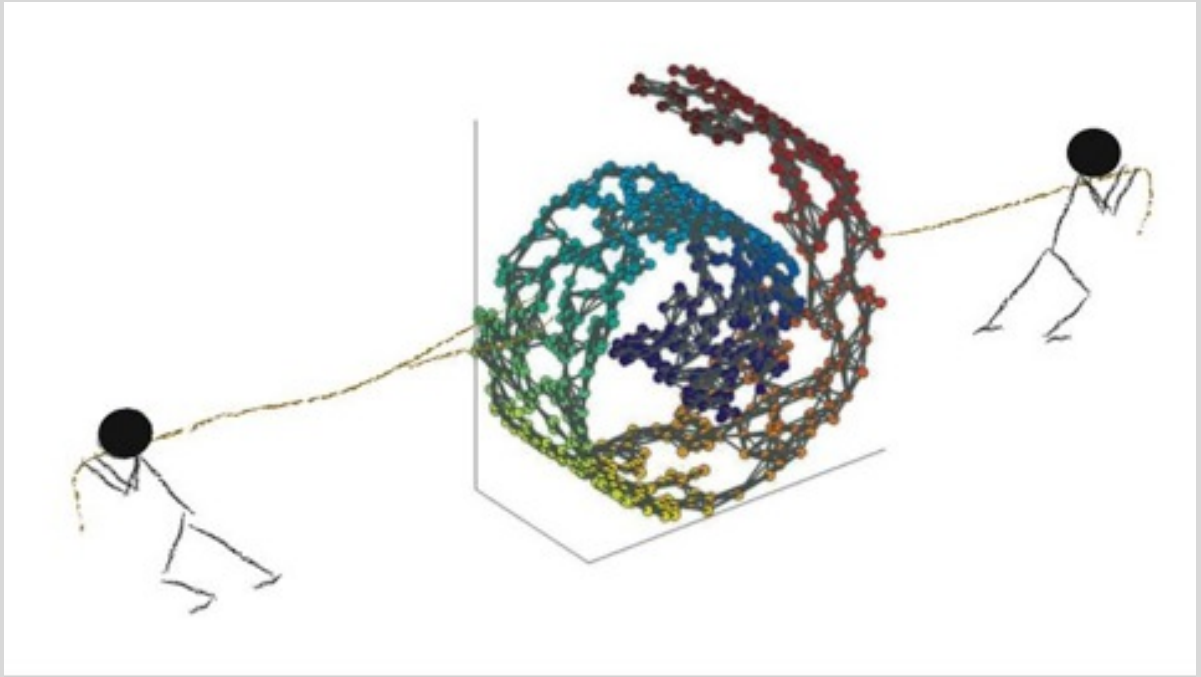


DigitalHouse > Coding School

DATA SCIENCE

Clase 54

Introducción a Manifold Learning

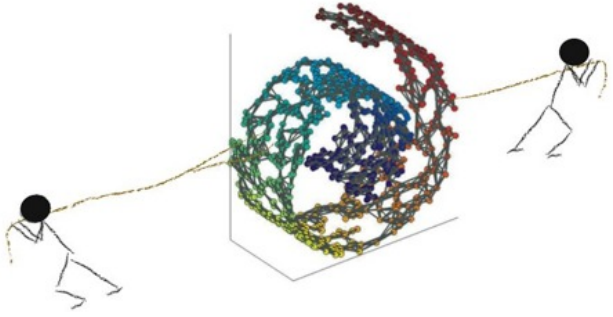
2017

Manifold Learning

OBJETIVOS DE LA CLASE

1. **Presentar algunas técnicas de Manifold Learning: MDS, LLE y IsoMap**

2. **Marcar las diferencias fundamentales con PCA**



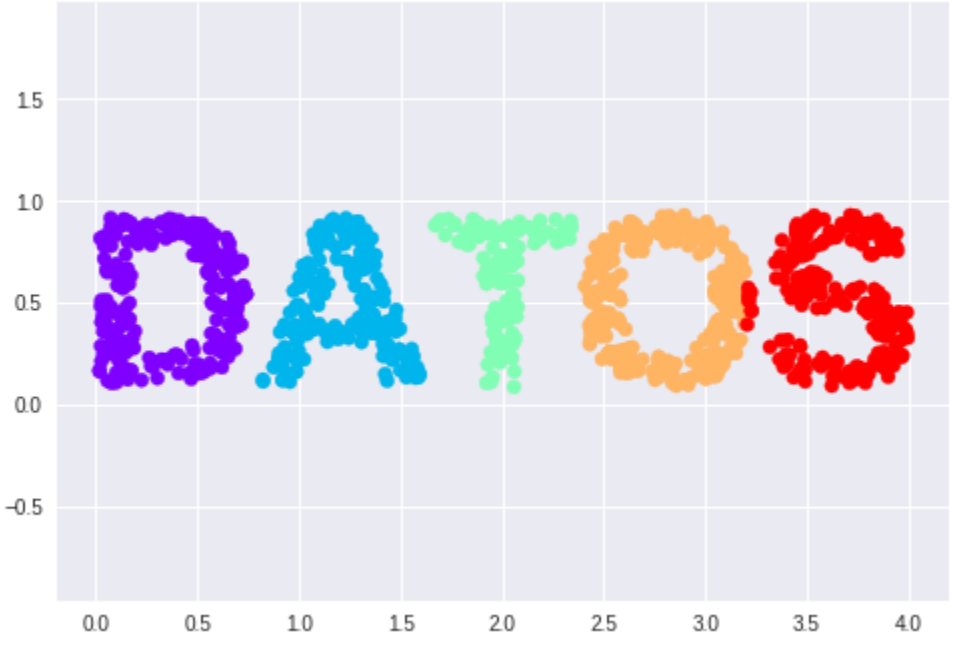
Manifold Learning como reducción de la dimensionalidad

- PCA puede encarar tareas de reducción de la dimensionalidad. PCA no funciona demasiado bien si las relaciones entre los datos exhiben fuertes rasgos de no linealidad.
- Otros métodos llamados "Manifold Learning"
  - Buscan describir un dataset como un geometría de baja dimensionalidad "embebida" en un espacio multidimensional.
  - Podría pensarse en una analogía con una hoja de papel: espacio de dos dimensiones, pero que puede ser rotado, doblado, arrugado sobre esas dos dimensiones en un espacio tridimensional.
  - Rotar o estirar la hoja no altera la geometría "plana" del papel. Se trata de transformaciones lineales.
  - En cambio, doblarla, hacerla un tirabuzón, abollarla son transformaciones no lineales de una geometría bidimensional en un espacio tridimensional.

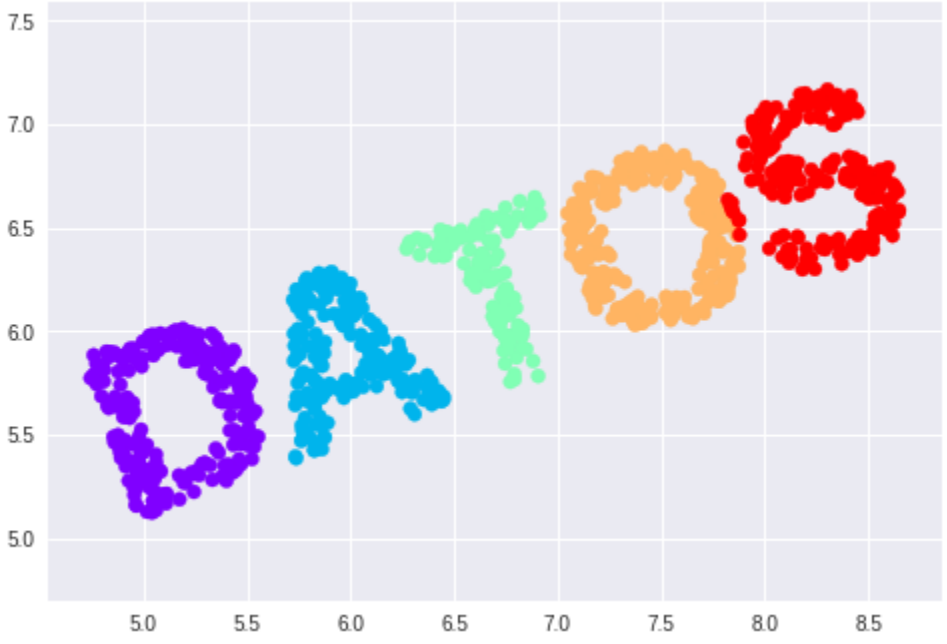
Manifold Learning como reducción de la dimensionalidad

- Los algoritmos basados en Manifold Learning van a tratar de aprender la naturaleza fundamentalmente bidimensional del papel aún si está realizada en un espacio tridimensional.
- Vamos a ver tres técnicas de Manifold Learning:
  - Multidimensional Scaling (MDS)
  - Locally Linear Embedding (LLE)
  - Isometric Mapping (IsoMap)

- Imaginemos que tenemos el siguiente dataset.
- Al observar el scatter, parece surgir que la relación entre X e Y no es la mejor representación de los datos
- Por ejemplo, si rotamos la matriz, la relación entre los puntos se mantiene inalterada



- Imaginemos que tenemos el siguiente dataset.
- Al observar el scatter, parece surgir que la relación entre X e Y no es la mejor representación de los datos
- Por ejemplo, si rotamos la matriz, la relación entre los puntos se mantiene inalterada.

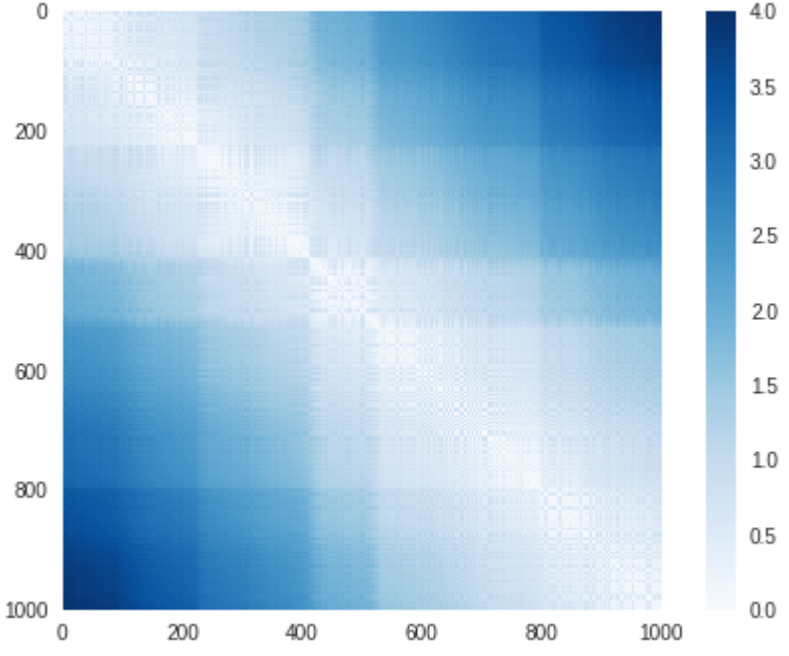


- Los valores X e Y no son fundamentales para extraer la información y las relaciones en los datos.
- En este caso, la distancia entre los diferentes puntos es la información fundamental:
  - la distancia de cada punto contra el resto de los puntos del dataset.
- Una manera común de representar esto (ya lo vimos) es una matriz de distancias: para NN puntos, generamos un array de NxN tal que cada entrada $(i,j)(i,j)$ contiene la distancia entre cada punto i y j.

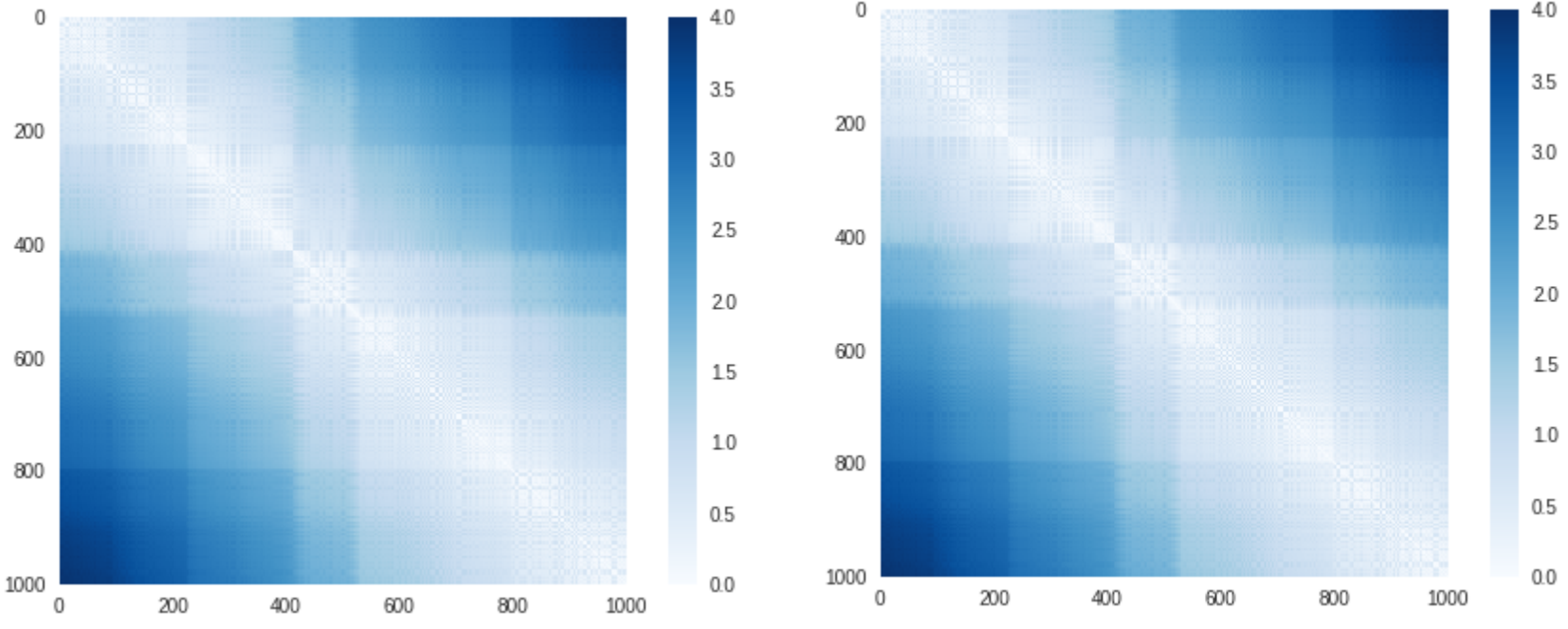
```
from sklearn.metrics import pairwise_distances
D = pairwise_distances(X)
```

- Una buena representación de la matriz de distancias puede generarse de la siguiente forma

```
plt.imshow(D, zorder=2, cmap='Blues', interpolation='nearest')
plt.colorbar()
```



- Si comparamos la matriz de distancias de los datos originales y la de los datos rotados… podemos ver que las mismas son iguales.

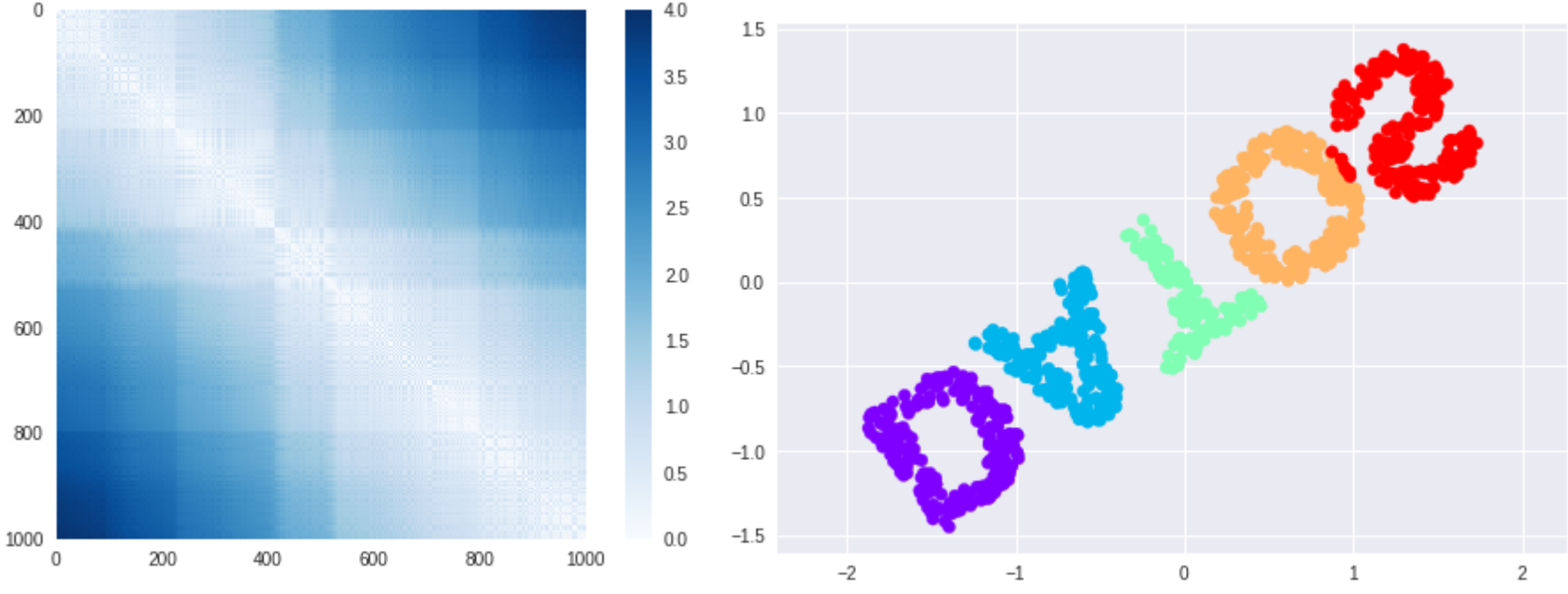


Multidimensional Scaling (MDS)

- La matriz de distancias nos da una representación de los datos que es invariante a las rotaciones y traslaciones.
- La visualización de la matriz de distancias no es totalmente intuitiva: hemos perdido todo signo visible de la estructura de los datos: el "DATOS" que veíamos antes.
- Computar la matriz de distancias de las coordenadas (x,y) es bastante directo (simplemente, aplicamos alguna función de distancia a los dos puntos).
- El proceso inverso (pasar de la matriz de distancias a las coordenadas) no es tan simple.
- El algoritmo de MDS va a hacer: recupera una representación D-dimensional de los datos a partir de una matriz de distancias. Veamos como funciona para nuestra matriz de distancia:

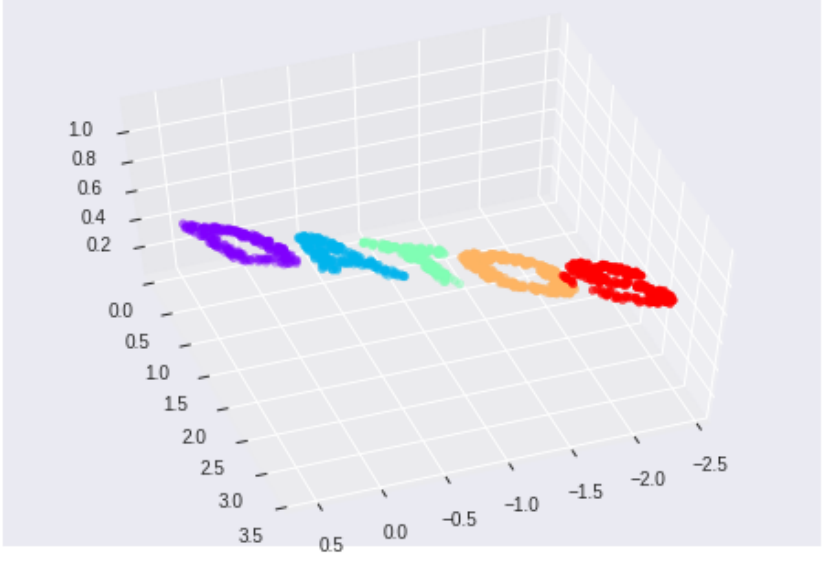
```
from sklearn.manifold import MDS
model = MDS(n_components=2, dissimilarity='precomputed', random_state=1)
out = model.fit_transform(D)
plt.scatter(out[:, 0], out[:, 1], **colorize)
plt.axis('equal')
```

Multidimensional Scaling (MDS)



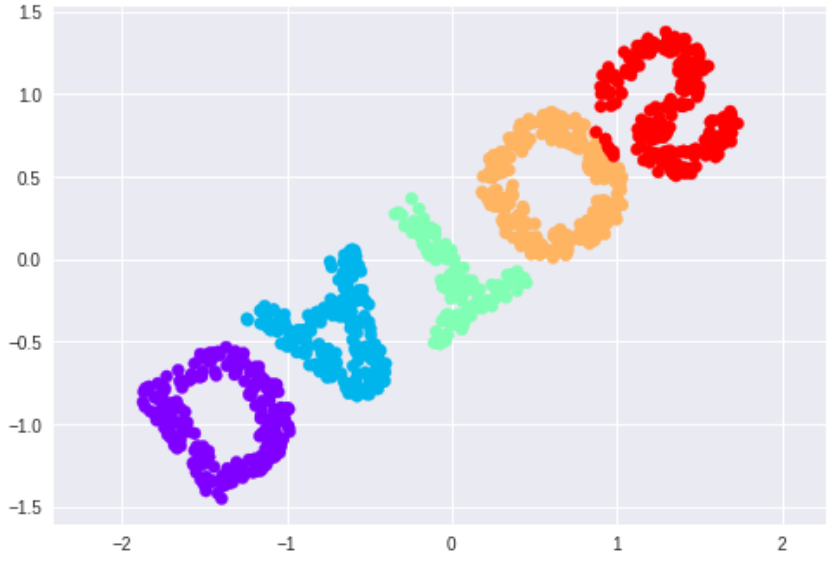
- Las matrices de distancias pueden ser computadas de datos con cualquier nivel de dimensionalidad.

- Por ejemplo, simplemente rotando los datos en el plano-bidimensional podemos proyectarlos en tres dimensiones usando la siguiente función (que es básicamente una generalización de la rotación de m



- Podemos ahora llamar al estimador MDS que utilizando como input esta estructura de datos tridmensional va a computar la matriz de distancias y luego determinar cuál es la representación geométrica en baja dimensionalidad (en este caso, dos dimensiones) para esta matriz. El resultado recupera la representación original de los datos:

```
model = MDS(n_components=2,
random_state=1)
out3 = model.fit_transform(X3)
plt.scatter(out3[:, 0], out3[:, 1], **colorize)
plt.axis('equal')
```



- Objetivo esencial de un estimador de manifold learning:
  - dado un set de datos en un espacio multimensional busca una representación de menor dimensionalidad que preserve ciertas relaciones dentro de los datos.

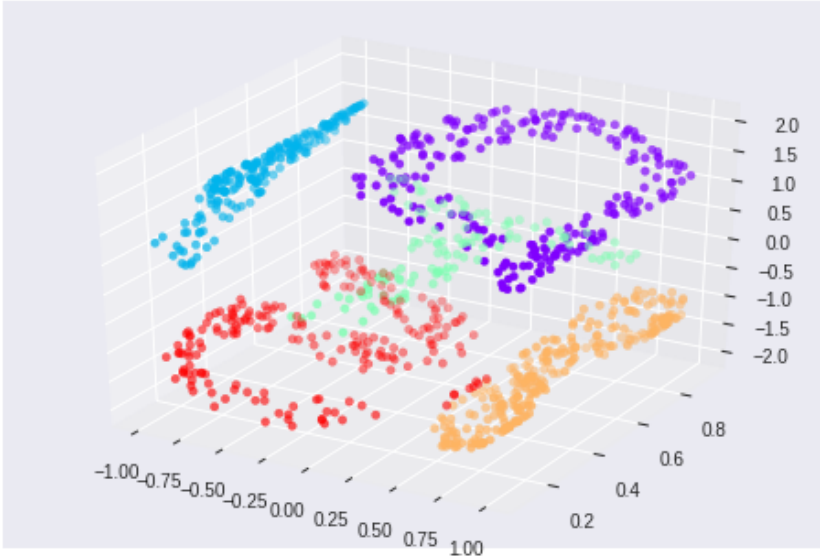
- En el caso de MDS la cantidad que se busca preservar es la **distancia entre todos los pares de puntos**.

Locally Linear Embedding (LLE)

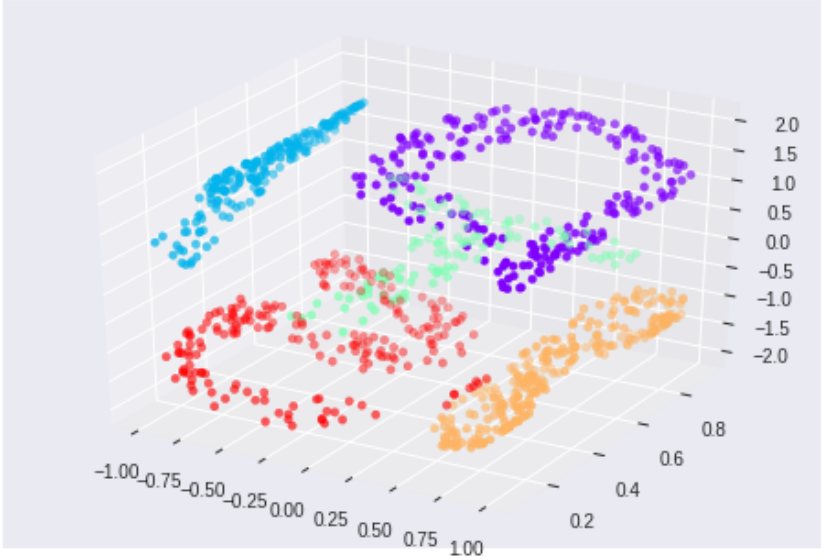
- Cuando estos "embeddings" son no lineales MDS comienza a funcionar mal. No alcanza con rotar o trasladar los datos para recuperar la estructura subyacente.

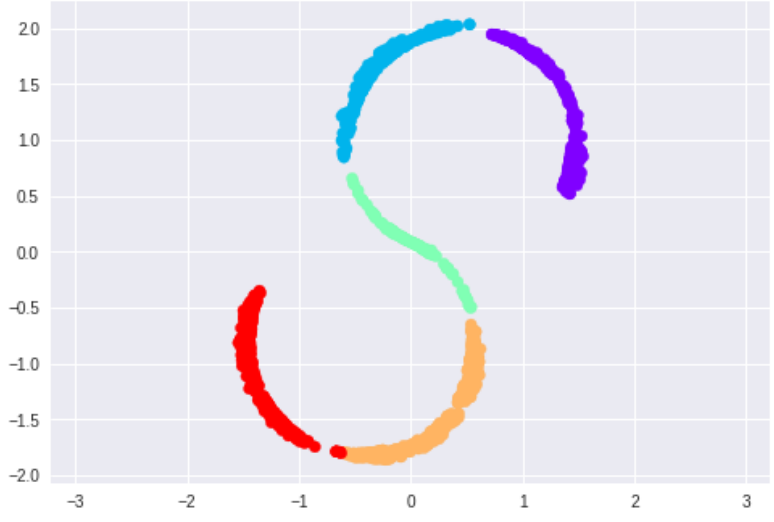
- Veamos el siguiente caso. La relación fundamental entre los datapoints todavía se puede recuperar pero en este caso los datos han sido transformados en una forma no lineal: fueron "envueltos" en la forma de una "S".

- Si tratamos un simple algoritmo de MDS en estos datos, el mismo no va a poder recuperar ("desenvolver") este "embedding" no lineal y se pierden las relaciones fundamentales en los datos.



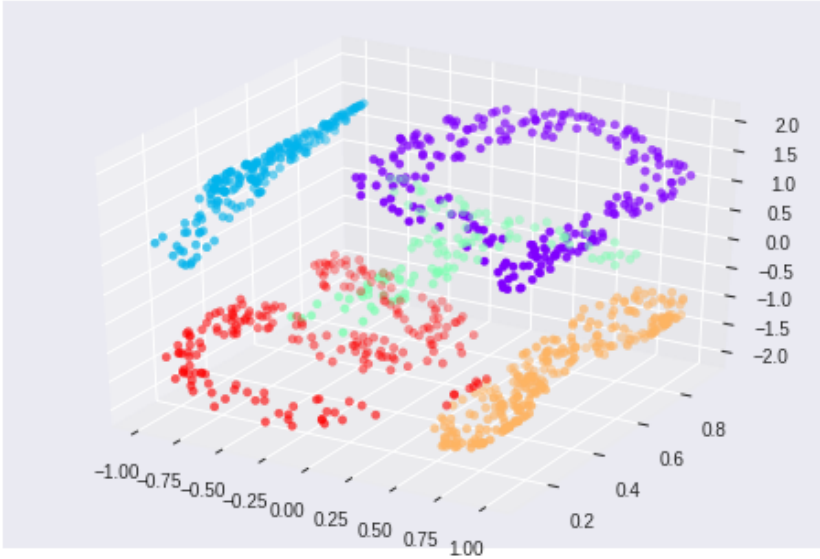
Locally Linear Embedding (LLE)

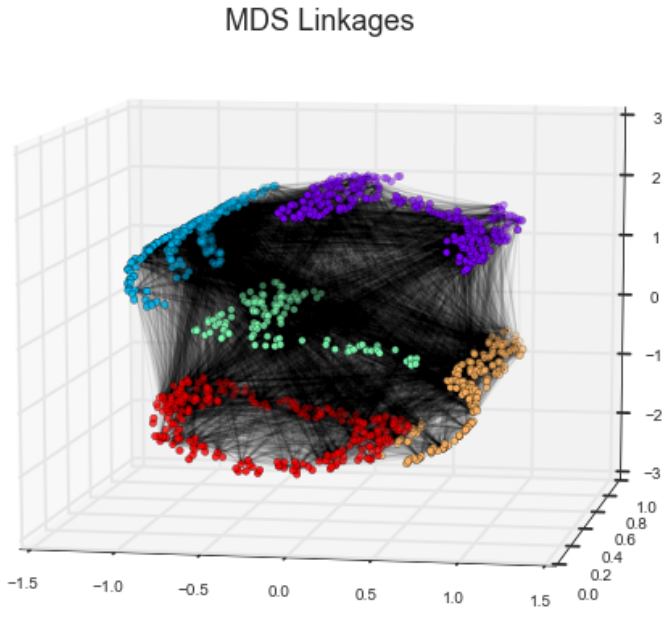


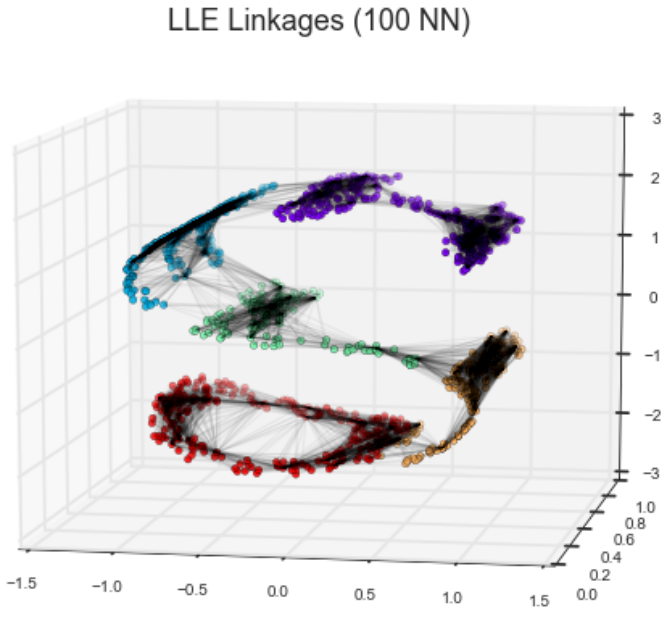


- La fuente del problema es que MDS trata de preservar las distancias entre todos los puntos del embedding... incluso aquellos que están muy lejos entre sí. ¿Pero qué pasaría si modificamos este algoritmo para que solamente preserve las distancias de los puntos cercanos entre sí?.

- El resultado va a ser un poco más cercano a los que estamos pensando.




MDS Linkages
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0


LLE Linkages (100 NN)
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0

- Cada línea representa la distancia que va a ser preservada en el embedding. A la izquierda hay una representación del modelo usando MDS: trata de preservar las distancias entre todos los pares de puntos.

- A la derecha aparece la representación del modelo utilizando un algoritmo de manifold learning llamado **Locally Linear Embedding (LLE)**: en lugar de preservar todas las distancias, trata de preservar solamente aquellas que aparecen entre vecinos cercanos. En este caso, se usaron 90 vecinos cercanos.

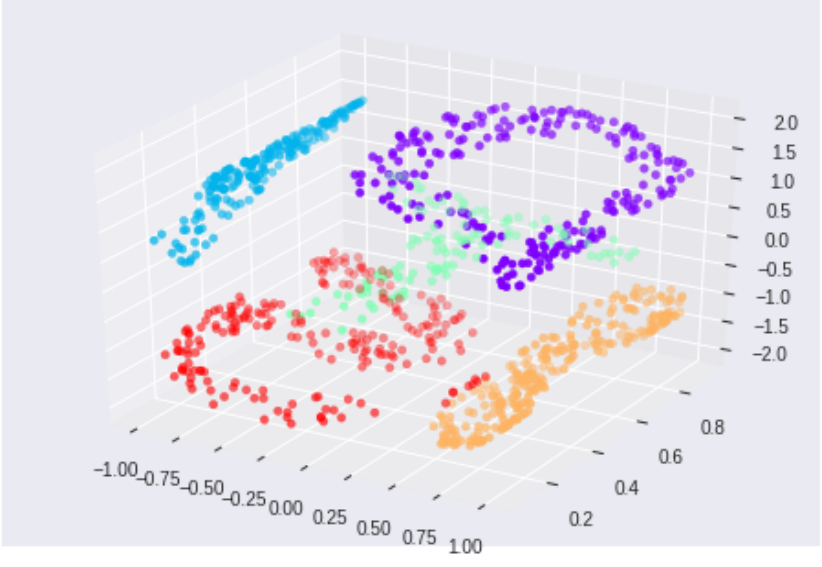
- Al observar el panel de la izquierda puede notarse por qué MDS falla:

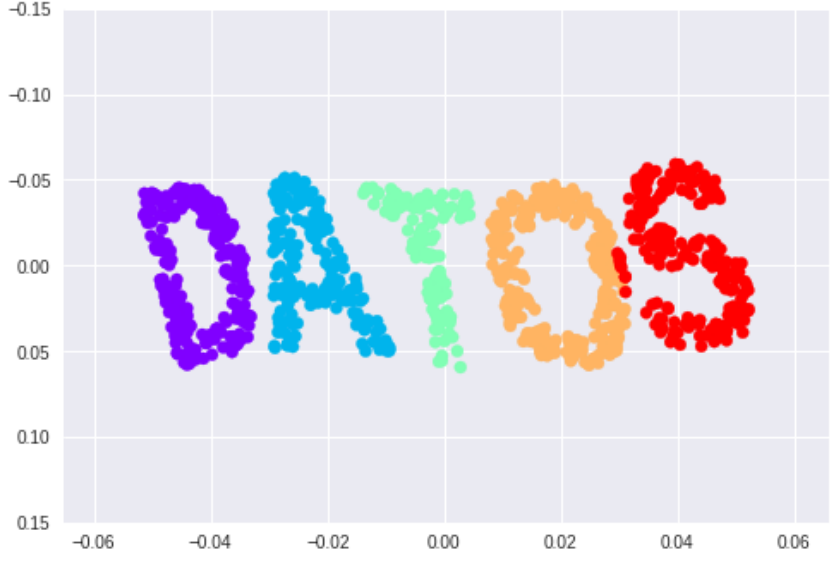
  - no hay manera de "aplanar" estos datos de forma adecuada preservando la longitud de cada línea de distancia entre dos puntos.

  - En cambio, en el de la izquierda parece un poco más fácil. Podemos imaginar "desenrollar" los datos de forma tal que se mantengan las longitudes de las líneas.

Locally Linear Embedding (LLE)

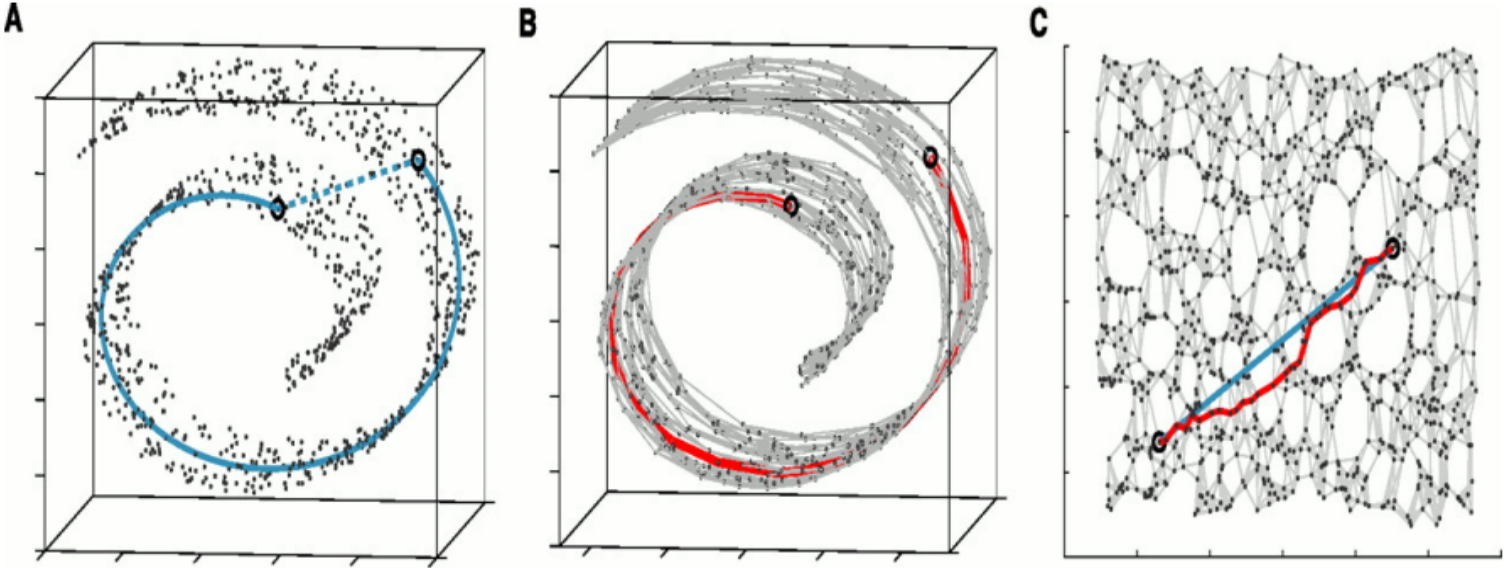
```
from sklearn.manifold import Isomap
model = LocallyLinearEmbedding(n_neighbors=90, n_components=2, method='modified',
                               eigen_solver='dense')
out = model.fit_transform(XS)
```





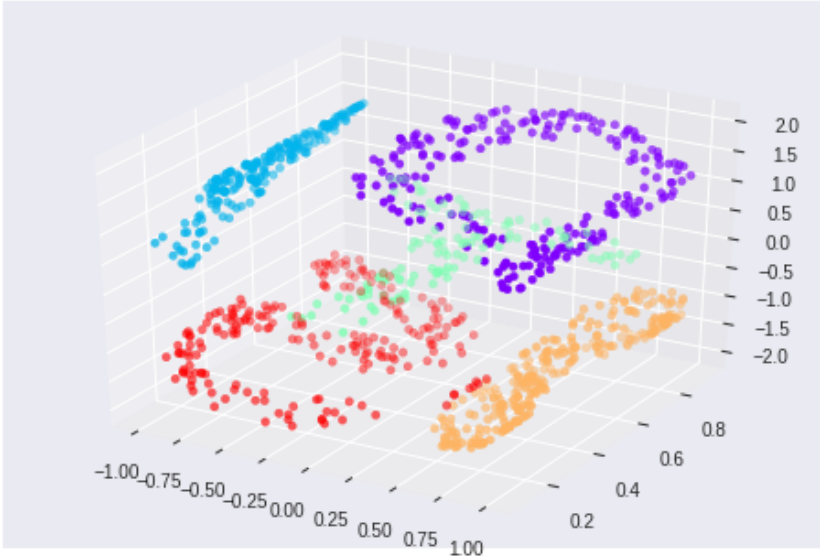
IsoMap

- Otro método para reducir la dimensionalidad de manera no lineal
- Diferencias con LLE: usa la matriz de distancias completas entre los puntos (no matrices de distancias locales)
- Diferencias con MDS: Computa las distancias geodésicas y no las distancias euclidianas

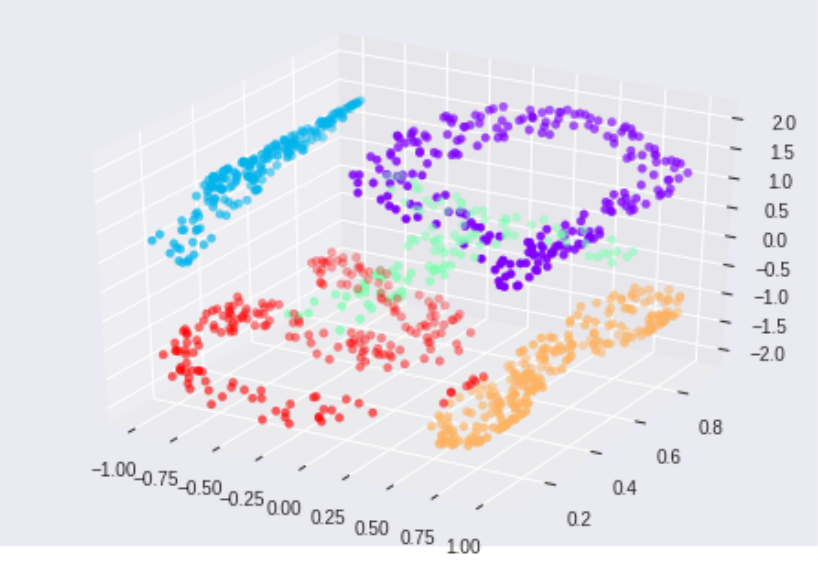


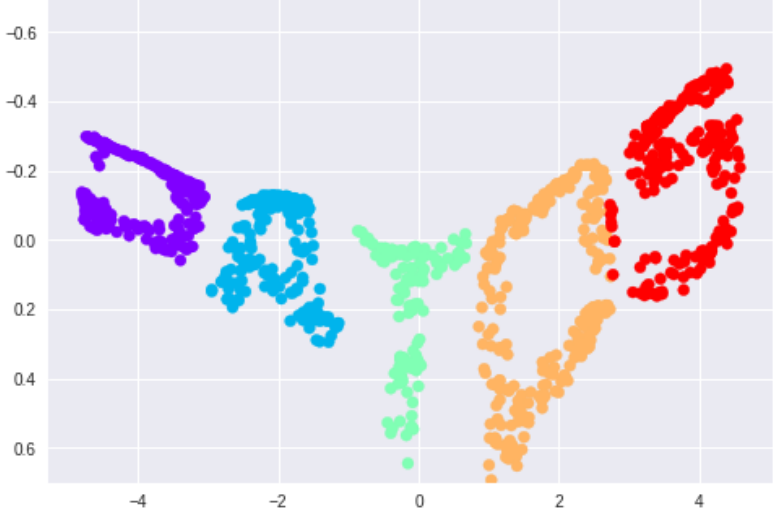
- Algoritmo general:

1. Define los vecinos cercanos a cada punto (puede hacerlo mediante diferentes métodos)
2. Construye un "grafo" que conecta a los vecinos más cercanos
3. Busca la distancia más cercana entre los nodos (puntos) - esta es la aproximación a la distancia geodésica
4. Sobre esa matriz, realiza MDS



```
from sklearn.manifold import Isomap
model = Isomap(n_neighbors=90, n_components=2, eigen_solver='dense')
out = model.fit_transform(XS)
```





Práctica Guiada: Manifold Learning en Faces y MNIST datasets

CONCLUSIÓN

- Uso fundamental de estas técnicas: visualización

- Diferencias con PCA:
  - No hay una buena forma de manejar datos perdidos.
  - La presencia de ruido en los datos puede alterar drásticamente los embeddings. PCA, por contraste, "filtra" los ruidos a través de los componentes principales.
  - El embedding resultante resulta altamente dependiente del número de vecinos elegidos y no suele haber métodos "sólidos" para hacer tunning de este parámetro.
  - El óptimo global en cantidad de dimensiones suele ser difícil de determinar. PCA, en cambio, tiene un criterio: la cantidad de varianza que explican los componentes.
  - Tienen complejidad cuadrática o cúbica. Para PCA existen aproximaciones randomizadas que generalmente son mucho más rápidas.

- Ventaja fundamental: capacidad de preservar relaciones no lineales. Por esta razón, suele ser útil realizar exploraciones de datos con manifold learning, luego de haberlos explorado con PCA.