Semi-Supervised Learning

Wen Zhang[1], Yun You[2], and Feng Lu[2]

[1]Department of Biomedical Engineering, Johns Hopkins University
[2]Department of Applied Mathematics and Statistics, Johns Hopkins University

May 6, 2024

1 Introduction

In this project, we aim to address the challenges of semi-supervised learning (SSL). Although various approaches exist for tackling SSL, such as generative models, consistency regularization, and pseudo-labeling methods [1], our focus lies on graph-based SSL. We will delve into different strategies for constructing weighted graphs and explore the application of various classifiers to SSL problems using both non-adaptive and adaptive kernels. Furthermore, we will conduct a theoretical analysis and provide insights into experimental results, highlighting the strengths and weaknesses of each method.

1.1 Problem description

Problem Description Suppose we have a large number of samples $x_1, ..., x_n \in \mathcal{X} \subseteq \mathbb{R}^d$. Without loss of generality, we can assume the first $m << n$ samples have labels $y_i = f(x_i) \in \mathcal{Y} = \{1, ..., \mathcal{C}\}$. Here there are $\mathcal{C}$ class and $f : \mathcal{X} \mapsto \mathcal{Y}$ is some labeling function that assign class labels to each sample x_i. In semi-supervised learning, we want to learn a function $\hat{f}$ using the entire 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 main idea is that we can exploit the total amount of samples we have to understand the underlying geometry of the data, and leverage this information to learn an appropriate classifier $\hat{f}$.

2 Laplacian Eigenmap based SSL

2.1 Overview

In the realm of semi-supervised learning (SSL), the method of Laplacian Eigenmaps is particularly effective in cases where the data is presumed to reside on a low-dimensional manifold within a higher-dimensional space. This approach leverages the manifold assumption, positing that classification functions should fundamentally operate within this manifold rather than across the entire ambient space. This theory allows for deriving a classification algorithm that utilizes a subset of labeled data points to classify a larger dataset.

Key ideas Given the advantage in classification problems of recovering the manifold and developing classifiers on it, we can use the unlabeled data to recover the manifold and the labeled data to construct classifiers. The model of a weighted graph can help utilize the manifold structure. The method proposed by Belkin and Niyogi [2]can use the graph Laplacian matrix to approximate the Laplace-Beltrami operator.

2.2 Algorithmic Description

For simplicity, assume that labels $y_i \in \{-1, 1\}, i = 1, ..., m$.

The Laplacian Eigenmaps algorithm begins by representing each data point as a vertex within a graph G, constructing this graph to reflect the manifold's estimated structure. Connections between vertices (data points) are established based on k-nearest neighbors criteria, using Euclidean distance as a measure. The graph's adjacency matrix is given by $W_{n\times n}$:

$$W_{ij} = \begin{cases} 1 & \text{if } x_i \in k\text{-nearest neighbor of } x_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The unnormalized graph Laplacian matrix L of G can be computed by $L = D - W$, where D is the degree matrix of G satisfying $D_{ii} = \sum_j W_{ij}$. It is important to note that L is positive semi-definite and can be thought of an operator on functions defined on vertices of the graph. Then we calculate the p smallest eigenvectors $e_1, ..., e_p$ of the Laplacian matrix.

Using he labeled data to build a linear classifier on the space spanned by $e_1, ..., e_p$, the error function is:

$$err(\vec{a}) = \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{p} a_j e_j(i) \right) \tag{2}$$

The minimization is considered over the space of coefficients $\vec{a} = (a_1, ..., a_p)^T$. The solution is given by

$$\vec{a} = (U^T U)^{-1} U^T \vec{y} \tag{3}$$

where $\vec{y} = (y_1, ..., y_m)^T$ and U is a $m \times p$ matrix with $U_{ij} = e_j(i)$. As a result, the classification of the unlabeled data can be performed by:

$$y_i = \hat{f}(x_i) = \begin{cases} 1 & \text{if } \sum_{j=1}^{p} a_j e_j(i) \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad ; \quad i = m+1, \ldots, n. \tag{4}$$

These eigenvectors form a new feature space in which data points are more distinctly separable.

2.3 Experiment Results

2.3.1 Dataset

Our experiment used two datasets, MNIST and COIL20. The dataset characteristics and preprocessing method we use is described below.

MNIST MNIST is a dataset comprising 70,000 labeled images of handwritten digits from 0 to 9. Each image is a 28x28 pixel grayscale representation. The original dataset of MNIST is shown in Fig.1.

Figure 1: original and smoothed MNIST data sample

We initially smooth each image twice using a 3x3 filter to expedite the graph construction process. This preprocessing step helps reduce noise and enhances the image features. Fig.2 gives an example of the smooth process.

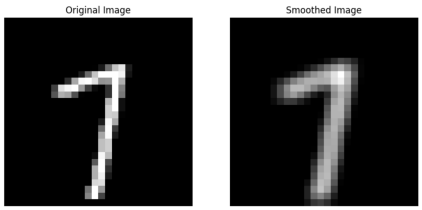


Figure 2: original and smoothed MNIST data sample

After smoothing, we apply Principal Component Analysis (PCA) to each image and extract the first 50 principal components, which capture the most significant features and variations in the dataset. Subsequently, we project each image onto the subspace defined by these 50 principal components, effectively reducing the dimensionality of the data. The resulting 50-dimensional vectors serve as the new coordinates for each image, enabling us to represent them compactly and informally. By leveraging these coordinates, we can construct the graph more efficiently while retaining the essential features for the semi-supervised learning task. The PCA components was shown at Fig.3.

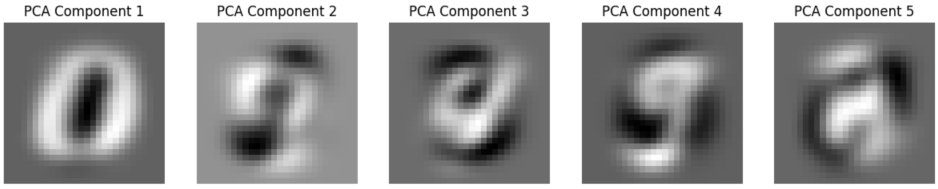


Figure 3: the 5 largest PCA component of the MNIST dataset

COIL20 COIL20 is a collection designed for object recognition tasks, containing 1,440 grayscale images of 20 different objects. Each object is captured from 72 different angles, providing a 360-degree view. The images are 128x128 pixels. Fig.4 is an example of this dataset.

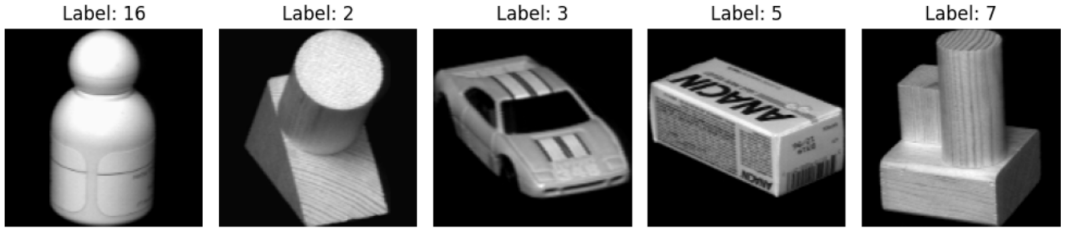


Figure 4: the 5 largest PCA component of the MNIST dataset

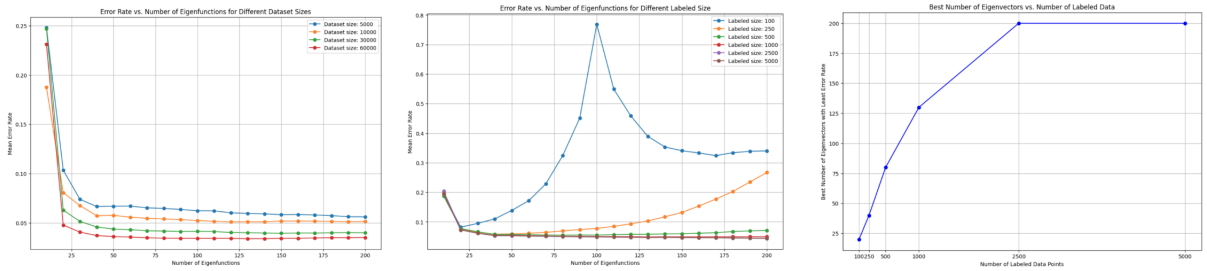
after using the same smoothing method we describe before, we take the first 100 principal components of the COIL20 dataset as shown in Fig.5.

Figure 5: the 5 largest PCA component of the COIL20 dataset

2.3.2 Result

For MNIST dataset, we set the nearest neighbors $k = 9$; for COIL20 dataset, we set the nearest neighbor to be 25.

Fig.6 shows the error rate vs. number of eigenvectors. In Fig.6(a) we fix the labeled data to be 1000. The error rate decreases significantly as the number of eigenvectors increases, then stabilizes. Larger datasets generally yield lower error rates due to improved data representation. The initial eigenvectors capture the most significant data features while adding more beyond a certain point provides diminishing returns. In Fig.6(b), we test 10,000 data with difference labeled size, the graph generally demonstrates that increasing the labeled data size generally results in lower error rates. However, with very few labeled points, the error rate initially increases before declining. This overfitting occurs when the number of eigenvectors exceeds the labeled data points, leading to poor generalization. In Fig.6(c), As the labeled data size increases, the optimal number of eigenvectors rises due to the model's ability to handle more complexity without overfitting.



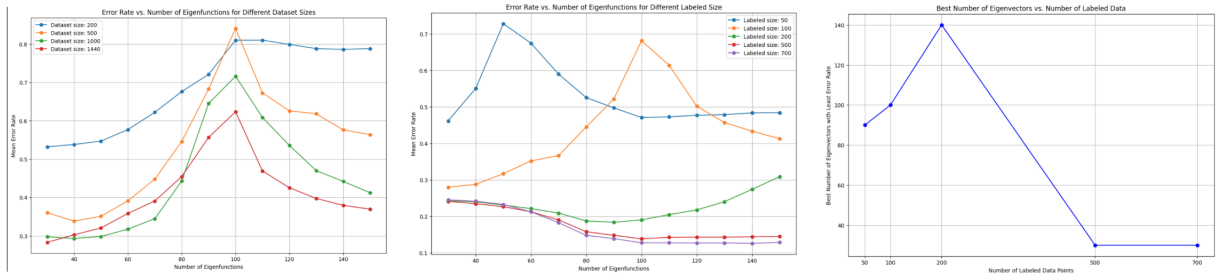
(a) Error rate vs. number of eigenvectors with different set sizes

(b) Error rate vs. number of eigenvectors with different labeled data sizes

(c) Best number of eigenvectors with different labeled data sizes

Figure 6: Influence of number of eigenvectors for MNIST dataset

For COIL20 dataset, we replicate the experiments we mentioned before. First, we fix labeled data to be 100 to generate Fig.7(a) and total number of data to be 1440 to produce Fig.7(b). Since we only use 100 labeled data, the error increases when the number of eigenvectors in less than 100 and decreases when the number exceeds 100. This suggests overfitting occurs with smaller datasets when too many eigenvectors are used. It is interesting in Fig.7(c), the optimal number of eigenvectors increases with more labeled data, allowing the model to capture intricate patterns without overfitting.



(a) Error rate vs. number of eigenvectors with different set sizes

(b) Error rate vs. number of eigenvectors with different labeled data sizes

(c) Best number of eigenvectors with different labeled data sizes

Figure 7: Influence of number of eigenvectors for COIL20 dataset

In conclusion, a balance between model complexity (number of eigenvectors) and data availability is essential to avoid overfitting. Besides, larger datasets and labeled data sizes enable more complex models, improving classification accuracy.

3 Eigenfunction Based Regularization on Graph

3.1 Overview

In advancing graph-based semi-supervised learning (SSL) methods, Szlam et al. [3] introduced function-adapted kernels to address scenarios where the target function may not be smooth relative to the original data geometry.

Their innovation lies in adjusting the data geometry such that the target function exhibits maximum smoothness in this new context. This approach has been applied both to Laplacian Eigenmap-based SSL and diffusion-based SSL. In this section, we primarily focus on the former, setting the stage for an exploration of diffusion-based methods later.

3.2 Algorithmic Description

The essence of this approach is to manipulate the manifold structure of the data to enhance function approximation and smoothing capabilities for SSL.

Weighted Graph Construction We construct a weighted graph with vertices of both labeled and unlabeled data points to model the manifold of the data. The edges of this graph are weighted by the similarity between data points, calculated using a Gaussian heat kernel based on Euclidean distance. The local similarity between x and y can be collected with a matrix W:

$$W_\sigma(x, y) = h\left(\frac{\rho(x, y)^2}{\sigma}\right) \tag{5}$$

where ρ is a distance metric(Euclidean in this project),σ is the local time parameter, $h(a) = exp(-a)$ is the Guassian kernel. This similarity is strictly local, defined only among k-nearest neighbors to maintain reliability over short distances, which better captures manifold structures.

Theoretical Support The operator $(I - D_\sigma^{-\frac{1}{2}} W_\sigma D_\sigma^{-\frac{1}{2}})/\sigma$, where D_σ is a diagonal matrix of row sums of weight matrix W, approximates the Laplacian on the manifold as σ approaches zero. This matrix formulation supports the idea of using a weighted graph to approximate manifold structures effectively.

Self-tuning Weight Matrix To accommodate variations in local density and scale, Zelnik-Manor and Perona [4] suggested a self-tuning mechanism that adjusts distances at each vertex such that the distance to its j-th nearest neighbor is normalized to one. Let $\rho_x(z, z') = \rho_x(z, z')/\rho_x(x, x_j)$, where x_j is the j-th nearest neighbor to x. The self-tuning weight matrix is defined as:

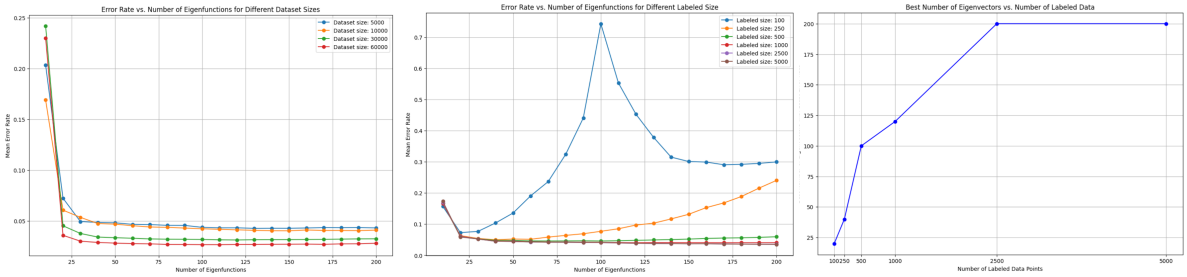
$$W_\sigma(x, y) = h\left(\frac{\rho_x(x, y)\rho_y(x, y)}{\sigma}\right) \tag{6}$$

Self-tuning can be useful in stabilizing the weight matrix. For example, if points are very close or far from all the nearest neighbors, the local similarity will be close to 1 or 0 in the former case. After self-tuning, the weights of different points can be brought to almost the same level. This normalization facilitates a more balanced and location-dependent similarity measure across the graph.

3.3 Experiment Results

Based on the algorithm we described in the previous part, we perform eigenfunction-based supervised learning with a self-tuning weight matrix. For MNIST dataset, we set the nearest neighbors $k = 9$ and the self-tuning neighbor index $m = 8$; for COIL20 dataset, we set the nearest neighbor to be 25 and the self-tuning neighbor index to be 20. The local time $\sigma = 1$ for both sets.

On both datasets, we study the influence of a number of eigenvectors on error rates with different set sizes and labeled data sizes. The result of MNIST dataset is shown in Fig.8. As shown in Fig.6(a), we fix the number of labeled data to be 1000 and change set sizes from 5,000 to 60,000. When we increase the number of eigenvectors, the error rate drops significantly and then stabilizes. This is likely because the initial eigenvectors capture the most important variations in the data, leading to a substantial reduction in error. However, after a certain point, additional eigenvectors provide diminishing returns as they represent less significant patterns. Then we fix the set size to be 10,000 and change the number of labeled data from 100 to 5,000. The result in Fig.8(b) shows that adding more labeled data generally reduced the error rate. However, with only 100 labeled points, the error rate initially increases before decreasing. This could be due to over-parameterization in the least squares regression problem when there are too few labeled examples relative to the number of eigenvectors, leading to overfitting. Fig.8(c) is the relationship between the optimal number of eigenvectors and different numbers of labeled data. This graph reflects the dependency of the model's complexity on the dimensionality (amount) of labeled data available. With more labeled examples, the model can effectively utilize a larger number of eigenvectors to capture more intricate patterns without overfitting.



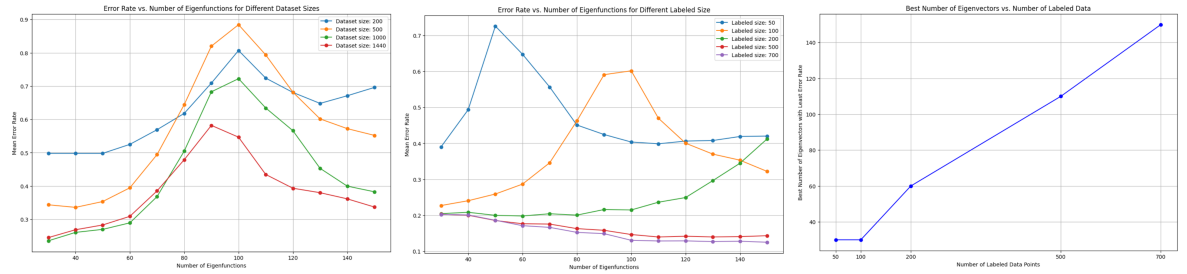
(a) Error rate vs. number of eigenvectors with different set sizes

(b) Error rate vs. number of eigenvectors with different labeled data sizes

(c) Best number of eigenvectors with different labeled data sizes

Figure 8: Influence of number of eigenvectors for MNIST dataset

The experiment results from COIL20 dataset resemble those from MNIST as can be seen in Fig.9. It is worth noting that in COIL20, when we fix the number of labeled data, the error rate increases at first and then drops after the number of eigenvectors exceeds 100. Since we only choose 100 labeled data, over 100 eigenvectors lead to over-parameterization and brings about the decrease in error rate. Compared with results from MNIST dataset, the error rates increase gradually in this case, suggesting that COIL20 can be more sensitive to overfitting. As for Fig.9(b), the COIL20 dataset shows an initial increase in error rate with the number of eigenvectors, especially for smaller labeled data sizes (50 and 100). This is attributed to overfitting when the number of eigenvectors exceeds the number of labeled points. The effect is more pronounced in COIL20 than MNIST, possibly due to the higher complexity of the COIL20 dataset or the smaller number of labeled examples used. Similar to MNIST, the optimal number of eigenvectors increases with the number of labeled data points in Fig.9(c). However, the growth is more rapid for COIL20, with the best number of eigenvectors reaching around 140 for 700 labeled points, compared to about 175 for 5,000 labeled points in MNIST. This difference could be due to the intrinsic dimensionality of the datasets, with COIL20 requiring fewer eigenvectors to capture its essential patterns.



(a) Error rate vs. number of eigenvectors with different set sizes

(b) Error rate vs. number of eigenvectors with different labeled data sizes

(c) Best number of eigenvectors with different labeled data sizes

Figure 9: Influence of number of eigenvectors for COIL20 dataset

These findings highlight the importance of considering dataset characteristics when applying semi-supervised learning techniques. The number of eigenvectors and labeled data points should be carefully tuned to balance model complexity and available data, avoiding underfitting and overfitting.

Comparison

In our evaluation of semi-supervised learning (SSL) techniques using the Laplacian eigenmap-based approach, we observed marginal improvements when transitioning from an unweighted to a regularized graph. This comparison, as shown in Fig.6 and Fig.8 for the MNIST dataset and Fig.7 and Fig.9 for the COIL20 dataset, indicates slight enhancements in performance. Although the changes in error rates are minimal—typically less than a 3% reduction—the regularized graph consistently outperforms the unweighted graph in most scenarios. Notably, there are exceptions where the unweighted graph performs slightly better, such as when the labeled data size is 200 and the number of eigenfunctions is 90. This improvement with the regularized graph is anticipated since it captures more

detailed relationships within the data. However, the general patterns of underfitting and overfitting persist across both graph models, indicating that while the graph structure influences performance, the fundamental dynamics of model complexity relative to data availability remain unchanged.

Time Analysis

Finally, we evaluate the time complexity of this method on both datasets by calculating the runtime of graph construction, eigenvalue decomposition, and classification as presented in Fig.10. Constructing the graph involves computing pairwise similarities or distances between all data points, which has a time complexity of $O(n^2)$, where n is the number of data points. During the whole process, the graph construction and eigenvalue decomposition steps dominate the runtime.

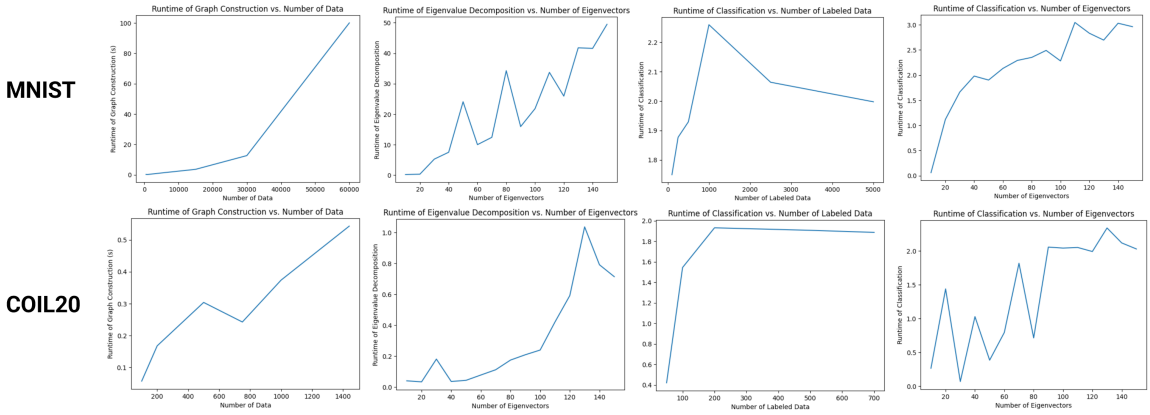


Figure 10: Run time analysis

4 Diffusion Based Regularization on Graph with Function-Adapted Kernels

4.1 Overview

In this part, we extend the graph-based SSL methodologies by integrating diffusion-based regularization, which simulates heat diffusion processes to propagate label information across a graph. This method hinges on the hypothesis that label information, when diffused properly across the graph's structure, can significantly improve the accuracy of semi-supervised learning models, especially in scenarios where labeled data is sparse. The approach leverages both the intrinsic geometry of the data and the known labels to enhance the estimation of unknown labels through iterative smoothing processes.

4.2 Algorithmic Description

Diffusion Process First, we construct a weighted graph and local similarity W_σ as described in the previous part. Let

$$D(x) = \sum_{y \in V} W_\sigma(x, y) \tag{7}$$

be the degree matrix, V is the set of vertices. The filter matrix K is defined by normalizing W_σ using D:

$$K(x, y) = \frac{W_\sigma(x, y)}{D(x)} \tag{8}$$

This normalization ensures that each row of K sums to 1, making K a proper stochastic matrix. In the context of a diffusion process, this means that K describes the transition probabilities of a random walk on the graph, where the probability of moving from x to y is proportional to their similarity. Applying K to a vector or function f (thought of as signal values at each point or node) yields a new vector where each element is a weighted average of its neighbors' values, with weights given by the similarities:

$$(Kf)(x) = \sum_{y \in V} K(x, y) f(y) \tag{9}$$

This operation is central in smoothing, denoising, or spreading information across the graph, where the Gaussian similarities between points guide the diffusion at each step. Then use a power of K to smooth χ_i^{lab} which is the indicator function of data belong to class i.

This diffusion process is repeated for a predetermined number of iterations obtaining $\bar{\chi_i^{lab}} = K^t \chi_i^{lab}$. After diffusion, classify each unlabeled point by assigning

$$x = \text{argmax}_i \bar{\chi_i^{lab}}(x) \tag{10}$$

This step effectively spreads the label information across the graph according to the connectivity and similarity of points.

Function-Adapted Kernels To further refine the geometry for smoother function approximation, function-adapted kernels [5] are incorporated. Let

$$W^f(x,y) = exp\left(-\frac{||x-y||^2}{\sigma_1} - \frac{||f(x)-f(y)||^2}{\sigma_2}\right) \tag{11}$$

to be the function-adapted weight matrix, where $\tilde{f}$ is the estimation of true classifier f. These kernels adjust weights based on both the Euclidean distance and the difference in function values estimated across data points. This dual consideration helps align the data geometry with the function's structure, promoting smoother estimations where the function varies minimally.

This sophisticated interplay between geometric manipulation and kernel adaptation forms the backbone of the proposed method, promising enhanced performance in semi-supervised learning scenarios by aligning data geometry with target function characteristics more effectively.

Modify the graph's structure based on the initial classification with $\bar{\chi_i}$ for binary problems and c_i as below for multi-class problems.

$$c_i(x) = \frac{\bar{\chi_i}(x)}{\sum_i |\bar{\chi_i}(x)|} \tag{12}$$

Adding c_i or χ_i to original dataset to construct a new graph with K^t from (11) with $\sigma_2 = \beta\sigma_1$. Finally. classify points based on the outputs of the adapted diffusion process.

This approach iteratively refines both the classification function and the graph structure on which the diffusion is performed. By adapting the graph to initial classification results, it seeks to make the diffusion process more efficient and aligned with the actual data distribution and inherent class separations.

4.3 Classifiers

We will introduce two similar classifiers, both utilizing the idea of diffusion. Both classifiers will use known labels as initial conditions and apply the diffusion process. We will name one the 'smoothing classifier' and the other the 'harmonic classifier'. The main difference will be in the harmonic classifier, as it will re-update the initial conditions at each iteration, updating the probability of known labels to 1.

4.4 Experiment results

4.4.1 Non-adapted function

We use the MNIST dataset and the COIL20 dataset to assess performance, employing the same pre-processing method as described in previous sections. Initially, we evaluate performance using a non-adaptive function to build a weighted graph, with parameters chosen to match those used in previous experiments. Subsequently, we compare the performance of two classifiers by varying the selection of labeled samples and the number of iterations. Through cross-validation, we find that at around 20 iterations, both classifiers exhibit improved performance, with the harmonic classifier generally outperforming the others. This classifier provides more accurate initial conditions by consistently updating the probabilities of known labels, which can expedite convergence and lead to stability as shown in the right images of Fig. 11 and Fig.12. In the left figures, we measure the number of labeled samples provided for training the classifiers, ranging from 0 to 5000 samples. The trend is obvious: as we increase the number of labeled samples, performance should improve since the classifier will have more information. However, the harmonic classifier significantly benefits from more data, while the kernel smoothing method is less responsive to increases in labeled samples, suggesting it might reach its performance limit earlier.

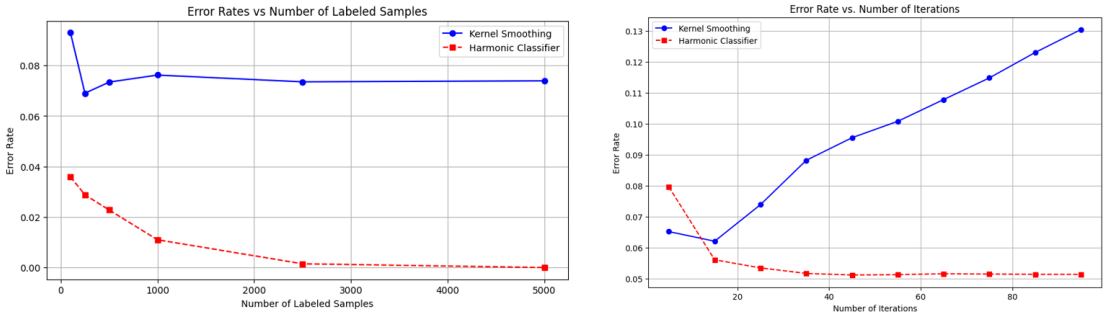


Figure 11: Influence of number of Iterations and labeled for MNIST dataset

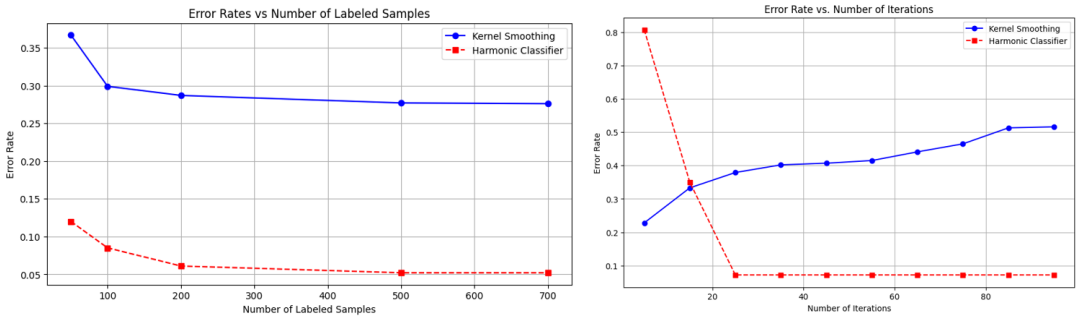


Figure 12: Influence of number of Iterations and labeled for COIL20 dataset

4.4.2 Function-Adapted kernel

Now, we apply the same analysis but use a function-adapted kernel to create a graph. In other words, we add an extra dimension c to the original data set, which is created by using a harmonic classifier to predict the label on the original data first, and then create the graph.". We first define the value of β. For the MNIST dataset, both classifiers have optimal solutions when $\beta = 5$, shown in Fig. 13. For the COIL20 dataset, the harmonic classifier also achieves an optimal solution when $\beta = 5$. However, the kernel smoothing classifier shows worse performance at $\beta = 5$ and does not exhibit any specific optimal solution as indicated in the graphs. Moreover, we did not find any significant performance change based on different β values.

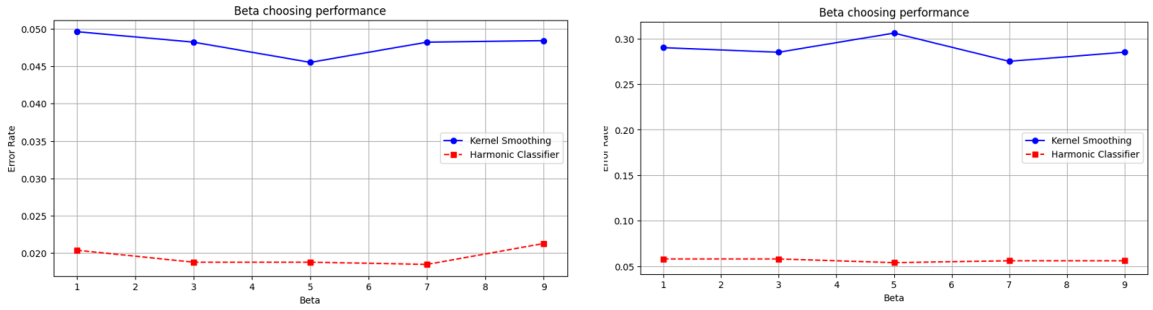


Figure 13: Influence of different Beta for MNIST dataset(left) and COIL20 dataset(right)

Then, we set $\beta = 5$, and test the classification error rates of the smoothing and harmonic classifiers as the number of labeled samples and iterations vary. The overall trend is similar to before, and the performance shows a slight improvement. The results were not readily apparent when looking at the graphical results, but when we examined the final classification error, we observed that the error rate decreased after applying the function-adapted kernel. For the MNIST dataset, the error rate when changing the number of labeled samples to 500 is 0.0228 for the harmonic classifier and 0.0734 for the smoothing classifier. After applying the adapted kernel, the error rate for the

harmonic classifier decreases to 0.0197, and for the smoothing classifier, it decreases to 0.0713. The same decreasing trend applies to both datasets, as in Fig. 14,Fig. 15 .

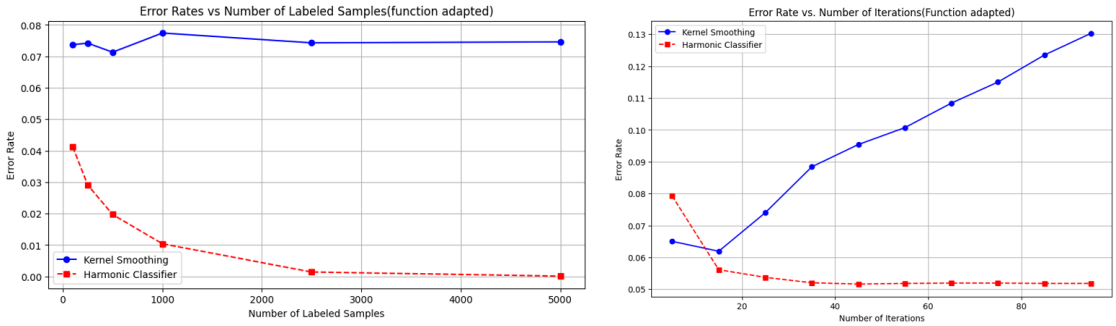


Figure 14: Influence of number of Iterations and labeled for MNIST dataset

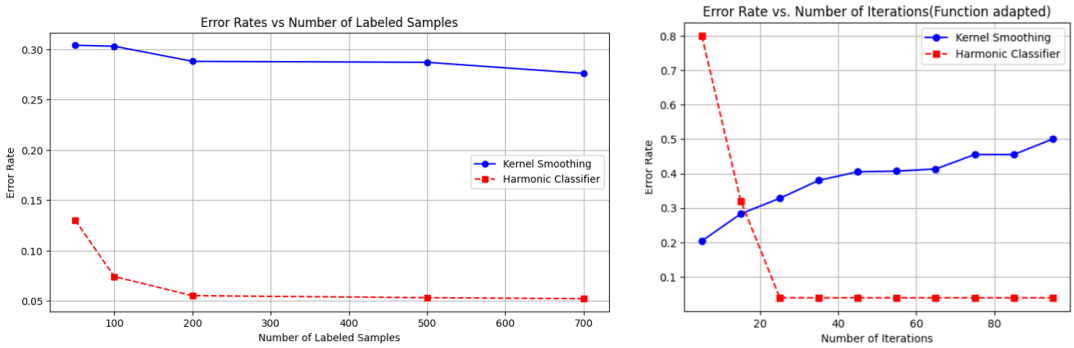


Figure 15: Influence of number of Iterations and labeled for COIL20 dataset

4.5 Comparison

By comparing the results, using function-adapted methods to build weighted graph is outperforming non-adapted methods. Both methods consider the distance between data points in the graph, but the function-adapted method also leverages additional information about the relationships between data points. It considers how each data point is relevant to the others by incorporating the probabilities of data labels after diffusion. This provides more precise information for improvement.

Moreover, using a harmonic classifier results in better performance in terms of error rate, indicating convergence in performance. This improvement is due to the classifier updating the probability of known labels to 1, which stabilizes the results. This approach ensures that the influence of known labels effectively propagates through the graph, reducing uncertainty.

5 Conclusion

In this report, we discuss the performance of the eigenfunction-based model and the diffusion-based model. We test their performance on two datasets, MNIST and COIL20, and discuss the results. We also discuss the advantage of adding extra function information by using a function-adapted kernel to create a graph, as well as investigate its ability to enhance performance.

6 Group Contribution

Every team member goes through the theoretical concepts. **Feng Lu** focuses on the basic Laplacian method of building a graph. **Wen Zhang** performs pre-processing of the datasets, builds the self-tuning weighted graph, applies the eigenfunction-based method, and writes the first 3 parts of the report. **Yun You** focuses on building

diffusion classifiers and compares the performance of using non-adapted and function-adapted kernels, and writes part 4 of the report.

References

[1] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A survey on deep semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(9):8934–8954, 2022.

[2] Mikhail Belkin and Partha Niyogi. Using manifold stucture for partially labeled classification. *Advances in neural information processing systems*, 15, 2002.

[3] Arthur Szlam, Mauro Maggioni, and Ronald R Coifman. A general framework for adaptive regularization based on diffusion processes on graphs. *Yale technichal report*, 2006.

[4] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. *Advances in neural information processing systems*, 17, 2004.

[5] Arthur D Szlam, Mauro Maggioni, and Ronald R Coifman. Regularization on graphs with function-adapted diffusion processes. *Journal of Machine Learning Research*, 9(8), 2008.