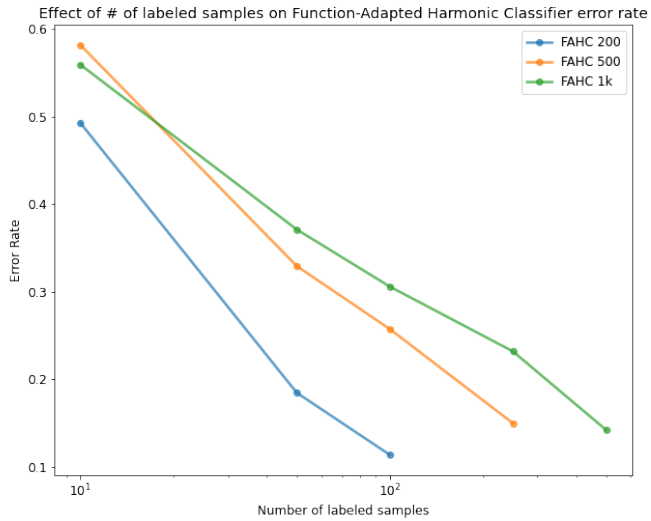
Figure: The classification error rate of function-adapted diffusion methods on MNIST.

- Similar trends as non-adapted
- Function-adapted classifiers perform better than non-adapted

Error rate vs. Number of Labeled Points, Function-adapted, Science News



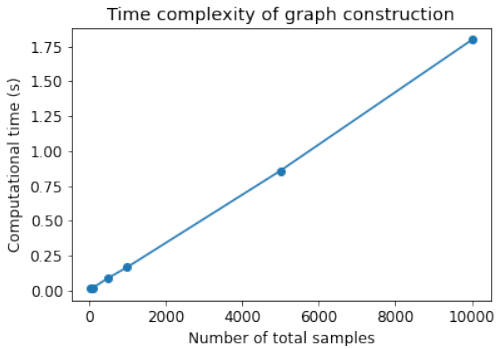
(a) KS.



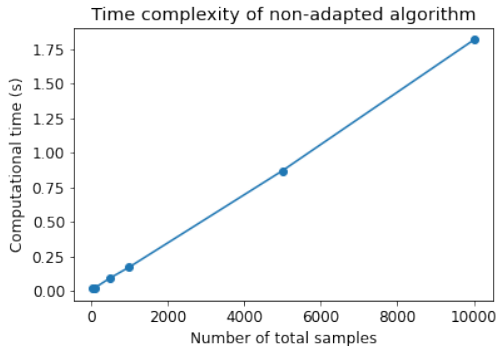
(b) HC.

Figure: The classification error rate of function-adapted diffusion methods on Science News.

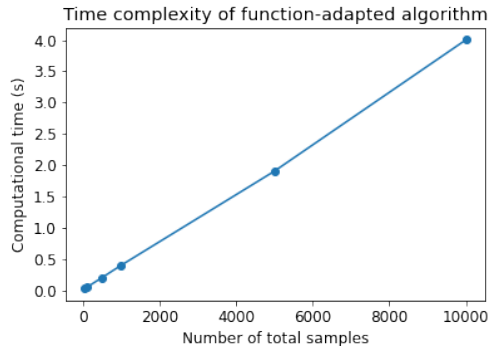
Time Complexity



(a) Graph construction.



(b) Non-adapted runtime.



(c) FA runtime.

Figure: Runtime complexity for diffusion-based SSL.

- Nearest-neighbor search and entire algorithm are linear in total points
- Graph construction dominates runtime

Poisson Learning

Overview

- Poisson Learning, published in 2020, attempts to solve the problem of SSL in *very low label rate* environments.
- Laplacian-based learning performs well with many labels, but degrades rapidly at low label rates (<50% performance on MNIST with one label per class).
- Enter Poisson Learning: still uses the graph Laplacian, but alters boundary conditions.

Poisson Learning

The Poisson Equation

- In Poisson learning, the objective is to solve the following Poisson equation:

$$\mathcal{L}u(x_i) = \sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} \text{ for } i = 1, ..., n \tag{6}$$

  satisfying $\sum_{i=1}^{n} d_i u(x_i) = 0$, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise, and $\bar{y} = \frac{1}{m}\sum_{j=1}^{m} y_j$ is the average label vector.
- This is a graph Poisson equation, with the labeled points corresponding to "sources" in the equation to be solved.

Poisson Learning

Random Walk Interpretation

The random walk interpretation of Poisson learning is as follows: suppose at time 0 the labeled vertices are released to walk around the graph, defining $x \in X$ and let $X_0^x, X_1^x, \ldots$ be a random walk on X starting at $X_0^x = x$ with transition probabilities

$$\mathbb{P}(X_k^x = x_j | X_{k-1}^x = x_i) = d_i^{-1} w_{ij} \tag{7}$$

We are interested in a random walk where the labeled vertices explore the graph, and each time a labeled vertex "visits" an unlabeled vertex, its label information is recorded.

Poisson Learning

Random Walk Interpretation

We are interested only in the short-time behavior of the random walk (as $T \to \infty$ we are capturing only the average of the labels). Thus, we subtract off the tail of the distribution (i.e. the average of the labels, $\bar{y}$), and we further normalize by the degree of the vertex, d_i, yielding the following expression:

$$u_T(x_i) = \mathbb{E}\left[\sum_{k=0}^{T} \frac{1}{d_i} \sum_{j=1}^{m} (y_j - \bar{y}) \mathbb{1}_{\{X_k^{x_j} = x_i\}}\right] \tag{8}$$

It turns out that as $T \to \infty$, the above converges to the solution of the Poisson equation posed earlier.

Poisson Learning

Variational Interpretation

The problem in Poisson Learning can also be posed as the following variational problem:

$$\min_{u \in \ell_0^2(X)} \left\{ \sum_{i,j=1}^{n} w_{ij} |u(x_i) - u(x_j)|^2 - \sum_{j=1}^{m} (y_j - \bar{y}) \cdot u(x_j) \right\} \quad (9)$$

Per Theorem 2.3[5], the solution to the above is also the solution of the Poisson equation posed earlier.

[5] Calder, Jeff, et al. "Poisson learning: Graph based semi-supervised learning at very low label rates." International Conference on Machine Learning. PMLR, 2020.

Poisson Learning

Algorithm

We can solve our Poisson equation through the following iterative process:

$$u_{T+1}(x_i) = u_T(x_i) + d_i^{-1}\left(\sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} - \mathcal{L}u_T(x_i)\right) \tag{10}$$

Also expressed as:

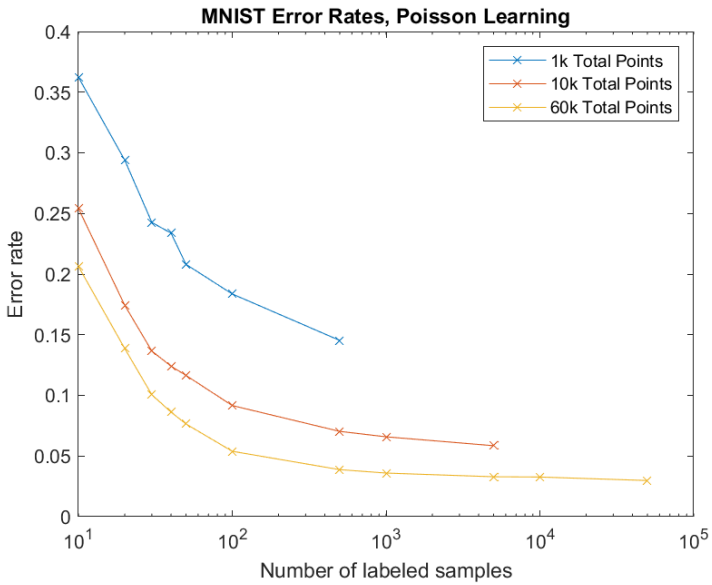
$$\mathrm{U} = \mathrm{U} + \mathrm{D}^{-1}(\mathrm{B}^T - \mathrm{LU}) \tag{11}$$

Per Theorem 2.1, the above iterative process yields the unique solution to equation (6) satisfying $\sum_{i=1}^{n} d_i u(x_i) = 0$. The authors found that this typically converged by $T = 500$ so I used this parameter for my experimentation

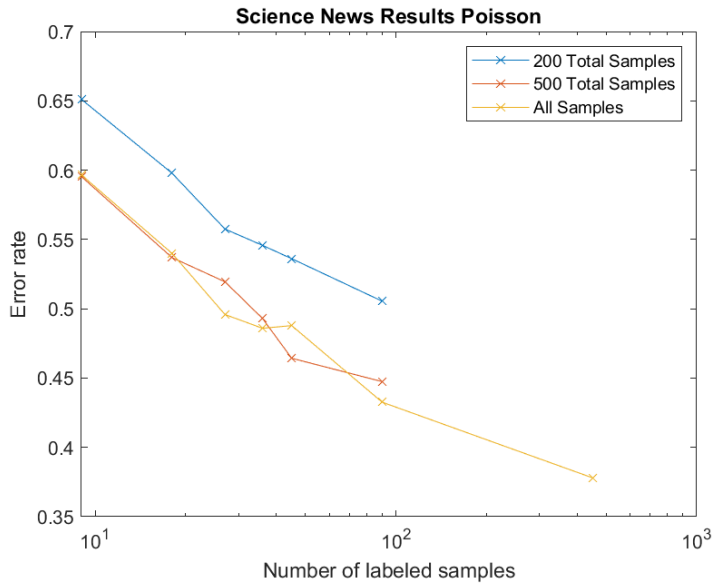
Poisson Learning

Experimental Results

On the MNIST dataset, this algorithm performed very well even at very low label rates, but it was less successful on the Science News Dataset:



(a) MNIST



(b) Science News

Poisson Learning

Experimental Results

The algorithm also performs reasonably well on Fashion MNIST, even at low label rates:

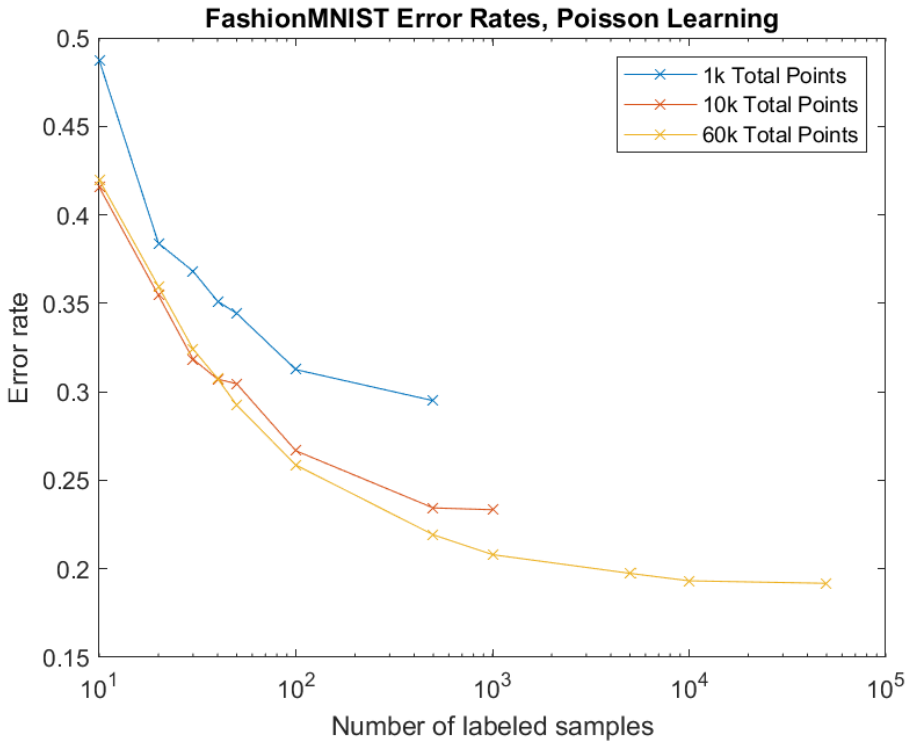


Figure: Poisson Learning: Fashion MNIST

Poisson Learning

Varying the Weight Kernel

In the prior experiments, the weight matrix was constructed using Gaussian weights:

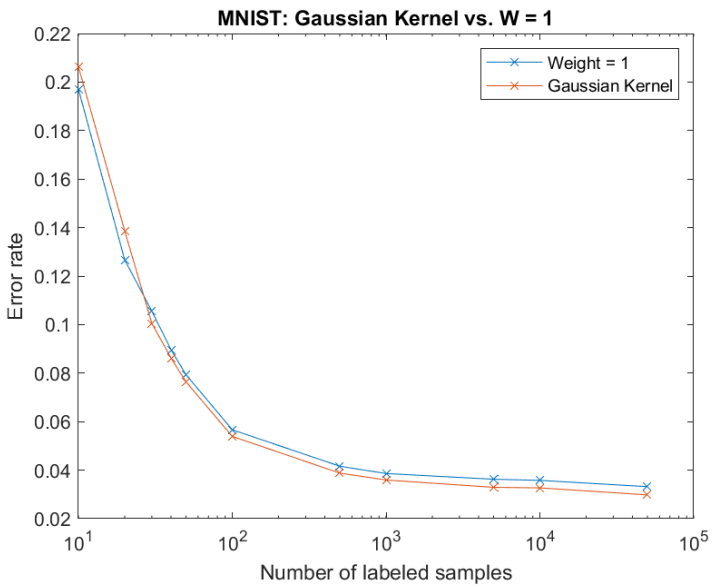
$$w_{ij} = \exp(-4|x_i - x_j|^2 / d_K(x_i)^2) \tag{12}$$

where $d_K(x_i)$ is defined to be the distance between x_i and its $k - th$ nearest neighbor. In principle, though, many kernels can be used. What if we set $W_{ij} = 1$ for each nearest neighbor?

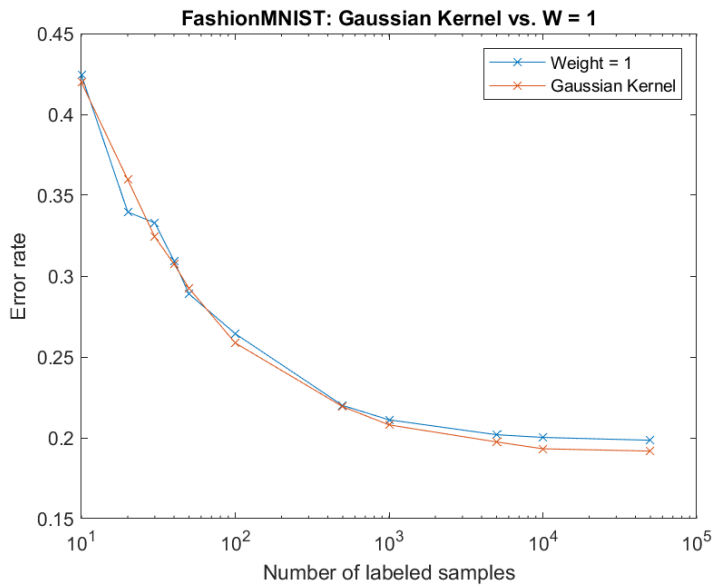
Poisson Learning

Experimental Results

I tested this on MNIST and FashionMNIST, and found almost no difference between the two weight kernels:



(a) MNIST

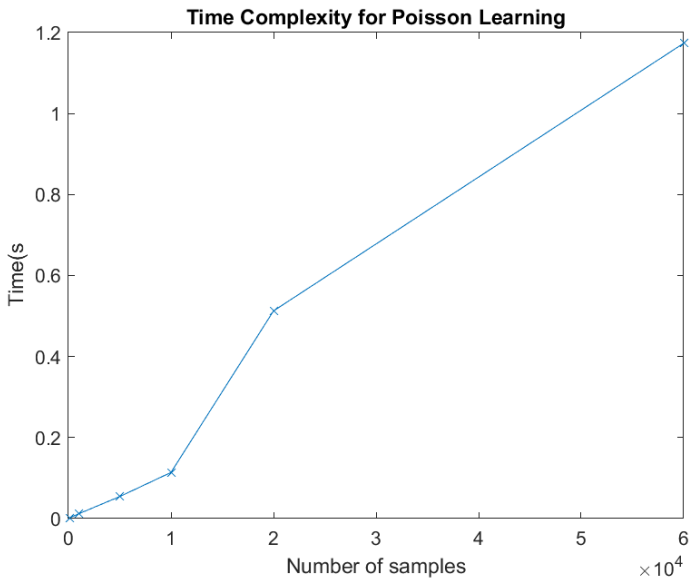


(b) FashionMNIST

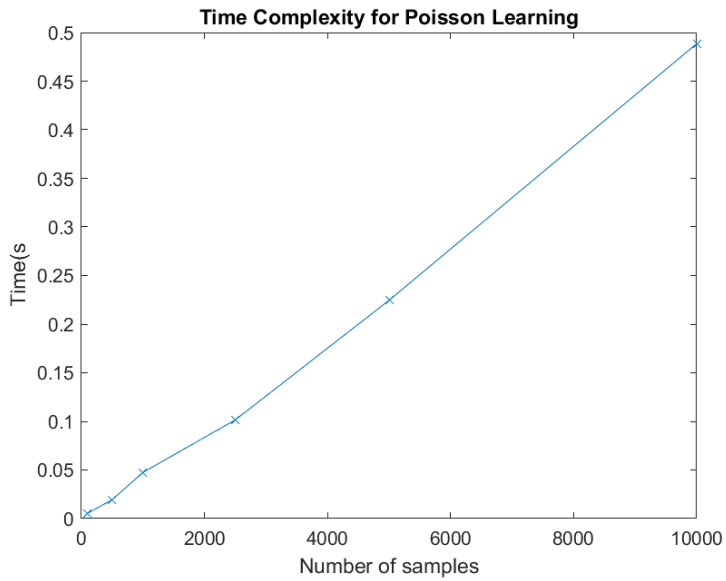
Poisson Learning

Time Complexity

Similar to the other algorithms, Poisson Learning has approximately $O(n)$ time complexity.



(a) FashionMNIST



(b) MNIST