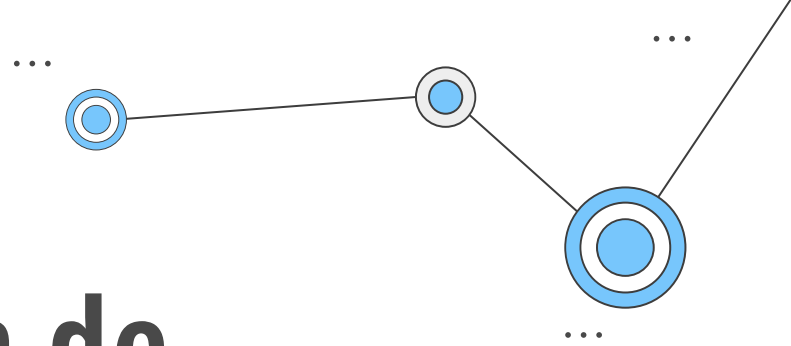


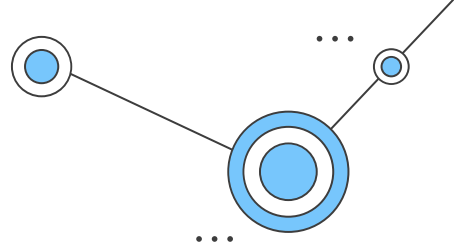
Reducción de dimensionalidad



**UNIVERSIDAD
CATÓLICA**
DE CÓRDOBA
JESUITAS

Dr. Francisco Arduh
2023

Introducción

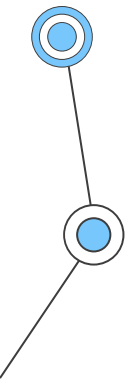


Reducir la dimensionalidad, además de acelerar el entrenamiento, puede mejorar el desempeño del algoritmo.

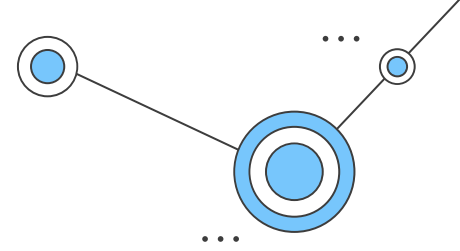
Un caso de reducción de dimensionalidad lo podríamos ver en el ejemplo del dataset MNIST:

- Eliminar los bordes ya que no aportan información.
- Se podrían combinar los píxeles, ya que la información de los píxeles cercanos están correlacionados.

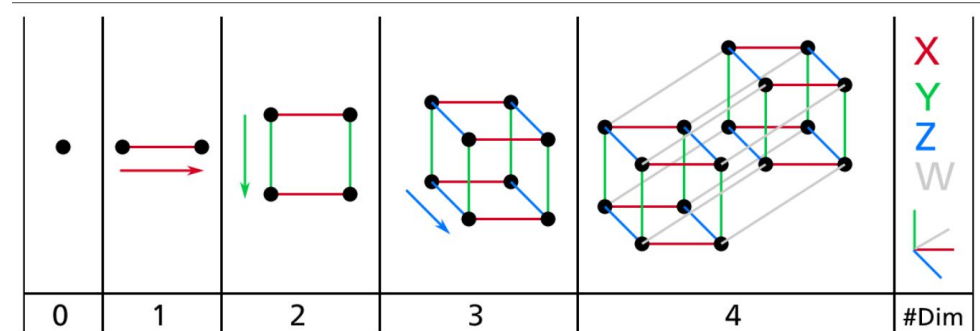
Reducir la dimensionalidad a dos o tres dimensiones puede ser muy útil al momento de la visualización de la información.



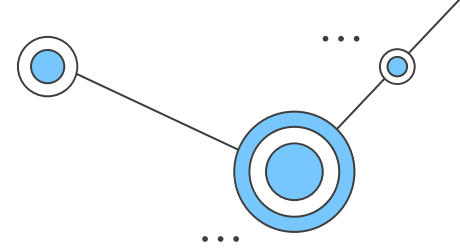
La maldición de la dimensionalidad



- La mayoría de los problemas con los que tienen una dimensionalidad mayor a 3 dimensiones.
- A mayor dimensionalidad existe “más espacio” que “llenar” con instancias.
 - La distancia media entre dos puntos de un hipercubo de lado 1 de un millón de dimensiones es ~ 408.25 .
 - La probabilidad de elegir un punto extremo aumenta.
- Una nueva instancia es probable que esté “lejos” de nuestro training set, por lo que la probabilidad de sobreajuste es muy alta.

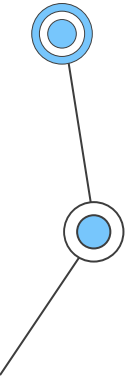


Principales enfoques para la reducción dimensional



Proyección

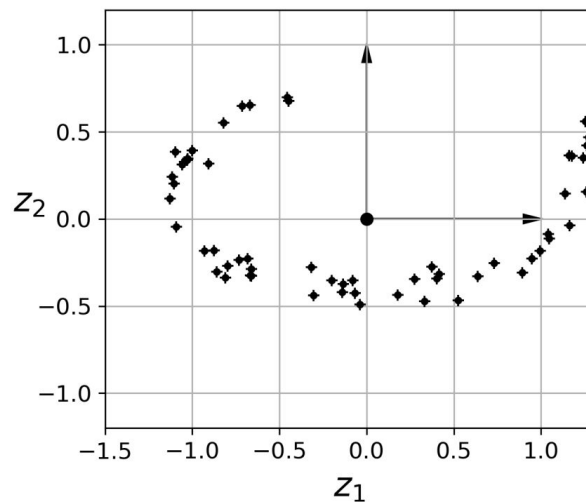
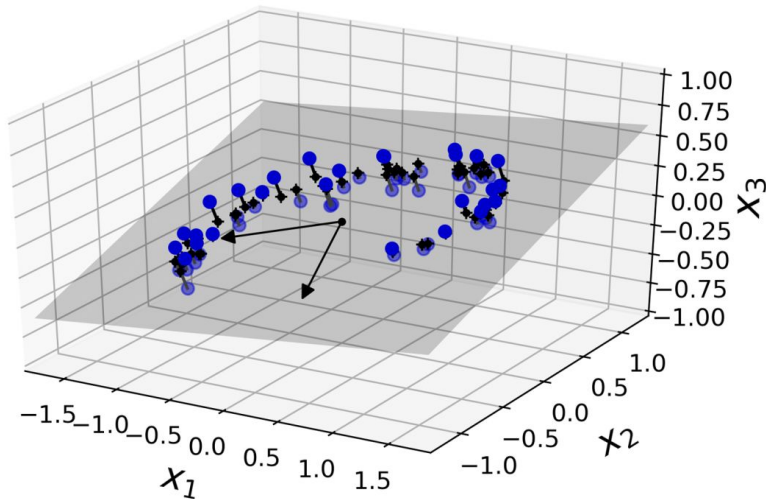
Manifold Learning



Proyección

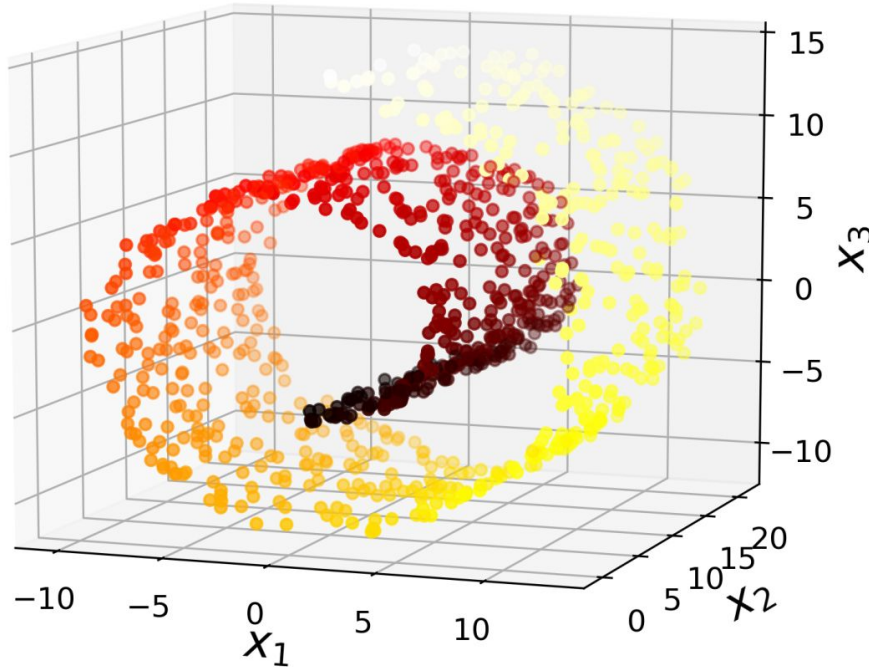
La instancias de entrenamiento generalmente no están dispersas uniformemente por el espacio:

- Algunas variables son casi constantes.
- Otras variables están altamente correlacionadas.

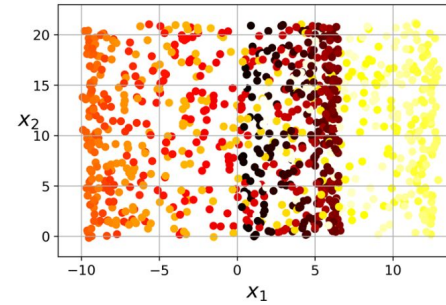


Proyección

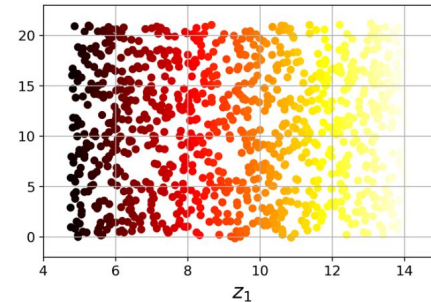
¿Qué pasaría con este ejemplo?



Proyección:

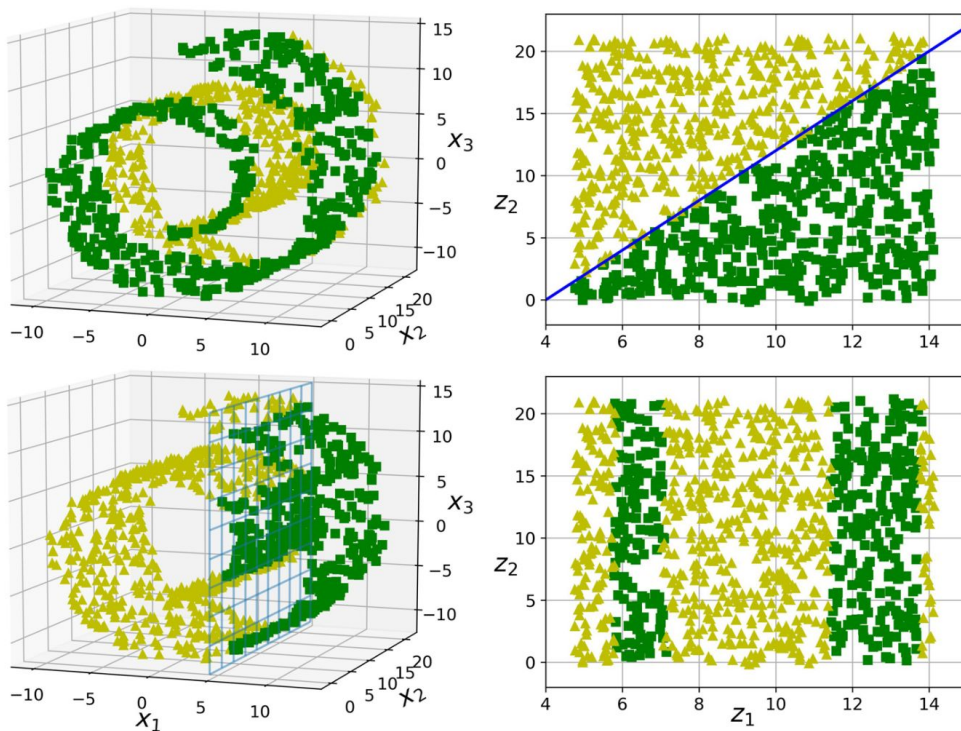


Lo que quisiera:

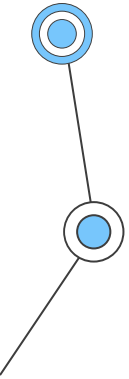
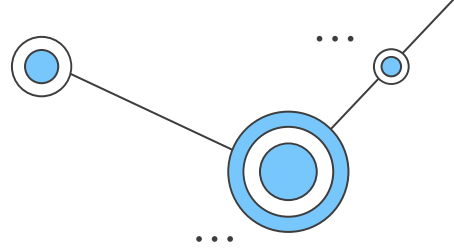


Manifold learning (Reducción dimensional no lineal)

Un manifold d-dimensional es parte de un espacio n-dimensional, que puede ser doblado y retorcido.

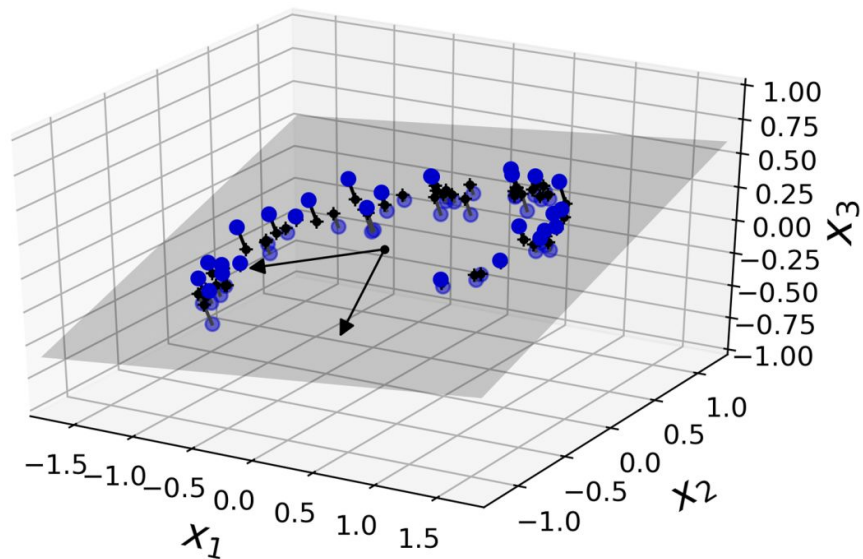


Técnicas de reducción dimensional



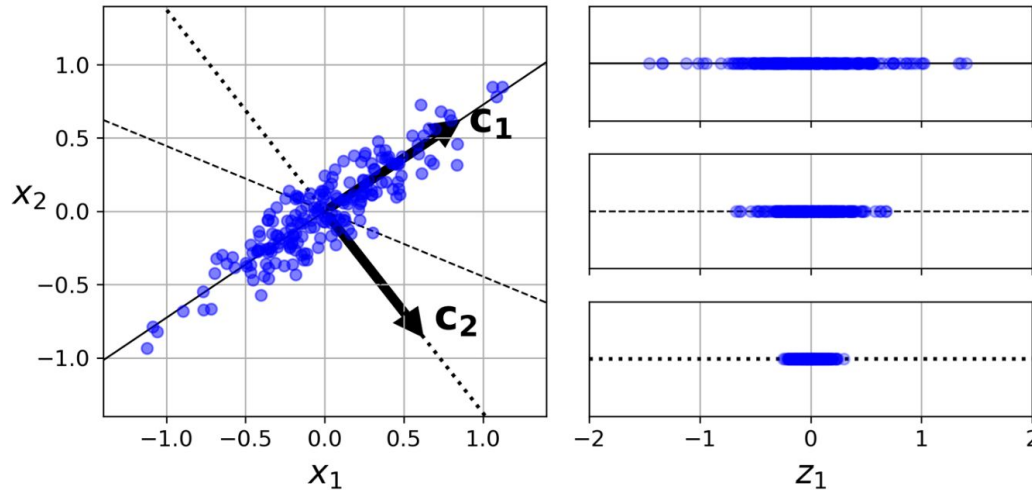
PCA (Análisis de componente principal)

- Identificar el hiperplano que está más cerca de los datos.
- Proyectar los puntos en ese plano.

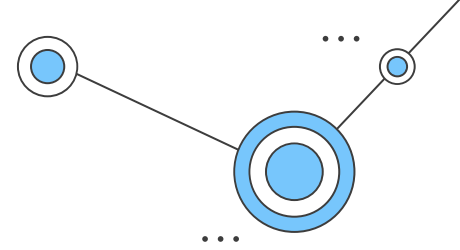


PCA: Preservar la varianza

Al momento de seleccionar el hiperplano en el cual proyectar se elige el de mayor varianza.



PCA: componente principal



- La componente principal es la de mayor varianza,
- Una vez seleccionada, se toma el plano con las dimensiones restantes y se vuelve a repetir el procedimiento.
- Se repite el punto anterior hasta llegar a la cantidad de componentes igual a las dimensiones.

Para encontrar las componentes principales se utiliza *Descomposición en valores singulares*, en la cual a la matriz del dataset se la descompone en 3 matrices. $\mathbf{V}\Sigma\mathbf{V}^T$, donde \mathbf{V} contiene vectores unitarios, que definen las componentes principales.

$$\mathbf{V} = \begin{pmatrix} | & | & \cdots & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & \cdots & | \end{pmatrix}$$



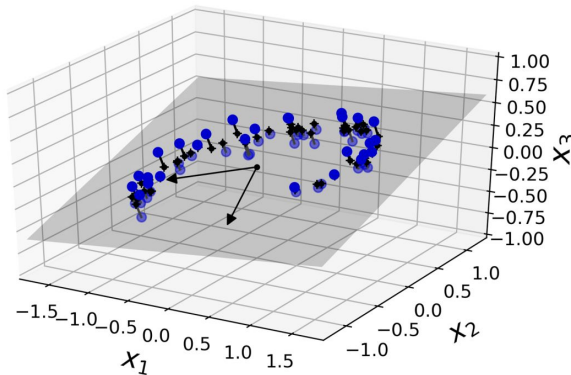
PCA: proyecta a una dimensión menor

Para proyectar el dataset a d dimensiones , se eligen las d primeras componentes principales (conservando la mayor varianza posible).

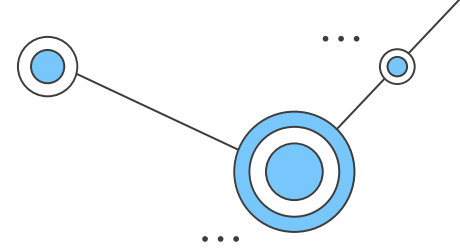
Matemáticamente se puede expresar como:

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

Donde \mathbf{W}_d es la matriz que contiene las d primeras columnas de \mathbf{V} .



PCA en Scikit-learn



```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 2)  
X2D = pca.fit_transform(X)
```

- Se puede acceder a la matriz traspuesta de \mathbf{W}_d , a través del atributo `components_`.
`pca.components_.T`
- También se puede a proporción de varianza explicada por cada uno de los componentes principales, a través del atributo `explained_variance_ratio_`.

```
>>> pca.explained_variance_ratio_  
array([0.84248607, 0.14631839])
```

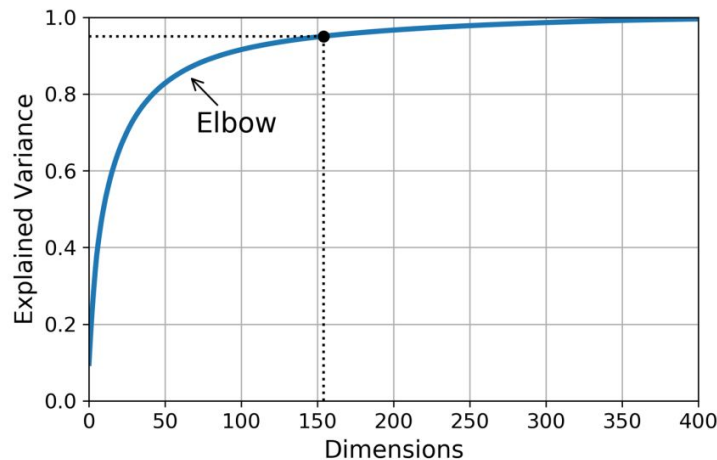


Eligiendo el número correcto de dimensiones

Se puede elegir con cuantas dimensiones me quedo en función de la suma de las proporciones de la varianza de las primeras componentes principales:

```
pca = PCA(n_components=0.95)  
X_reduced = pca.fit_transform(X_train)
```

Otra opción es determinar gráficamente el número de dimensiones:

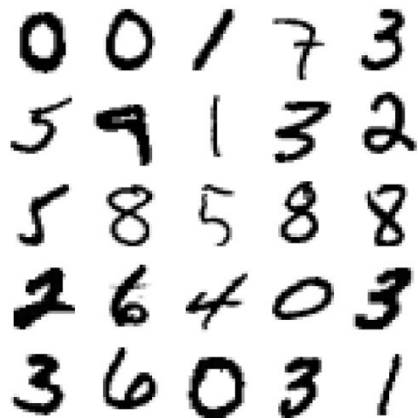


PCA para compresión

```
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA

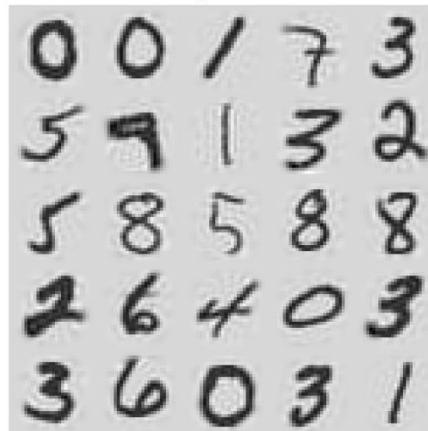
pca = PCA(n_components = 154) # PCA(n_components = 0.95)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
reconstruction_error = mean_squared_error(X_train, X_recovered)
```

Original



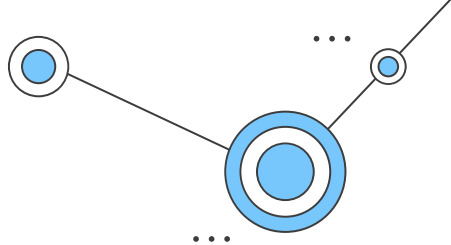
0	0	/	7	3
5	9	1	3	2
5	8	5	8	8
2	6	4	0	3
3	6	0	3	1

Compressed



0	0	/	7	3
5	9	1	3	2
5	8	5	8	8
2	6	4	0	3
3	6	0	3	1

PCA incremental



- Se utiliza en caso que cuente con un training set muy grande o el algoritmo necesita aprender online.
- Utiliza minilotes como se puede observar en el siguiente código:

```
from sklearn.decomposition import IncrementalPCA

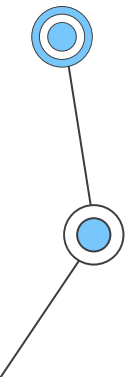
n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    inc_pca.partial_fit(X_batch)
```

```
X_reduced = inc_pca.transform(X_train)
```

- Es posible utilizar fit() junto a clase memmap de NumPy que, permite levantar en memoria sólo la parte de la muestra que se utilizar:

```
X_mm = np.memmap(filename, dtype="float32", mode="readonly", shape=(m, n))

batch_size = m // n_batches
inc_pca = IncrementalPCA(n_components=154, batch_size=batch_size)
inc_pca.fit(X_mm)
```

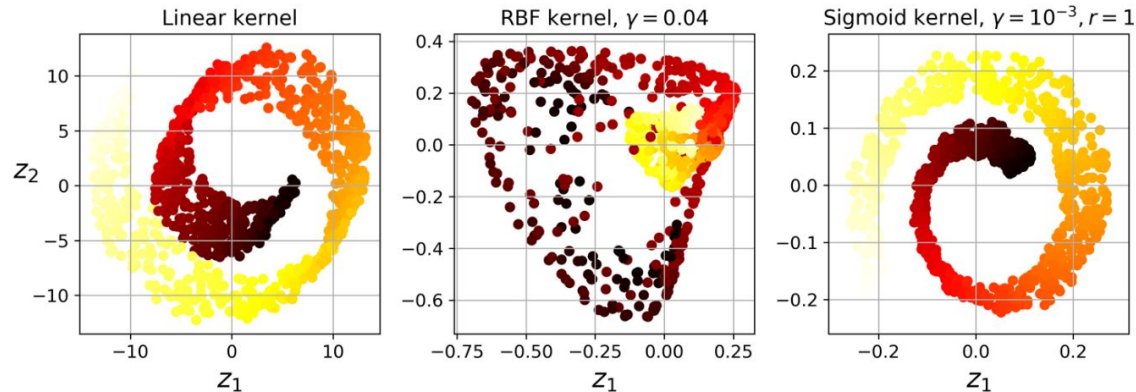


Kernel PCA

- Al igual como vimos anteriormente es posible utilizar el truco del kernel con PCA. Haciendo posible proyecciones no lineales para reducción dimensional.

```
from sklearn.decomposition import KernelPCA
```

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)  
X_reduced = rbf_pca.fit_transform(X)
```

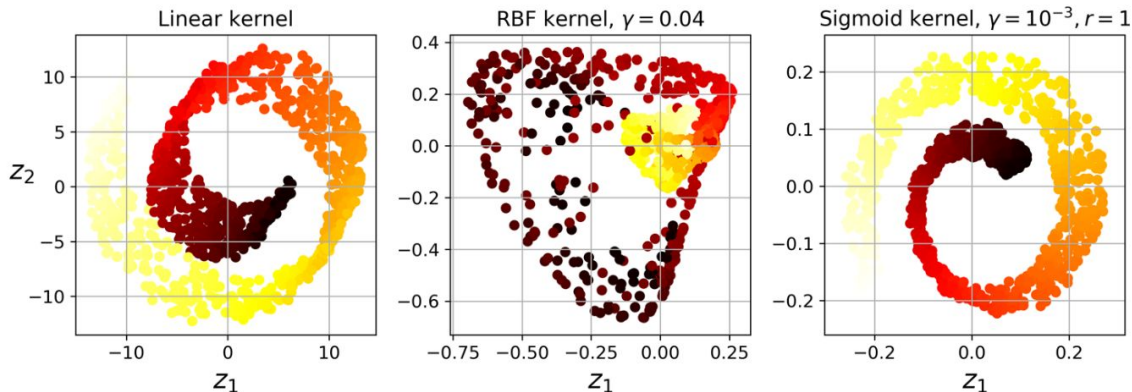


Kernel PCA

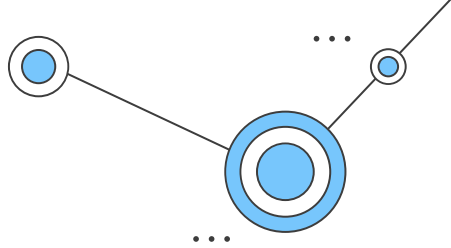
- Al igual como vimos anteriormente es posible utilizar el truco del kernel con PCA. Haciendo posible proyecciones no lineales para reducción dimensional.

¿Cómo elegimos el kernel correcto?

```
from sklearn.decomposition import KernelPCA  
rbf_pca = KernelPCA(n_components = 2, kernel = "rbf", gamma = 0.04)  
X_reduced = rbf_pca.fit_transform(X)
```



Seleccionando un Kernel: Grid Search



¿Qué métrica sería más apropiada para evaluar un algoritmo no supervisado?

Una posible solución sería evaluar junto a un algoritmo supervisado y utilizar la métrica que utiliza este para evaluar.

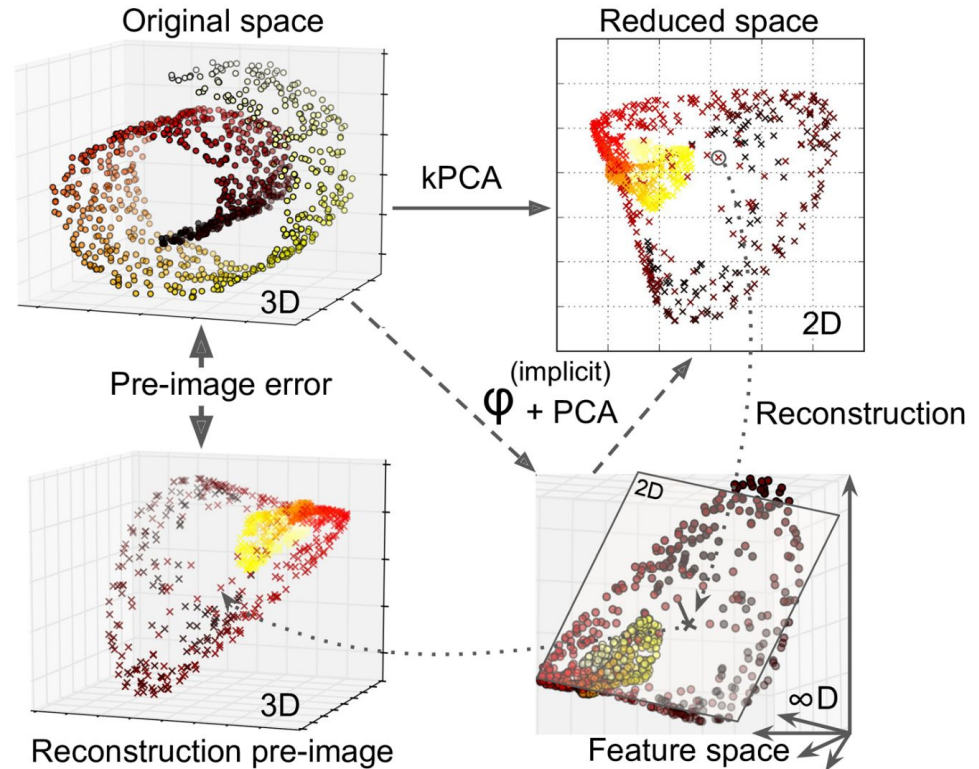
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
    ("kpca", KernelPCA(n_components=2)),
    ("log_reg", LogisticRegression())
])
param_grid = [{
    "kpca__gamma": np.linspace(0.03, 0.05, 10),
    "kpca__kernel": ["rbf", "sigmoid"]
}]
grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

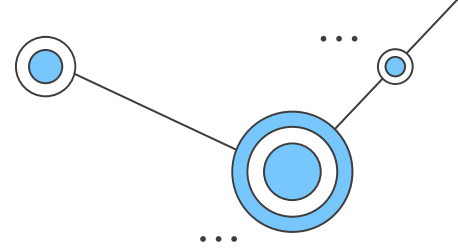


Seleccionando un Kernel: Error de reconstrucción

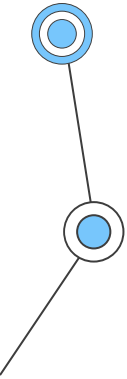
```
>>> rbf_pca = KernelPCA(n_components = 2,  
...                      kernel="rbf",  
...                      gamma=0.0433,  
...                      fit_inverse_transform=True)  
>>> X_reduced = rbf_pca.fit_transform(X)  
>>> X_preimage = rbf_pca.inverse_transform(X_reduced)  
>>> mean_squared_error(X, X_preimage)  
32.786308795766132
```



Manifold learning: Locally Linear Embedding (LLE)



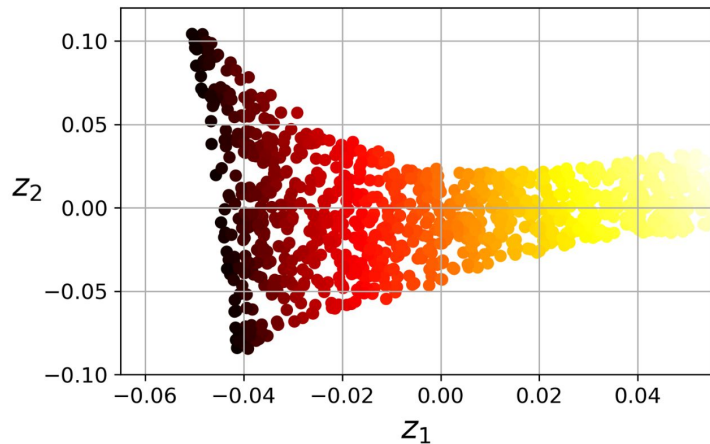
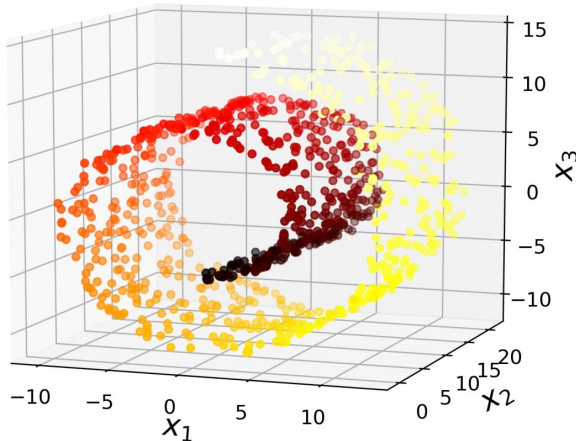
- LLE es una técnica de reducción dimensional no lineal.
- Consiste en medir la relación lineal entre una instancia y su vecina más cercana, y luego buscar una representación en una dimensión más baja que mantenga esa relación.
- Funciona bien para desenrollar espacios con poco ruido



Locally Linear Embedding (LLE): en Scikit-learn

```
from sklearn.manifold import LocallyLinearEmbedding
```

```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)  
X_reduced = lle.fit_transform(X)
```



Preserva las distancias entre instancias locales, pero a distancias grandes.

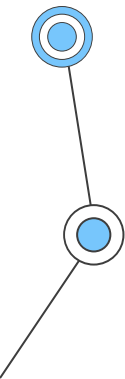
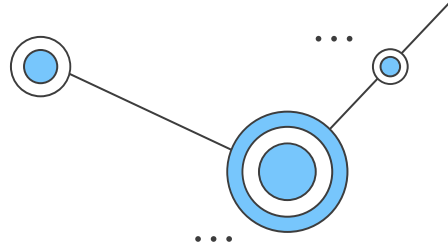
Locally Linear Embedding: ¿Cómo funciona?

Primer paso, encontrar los pesos \mathbf{w} tal que minimicen la relación lineal con sus vecinos sujeto a algunas restricciones:

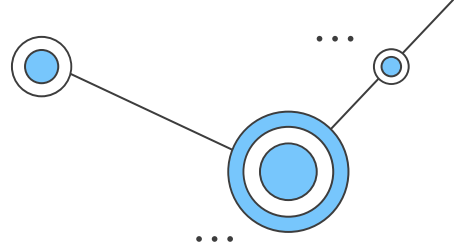
$$\begin{aligned} \widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \quad & \sum_{i=1}^m \left(\mathbf{x}^{(i)} - \sum_{j=1}^m w_{i,j} \mathbf{x}^{(j)} \right)^2 \\ \text{subject to} \quad & \begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^m w_{i,j} = 1 & \text{for } i = 1, 2, \dots, m \end{cases} \end{aligned}$$

Segundo paso, consiste en encontrar las nuevas coordenadas manteniendo la relación con los pesos encontradas en el paso anterior:

$$\widehat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \quad \sum_{i=1}^m \left(\mathbf{z}^{(i)} - \sum_{j=1}^m \widehat{w}_{i,j} \mathbf{z}^{(j)} \right)^2$$



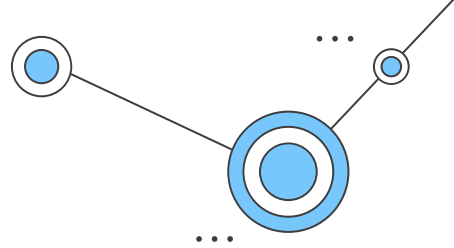
Otros métodos de reducción dimensional



- **Random Projections:** Realiza proyecciones lineales aleatorias. Ver `sklearn.random_projection`.
- **Multidimensional Scaling (MDS):** Reduce dimensionalidad preservando la distancia entre instancias. Ver `sklearn.manifold.MDS`.
- **Isomap:** Preserva la distancia geodésica entre los puntos. Ver `sklearn.manifold.Isomap`
- **t-Distributed Stochastics Neighbor Embedding (t-SNE):** Reduce la dimensiones tratando de mantener instancias similares cerca. Se utiliza principalmente para visualización. Ver: `sklearn.manifold.TSNE`
- **Linear Discriminant Component:** Es un algoritmo de clasificación, durante el entrenamiento aprende que eje más discriminante. Es una técnica utilizada para reducir la dimensionalidad antes de utilizar otro algoritmo de clasificación. Ver: `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`



Enlaces para tener en cuenta



Guía de Scikit-learn de Random Projections:

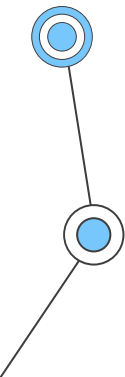
https://scikit-learn.org/stable/modules/random_projection.html#random-projection

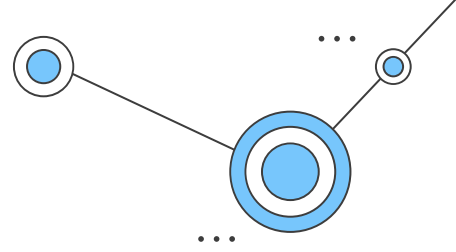
Guía de Scikit-learn de manifold:

<https://scikit-learn.org/stable/modules/manifold.html>

Guía de Scikit-learn de LDA:

https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda





¿Dudas?

