Spectral Nonlinear Dimensionality Reduction: Analysis & Comparison of Proposed Extensive Methods

Josh Wang | wang.12432
Scott Caley | caley.14
Min Shi | shi.1532

Accompanying Repository: https://github.com/scottcaley/Spectral_NDR

Abstract

Our project research group explores the developments into dimension reduction methods. Dimensionality reduction remains a vital optimization problem in modern machine learning fields due to the high dimensionality and presence of noise in industrially produced data. Modern methods such as Locally Linear Embedding (LLE) and Laplacian Eigenmaps reduce dimensions by creating a low-dimensional embedding assuming the data lies on a lower-dimensional manifold. They encounter issues, however, when the need for estimating the intrinsic dimensionality of the data or preserving shared local geometric features arises. Such features can be angles, scale, or translational relationships.

A proposed method by Sha & Saul [5] aims to solve a small-scale semidefinite programming problem in order to adjust the embeddings produced by such spectral dimensionality reduction methods to better preserve local geometric features. In the project, we implement such extended spectral methods and demonstrate on a generated dataset to test the accurateness and qualitative performance.

1. Introduction

One of the broadest areas of research within Machine Learning and Artificial Intelligence deals with the quality and optimization of data. For this research project, our group wanted to explore the topic of dimensionality reduction. When surveying the efforts of Machine Learning research across the past several years, we may attribute impactful improvements in performance to many qualities, but the dimension reduction of data and tailored-model performance, especially of LLMs within recent years, play a significant role.

Dimension reduction plays such a vital role in model and general classification performance because it is inherently the response to one of the downsides of robust data in the modern world: “the curse of dimensionality”. In today’s

world, usage of machine learning involves processing hundreds of millions of terabytes a day. Modern day may range from simple relational tables, albeit consisting of potentially thousands of attributes, or even complex images generated from sources like car cameras or topological maps. In all these usages, however, the data dimensionality can be exceedingly large. Often models and usage of machine learning, especially within image and geometrically-based data, have a large amount of noise existent within its total dimensions, in which the discoverable trends and noise-resistant data is only within a relative minority subset of dimensions.

Because of such importance being placed on dimension reduction, the project group aims to discover, test, and implement proposed strategies for dealing with the inefficiencies of current popular methods for dimension reduction. Two such methods we take a look at in the project are Locally Linear Embedding and Laplacian Eigenmaps. While the dimensions yielded by these two such methods have noticeable performance improvement when employed with machine learning models, the underlying methods are potentially problematic in that they inherently do not act on or intentionally preserve local geometric properties existent among the data. Not that the data points themselves may specifically hold specific geometric data, but rather the relationships they hold to one another in local settings, especially when viewed within a high dimension space, may exhibit similarities such as angles or isometric embeddings.

The paper that we focus on in this study is “Analysis and Extension of Spectral Methods for Nonlinear Dimensionality Reduction” (Sha & Saul, 2005). The authors seek to analyze both LLE and Laplacian Eigenmaps, in particular their shortcomings when it comes to the aforementioned properties. They then propose extended methods that are built on top of the eigenvectors produced by LLE and the Laplacian eigenmaps that aim to preserve local geometric features such as angles, translation, or scaling. “The new embeddings explicitly optimize the degree of neighborhood similarity—that is, equivalence up to rotation, translation, and scaling—with the aim of discovering conformal (angle-preserving) mappings” (Sha & Saul, 2005). We have studied their proposed implementation and seek to test the supposed performance improvements using a variation of data.

Our first order of research was to implement the LLE and Laplacian Eigenmap methods using Python so that they were able to be extended and our performance metrics would have a baseline of comparison. Afterwards, the authors detail a proposed extension based on their analysis of the properties of the eigenvectors and eigenmaps produced by the methods. They describe “a d-dimensional embedding is computed from the m bottom eigenvectors of LLE or Laplacian eigenmaps with m>d, thus incorporating information that the original algorithm would have discarded for a similar result” (Sha & Saul, 2005) along with the additional solving of a proposed semidefinite program which yields an estimate of the underlying dimensionality of the data rather than an estimated, representative mapping of such.

We implement both the embedding computation and semidefinite program solution using python that can work either via LLE or Laplacian Eigenmaps for its foundation. Then, we test the proposed solution using randomly generated data fitted to a spiralling gaussian distribution. While this method of testing is simple, it becomes easy to compare and visualize the reduced dimension output of such methods.

2. Problem Statement

Through the project, we hope to assess the question “What is the qualitative improvement of extending existing dimensionality reduction methods to include a basis of local geometric features?” In order to do so, we must assess the proposed inadequacies of popular methods employed in machine learning today, and what lacking elements of each can lead to higher performance.

The two methods that are primarily discussed are LLE and Laplacian Eigenmaps. Sha & Saul comprehensively list the descriptions of both methods: “LLE appeals to the intuition that each high dimensional input and its k-nearest neighbors can be viewed as samples from a small linear ‘patch’ on a low dimensional submanifold… Laplacian eigenmaps also appeal to a simple geometric intuition: namely, that nearby high dimensional inputs should be mapped to nearby low dimensional outputs.” The embeddings, however, often fail to preserve local geometric features, such as angles or scale. This, in part, is due to the nature of trying to construct sparse matrices and derive embeddings from eigenvectors. However, for model performance and trend discovering, important implicit features of the data need to be preserved whilst tackling the curse of dimensionality. One foundational problem is that prior methods do not directly estimate the dimensionality of the underlying manifold, thus exhibiting unpredictable behavior depending on sampled data and boundary conditions.

This is solved through the proposed extensions by Sha & Saul as previously discussed. They work by building upon the embeddings produced by LLE and Laplacian Eigenmaps by solving a small-scale semidefinite programming problem to adjust the embeddings to better preserve local clustered features, such as angles, rotation, translational relations, and scalings.

The question then becomes how can these changes be thus visualized and compared to their non-adjusted counterparts. Additionally, this involves the question of how the spectral methods should be implemented and which datasets best fit or demonstrate the preservation of geometric features.

3. Proposed Method

3.1 Locally Linear Embedding (LLE) & Laplacian Eigenmaps Implementation

We first begin the dimensionality reduction process by performing an already-established nonlinear dimensionality reduction technique. The two given choices are LLE (locally linear embeddings) and Laplacian eigenmaps.
In LLE, we first construct a n by n matrix weight matrix, W, that relates each data point to its k nearest neighbors. The weights were chosen in such a manner to minimize distance between a data point and its weighted nearest neighbors. The objective function

$$\varepsilon(W) = \sum_i \left| x_i - \sum_j W_{ij} x_j \right|^2 \qquad (1)$$

allowed us to optimize each row of W independently. To normalize this matrix, the SMO (sequential minimal optimization) algorithm was used. The new data is the d eigenvectors corresponding to the least d non-zero eigenvalues of

$$\mathbf{\Phi} = (I - W^T)(I - W). \qquad (2)$$

```python
def SMO(A, b, knn, num_iterations = 10):
    """
    LLE SMO algorithm to calculate a row vector, w, of weight matrix W
    We are trying to minimize w^T Aw + b^T w, where A is positive definite
    knn contains the only indices that are allowed to be non-zero
    num_iterations is the amount of times we run the algorithm
    """

    # setup
    n = b.shape
    k = len(knn)
    w = np.zeros(n)
    for i in knn:
        w[i] = 1.0 / k

    for iteration in range(num_iterations):

        # In every iteration, go over every ordered pair of i, j in knn where i is not j
        for i in knn:
            for j in knn:
                if i==j: continue

                # Want to find optimal difference t to add to w[i] and subtract from w[j]
                # I worked this out on paper
                quadratic_coef = A[i, i] - 2 * A[i, j] + A[j, j]
                linear_coef = 2 * A[i, i] * w[i] + 2 * A[i, j] * w[j] - 2 * A[i, j] * w[i] - 2 * A[j, j] * w[j] + b[i] - b[j]
                t = - linear_coef / (2.0 * quadratic_coef)

                # boundary adjustments
                if t > 0:
                    max_change = min(1 - w[i], w[j])
                    if t > max_change: t = max_change
                elif t < 0:
                    max_change = min(w[i], 1 - w[j])
                    if abs(t) > max_change: t = - max_change

                w[i] += t
                w[j] -= t

    return w
```

```python
def NDR(X, d, k=5):
    """
    X is n by p data matrix
    d is the desired embedding dimension
    k is the number of neighbors in KNN
    """
    n, p = X.shape
    kdtree = KDTree(X)

    W = np.zeros((n, n))
    A = X @ X.T # for optimization later
    for i in range(n): # To find the weight matrix, each row is independently optimized
        knn = kdtree.query([X[i,:]], k=k+1)[1][0] # get k neighbors that aren't itself
        knn = knn[knn != i] # filter out itself
        b = -2 * X @ X[i]
        W[i, :] = SMO(A, b, knn)

    Phi = (np.eye(n) - W.T) @ (np.eye(n) - W)

    eigenvalues, eigenvectors = np.linalg.eig(Phi)
    idx = np.argsort(eigenvalues)
    eigenvectors_sorted = eigenvectors[:, idx]

    return eigenvectors_sorted[:, 1:d+1]
```

Figure 1: LLE Implementation

Laplacian eigenmaps take a similar approach to LLE in terms of the weight matrix W, where data points are related to their k nearest neighbors. For this process, we stuck to the choices of either a constant 1/k weight, or a weight assigned by a gaussian pdf. Again, a **Φ** matrix is constructed formulaically, and then the new data is transformed to the d eigenvectors corresponding to the least d non-zero eigenvalues of **Φ**.

```python
def NDR(X, d, k=5, variance=0.0):
    """
    X is n by p data matrix
    d is the desired embedding dimension
    k is the number of neighbors in KNN
    variance is a scalar parameter, and its value determines which weight method is chosen
    """
    n, p = X.shape
    kdtree = KDTree(X)

    W = np.zeros((n, n))
    for i in range(n):
        knn = kdtree.query([X[i,:]], k=k+1)[1][0] # get k neighbors that aren't itself
        knn = knn[knn != i] # filter out itself
        for j in knn:
            if (variance <= 0.0):
                W[i, j] = 1.0 / k
            else:
                diff = X[i] - X[j]
                W[i, j] = np.exp(-1.0 * np.dot(diff, diff) / variance)

    D = np.diag(np.sum(W, axis=1))
    D_inv_sqrt = np.diag(1 / np.sqrt(np.diag(D)))
    Phi = np.eye(n) - D_inv_sqrt @ W @ D_inv_sqrt

    eigenvalues, eigenvectors = np.linalg.eig(Phi)
    idx = np.argsort(eigenvalues)
    eigenvectors_sorted = eigenvectors[:, idx]

    return eigenvectors_sorted[:, 1:d+1]
```

Figure 2: Laplacian Eigenmap Implementation

3.2 Conformal Eigenmaps Extension

The nonlinear dimensionality reduction technique proposed by Sha and Saul first makes use of one of the aforementioned techniques to reduce dimensionality from p down to m. It then does a linear transformation on the data in such a way that attempts to restore its angle data, before further reducing the data down to dimension p. Once the dimensionality of X is reduced to Y by either LLE or Laplacian eigenmaps, the transformation matrix L is determined from the differences in X and Y. Data is preserved based on whether angle differences are preserved. We want to produce $z_1, z_2, \ldots, z_n$ that correspond to $x_1, x_2, \ldots, x_n$ where local angles stay similar. In other words, for local points x_1, x_2, x_3, we want to produce z_1, z_2, z_3 in an attempt to approximately obtain

$$\frac{|x_1 - x_2|^2}{|z_1 - z_2|^2} = \frac{|x_1 - x_3|^2}{|z_1 - z_3|^2} = \frac{|x_2 - x_3|^2}{|z_2 - z_3|^2} \quad (3)$$

i.e. form a similar triangle. We do this by relating distance lengths by a factor s_i. The final product is a semidefinite programming problem dependent on both X and Y.

```python
def semidef_prog(X, Y, k=5):
    """
    X is the n by p data matrix
    Y is the n by m transformed data matrix (either by LLE or Laplacian Transform)
    k is the number of neighbors
    returns L, a m by m matrix, the next transformation proposed by Sha, Saul
    """

    n, m = Y.shape
    kdtree = KDTree(X)

    # eta is the matrix of eta_{ij}'s
    eta=[[0 for _1 in range(n)] for _2 in range(n)]
    for i in range(n):
        knn = kdtree.query([X[i,:]], k=k)[1][0]
        for j in knn:
            eta[i][j]=1

    # Define and solve the CVXPY problem.
    # Create a symmetric matrix variable.
    P = cp.Variable((m,m), symmetric=True)
    # creating the objective function
    objective=0
    for i in range(n):
        #setup s_i
        numer=0
        denom=0
        for j1 in range(n):
            for j2 in range(n):
                if j1!=j2:
                    vecY=Y[j1,:]-Y[j2,:]
                    vecX=X[j1,:]-X[j2,:]
                    numer+=2*eta[i][j1]*eta[i][j2]*cp.quad_form(vecY, P)*(np.linalg.norm(vecX)**2)
                    denom+=2*eta[i][j1]*eta[i][j2]*(np.linalg.norm(vecX)**4)
        s=numer/denom
        for j1 in range(n):
            for j2 in range(n):
                if j1!=j2:
                    vecY=Y[j1,:]-Y[j2,:]
                    vecX=X[j1,:]-X[j2,:]
                    expression=eta[i][j1]*eta[i][j2]*(cp.quad_form(vecY, P)-s*np.linalg.norm(vecX)**2)**2
                    objective+=expression
    # The operator >> denotes matrix inequality.
    constraints = [P >> 0]
    constraints += [cp.trace(P) == 1]
    prob = cp.Problem(cp.Minimize(objective), constraints)
    prob.solve()

    L = sqrtm(P.value)
    return L
```

```python
def NDR(X, m, d, use_LLE=True, k=5, variance=0.0):
    """
    X is n by p data matrix
    m is the desired embedding after LLE or Laplacian Eigenmap
    d is the final desired embedding
    use_LLE determines whether LLE or Laplacian Eigenmap is used
    k is number of neighbors
    variance is gaussian variance for Laplacian Eigenmap weight matrix
    """

    Y = LLE.NDR(X, m, k) if use_LLE else Laplacian_Eigenmap.NDR(X, m, k, variance)
    L = semidef_prog(X, Y, k)
    Z = Y @ L # multiply row vectors by symmetric matrix L

    pca = PCA(n_components=d)
    Z_reduced = pca.fit_transform(Z)
    return Z_reduced
```

Figure 3: Sha-Saul Implementation

In this problem, the objective function is

$$D(s) = \sum_{i} \sum_{j,j'} \eta_{ij}\eta_{ij'}(|z_j - z_{j'}|^2 - s_i|x_j - x_{j'}|^2)^2. \quad (4)$$

where z_j's are a linear combination of the output $y_i \in R^m$ from LLE, i.e., $z_i = Ly_i$ for some $m \times m$ matrix L and for all i. We can replace all the z_j's by Ly_i in the above function, and solve the equations

$$D_{s_i}(s) = 0,\ i = 1, \dots, n \quad (5)$$

to replace s_i by some expression of z_j's, and ultimately some expressions of L. Then the objective function only depends on L. Since a semidefinite programming program is included in the convex optimization problems, I can consider this problem as a convex optimization problem. There is a library called cvxpy that can solve such problems and we did a detailed computation to solve Equation (5) to formulate this problem in the language of cvxpy, and thereby complete the final semidefinite programming part.

Numerical Result

To test this method, we will demonstrate it on a non-uniformly curved 1D manifold in 2D space.

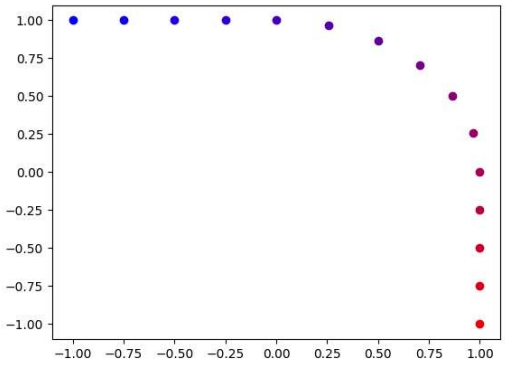

Figure 4: Original data

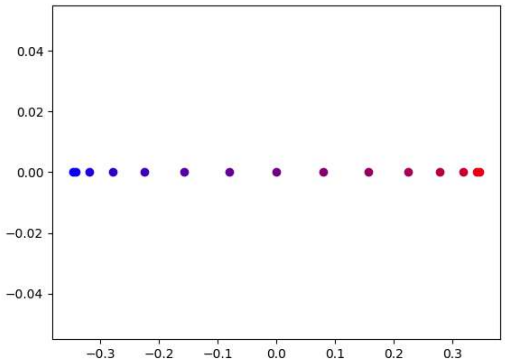

Figure 5: Data transformed with LLE, d=1, k=2

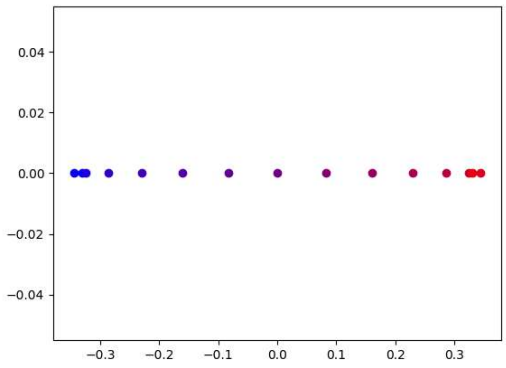

Figure 6: Data transformed with Laplacian eigenmap, d=1, k=2, variance=1.0

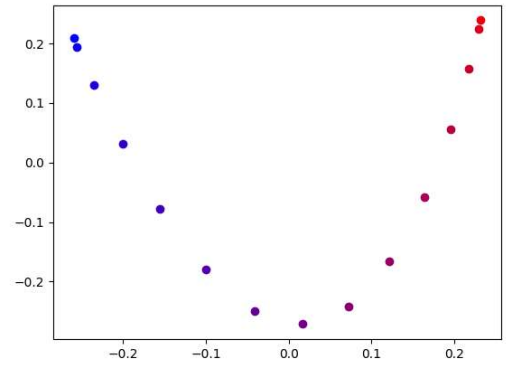

Figure 7: Data transformed with Sha-Saul method, m=2, d=2, k=2, LLE implementation

Conclusion

Through our implementation, we see that the former implementations (LLE and Laplacian eigenmaps) are capable of straightening out a manifold in a higher-dimensional space. Comparatively, the Sha-Saul can record both position along the manifold (X-axis) and angle along the manifold (Y-axis). Despite the qualitative correctness of the dimension reduction in the proposed method, the extended spectral method suffers greatly in terms of runtime. In our tests using randomly generated data along a simple 2D distribution, we can already see suboptimal performance arising from the implementation due to the drawback of unoptimized semidefinite programming. With respect to n, the runtime of our implemented semidefinite programming method has a comparatively high time complexity. This is expected, though, by the description of the algorithm for semidefinite programming in Vandenberghe and Boyd's paper "semidefinite programming", [5].

In future research, the group would like to explore possible optimizations in runtime and space complexity for the semidefinite programming problem solution. Additionally, we recognize that our dataset, while easily large in sample size, is limited by our ability to produce real-sourced data with high dimensionality. Any constant overhead costs, though maybe spanning minutes of compute time in our small sampled tests, are not currently compared against datasets with an inherently large compute time and size when applying LLE and Laplacian Eigenmaps. Thus, we would like to explore the runtime comparisons of extended and unextended spectral methods using vastly large datasets, especially ones that stress the importance of angle and geometric relationship preservation, such as facial or object images for classification available via sources such as Kaggle. Other datasets that may viably show an accurate comparison of dimension reduction methods can also be polygon-shape data or 2D black-and-white hue images like Sha & Saul use in their Experimental Results through their 2D crack images.

Finally, we would like to compare and possibly build our implementations on previously optimized and widely available libraries such as scikit-learn's spectralEmbedding [6] and LocallyLinearEmbedding [4] methods. These would offer an optimized baseline of comparison so that, depending on the further optimization of our semidefinite programming solution, we

could have an accurate runtime comparison that most closely reflects usage amongst those using these machine learning libraries.

References

[1] Caley, S., Shi, M., & Wang, J. (2024, November). *Spectral NDR Group Repository*. GitHub. https://github.com/scottcaley/Spectral_NDR

[2] Cohen, M. B., Lee, Y. T., & Song, Z. (2019). Solving linear programs in the current matrix multiplication time. *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 938–942. https://doi.org/10.1145/3313276.3316303

[3] *CVXPY API Documentation*. cvxpy.org. (n.d.). https://www.cvxpy.org/api_reference/cvxpy.html

[4] *LLE Scikit Documentation*. scikit. (n.d.-a). https://scikit-learn.org/1.5/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html

[5] Sha, F., & Saul, L. K. (2005). Analysis and extension of spectral methods for nonlinear dimensionality reduction. *Proceedings of the 22nd International Conference on Machine Learning - ICML '05*, 784–791. https://doi.org/10.1145/1102351.1102450

[6] *Spectral Embedding Scikit-Learn Documentation*. scikit. (n.d.-b). https://scikit-learn.org/1.5/modules/generated/sklearn.manifold.SpectralEmbedding.html