Project 3 - Unsupervised learning

August Jonasson

November 1, 2024

Task 1 (ISOMAP or LLE)

We are going to use LLE for the first task.

A)

As opposed to the CTD embedding, the graph when using LLE doesn't have to be fully connected. It also doesn't have to be undirected (symmetric adjacency matrix). Because of this we can use the regular (not mutual) k nearest neighbor in order to connect the data points. Using this method, note that the dimensionality of the embedding will be strictly less than the number k (Roweis and Saul 2000). Because of this, since we expect to unfold the swiss roll into a two-dimensional representation, the smallest number we can use is $k = 3$. On the other hand, we also have to select k sufficiently small such that points in different "layers" (see Figure 1) of the roll do not connect directly to each other. This would defeat the purpose of reconstructing points based on local linearity, since these connections would connect through space that isn't part of the manifold. Some experimentation yields that $k = 20$ is the largest k we can choose such that this still holds.

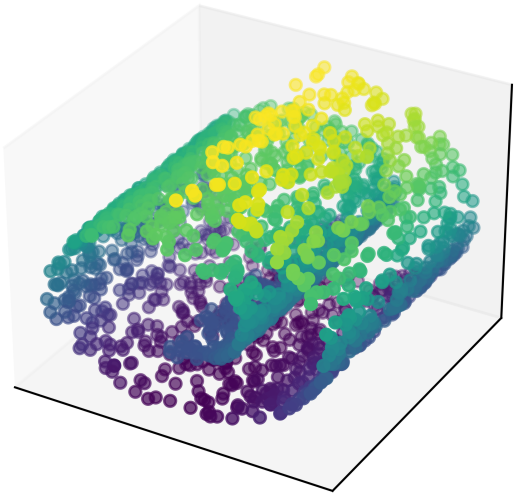

Figure 1: Swiss roll data

B)

The LLE embedding onto two dimensions was made using the `LocallyLinearEmbedding` function from the `sklearn.manifold` library and the result can be viewed in Figure 2. The only hyperparameter in this method is k and this was set to 20, as explained in the previous sub-task.

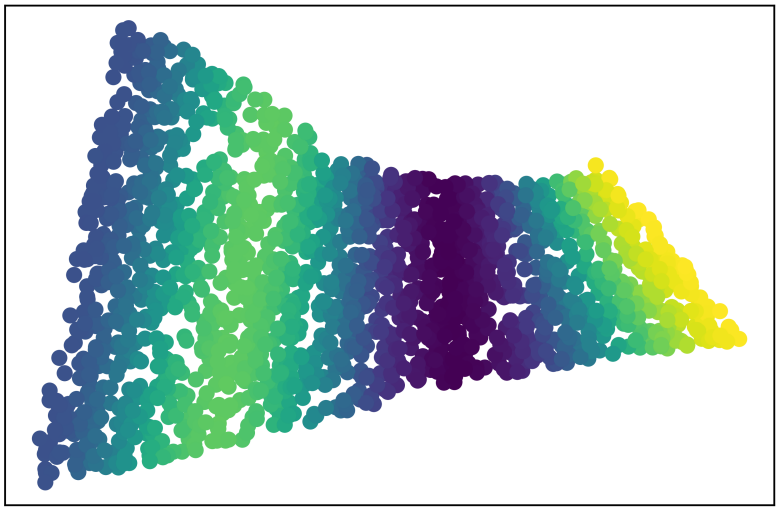

Figure 2: LLE embedding of swiss roll data using nearest neighbor method with $k = 20$

C)

Comparing the colors in Figure 1 to the colors in Figure 2 we can tell that the swiss roll was successfully unfolded, albeit a bit distorted. In the LLE embedding we see a rather uniform distribution of the data points as opposed to the "bubbles" (or "holes in the web") that were produced by the CTD embedding. We could also mention that whereas the CTD embedding was pretty sensitive to the number k - increasing or decreasing k by a single unit could drastically change the results - the LLE embedding produce rather similar results for a relatively speaking wide range of k:s, e.g. $k \in (10, 11, \ldots, 30)$.

D)

Varying the number k by ± 15 yields the following results. Figure 3 tells us that setting k too low fails miserably in preserving the geometric properties of the manifold. A k this low leads to fragmenting of the data structure and as such does not lead to good representations of local neighborhoods, and the following linearity assumption fails.

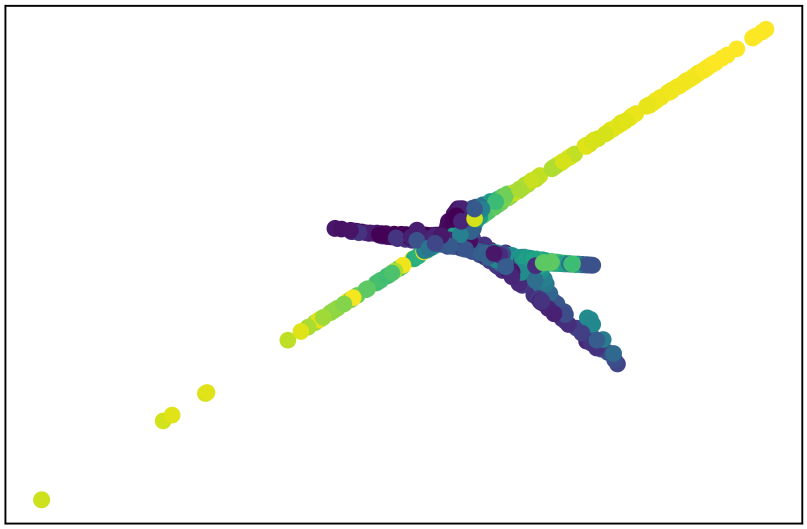

Figure 3: LLE embedding of swiss roll data using kNN $k = 5$

When increasing k too much, as we have mentioned earlier, we start seeing connections between points that are not "along" the swiss roll. This also heavily breaks the assumption of local linearity since we introduce far-away points that go directly against the linearity. In Figure 4 we can see the resulting embedding for such a k. Instead of unfolding the swiss roll, this more resembles what would happen if one were to step on it, thus not preserving its topology.

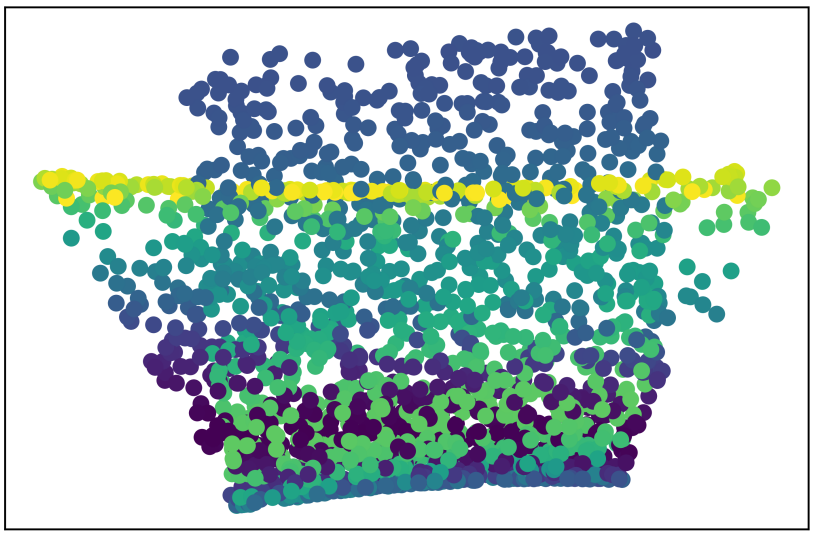

Figure 4: LLE embedding of swiss roll data using kNN $k = 35$

Task 2 (Density-based Clustering)

Based on Figure 5 we decide to go with the regular DBSCAN method for clustering the data. We choose this method over the density peak method primarily because of the fact that using density peaks looks like it might be susceptible to returning several density peaks for one and the same cluster. Also, the clusters look to be of similar densities, which is very appropriate for DBSCAN.

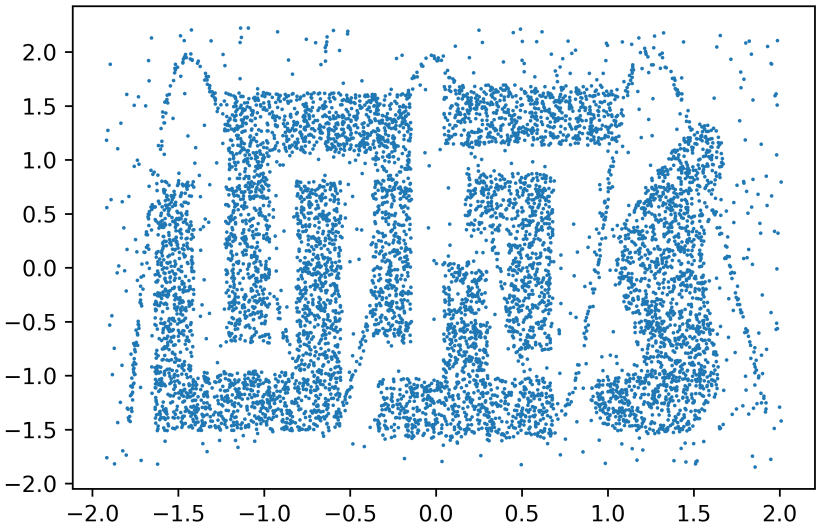

Figure 5: Arbitrary shape data set (standardized)

A)

For DBSCAN there are two hyperparameters to consider: the distance that defines neighborhoods ϵ, and the minimum number of points to define a cluster core point `minPoints`. The distance measure that we will be using is the Euclidean distance.

The formal way of tuning the hyperparameters for the DBSCAN is to perform a grid search in combination with clustering validation. As we shall see in a later sub-task, however, using internal measures such as the silhouette score (based on cohesion and separation) doesn't work very well for these clusters because of how they sort of link together. Consider the within-cluster-distances and between-cluster-distances of the two leftmost clusters in Figure 5. They will not provide us with useful information.

We still turn to the grid search, but instead of using the validation in each step we simply rely on the fact that this data set is easy to inspect. We can tell from Figure 5 that there are six clusters. As such, we only have to look at the values of ϵ and `minPoints` that return six clusters, and then choose whichever one looks the best. Doing this we arrive at the parameter settings $\epsilon = 0.08$ and `minPoints` $= 14$. We could have also used the heuristic approach from Ester et al. 1996, but since this also lacks validation, and requires us to fix `minPoints` ourselves, we deem this to be equivalent to our method in this case, since we can actually look at the data.

B)

From Figure 6 we can see the resulting clustering of the DBSCAN method with the set hyperparameters. The black points indicate noise.

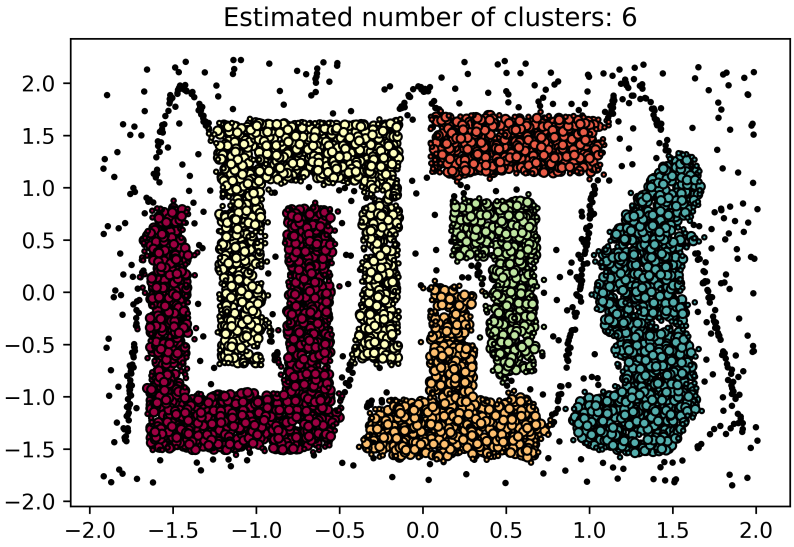

Figure 6: DBSCAN clustering of the 'arbitrary shape' using $\epsilon = 0.08$ and `minPoints` $= 14$

C)

Before performing validation on the clustering of the previous task, we remove all of the noise points. We perform the validation on the data set that can be seen in Figure 7.

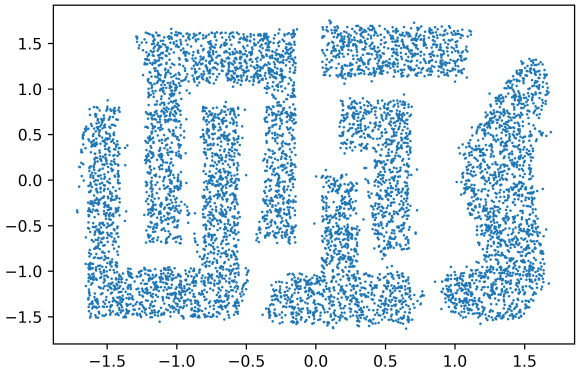

Figure 7: Arbitrary shape with noise removed according to DBSCAN result

In Figure 8 we can see the silhouette plot that results from the DBSCAN clustering and with the

noise removed. If we were to trust this image, we would say that the clustering is pretty bad. The negative values indicate overlapping clusters, which, from Figure 6, we already know not to be true. The average silhouette score over all data points, just exceeding 0.2, also reflects a poor clustering result.

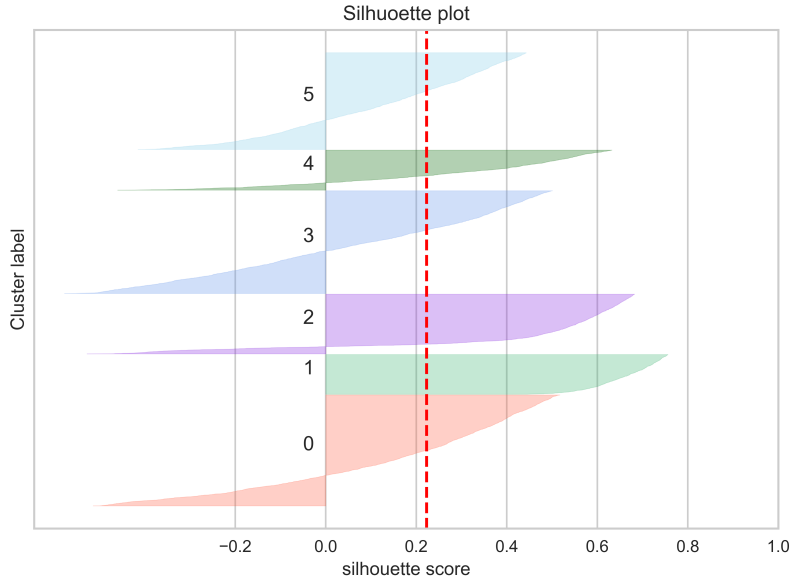

Figure 8: Silhouette plot of DBSCAN clustering

The reason for wrongfully getting overlapping clusters from the silhouette plot is that our clusters are separated in a non-linear manner whereas the distance measure that we use is Euclidean, hence leading to scenarios such as measuring across different clusters when calculating within-cluster distances - which is unreasonable. As such, it is inappropriate to use this validation technique while using the Euclidean distance. In this scenario, they are wrong for each other.

D)

Fixing `minPoints` $= 14$ and varying ϵ with ± 0.03 we can see from Figure 9 that when lowering ϵ both the number of clusters and the number of noise points have increased drastically. This is because there are still sufficiently dense regions such that some clusters are recognized, but the "real" clusters are not dense enough to bind together.

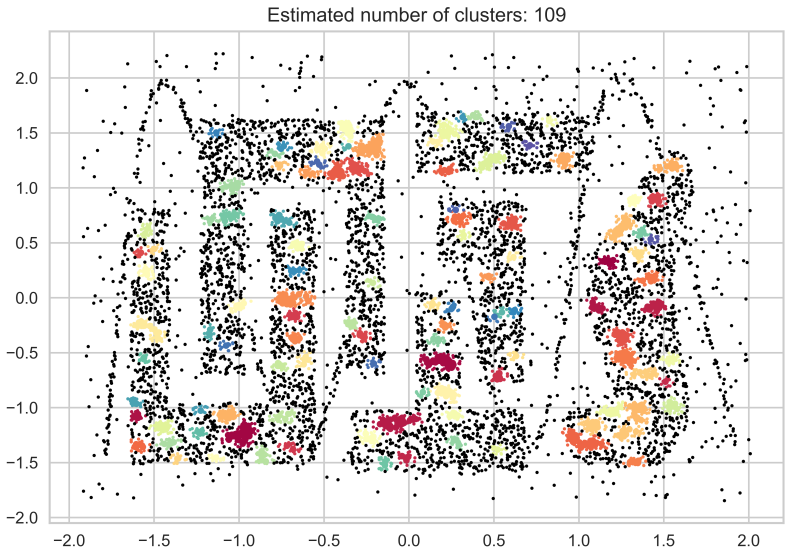

Figure 9: DBSCAN clustering using $\epsilon = 0.05$

When increasing ϵ in Figure 10 the amount of noise remains somewhat unchanged in the outskirts of the data, and the number of clusters have only increased by one. The biggest change here, however, is the cluster assignment. The neighborhood distance is now so far that it binds some of the large clusters together. We can also tell that there are several fake clusters that have sprung up due to this as well. Those regions now include sufficiently many points to be clusters.

Overall, if we set ϵ too low, then everything will be interpreted as noise, and if we set it too high, clusters start blending together and noise turns into clusters.

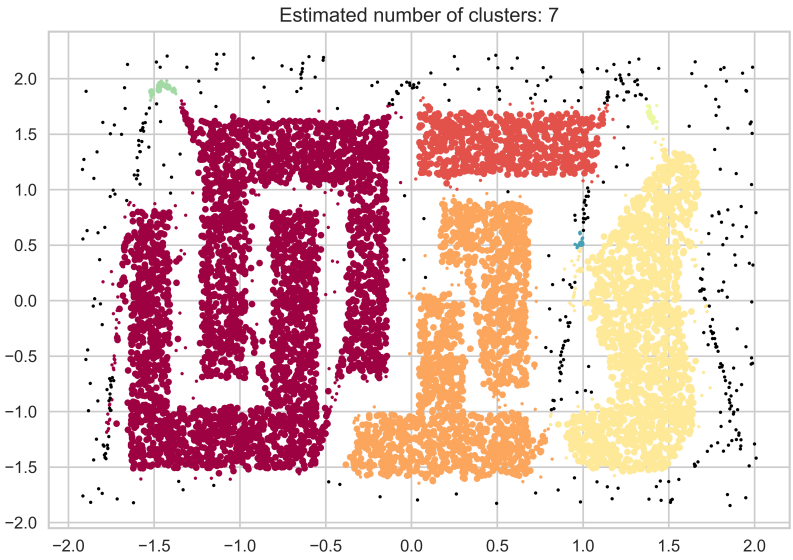

Figure 10: DBSCAN clustering using $\epsilon = 0.11$

Python Code

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.manifold import LocallyLinearEmbedding as LLE

"""
TASK 1
"""

# reading the data into a data frame
df = pd.read_csv("Swiss_Roll.txt", sep='\s+', header=None)
df.columns = ["y1", "y2", "y3"]

# 3d plot of the swiss roll, colored based on z-value ("y3")
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
coloring = df["y3"]
ax.scatter(df["y1"], df["y2"], df["y3"], c=coloring)
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])
ax.figure.savefig("swissRoll", dpi=600)

lle = LLE(n_components=2, n_neighbors=35).fit(df)
embedding_lle = lle.embedding_
reconstruction_error_lle = lle.reconstruction_error_

fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(embedding_lle[:,0], embedding_lle[:,1], c=coloring)
ax.set_xticks([])
ax.set_yticks([])
ax.figure.savefig("higherK", dpi=600)

"""
TASK 2
"""

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("Arbitrary_Shape.txt", sep='\s+', header=None)
df.columns = ["y1", "y2"]
df = np.array(df)
df = StandardScaler().fit_transform(df)

fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(df[:,0], df[:,1], s=0.4)
ax.figure.savefig("arbitraryShape", dpi=600)

# e_incrs = np.linspace(0.05, 0.15, num=15)
# minPoints_incrs = [i for i in range(4,20)]
```

```
57
58 # silh_scores = []
59 # for e in e_incrs:
60 #     for mP in minPoints_incrs:
61 #         db = DBSCAN(eps=e, min_samples=mP).fit(df)
62 #         labels = db.labels_
63 #         silh_scores.append([e, mP, silhouette_score(df, labels)])
64
65 # silh_scores = np.array(silh_scores)
66 # fig = plt.figure()
67 # ax = fig.add_subplot()
68 # ax.scatter(silh_scores[:,0], silh_scores[:,1], s= 0.4)
69
70 db = DBSCAN(eps=0.11, min_samples=14).fit(df)
71 labels = db.labels_
72
73 unique_labels = set(labels)
74 core_samples_mask = np.zeros_like(labels, dtype=bool)
75 core_samples_mask[db.core_sample_indices_] = True
76 n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
77 n_noise_ = list(labels).count(-1)
78
79 colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
80 for k, col in zip(unique_labels, colors):
81     if k == -1:
82         # Black used for noise.
83         col = [0, 0, 0, 1]
84
85     class_member_mask = labels == k
86
87     xy = df[class_member_mask & core_samples_mask]
88     plt.plot(
89         xy[:, 0],
90         xy[:, 1],
91         "o",
92         markerfacecolor=tuple(col),
93         markeredgecolor="k",
94         markersize=4,
95     )
96
97     xy = df[class_member_mask & ~core_samples_mask]
98     plt.plot(
99         xy[:, 0],
100        xy[:, 1],
101        "o",
102        markerfacecolor=tuple(col),
103        markeredgecolor="k",
104        markersize=2,
105    )
106
107 plt.title(f"Estimated number of clusters: {n_clusters_}")
108 plt.savefig("higherEpsilon.png", dpi=600)
109 plt.show()
110
111 # removing the noise from the data
112 idx = np.where(labels != -1)[0]
113 labels_wo_noise = labels[idx]
114 df_wo_noise = df[idx, :]
```

```
115
116 fig = plt.figure()
117 ax = fig.add_subplot()
118 ax.scatter(df_wo_noise[:,0], df_wo_noise[:,1], s=0.4)
119 ax.figure.savefig("arbitraryShapeNoNoise", dpi=600)
120
121 silhouette_avg = silhouette_score(df_wo_noise, labels_wo_noise)
122 each_silhouette_score = silhouette_samples(df_wo_noise, labels_wo_noise)
123
124 colorlist =["tomato","mediumseagreen","blueviolet","cornflowerblue",
125             "darkgreen","skyblue"]
126
127 #Visualization
128 fig =plt.figure()
129 ax = fig.add_subplot(1,1,1)
130 y_lower =10
131 for i in range(n_clusters_):
132     ith_cluster_silhouette_values = each_silhouette_score[labels_wo_noise == i]
133     ith_cluster_silhouette_values.sort()
134     size_cluster_i = ith_cluster_silhouette_values.shape[0]
135     y_upper = y_lower + size_cluster_i
136
137     color = colorlist[i]
138     ax.fill_betweenx(np.arange(y_lower,y_upper),0,ith_cluster_silhouette_values,
    facecolor=color,edgecolor=color,alpha=0.3)
139
140     #label the silhouse plots with their cluster numbers at the middle
141     ax.text(-0.05,y_lower + 0.5 * size_cluster_i,str(i))
142
143     #compute the new y_lower for next plot
144     y_lower = y_upper +10
145
146 ax.set_title("Silhuoette plot")
147 ax.set_xlabel("silhouette score")
148 ax.set_ylabel("Cluster label")
149
150 #the vertical line for average silhouette score of all the values
151 ax.axvline(x=silhouette_avg,color="red",linestyle="--")
152
153 ax.set_yticks([])
154 ax.set_xticks([-0.2,0,0.2,0.4,0.6,0.8,1])
155 ax.figure.savefig("silhouettePlot", dpi=600)
```

References

Ester, Martin et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34, pp. 226–231.

Roweis, Sam T and Lawrence K Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500, pp. 2323–2326.