Reducciones dimensionales (aprendizaje No supervisado)

Si tenemos datos en dimensión alta buscamos proyectar esos datos en un espacio de dimensión menor sin perder información importante.

Reducir dimensionalidad: cuando tenemos una alta dimensionalidad, muchas columnas, features, se vuelve difícil de manejar en tiempo y espacio (computacionales). Además comienzan a ocurrir problemas matemáticos porque hay muchas dimensiones cuya abstracción se vuelve compleja, y eso hace que sea difícil de entender.

Existen principalmente dos formas de reducir dimensión:

- Seleccionando variables
- Transformando las variables (linealmente o no)

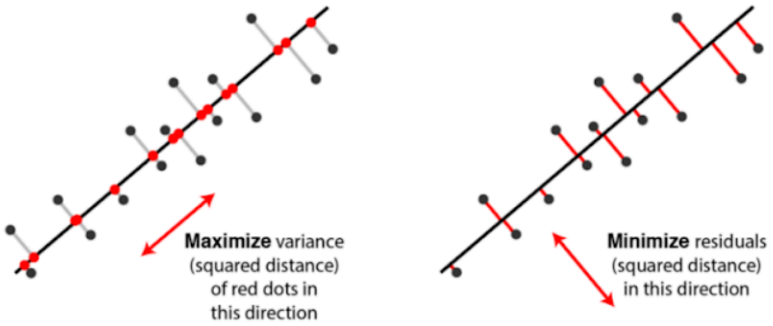
Queremos realizar esta reducción conservando algún tipo de propiedad de optimalidad.

Objetivos de la reducción dimensional:

- Visualizar los datos.
- Reducir el ruido.
- Regularización de datos (por ej llevarlos a misma escala)
- Comprensión de la información.
- Reducción del cómputo de los modelos.
- Sirve como resumen de los datos.

PCA

Buscamos un conjunto de proyecciones ortogonales de X = (x1, x2, . . . , xp) donde las proyecciones esten ordenadas de forma decreciente según su varianza. Una proyección es buena si mantiene la varianza de mis datos. PCA me devuelve todas las proyecciones. Tengo tantas proyecciones como dimensiones.



Dos puntos importantes:

- Varianza: es la medida que usa esta técnica para cuantificar la información. Lo importante es conservar la varianza de mis datos y me va a decir cuánto conserva de varianza cada proyección.

- El resultado provoca una rotación o cambio del sistema de coordenadas de modo que las proyecciones no esten correlacionadas.

Idea: rotar el dataset de manera de maximizar la varianza de los datos en proyecciones ortogonales.

Suposición: los datos se encuentran mayormente en un subespacio lineal de dimensión menor a la original.

Lo que hago es rotar para ver la mejor posición para proyectar los datos.

¿Pueden dos proyecciones distintas dar la misma varianza? Si. El área de la "sombra" es la misma pero la orientación no.

¿Qué pasa si nos queda una proyeccion con varianza 0? Si hacemos una proyección mala, todos los puntos caen en el mismo lugar. **Si todos caen en el mismo lugar, la varianza es 0.**

Quiero minimizar la proyección, minimizar los residuos de la proyección.
Lo que se ordena son los autovectores y los autovalores explican la varianza conservada.

¿Cómo sabemos en qué direccion proyectar? la direccion del autovector del autovalor más alto de la matriz de covarianza

- Obtenidas las componentes principales, nos quedamos con las N(cantidad de dimensiones) componentes que mayor varianza expliquen.

- Podemos reducir y reconstruir.

¿Cómo se reconstruye?

- Dada nuestra matriz de datos X de n x p, aplicamos PCA para obtener los k componentes

- Llamemos V a la matriz de autovectores resultante de p x k

- Luego la proyección de X en k dimensiones está dada en la matriz Z=XV

- Finalmente para reconstruir la matriz X basta con hacer X = ZV^T

- Si la varianza abarcada en V es < 1 entonces la reconstrucción no será perfecta

Luego se pueden reconstruir los datos, aunque depende de cuánta varianza se conserve. Si la varianza abarcada en V es < 1 entonces la reconstrucción no será perfecta.

Ventajas:
- No necesita establecer hiperparametros.

- No tiene iteraciones.
- Siempre me da el mismo resultado, es deterministico.

Desventajas:

- **Limitado a proyecciones lineales.**

Resumencito:
¿Cómo funciona? dada una variable buscamos un conjunto de proyecciones lineales ortogonales donde las proyecciones esten ordenadas segun su varianza.

¿De qué nos sirve la varianza acá? la varianza nos cuantifica la inforamcion. A traves de ella sabemos que tan optimo es.

¿Cómo es el resultado? Es una rotación o cambio de coordenadas de modo que las proyecciones no están correlacionadas .

MDS Multidimensional scaling

La idea es mantener la distancia entre los datos. Ubica los datos en una dimensión menor tal que las distancias se parezcan lo más posible (con distancia euclidiana tiende a PCA)

Ventajas:

- Soporta varios tipos de distancias.

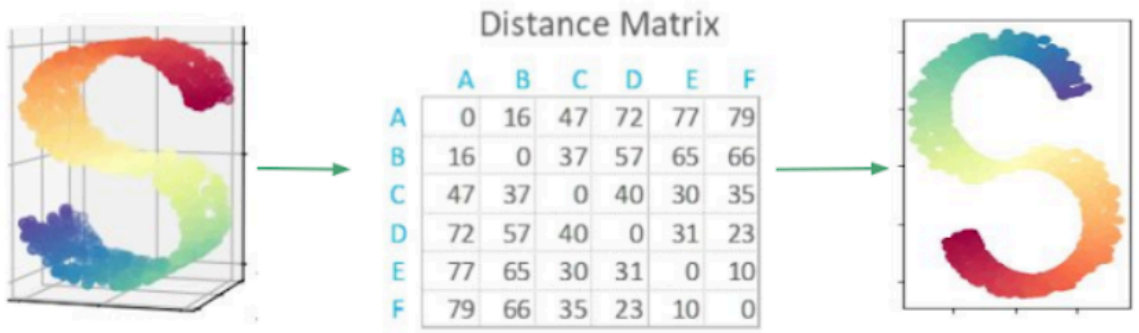
- Permite transformaciones no lineales.

Desventajas:

- Iterativo a diferencia de PCA, tarda más y puede tener mínimos locales (puede dar soluciones no óptimas y distintas en cada corrida).

- Difícil determinar que distancia a usar es la mejor

Usa una matriz de distancia para conservar las distancias del dataset. Creo un set de datos que respete las distancias de esa matriz.



Isomap

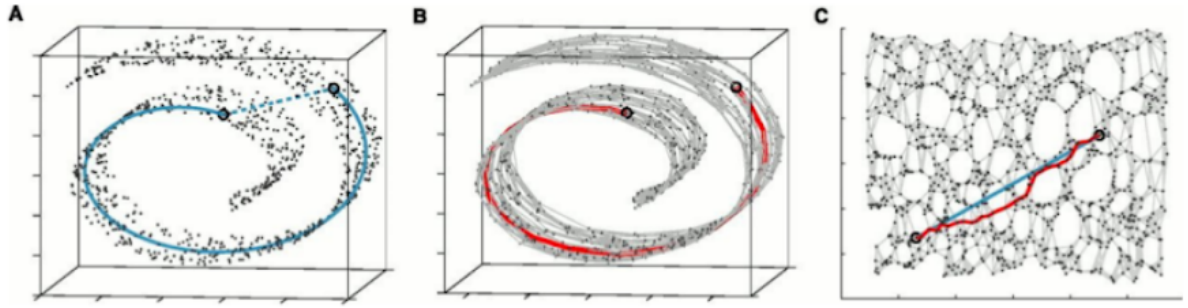
¿Cuál es la idea?

- preservar la geometría del mainfold usando la **distancia geodésica**

¿Cómo funciona?:

- Voy saltando de un punto a otro (se genera un grafo dentro de esa vecindad). Va creando aristas para unir a los puntos. Computo el camino más cercano entre todo par de puntos. Se genera una matriz de distancias.

Graficamente:



A: La distancia puede ser la línea punteada (directa) o a través del manifold (línea llena)

B: La línea roja muestra cómo es la distancia a través del manifold: saltando de nodo (cercano) en nodo

C: Cómo queda al reducción a dos dimensiones

Algoritmo:

1) Determinar vecinos más cercanos por cada punto. (OJO: Al incrementar demasiado la cantidad de vecinos a observar se pierde la noción de distancia geodésica, ya que muchos nodos pasan a tener vecinos a los que llegan saliéndose del mainfold)

2) Construir un grafo de vecindad
    a) Se conectan todos los vecinos.
    b) Cada eje con longitud distancia euclidiana entre los puntos.

3) Computar el camino más cercano entre todo par de puntos: Dijkstra's / Floyd–Warshall

4) Generar matriz de distancias en base al punto 3 y aplicar MDS

Ventajas:

- Mantiene la estructura del mainfold.
- Permite las transformaciones no lineales.

Desventajas:

- Hay que determinar la cantidad de vecinos a mirar.

- Es sensible al ruido y a los outliers.

t-SNE *(sirve solo para **visualización**)*

¿Cuál es la motivacion?
- poder ver las cosas que están en altas dimensiones bajandolas a R2

Primero surge **SNE**
Buscamos una función que reduzca la dimensión. Hay que lograr que los puntos que estaban cerca en Rn esten también en R2.

Construimos una matriz de probabilidad de que dos puntos están cerca, de que sean vecinos.
Usamos la perplexity para distribuir más las probabilidad entre los puntos. Como que hay más vecinos que probablemente estén cerca de un punto.

3 pasos:
- CON PUNTOS DE ENTRADA hacemos matriz con la probabilidad de que 2 puntos estén cerca (filas suman 1) + perplexity + puntos son vecinos osea que todos los que están más o menos cerca tiene la misma proba

- CON PUNTOS DE SALIDA los iniciamos al azar y calculamos matriz de distancias que obviamente va a ser ridicula. Calculamos la distancia entre las 2 matrices y nos movemos por descenso de gradiente para que baje esa distancia. No puede llegar a 0.

- Ajustamos perplexity a ojo viendo cual nos parece que tiene más sentido, la mejor foto.

El mejor resultado es el que devuelve la menor distancia entre las matrices.
Se recomienda correr varias veces el algoritmo.

Parecido a MDS, la diferencia es que uno usa matriz de distancia y el otro una matriz de probabilidad.

Para algunos casos anda mal entonces cambiaron la forma de calcular las distancias y la forma en la que se expresa la matriz de probabilidad.

t-SNE

¿Qué cambia?
- La forma de calcular las distancias.
- La forma en la que expresamos la matriz de probabilidad.

t-SNE es una especie de patch a SNE que soluciona algunos problemas que tenía originalmente.

Fortalezas:

- De lo mejor para visualizar datos
- Conserva estructuras no lineales globales y locales

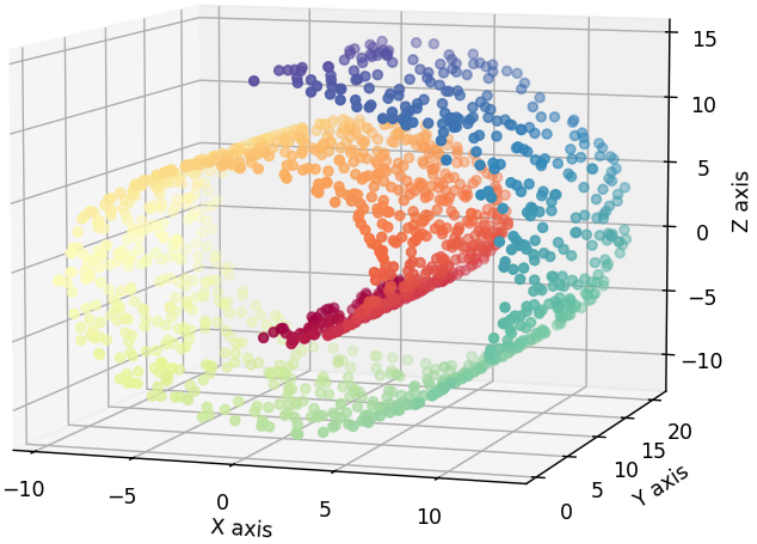
Debilidades:

- Es estocástico
- Escala mucho en tiempo con dimensiones y puntos
- No se puede usar para nuevos puntos

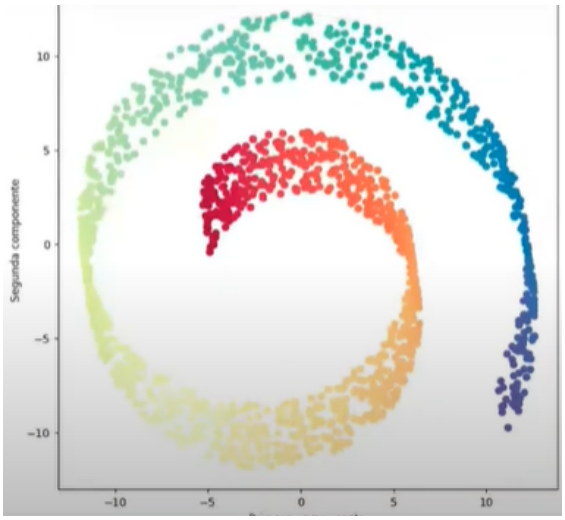
Se puede a veces reducir dimensionalidad con PCA y luego aplicar t-sne para que no tarde tanto.

COMPARANDO REDUCCIONES:

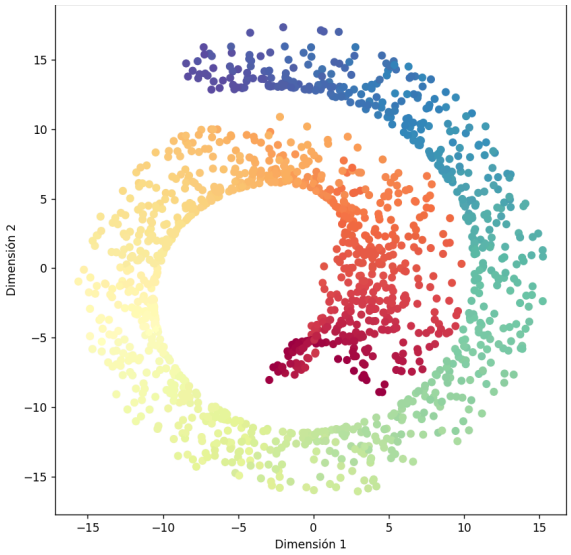
Original: Swiss roll (penseolos como un pionono en 3D tipo posta posta)



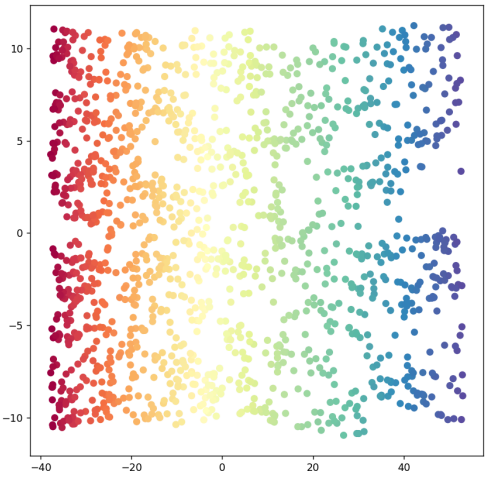
Si aplicamos PCA:



Si aplicamos MDS:



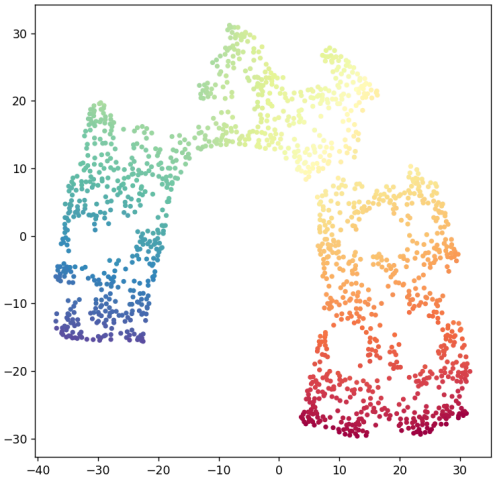
Si aplicamos ISOMAP con una cantidad de vecinos razonable:



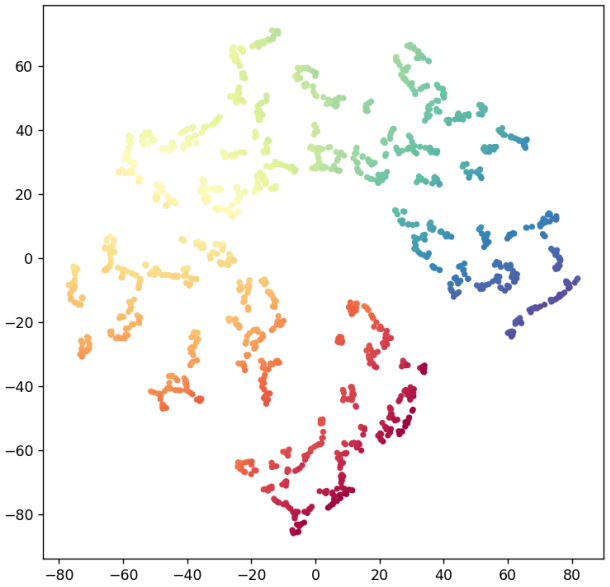
ISOMAP con **baja** cantidad de vecinos veriamos algo parecido aunque más "esparso", eso es debido a que el grafo está menos contactado y por lo tanto algunas distancias se agrandan.

ISOMAP con **alta** cantidad de vecinos veriamos algo parecido a MDS, se pierde la noción de distancia geodésica, ya que muchos nodos pasan a tener vecinos a los que llegan saliéndose del manifold.

Si aplicamos TSNE con un perplexity razonable:

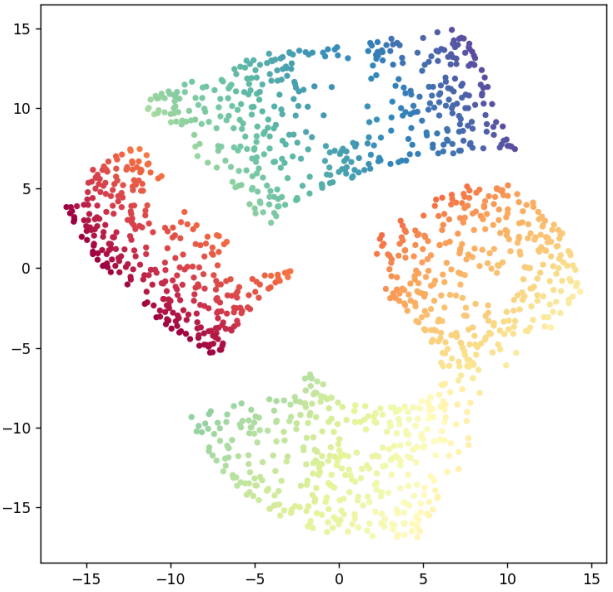


TSNE con perplexity baja:



Los puntos dentro de los conglomerados se encuentran más juntos pero los conglomerados entre sí se encuentran más separados

TSNE con perplexity altísima:



Si observamos los datos orginiales, se observa una cercanía entre los puntos de color rojo y verde, y los puntos de color naranja y azul. Al aumentar el valor de *perplexity*, el algoritmo toma más vecinos cercanos e intenta reproducir esa cercanía en el gráfico 2D, siendo la *perplexity* tan grande que llega a tomar puntos cercanos de la otra parte del *roll* e intenta reproducir esa cercanía.