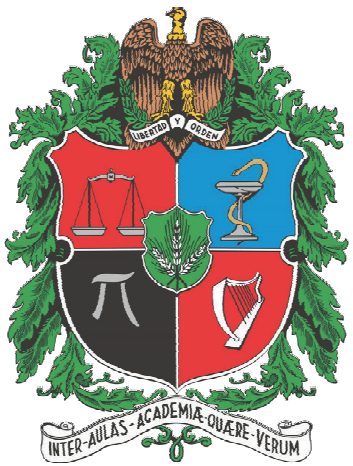
Técnicas de Reducción de Dimensión

A. M. Alvarez-Meza, Ph.D.
D. F. Collazos-Huertas, Ph.D(c)
amalvarezme@unal.edu.co, dfcollazos@unal.edu.co

Universidad Nacional de Colombia-sede Manizales



Contenido

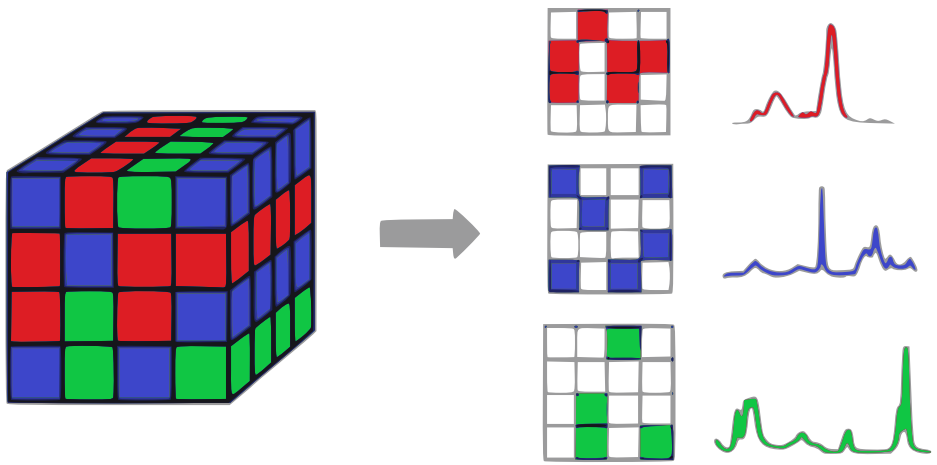
Contenido

Reducción de dimensión

Muchos problemas de **Machine Learning** implican **millones de características** para cada instancia de entrenamiento.

- Extremadamente lento
- Es mucho más difícil encontrar una buena solución.

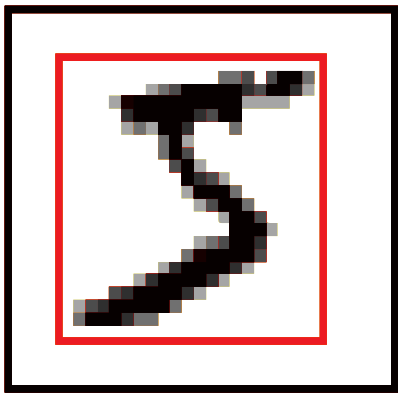
Este problema a menudo se conoce como *curse of dimensionality*.

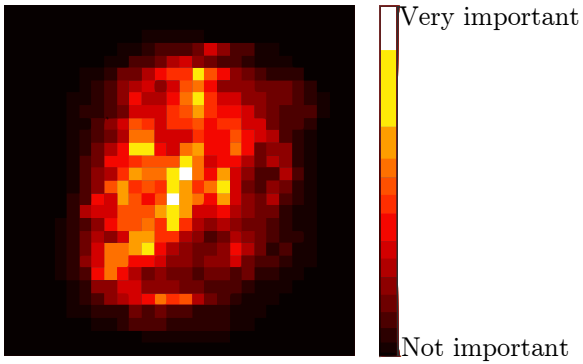


En el mundo real, es posible reducir considerablemente el número de características, convirtiendo un "**intractable problem**" en un "**tractable problem**".

Reducción de dimensión: Ejemplo

Consideremos las imágenes de la base de datos **MNIST**:





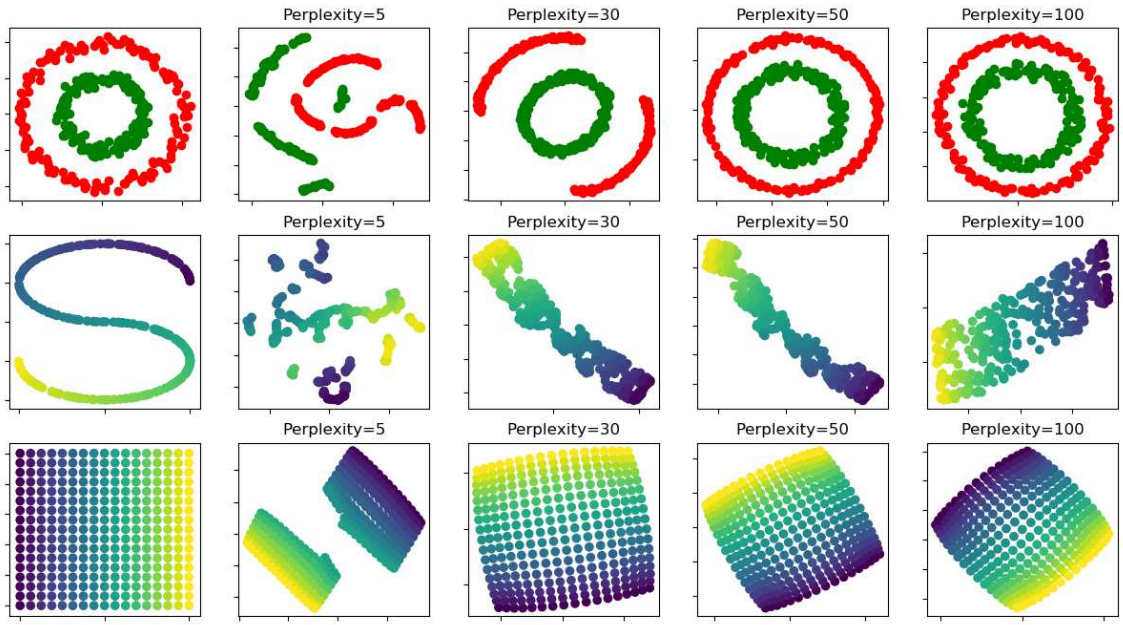
- Los píxeles en los bordes de la imagen **casi siempre son blancos**, por lo que puede eliminar estos píxeles **sin perder mucha información**.
- La imágen de relevancia, confirma que **estos píxeles no son importantes para la tarea de clasificación**.

En el proceso de reducción de la dimensionalidad se pierde algo de información.

Reducción de dimensión: TIPS

- En algunos casos, reducir la dimensionalidad de los datos de entrenamiento puede filtrar algunos ruidos y detalles innecesarios y, por lo tanto, **generar un mayor rendimiento**.

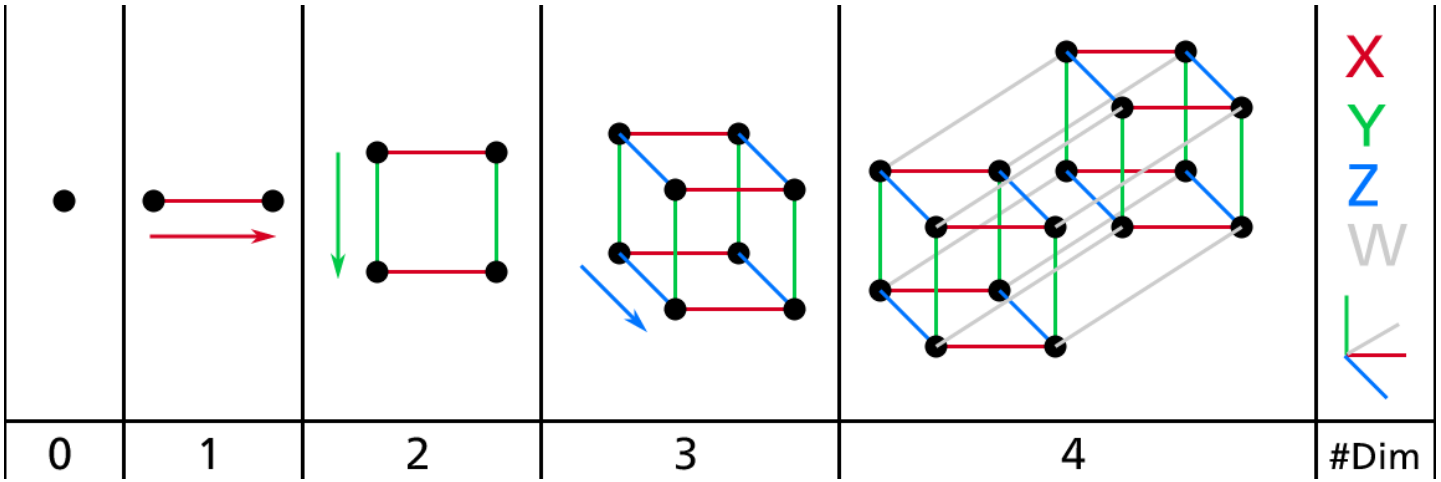
- Además de acelerar el entrenamiento, la reducción de dimensionalidad también es extremadamente útil para la visualización de datos.



Contenido

Curse of Dimensionality

Estamos tan acostumbrados a vivir en **tres dimensiones** que nuestra intuición nos falla cuando tratamos de imaginar **un espacio de alta dimensión**.



Resulta que muchas cosas se comportan de manera muy diferente en **espacios de alta dimensión**

Curse of Dimensionality: Ejemplo

- En un hipercubo de unidad de 10,000 dimensiones, la mayoría de los puntos en un hipercubo de alta dimensión están muy cerca del borde.

- En un **cuadrado de la unidad**, la distancia entre estos dos puntos será, en promedio, aproximadamente 0,52. Si elige dos puntos aleatorios **en un cubo 3D de unidad**, la distancia promedio será aproximadamente 0,66.

- ¿Qué pasa con dos puntos elegidos al azar en un hipercubo de un millón de dimensiones? ¡Bueno, la distancia promedio, será de 408,25. **Es decir, que la mayoría de las instancias de entrenamiento estén muy lejos unas de otras.**

Cuantas más dimensiones tenga el conjunto de entrenamiento, mayor será el riesgo de **"over-fitting"**.

Curse of dimensionality: maldición de la dimensión

Solución a la maldición de la dimensionalidad: aumentar el tamaño del conjunto de entrenamiento para alcanzar una densidad suficiente de instancias de entrenamiento.

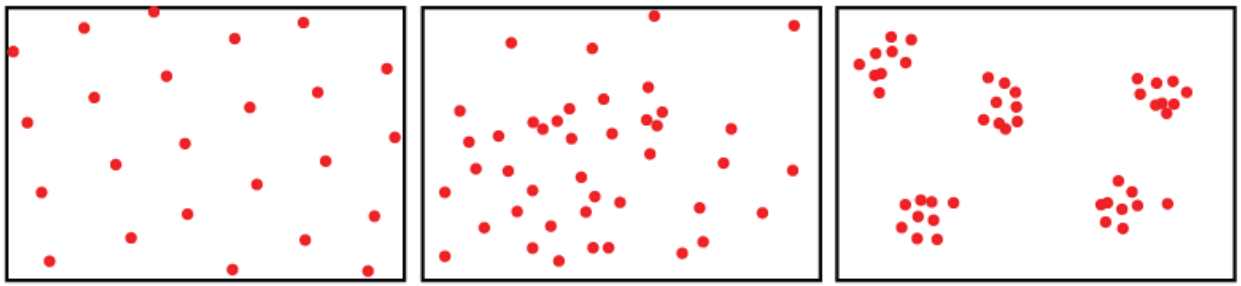
Sin embargo, el número de instancias de entrenamiento requeridas para alcanzar una densidad dada crece exponencialmente con el número de dimensiones.

- Con solo 100 características, necesitaría más instancias que átomos en el universo.

Contenido

Enfoques principales: Proyección

- En los problemas del mundo real, las instancias de entrenamiento no se distribuyen de manera uniforme en todas las dimensiones.

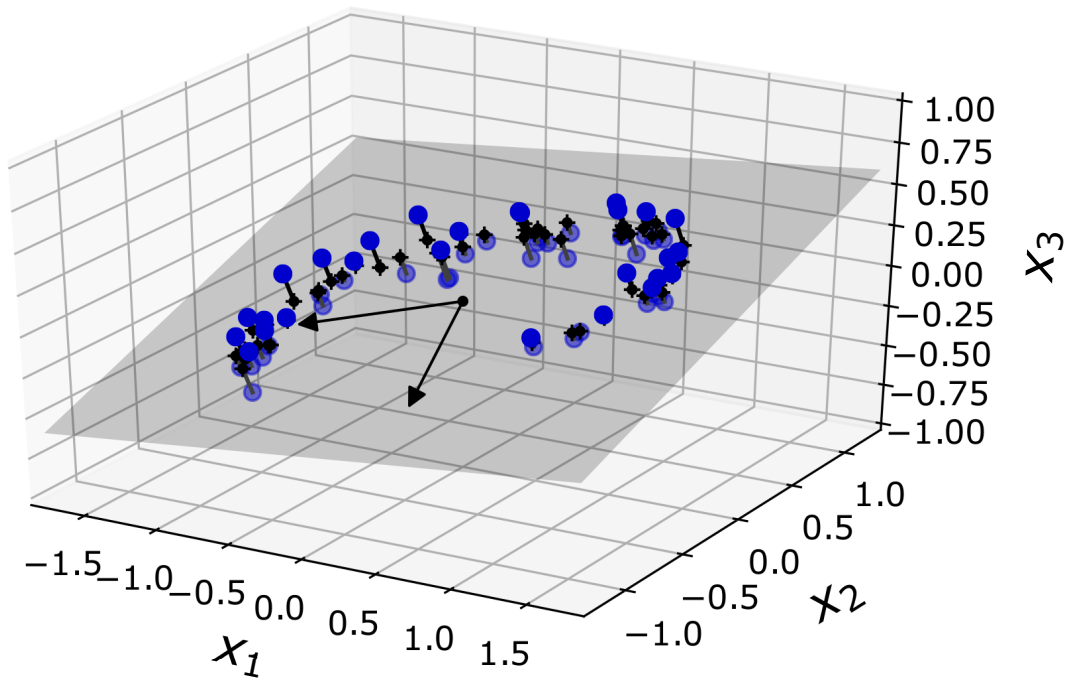


- Muchas características **son casi constantes**, mientras que otras están **altamente correlacionadas**.

Todas las instancias de entrenamiento realmente se encuentran dentro (o cerca) de un subespacio de dimensiones mucho más bajas.

Proyección: Ejemplo I

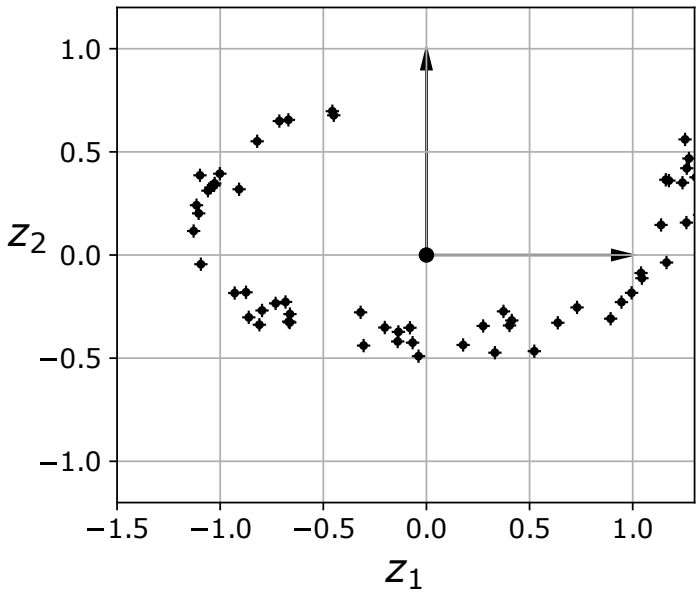
En la Figura se puede ver un conjunto de datos en 3D representado por círculos.



Todas las instancias de entrenamiento se encuentran cerca de un plano: este es un subespacio de menor dimensión (2D) del de alta dimensión (3D).

Proyección: Ejemplo I

Si **proyectamos cada instancia de entrenamiento perpendicularmente en este subespacio**, obtenemos el nuevo conjunto de datos 2D:

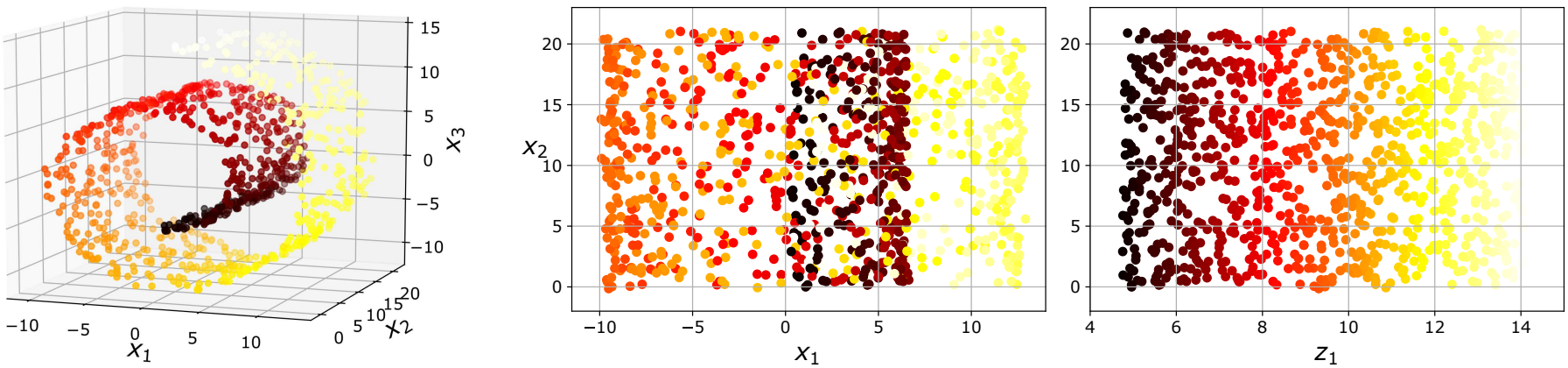


- Tengamos en cuenta que los ejes corresponden a las nuevas características z_1 y z_2.

- Sin embargo, la proyección no siempre es el mejor enfoque para la reducción de la dimensionalidad.

Proyección: Ejemplo II

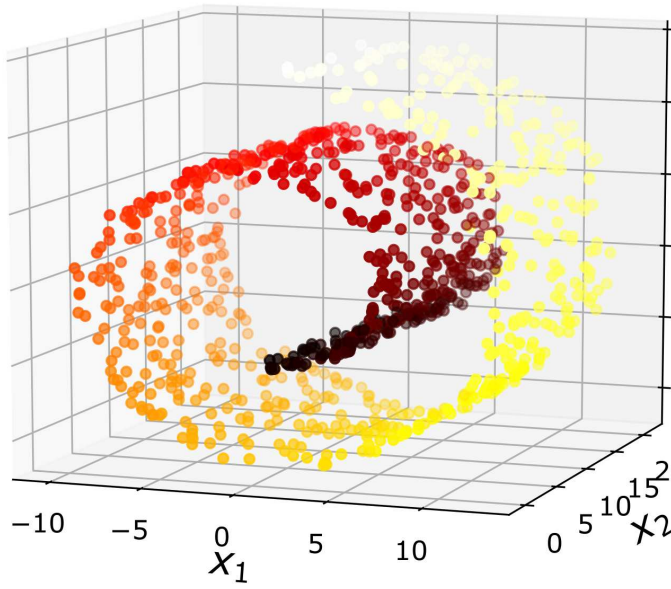
En muchos casos, el subespacio puede girar y girar, como en el famoso conjunto de datos "*Swiss roll*":



- Proyectar en un plano (por ejemplo, quitando x_3) aplastaría diferentes capas del rollo suizo (Centro).
- Lo que realmente deseamos es desenrollar el rollo suizo para obtener el conjunto de datos 2D (Derecha).

Enfoques principales: Manifold learning

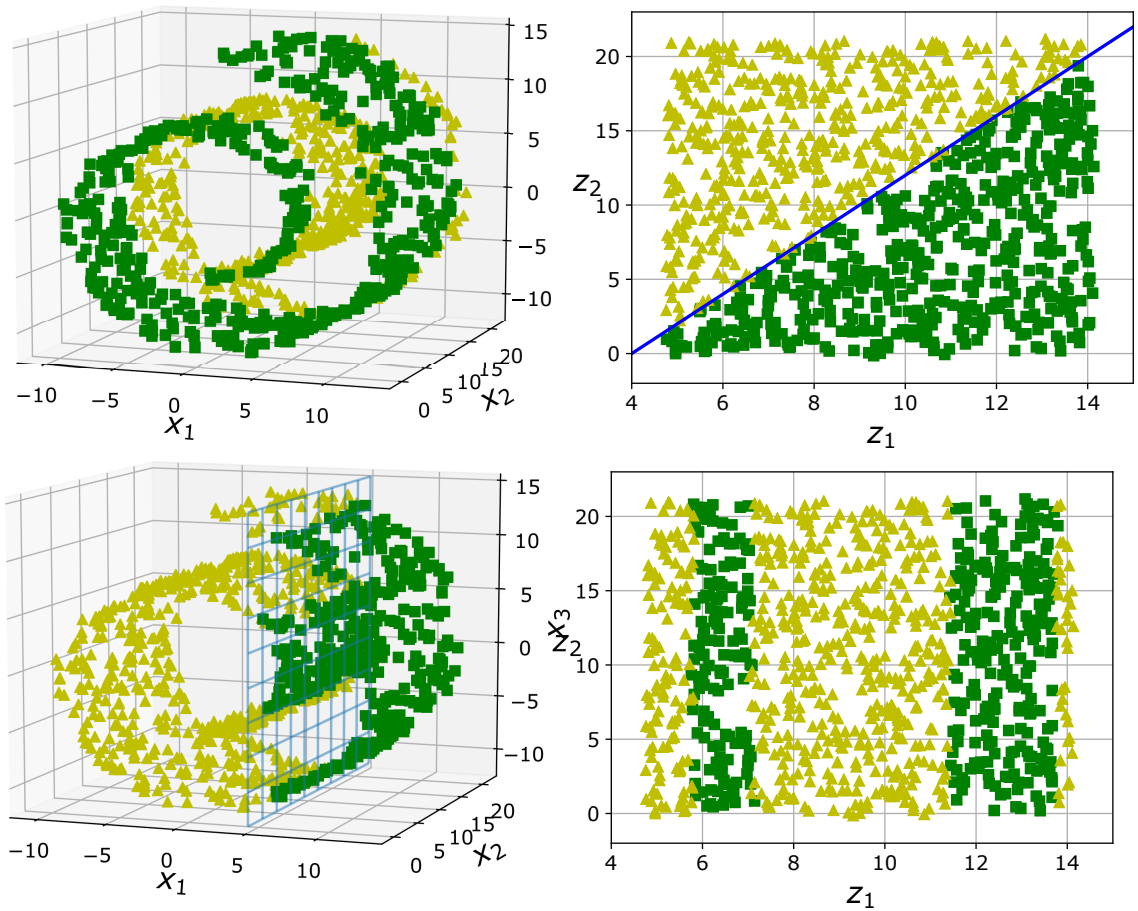
- El "*Swiss roll*" **es un ejemplo de una variedad 2D**.
- Un múltiple 2D es una forma 2D que se puede doblar y torcer en un espacio de dimensiones superiores.
- Una variedad d-dimensional es una parte de un espacio n-dimensional **(donde $d < n$)** que localmente se parece a un hiperplano d-dimensional.



Manifold assumption: la mayoría de los conjuntos de datos de alta dimensión del mundo real se encuentran cerca de un manifold de mucha más baja dimensión.

Manifold learning: (Variedades) Ejemplo I

El **Implicit assumption**: la tarea en cuestión (por ejemplo, clasificación o regresión) será más simple si se expresa en el espacio de dimensiones inferiores del manifold.



Manifold learning: Ejemplo I

- En la **fila superior de la Figura**, el rollo suizo se divide en dos clases: en el espacio 3D (a la izquierda), el límite de decisión sería bastante complejo, pero en el espacio múltiple desenrollado 2D (en el derecha), el límite de decisión es una simple línea recta.

- De lo contrario, en la **fila inferior de la Figura**, el límite de decisión se encuentra en $x_1 = 5$. Este límite de decisión parece muy simple en el espacio 3D original, pero se ve más complejo en el múltiple desenrollado.

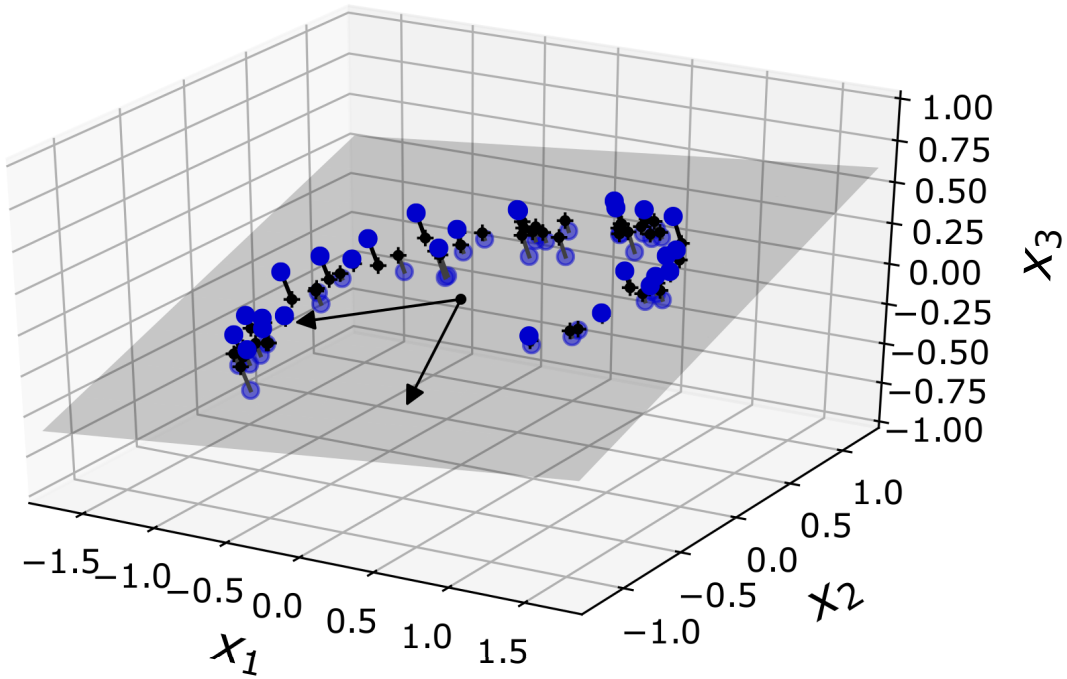
Si se reduce la dimensionalidad antes de entrenar un modelo, acelerará el entrenamiento, pero **no siempre** puede conducir a una solución mejor.

Contenido

PCA

PCA es el algoritmo de reducción de dimensionalidad más popular. ¿Cómo funciona?:

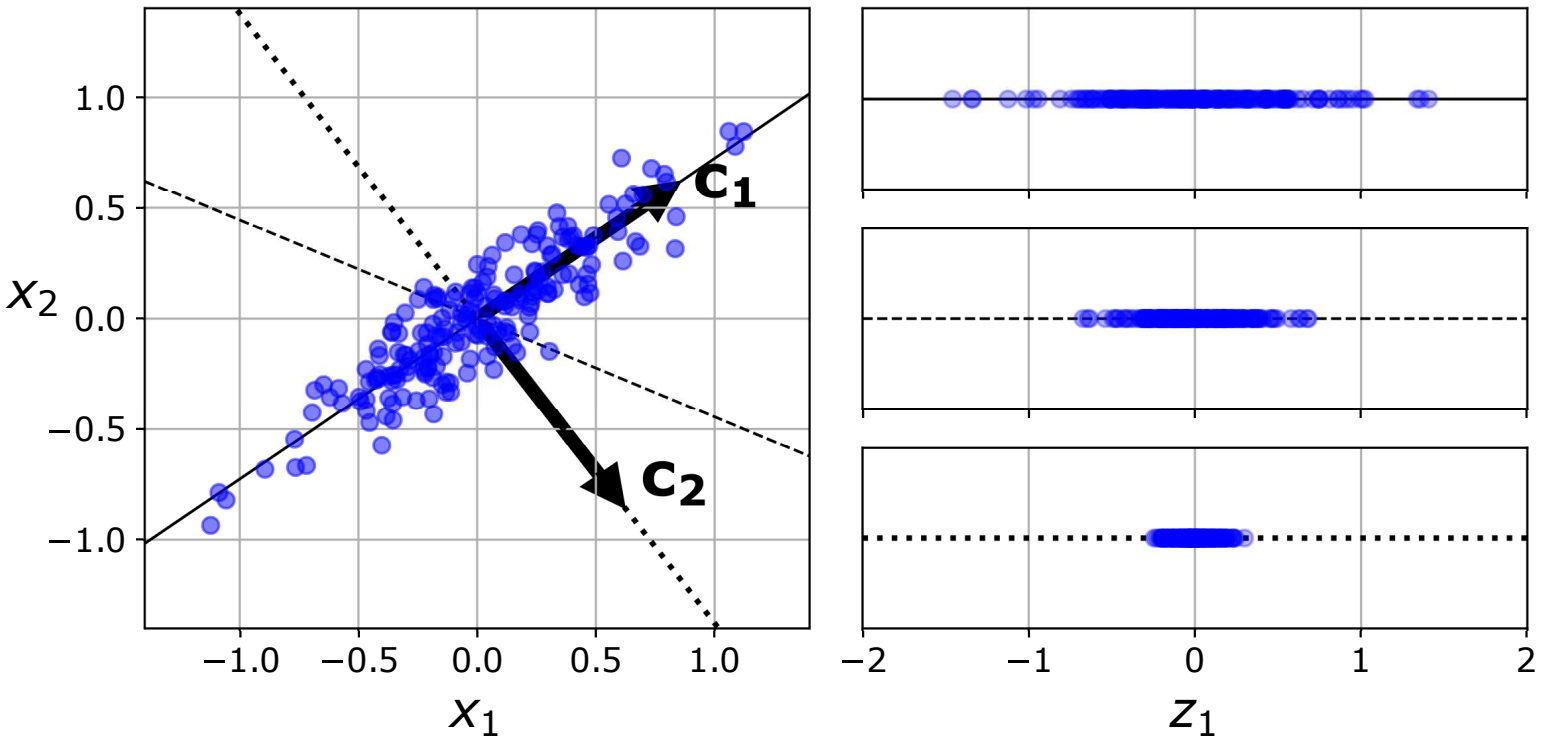
1. Identifica el hiperplano que se encuentra más cerca de los datos.
2. Luego proyecta los datos sobre él:



PCA: Preservando la varianza I

Antes de poder **proyectar** el conjunto de entrenamiento en un hiperplano de dimensiones inferiores, primero debemos elegir el hiperplano correcto.

- **Ejemplo:**

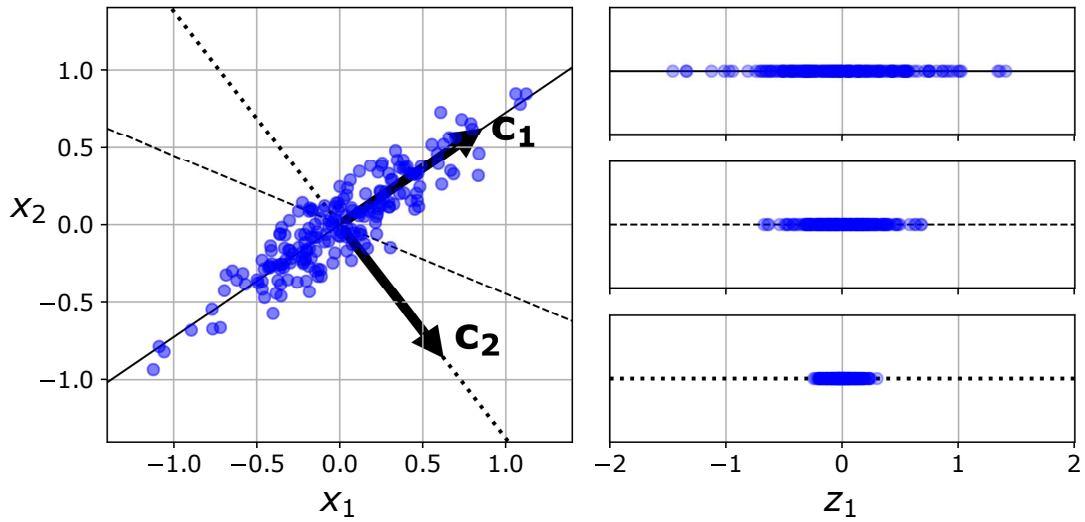


PCA: Preservando la varianza II

- En la **Figura de la izquierda** tenemos un conjunto de datos 2D, junto con tres ejes diferentes.
- En la **Figura de la derecha** está el resultado de la proyección de los datos en cada uno de estos ejes.
- Como podemos ver:
  - La proyección en la línea continua conserva la varianza máxima.
  - La proyección en la línea punteada conserva muy poca varianza.
  - La proyección en la línea discontinua conserva una cantidad intermedia de varianza.

Se selecciona el eje que **preserva la cantidad máxima de variación**, el que **minimiza la distancia cuadrática media** entre el conjunto de datos original y su proyección sobre ese eje.

PCA: Componentes principales I

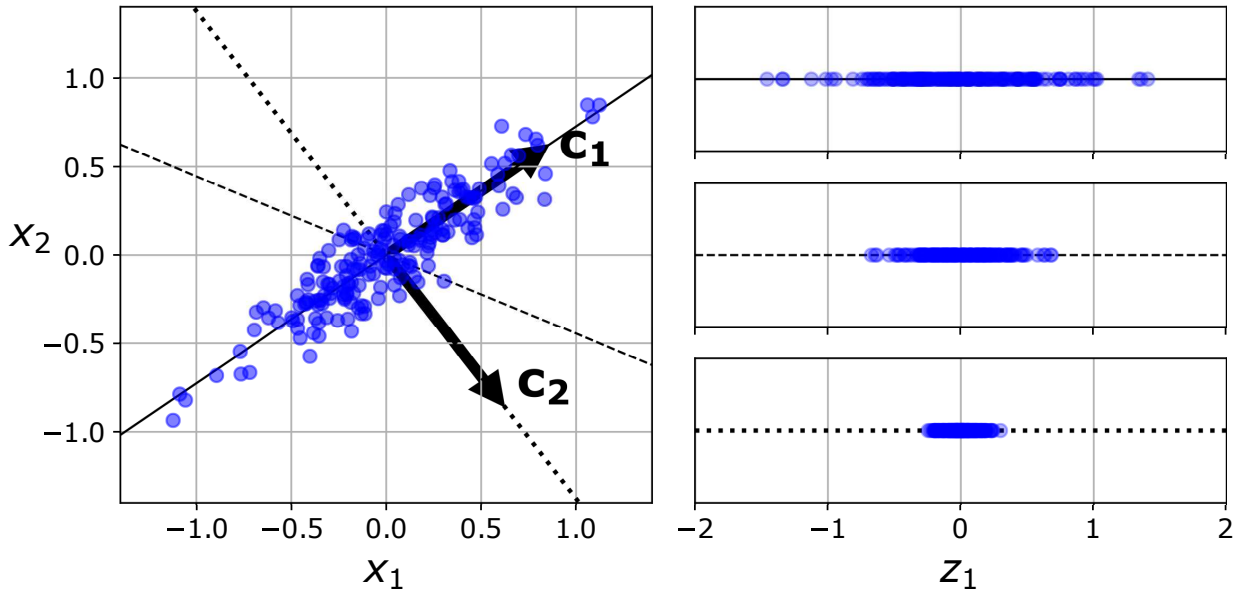


- PCA identifica el eje que representa la mayor cantidad de variación en el conjunto de entrenamiento (**línea continua**).
- También encuentra un segundo eje, **ortogonal al primero**, que representa la mayor cantidad de varianza restante (**línea punteada**).

PCA encontraría tantos ejes como el número de dimensiones en el conjunto de datos.

PCA: Componentes principales II

- El vector unitario que define el i-ésimo eje se llama i-ésimo componente principal **(PC)**. En la Figura, la primera PC es c_1 y la segunda PC es c_2.
- Las dos primeras PC están representadas por las flechas ortogonales en el plano, y la tercera PC sería ortogonal al plano (apuntando hacia arriba o hacia abajo).



PCA: Componentes principales III

* **Tip:** La dirección de los componentes principales no es estable. Sin embargo, generalmente seguirán en los mismos ejes.

Entonces, **¿cómo encontrar los componentes principales de un conjunto de entrenamiento?**.

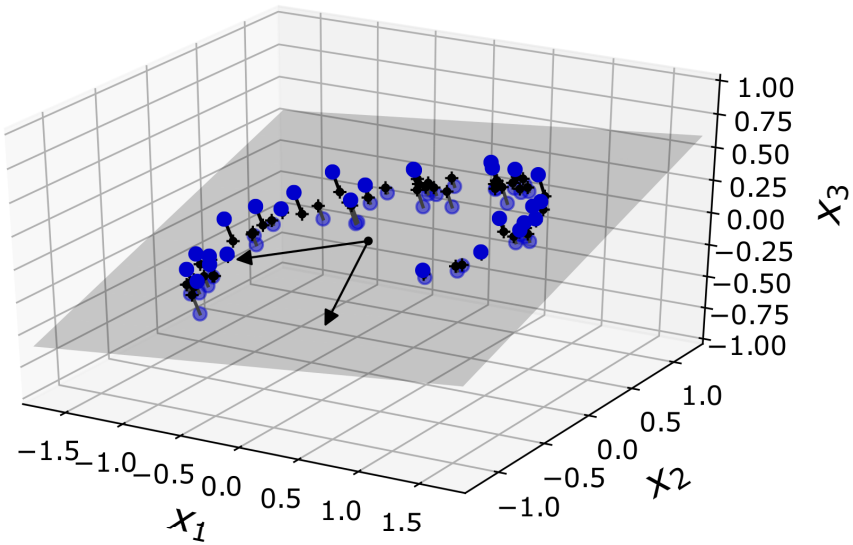
R/ Existe una técnica estándar de factorización de matriz llamada **Singular Value Decomposition (SVD)** que puede descomponer la matriz del conjunto de entrenamiento $\boldsymbol{X}$ en la multiplicación matricial de tres matrices $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, donde $\boldsymbol{V}$ contiene todos los componentes principales que estamos buscando:

$$\boldsymbol{V} = \begin{pmatrix} | & | & & | \\ c_1 & c_2 & \cdots & c_n \\ | & | & & | \end{pmatrix}$$

PCA: Proyectando a d-dimensiones I

- Se reduce la dimensionalidad a d-dimensiones **proyectándolo en el hiperplano definido por los primero d componentes principales**. Garantizando que la proyección conservará la mayor variación posible.

- En la Figura, el conjunto de datos 3D se proyecta hacia abajo en el plano 2D definido por los dos primeros PCs, **preservando una gran parte de la varianza del conjunto de datos**.



PCA: Proyectando a d-dimensiones II

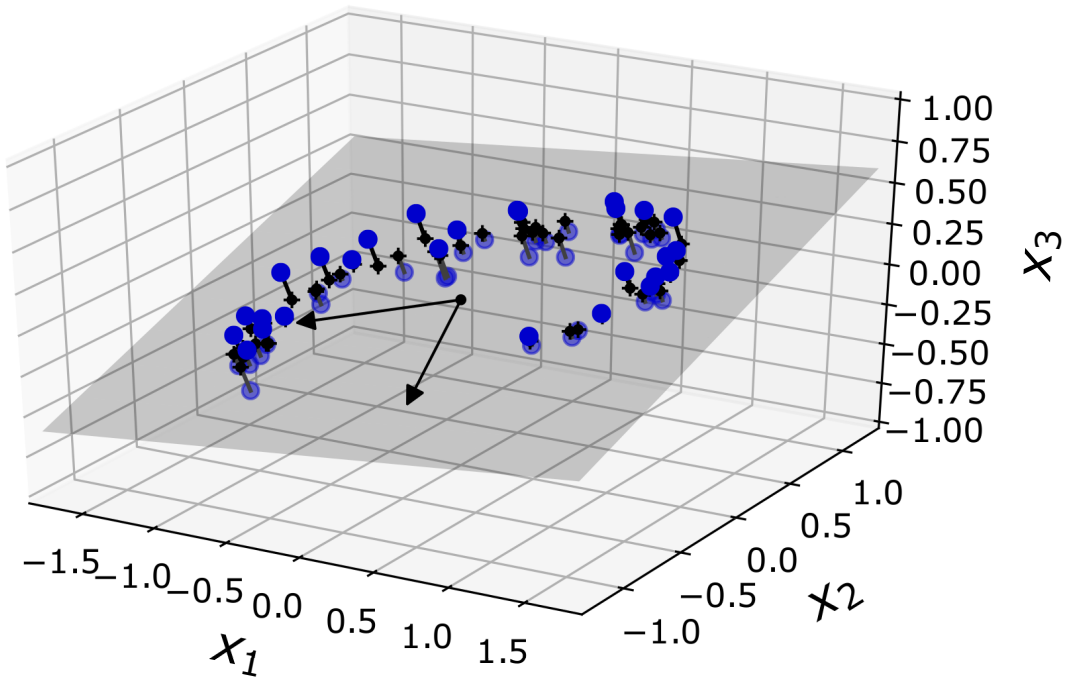
Para proyectar el conjunto de entrenamiento en el hiperplano, se calcula la multiplicación matricial de la matriz $\boldsymbol{X}$ del conjunto de entrenamiento por la matriz $\boldsymbol{W}_d$, que contiene los primeros d componentes principales, como se muestra en la siguiente ecuación:

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

PCA: Explained variance ratio

La **relación de varianza explicada** de cada componente principal, está disponible a través de la variable `explained_variance_ratio_`. Esta indica la proporción de la varianza del conjunto de datos que se encuentra a lo largo del eje de cada componente principal.

Ejemplo:



PCA: Elegir el número correcto de dimensiones

- **Generalmente es preferible elegir el número de dimensiones que se suman a una porción suficientemente grande de la varianza (por ejemplo,** $95\,\%$**).**

A menos, que el objetivo sea la visualización de datos; en ese caso, generalmente querrá reducir la dimensionalidad a 2 o 3.

PCA para compresión

- Obviamente, después de la **reducción de dimensionalidad**, el conjunto de entrenamiento ocupa mucho menos espacio.

- También **es posible descomprimir el conjunto de datos reducido** de nuevo a las dimensiones originales aplicando la transformación inversa de la proyección PCA. Por supuesto, esto no le devolverá los datos originales, ya que la proyección perdió un poco de información.

$$\boldsymbol{X}_{recovered} = \boldsymbol{X}_{d-proj} \boldsymbol{W}_d^\top$$

La distancia cuadrática media entre los datos originales y los datos reconstruidos se denomina **error de reconstrucción**

Variantes de PCA

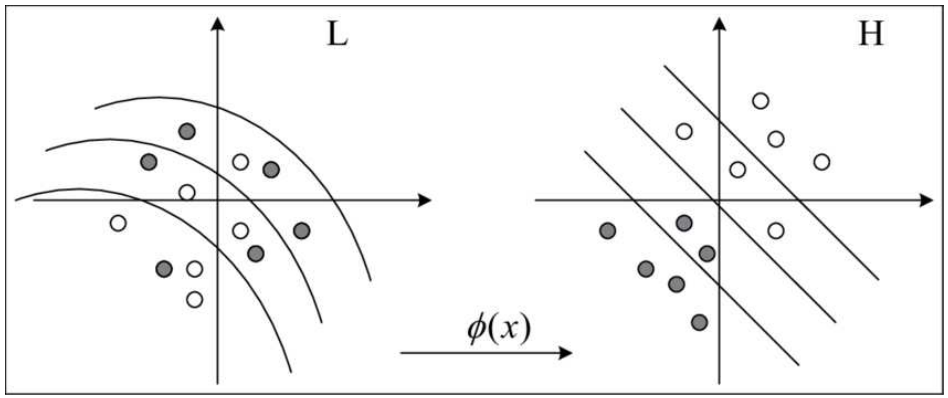
- **Randomized PCA:** Si configuramos el hiperparámetro `svd_solver` en "*randomized*", Scikit-Learn usa un algoritmo estocástico llamado PCA aleatorio que encuentra rápidamente una aproximación de los primeros d componentes principales.

- **Incremental PCA:** Un problema con las implementaciones anteriores de PCA es que requieren que todo el conjunto de entrenamiento se ajuste a la memoria para que el algoritmo se ejecute. PCA incremental **(IPCA)** divide el conjunto de entrenamiento en mini-lotes y alimentar un algoritmo de IPCA un mini-lote a la vez.

Kernel PCA

El *kernel trick*, mapea implícitamente las instancias en un espacio de muy alta dimensión (espacio de características), permitiendo la clasificación y regresión no lineal con SVMs.

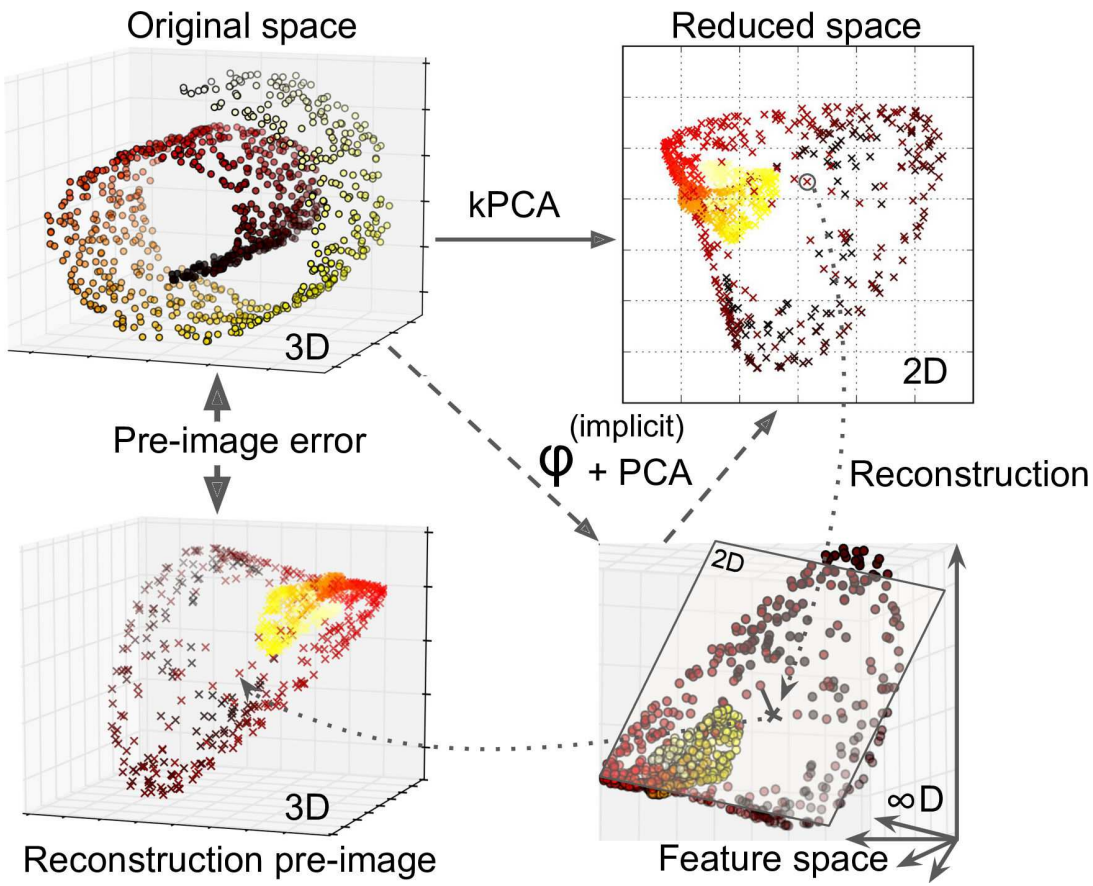
- **TIP:** Recuerde que **un límite de decisión lineal** en el espacio de características de alta dimensión corresponde a un **límite de decisión no lineal complejo** en el espacio original.



En PCA, se realizan proyecciones **no lineales complejas** para la reducción de la dimensionalidad. **Esto se llama Kernel PCA (kPCA)**

KPCA: selección de hiperparámetros I

Otro enfoque, esta vez **completamente sin supervisión**, es seleccionar el kernel y los hiperparámetros que producen el error de reconstrucción más bajo. Sin embargo, la reconstrucción no es tan fácil.



KPCA: selección de hiperparámetros II

1. Esto es mapear los datos en un espacio de características de dimensiones infinitas usando el feature map φ, y luego **proyectarlo usando PCA lineal**.

2. Al invertir el paso lineal de PCA para una instancia dada en el espacio reducido, el punto reconstruido estaría en el espacio de características, no en el original.

3. Dado que el espacio de características es de dimensión infinita, no podemos calcular el punto reconstruido, ni el error de reconstrucción.

4. Es posible encontrar un punto en el espacio original que se mapearía cerca del punto reconstruido. **Esto se llama reconstrucción pre-imagen**.

Se puede seleccionar el kernel y los hiperparámetros **que minimizan este error de reconstrucción** pre-imagen.

Contenido

Locally Linear Embedding (LLE)

- LLE es otra técnica de reducción de dimensionalidad **no lineal** (NLDR).
- LLE funciona midiendo primero cómo cada instancia de entrenamiento se relaciona linealmente con sus vecinos más cercanos.
- Luego busca una representación de baja dimensión del conjunto de entrenamiento donde estas **relaciones locales se conservan mejor**.
- Esto lo hace particularmente bueno para **desenrollar manifolds retorcidos**, especialmente cuando no hay demasiado ruido.

LLE: Cómo funciona? I

1. Para cada instancia de entrenamiento $\boldsymbol{x}^{(i)}$, el algoritmo identifica a sus k vecinos más cercanos. Luego intenta reconstruir $\boldsymbol{x}^{(i)}$ como una función lineal de estos vecinos. Específicamente, encuentra los pesos $w_{i,j}$ tal que la distancia al cuadrado entre $\boldsymbol{x}^{(i)}$ y $\sum_{j=1}^{m} w_{i,j}\boldsymbol{x}^{j}$ es lo más pequeña posible, suponiendo $w_{i,j} = 0$ si $x^{(j)}$ no es uno de los k vecinos más cercanos de $x^{(i)}$.

Por lo tanto, **el primer paso de LLE es un problema de optimización restringida**, donde $\boldsymbol{W}$ es la matriz de peso que contiene todos los pesos $w_{i,j}$. La segunda restricción **simplemente normaliza los pesos** para cada instancia de entrenamiento $\boldsymbol{x}^{(i)}$.

LLE: Cómo funciona? II

$$\widehat{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \sum_{i=1}^{m} \left(\boldsymbol{x}^{(i)} - \sum_{j=1}^{m} w_{i,j} \boldsymbol{x}^{(j)} \right)^2$$

sujeto a:

$$\begin{cases} w_{i,j} = 0 \\ \sum_{j=1}^{m} w_{i,j} = 1, \forall i = 1, 2, \cdots, m \end{cases}$$

$w_{i,j} = 0$ si $\boldsymbol{x}^{(j)}$ no es uno de los k vecinos cercanos de $\boldsymbol{x}^{(i)}$. La matriz de peso $\boldsymbol{W}$ codifica las relaciones lineales locales entre las instancias de entrenamiento.

2. El segundo paso es **mapear las instancias de entrenamiento en un espacio d-dimensional** (donde $d < n$) mientras se preservan estas relaciones locales tanto como sea posible. Si $\boldsymbol{z}^{(i)}$ es la imagen de $\boldsymbol{x}^{(i)}$ en este espacio d-dimensional, entonces queremos que la distancia al cuadrado entre $\boldsymbol{z}^{(i)}$ y $\sum_{j=1}^{m} \widehat{w}_{i,j} \boldsymbol{z}^{(j)}$ **sea lo más pequeña posible**.

LLE: Cómo funciona? III

Esta idea nos lleva ahora a un problema de optimización **sin restricciones** descrito a continuación:

$$\widehat{\boldsymbol{Z}} = \arg\min_{\boldsymbol{Z}} \sum_{i=1}^{m} \left(\boldsymbol{z}^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j} \boldsymbol{z}^{(j)} \right)^2$$

En lugar de mantener las instancias fijas y encontrar los pesos óptimos, estamos haciendo lo contrario: mantener los pesos fijos y encontrar la posición óptima de las imágenes de las instancias en el espacio de baja dimensión. Tengamos en cuenta que $\boldsymbol{Z}$ es la matriz que contiene todo $\boldsymbol{z}^{(i)}$.

Desafortunadamente, el algoritmo LLE escala escasamente a conjuntos de datos muy grandes.

Contenido

Otras técnicas de reducción de dimensión

- El **escalado multidimensional (MDS)** reduce la dimensionalidad al intentar preservar las distancias entre las instancias.
- **Isomap** crea un gráfico conectando cada instancia a sus vecinos más cercanos, luego reduce la dimensionalidad mientras intenta preservar las distancias geodésicas entre las instancias.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** reduce la dimensionalidad al tratar de mantener instancias similares cercanas e instancias distintas lejos. Se utiliza principalmente para la visualización, en particular para visualizar grupos de instancias en espacios de alta dimensión.

t-distributed stochastic neighbor embedding (t-SNE) I

- Dado un conjunto de N objetos de alta dimensión $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, t-SNE **calcula primero las probabilidades p_{ij} que son proporcionales a la similitud de los objetos $\boldsymbol{x}_i$ y $\boldsymbol{x}_j$**, de la siguiente manera:

$$p_{j|i} = \frac{\exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2 / 2\sigma_i^2\right)} \tag{1}$$

- "La similitud del datapoint $\boldsymbol{x}_j$ con el datapoint $\boldsymbol{x}_i$ es la probabilidad condicional, $p_{j|i}$, que $\boldsymbol{x}_i$ elegiría $\boldsymbol{x}_j$ como vecino si los vecinos fueran seleccionados en proporción a su densidad de probabilidad bajo una Gaussiana centrada en $\boldsymbol{x}_i$".

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2}$$

t-distributed stochastic neighbor embedding (t-SNE) II

- Además, las probabilidades con $i = j$ se establecen en cero: $p_{ii} = 0$

– El ancho de banda del kernel Gaussiano se adapta a la densidad de los datos: se utilizan **valores más pequeños** de σ_i en partes **más densas** del espacio de datos.

– Dado que el kernel Gaussiano usa la distancia euclidiana $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, se ve afectado por la maldición de la dimensionalidad, y en datos de alta dimensión **cuando las distancias pierden la capacidad de discriminar**, la p_{ij} se vuelve **demasiado similar**.

t-distributed stochastic neighbor embedding (t-SNE) III

- t-SNE tiene como objetivo aprender un mapa d-dimensional $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ (con $\boldsymbol{y}_i \in \mathbb{R}^d$) que **refleja las similitudes** p_{ij} lo mejor posible. Con este fin, mide las similitudes q_{ij} entre dos puntos en el mapa $\boldsymbol{y}_i$ y $\boldsymbol{y}_j$, utilizando un enfoque muy similar. Específicamente, q_{ij} se define como:

$$q_{ij} = \frac{\left(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2\right)^{-1}} \tag{3}$$

- Se usa una **distribución Student-t de cola pesada** para medir similitudes entre puntos de baja dimensión para permitir que objetos diferentes se modelen muy separados en el mapa. Tenga en cuenta que también en este caso configuramos $q_{ii} = 0$

t-distributed stochastic neighbor embedding (t-SNE) IV

- Las ubicaciones de los puntos $\boldsymbol{y}_i$ en el mapa se determinan **minimizando la divergencia (no simétrica)** *KullbackLeibler* de la distribución Q de la distribución P, es decir:

$$KL\left(P||Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{4}$$

- La minimización de la divergencia *Kullback-Leibler* con respecto a los puntos $\boldsymbol{y}_i$ se realiza usando **Gradiente Descendente**. El resultado de esta optimización es un mapa que refleja bien las similitudes entre las entradas de alta dimensión.

Referencias I

Géron, A., (2019).

Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.
O'Reilly Media.

Murphy, K. (2012).

Machine Learning: A Probabilistic Perspective.
The MIT Press. 1st Edition. 2012

Bishop, C. (2006).

Pattern recognition.
Ed. Springer. 2006