Manifold Learning Based Image Classification

ECE-656

SUBMITTED BY: SONU DILEEP

SUBMITTED ON : 05/15/2020

I. INTRODUCTION

Image classification is one of the most researched topics in the field of computer vision and image processing. It basically classifies an image according to the content in an image. There are many traditional machine learning algorithms as well as highly efficient deep learning models which helps in image classification. Accuracy and speed are the two of the important metrics we consider in an image classification model. So therefore, we need to keep a balance between these 2 metrics to choose a classifier model. We cannot simply increase one at the cost of other. We know that images are points in high dimensional space. For example, an image of size 600 x 600 pixels is a point in 360,000-dimension space. This high dimension space considerably affects the speed of any image classification model. This also takes a lot of memory space. So many studies have been carried out which showed that this high dimensional representation can be mapped to a relatively lower space but still preserve most of the topological information in an image. Many dimensionality reduction techniques exist and one of the most widely used Nonlinear dimensionality reduction approach is Manifold Learning. This project focuses on analyzing and studying three different manifold learning techniques for image classification. The three manifold learning technique studied in this project are :- **1)** Multidimensional Scaling (MDS) **2)** Isomap **3)** Locally Linear Embedding(LLE). All three algorithms map the original high dimensional data into a low dimension differently. Finally, this low dimension data is used to train a k-Nearest Neighbor(k-NN) image classifier. The speed and accuracy of these models will be compared to a model trained using original high dimensional dataset.

II. THEORY

Manifold Learning is one of the most exciting topics in machine learning. It is a technique for non-linear dimensionality reduction. They belong to a class of unsupervised estimators which helps to capture the non-linear structure in data. Algorithms for this task are based on the idea that the dimensionality of most data sets is only artificially high. Manifold learning algorithms map high dimensional data to a low dimensional manifold space which captures all the essential characteristics of the data. The basic idea of this project is to see whether we can apply manifold learning techniques for image classification. If the accuracy of the model using high dimensional data and the low dimensional data are almost same, then it could save a lot of training time as well as classifying time. Also, it could also save a lot of computing power and memory. The dataset that is used for this project is Fruits-360 dataset from Kaggle. The image classifier that is used is k-NN. It is a simple machine learning algorithm that can be used for image classification. For this project I am working with three manifold learning algorithms and they are:- MDS, Isomap and LLE. I will create 3 low dimensional datasets of the original Fruits-360 dataset using these three algorithms. Then I will train 4 k-NN image classifiers, one with the original dataset and the rest with the dimensionally reduced datasets. The training speed, classifying speed and accuracy of all four models will be compared. Before going further, I will explain how the three manifold learning algorithms and the k-NN classifier works.

Multidimensional Scaling (MDS)

MDS is a multidimensional representation of the distances or dissimilarities between a set of data. It uses the pairwise distance between each data in a set and maps it to a low dimensional space so that their distances are preserved. They are mainly used for visualizing information lying in a high dimension space in a low dimension space (usually 2-D or 3-D) where it will be easy for us to analyze. So, the first step to use MDS is to make a distance matrix of the given dataset. Distance matrix basically contains the distance between a pair of points in the given high dimensional space. In classical MDS, Euclidean distance is used for calculating distances. These distances are invariant to linear transformation of images/data like rotation, translation. Then we choose a value for ‘N’, where n represents the dimension space to which we want to map the given data. Let ‘M’ be the total number of samples in the given dataset. Let $d_{i,j}$ represents the distance between data point i and j in some high dimensional space. Let xi, xj be the representation of the points ‘i’ and ‘j’ in low dimensional space. The optimal representation in the low dimension space i.e. finding optimal values of xi and xj can be formulated as an optimization problem with a cost function shown below.

$$\min_{x_1,\dots,x_M} \sum_{i<j} (\|x_i - x_j\| - d_{i,j})^2$$

$x_i - x_j$ is the distance between those points in the new space. So, minimizing the error between distances helps to preserve the distance when we map from high dimension to low dimension space. If we are mapping into a low dimension space like 1,2 or 3-D (a space where humans can understand visually) then we can see that similar images will be grouped together. After finding the distance matrix D there is an optimization technique which helps us to find ideal low dimension representation X using eigen decomposition. For that we first apply double centering on our distance matrix using a centering matrix and we get a matrix B. Find ‘m’ largest eigen values and their corresponding eigen vectors of the matrix B, where m corresponds to the dimension to which we want to map the high dimension data. Using this eigen value and vectors we can find the X matrix using the equation shown below.

$$X = E_m * (\Lambda_m)^{1/2}$$

where E_m is the eigen vectors and Λ_m corresponding eigenvalues of B. Training a k-NN model using these low dimensions dataset helps to classify the images fast.

Locally linear embedding (LLE)

LLE is an unsupervised learning technique which computes the low dimensional, neighborhood preserving embeddings of a high dimensional space. It uses the local symmetry of linear reconstructions to find the nonlinear patterns in high dimensional space. They are a class of eigen vector method for nonlinear dimensionality reduction. The algorithm basically works this way:-

1. The algorithm first finds the neighbors of each data point,Xi . We can specify the number of neighbors we want to choose before running the algorithm. We find the distance between points Xi and Xj and choose k closer neighbors.
2. The next step is to find the weight vector Wij that best reconstruct each data point Xi from the neighbors. We see this as an optimization problem where we try to optimize the cost function shown

below. Here Xi is a data point and Xj are the neighbors. For optimum value of Wij, Wij * Xj will be equal to Xi. These weight vectors are invariant to linear transformations like rotation, translation.

$$\mathcal{E}(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

3. Now have a matrix of optimum weight vectors. The next step is to find low dimensional embedding points Yi .Here also we use optimization techniques which minimizes the cost function given below . We keep W_{ij} fixed and adjust values for Yi and Yj that minimizes the cost function given below.

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2$$

This minimization problem can be solved by solving sparse N X N eigen value problem (N being the number of data points), whose bottom d nonzero eigen vectors provide an orthogonal set of coordinates. Generally the data points are reconstructed from K nearest neighbors, as measured by Euclidean distance.

Isomap

Isomap is also a nonlinear dimensionality reduction technique. Isomap stands for isometric mapping. It is based on spectral theory which tries to preserve the geodesic distances in a low dimension space. They create a neighborhood network and approximate the geodesic distance between pair of data points. They then form a distance matrix. Isomap uses eigen value decomposition of the distance matrix to find the low dimension encoding of the high dimension space. Isomap works in similar way to MDS algorithm except that in classical MDS we use Euclidean distance and in Isomap we use geodesic distances. The term geodesic comes from graph theory and it means that distance between two vertices is the number of edges in a shortest path. After finding the geodesic distance matrix we follow the same steps as in MDS for finding the X matrix using eigen value decomposition. So, the algorithm in Isomap follows like this:-

1. First choose a value of k and it determines the number of neighbor points we want to consider.
2. Then construct a neighborhood graph where each data point is connected to k nearest neighbors.
3. Compute the shortest path between pair of nodes.
4. Formulate the distance matrix and follow the same steps as in MDS to calculate X matrix. Here X is the

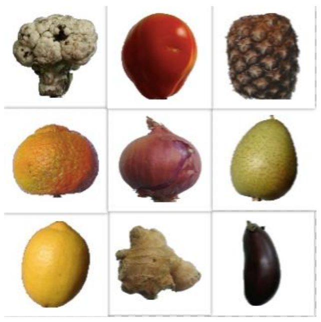
matrix with low dimension embedding of the high dimensional data.

K-Nearest Neighbors (k-NN)

k-NN is one of the most fundamental algorithms in machine learning. They are a class of supervised ML algorithm and they can be used for both classification as well as regression. Their outputs are easy to interpret as well is fast. Classification using k-NN is performed using votes of its neighbors. k-NN uses feature similarity to classify new data which means that it checks how the new data matches the points in training data. k-NN algorithm works as follows:-

1. The first phase is training/loading the train and test dataset. We need to choose a value for k such that k>1. K corresponds to the number of neighbor data points to include in the voting process.
2. For testing, calculate the distance between test data and all the training set. Distance can be calculated using Euclidean, Manhattan or Hamming equations. Then find k points in training set which are closer to the testing data.
3. Predictions are made through voting from those ‘k’ nearest neighbors. If a greater number of votes are received from neighbors belonging to a particular class, then that data point belongs to that class.

So far, we discussed the different manifold learning algorithms as well as about the k-NN classifier used in the project. I used Fruits-360 dataset from Kaggle for testing all the models. It is a dataset which contain images of 100+ different fruits. The first step in the project is to implement a k-NN classifier. I used python to implement the classifier. I split the Fruits 360-dataset into training and validation and found the training/loading time, validation accuracy and validation speed. It is the original model test on the high dimensional dataset. After that I applied the three manifold learning algorithms, MDS, Isomap and LLE on the training and validation dataset and made three different datasets which lies in a low dimensional space. Then I tested the performance of k-NN algorithm using these three datasets. The performance of the three models are compared with the original model. I started with embedding the high dimension images into a 2-D space and compared their results. I mapped the dataset in to these dimensions- 2,4,8,16,32 and 64-D space. Compared the speed and accuracy of all the models on the original model. A detailed discussion of the results is given below. These are few of the images in the Fruits-360 dataset.



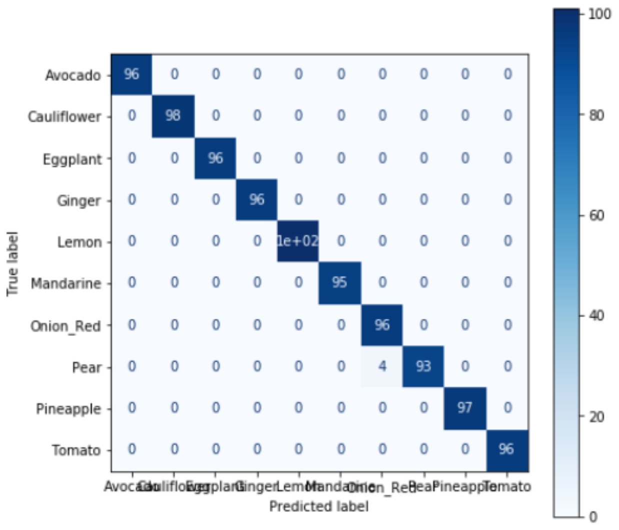
III. RESULTS

I started the experiment with implementing a k-NN classifier and then trained/loaded the Fruits-360 dataset into it. All implementation are done using python. Fruits-360 dataset comprises around 82,197 images of 120 Fruits. Each image is of size 100 x 100 pixel. I used only first 10 Fruits of the Fruits 360 dataset since there are more than 100+ fruits in the dataset and it will be difficult to plot them in the report.I split the dataset into 60% for training, 25% for validating and the rest 15% for testing. I tested the model for different values for k i.e the number of neighbor parameter and found the optimum model and finally checked the performance of model on validation set. Training speed, Validation speed and

Validation accuracy of the different models are compared. Finally a confusion matrix of the best performing model is plotted. For k-NN, training time actually means loading time. k-NN does not do any training at all. It basically memorizes all the training data. Shown below is a table showing the training time, validation accuracy and validation time that k-NN model took for different values of k.

Using Original Fruits-360 dataset			
k value (k-NN)	***Training time(sec)***	***validation time(sec)***	***validation accuracy(%)***
1	0.162	9.16	99.58
2	0.122	9.46	99.43
3	0.117	9.43	99.42
4	0.132	9.48	99.34
5	0.123	9.39	99.47
6	0.120	9.47	99.09

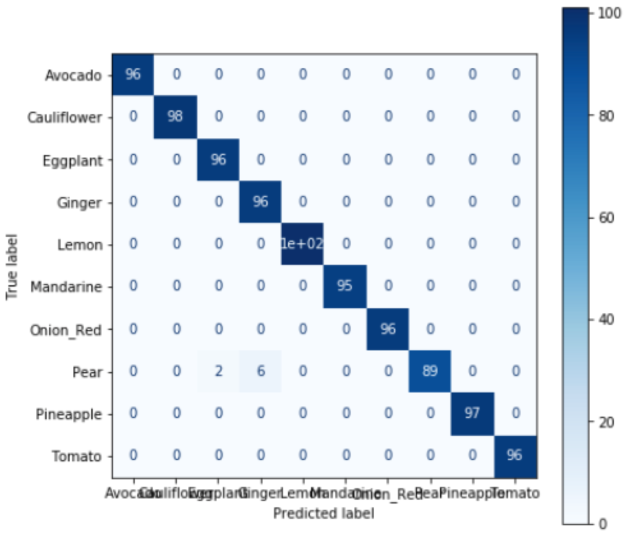
From the table we can see that a k-NN model with k=1 gave us the highest validation accuracy. The validation time for that model was around 9.16 seconds. Plotted below is the confusion matrix of that best performing model on validation dataset.



The next step in the project is to make the low dimensional embedding of the training and validation dataset using manifold learning algorithms. I started with locally linear embedding and made a dataset which lies in a low dimensional space. I tested different models by varying the number of components. Components represent the size of low dimensional space. I mapped the high dimensional space into 2,4,8,16 and 32 dimensional space. validation accuracy is shown below.

Using LLE Embedded dataset			
Number of components	***Training time(sec)***	***Validation time(sec)***	***Validation accuracy(%)***
2	0.0021	0.040	96.68
4	0.0019	0.046	98.50
8	0.0021	0.0420	99.17
16	0.0020	0.0413	97.72
32	0.0021	0.0494	96.90

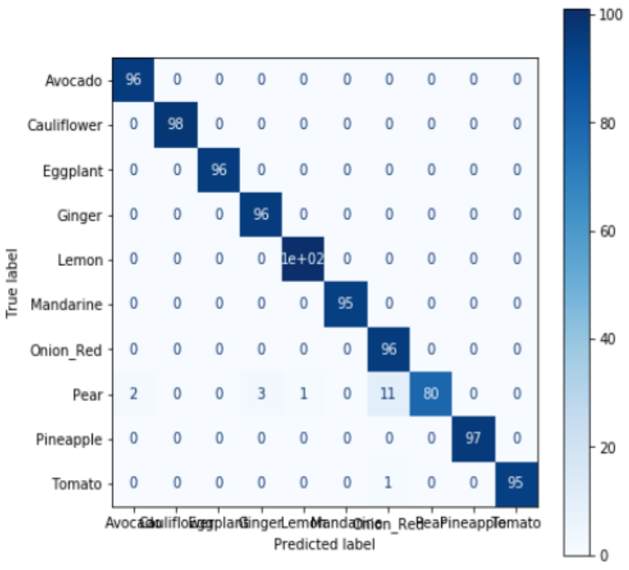
From the table we can see that the validation accuracy of LLE embedded dataset is close to the accuracy of original dataset. The model performed best when we mapped the images into an 8-D space. But compare the validation and training time for both models. We can see that best performing model with original dataset took around 9.16 seconds for validation, but our new dataset i.e. after applying manifold learning algorithms took only 0.040 seconds. It is very low compared to our original dataset, also if we compare the accuracy of the best performing model, we can see that the difference is very small. Plotted below is the confusion matrix of the best performing model.



In the next step I made a low dimensional embedding of the dataset using Multidimensional scaling algorithm. I followed the same steps as before. I tested the model for different number of components and found an optimal model with highest accuracy. Plotted below is table with results of each model.

Using MDS Embedded data set			
Number of components	***Training time(sec)***	***Validation time(sec)***	***Validation accuracy(%)***
2	0.003	0.034	94.97
4	0.0005	0.034	98.96
8	0.0005	0.0361	97.0085
16	0.00066	0.04023	98.07
32	0.00064	0.0427	98.504

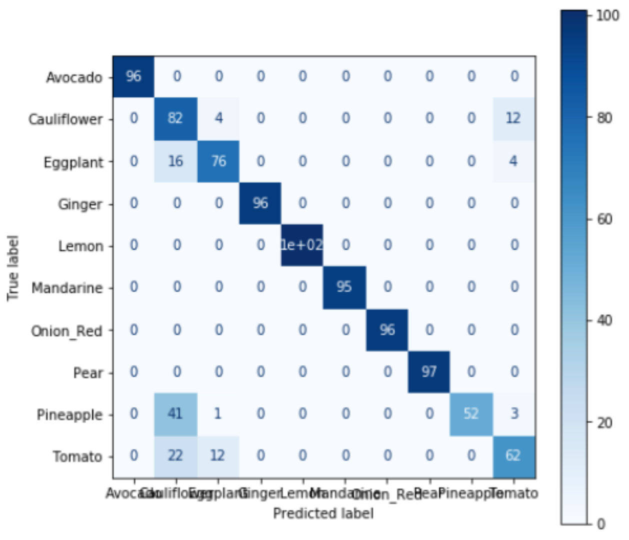
From the table we can see that the optimal model is one with ‘4’ number of components i.e. when the dataset is mapped to a 4-D space. The accuracy of the best performing model is 98.96% and the total time taken is 0.034 seconds. The accuracy is slightly low compared to the one where we applied locally linear embedding algorithm. Plotted below is the confusion matrices of the best performing model on validation set.



True label	Avocado	Cauliflower	Eggplant	Ginger	Lemon	Mandarine	Onion_Red	Pear	Pineapple	Tomato
Avocado	96	0	0	0	0	0	0	0	0	0
Cauliflower	0	98	0	0	0	0	0	0	0	0
Eggplant	0	0	96	0	0	0	0	0	0	0
Ginger	0	0	0	96	0	0	0	0	0	0
Lemon	0	0	0	0	1e+02	0	0	0	0	0
Mandarine	0	0	0	0	0	95	0	0	0	0
Onion_Red	0	0	0	0	0	0	96	0	0	0
Pear	2	0	0	3	1	0	11	80	0	0
Pineapple	0	0	0	0	0	0	0	0	97	0
Tomato	0	0	0	0	0	0	1	0	0	95

The next algorithm that I used is Isomap. I followed the same methods as before to find the optimal model. Plotted below is a table of results.

Using Isomap Embedded data set			
Number of components	***Training time(sec)***	***Validation time(sec)***	***Validation accuracy(%)***
2	0.0005	0.0351	82.79
4	0.0007	0.0354	85.491
8	0.0005	0.0371	86.77
16	0.0005	0.0368	87.08
32	0.0006	0.0427	91.63



True label	Avocado	Cauliflower	Eggplant	Ginger	Lemon	Mandarine	Onion_Red	Pear	Pineapple	Tomato
Avocado	96	0	0	0	0	0	0	0	0	0
Cauliflower	0	82	4	0	0	0	0	0	0	12
Eggplant	0	16	76	0	0	0	0	0	0	4
Ginger	0	0	0	96	0	0	0	0	0	0
Lemon	0	0	0	0	1e+02	0	0	0	0	0
Mandarine	0	0	0	0	0	95	0	0	0	0
Onion_Red	0	0	0	0	0	0	96	0	0	0
Pear	0	0	0	0	0	0	0	97	0	0
Pineapple	0	41	1	0	0	0	0	0	52	3
Tomato	0	22	12	0	0	0	0	0	0	62

From the table we can see that the optimal model has an accuracy of 91.63% on the validation dataset. It took only 0.0427 seconds to complete. This was the least performing model compared to MDS and LLE since it has the lowest accuracy. It gave the best results when mapped the dataset into an 8-D space. So, from all the results we can conclude that for the Fruits -360 dataset ,LLE algorithm gave us the best validation accuracy followed by MDS and then last Isomap. But if we are taking into consideration the time also with a bit tradeoff in accuracy, MDS performed better. The best model of MDS took only 1/4 time of the best performing model of LLE still with an accuracy of 98.96%. At last I checked the performance of optimum model using test dataset. Using LLE algorithm with '8' number of components gave a testing accuracy of 97.83% compared to 98.76% using the original dataset.

The advantages and disadvantages of all three algorithms are discussed below :-

Multidimensional Scaling (MDS)

Advantages

- Easy visualization and implementation.
- Does not require the raw data but a matrix of pairwise distance or dismilarity helps to visualize the points.
- Low computing Time
- Linear Transformation will not affect the result

Disadvantages

- It fails when there are nonlinear transformations in the data.

Locally Linear Embedding (LLE)

Advantages

- More accurate in preserving local structure than other two algorithms
- Less computationally expensive than ISOMAP.

Disadvantages

- Less accurate in preserving global structure than other two algorithms

Isomap

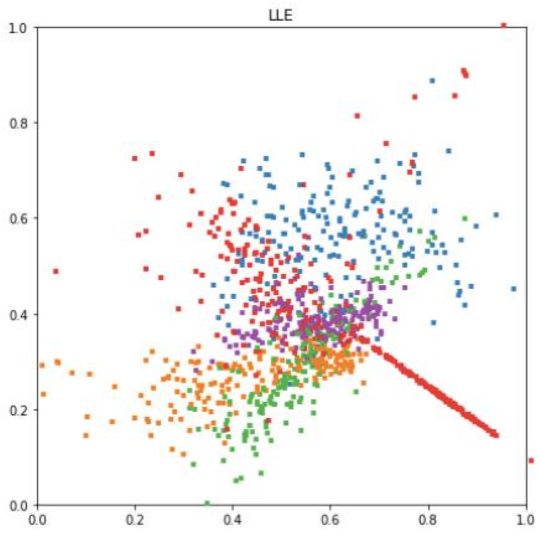
Advantages

- Easy to understand, they are a simple extension of MDS.
- Preserves true relationship between datapoints.

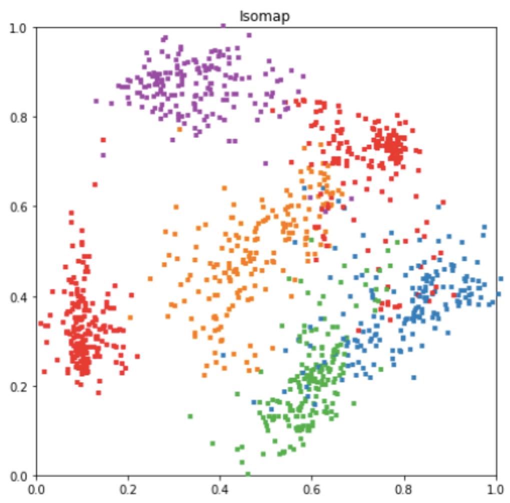
Disadvantages

- Computationally expensive
- They have difficulties with holes.

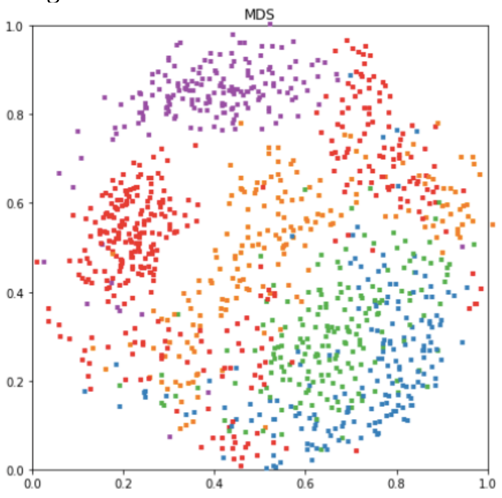
Manifold Learning algorithms are mainly used for visualization of the high dimensional data. Visualization in a high dimension does not make much sense for us humans. Normally what we understand is representations in 1-D, 2-D, or 3-D. Manifold learning can be used to visualize the data in a lower dimension. I have plotted the 2-D representation of the original 10,000-D images below. The different colors in the plot represents different fruits. I have plotted only a few images and few fruits for a compact representation in this report. The first plot shown below is the 2-D representation of the images after encoding using Locally Linear Embedding (LLE).



The second plot shown below is the 2-D embedding of data using Isomap.



The last plot shown below is the 2-D embedding of data using MDS.



IV. CONCLUSION

In this report we discussed three different manifold learning algorithms as well as we saw how it improves the image classification model. We used k-NN algorithm for image classification and saw how manifold learning techniques improves the performance of k-NN algorithm. Through the results we saw how reducing the dimensionality of the datasets considerably reduces the training time as well as testing time. We concluded that out of three manifold learning techniques LLE gave us the best classification accuracy. Also, the model took only 1/400th of the time taken on the original dataset. Considering the fact that there is a huge amount of data coming out every day and, storing and processing this much data in a high dimension space takes a lot of memory power and time, there is a much need to focus researches in the field of manifold learning techniques. For future research, an efficient way to use manifold learning with deep learning can be studied so that it opens new path for representing even the features in complex images.

V. REFERENCES

[1] (SIU), pages 1–4. IEEE, 2018.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] https://jakevdp.github.io/PythonDataScienceHandbook/05.10-manifold-learning.html

[3] https://scikit-learn.org/stable/modules/manifold.html

[4] An Introduction to Locally Linear Embedding, Lawrence K. Saul AT&T Labs – Research

[5] https://blog.paperspace.com/dimension-reduction-with-isomap/

[6] https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction#Locally-linear_embedding

[7] https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm