Reducción de la dimensionalidad

Índice

Referencia principal

Cap.8 de "Hands On Machine Learning with Scikit Learn and TensorFlow"
`(Todas las imágenes pertenecen a dicha referencia salvo que se indique lo contrario)`

Al terminar este tema conoceremos algunas técnicas para reducir la dimensionalidad de los datos. Esto da paso a técnicas de visualización de la información y permite acelerar los algoritmos.

1. Intuición

A lo largo del curso hemos utilizado el conjunto de datos '0' vs. '1' para explicar distintas técnicas de aprendizaje supervisado. En todas ellas, excepto en *Random Forest*, los ejemplos han sido bidimensionales, de manera que se podía presentar en una gráfica los resultados.

Sin embargo este conjunto de datos era, en origen, 784-dimensional, porque eran imágenes en escala de grises de resolución 28×28. En los códigos Python que se han proporcionado se realiza ingeniería de características sobre la imagen, obteniendo hasta 8.

Pero ni con ejemplos 8-dimensionales ni con 784-dimensionales es posible hacer una representación gráfica de estos. Lo más que podemos conseguir es dibujar en tres dimensiones.

¿Se puede reducir la dimensionalidad con la menor pérdida de información posible?

Sí. Hay varias posibilidades:

- Si hubiera características fuertemente correlacionadas, entonces tendríamos información redundante y podríamos quedarnos sólo con una de ellas.
- Si hay características que no aportan nada, podemos eliminarlas tranquilamente.
- Se pueden proyectar los ejemplos n-dimensionales en un subespacio 2 ó 3-dimensional. La Figura 1 representa el proceso de proyección de datos tridimensionales en un plano, o sea en un subespacio bidimensional. Con esta opción, obviamente, se pierde la información de las dimensiones que están en la dirección de la proyección, por lo que se trata de buscar el subespacio que minimiza esta pérdida.
- Proyectar en una variedad (*manifold* en inglés).

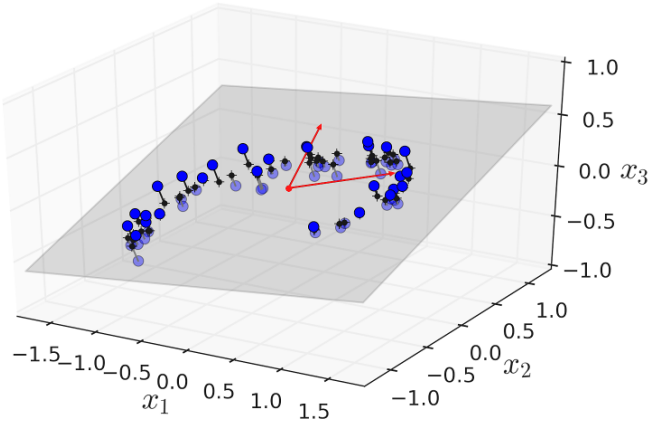


Figura 1: Reducción de 3 a 2 dimensiones proyectando sobre el subespacio que más información recoge

2. Análisis de Componentes Principales (PCA)

PCA es el método más popular de reducción de dimensión mediante proyección en un subespacio.

Este método identifica el subespacio donde la proyección pierde la menor información posible. Para dar una intuición sobre ello, supondremos un conjunto de datos bidimensional cuyos ejemplos son los puntos azules de la Figura 2. Cada ejemplo tiene, por tanto, dos características (x_1, x_2). Si proyectamos el conjunto de datos sobre el subespacio de dimensión 1 (o sea una recta), los ejemplos en este subespacio estarán descritos por 1 única característica, z. Pero según elijamos este subespacio llegaremos a un conjunto de datos proyectados diferente. Como se puede ver a la derecha de la Figura 2, cuando los datos se proyectan sobre el subespacio $\mathbf{c}_1$ se obtiene la máxima varianza de estos; mientras que cuando se proyectan sobre el subespacio $\mathbf{c}_2$ se obtiene la mínima. Cualquier proyección sobre otro subespacio da lugar a una varianza intermedia entre estas dos.

Parece lógico que, si tenemos que "bajar" de 2 dimensiones a 1, elijamos la proyección sobre $\mathbf{c}_1$ porque es la que mantiene la mayor información posible.

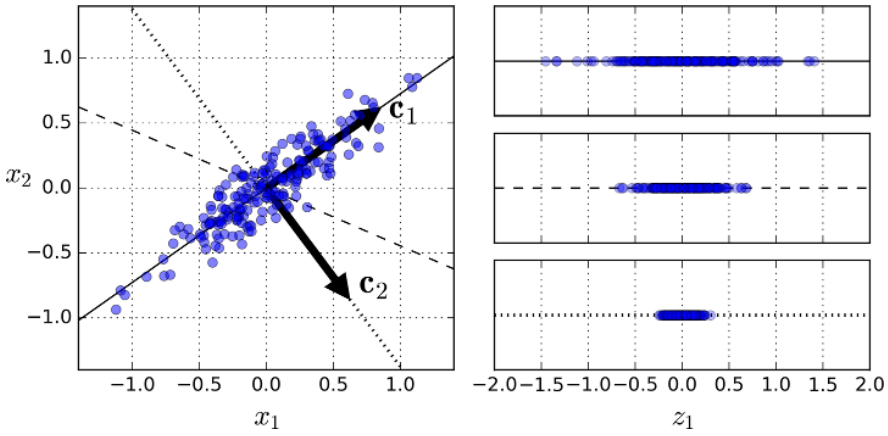


Figura 2: Proyecciones de un conjunto de datos 2D sobre tres subespacios 1D

Si vemos el conjunto de datos como una matriz $\mathbf{X}$ de $m \times n$ elementos podemos calcular su descomposición en valores singulares (SVD), es decir $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$. Tanto $\mathbf{U}$, Σ como $\mathbf{V}$ son matrices, por lo que la SVD es, a fin de cuentas, una factorización de la matriz $\mathbf{X}$. En concreto, si el conjunto de datos consta de m ejemplos n-dimensionales, entonces $\mathbf{X}$ es una matriz $(m \times n)$, $\mathbf{U}$ es una matriz $(m \times m)$, $\mathbf{V}$ es una matriz $(n \times n)$ y Σ es una matriz $(m \times n)$.

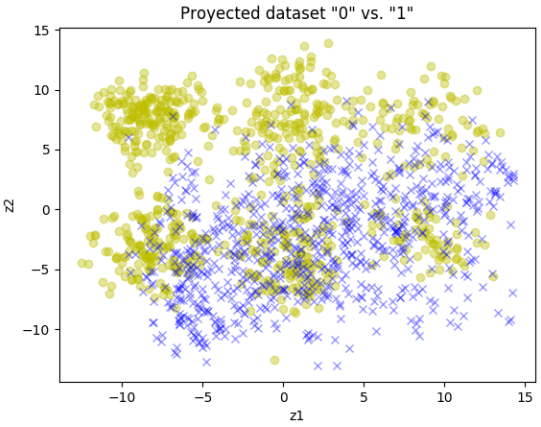
Las columnas de la matriz $\mathbf{V}^T$ son vectores denominados **componentes principales** y son una base del espacio en el que viven los ejemplos. Si recordamos el álgebra de espacios vectoriales, esto significa que cualquier punto del espacio n-dimensional se alcanza como una combinación lineal de los vectores de dicha base. Si sólo elegimos algunos vectores, entonces estos serán generadores de un subespacio.

La matriz Σ es diagonal y contiene los valores singulares, que son una generalización de los autovalores para el caso, muy habitual, de que la matriz que estamos intentando factorizar no sea cuadrada. Cada valor singular tiene asociado su componente principal correspondiente, que naturalmente es una generalización del autovector.

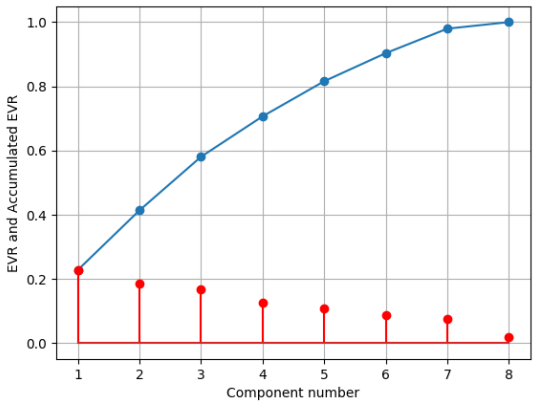
El módulo del valor singular da una medida de la importancia de dicho componente principal, por lo que la matriz Σ se suele ordenar de manera que el valor singular mayor ocupe la posición $(1, 1)$, el siguiente la posición $(2, 2)$, y así sucesivamente.

Sea $\mathbf{W}$ la matriz formada por los d primeros componentes principales, con $d < n$. Como hemos dicho, $\mathbf{W}$ base generadora de un subespacio d-dimensional. Recordando de nuevo el álgebra de los primeros cursos de grado, una vez que tenemos una base del subespacio obtener el conjunto de datos proyectado $\mathbf{X}_{\text{proy}}$ es simplemente $\mathbf{X}_{\text{proy}} = \mathbf{XW}$.

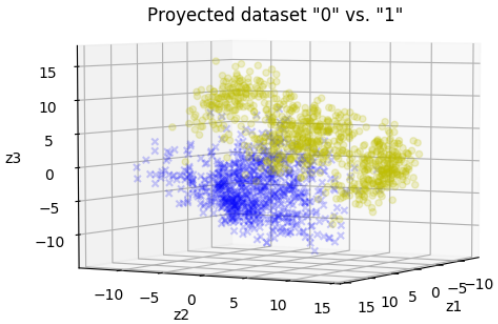
En la Figura 3 se muestran los resultados de proyectar el conjunto de datos '0' vs. '1' en el subespacio generado por los 2 y 3 primeros componentes principales. Además se realizó la descomposición en todos los componentes. Cada componente contribuye con un cierto porcentaje a la "reconstrucción" del conjunto original. Normalmente este porcentaje se llama Porcentaje de Varianza Explicada (*Explained Variance Ratio*, EVR). Como se puede ver en la Figura 3(d), los componentes principales se ordenan de mayor a menor EVR, o lo que es lo mismo, de mayor a menor valor singular. Además se puede hacer la suma acumulada de EVR, de modo que según se añaden componentes principales, aumentamos el porcentaje hasta el 100 %.



(a)



(c)



(b)

```
 First 8 principal components
[[-0.24  0.49  0.79 -0.02  0.20  0.08  0.00 -0.14]
 [ 0.01  0.01  0.07  0.53 -0.48  0.54  0.40 -0.00]
 [-0.12 -0.02 -0.15 -0.48  0.32  0.22  0.75  0.00]
 [-0.14 -0.03 -0.09 -0.38 -0.02  0.75 -0.50  0.02]
 [ 0.04 -0.06 -0.11 -0.00 -0.02  0.00 -0.01 -0.98]
 [-0.05 -0.84  0.43  0.13  0.27  0.07 -0.00 -0.00]
 [-0.67  0.07 -0.35  0.47  0.42  0.03 -0.07 -0.00]
 [-0.66 -0.18  0.07 -0.29 -0.59 -0.26  0.04 -0.01]]

 First 8 singular values
[ 259.1  233.7  221.8  192.6  179.4  160.5  149.8  77.6]

 Explained Variance Ratio
[ 0.22   0.18   0.16   0.12   0.10   0.08   0.07   0.02]

 Accumulated Explained Variance Ratio
[ 0.22   0.41   0.58   0.70   0.81   0.90   0.97   1.]
```

(d)

Figura 3: (a-b) Proyección del conjunto de datos '0' vs. '1' en el subespacio generado por los 2 y 3 primeros componentes principales respectivamente. (c) Porcentaje de varianza explicado según se añaden componentes principales, en rojo, y su acumulado, en azul. (d) Resultados numéricos. [Fuente: Original de A. Cuesta]

2.1. Consideraciones prácticas

Reconstrucción del conjunto de datos A partir de los componentes principales podemos reconstruir los datos. La Figura 4 muestra los ejemplos del conjunto de datos '0' vs. '1', pero utilizando las dos características habituales (W y H) ya que no podemos recrear una imagen 8-dimensional, tras la reconstrucción a partir de 3, 5, 7 y 8 componentes principales. Evidentemente, con los 8 componentes se reconstruye el conjunto completamente, y según los eliminamos perdemos información.

Este es el mismo proceso que se utilizaría para reconstruir una imagen a partir de sus componentes principales, que la codificarían comprimida. La perdida de información sería entonces la pérdida debida a la compresión, según qué formato se utilice.

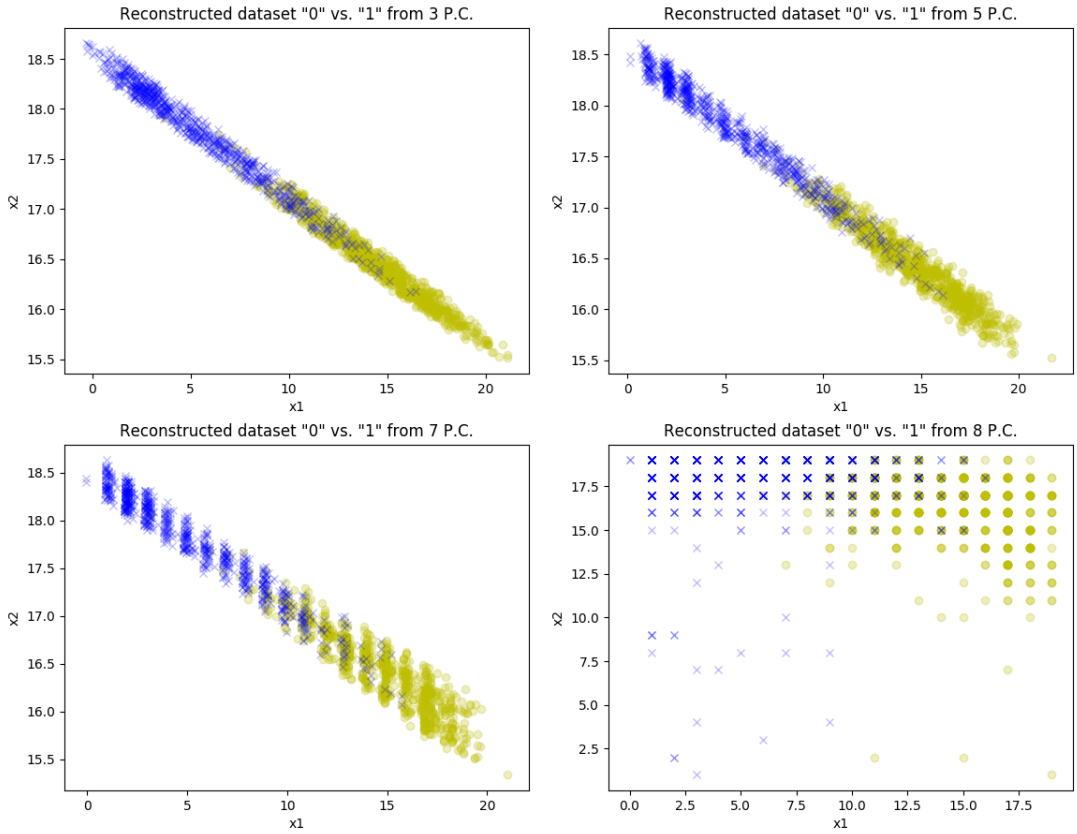


Figura 4: Reconstrucción del conjunto de datos '0' vs. '1' con 3, 5, 7 y 8 componentes principales. Los puntos mostrados se corresponden a las características W, en el eje horizontal, y H en el vertical. [Fuente: Original de A. Cuesta]

¿Cuántos componentes usar? Normalmente se fija un cierto porcentaje de varianza explicada acumulada y se utilizan los componentes necesarios hasta alcanzarla. Por ejemplo, para el caso de la Figura 3, si nos fijásemos un 80 % necesitaríamos utilizar los 5 primeros componentes principales.

En Python se puede fijar tanto el número de componentes a utilizar como el porcentaje de varianza explicada con el parámetro `n_components`.

- Si `n_components` es un número entero entre 1 y n, la dimensión de nuestro conjunto de datos, entonces fijaremos el número de componentes.
- Si `n_components` es un número fraccionario entre 0 y 1, entonces fijaremos el porcentaje de varianza explicada acumulada.

Preparación de los datos El algoritmo PCA presupone que los datos están centrados en cero, es decir tienen media nula. Como vimos a principio de curso esto se consigue restando al conjunto de datos su media.

Por otro lado, la descomposición SVD de una matriz está implementada en prácticamente cualquier lenguaje de cálculo numérico. Esto significa que se puede implementar el algoritmo desde cero en unas pocas lineas.

En caso de utilizar el método PCA de Python, se puede utilizar el conjunto de datos sin centrar porque ya lo hace él internamente. Pero si queremos construir el algoritmo debemos recordar centrarlo.

Diferencia entre PCA y RF La semana pasada aprendimos que con *Random Forest* (RF) podíamos estimar la importancia de las características. La idea de PCA es próxima, pero muy diferente. Es importante no confundir un componente principal con una característica.

- En el conjunto de datos original, cada característica es una columna. Si elegimos eliminar características entonces estamos eliminando columnas es decir "subcaracterizando".
- Cuando nos quedamos con k componentes principales, cada columna del conjunto de datos proyectado es una combinación lineal de todas las columnas del conjunto original. El conjunto de datos proyectado tendrá k columnas, pero no se corresponden con las k columnas que RF indicaría como más importantes.

PCA incremental Python incorpora la clase `IncrementalPCA`, que implementa una versión de PCA que se puede ejecutar sobre grandes conjuntos de datos que, previamente, han sido partidos en mini-lotes. `IncrementalPCA` recibe la secuencia de lotes en un bucle `for` y, al terminar, tenemos el mismo resultado que si hubiera estado todo el conjunto de datos en memoria desde el principio.

2.2. KernelPCA

Hemos explicado PCA a partir de la descomposición SVD de la matriz de datos. Igualmente se puede explicar a partir de la descomposición en autovalores y autovectores de la matriz de covarianza de los datos.

Sea $\mathbf{K} = \mathbf{X}^T\mathbf{X}/(m-1)$ dicha matriz de covarianza que, como sabemos, es cuadrada $(n \times n)$ y definida positiva; y cuya descomposición es $\mathbf{K} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$, donde $\mathbf{\Lambda}$ es una matriz diagonal que contiene los autovalores, ordenados de mayor a menor, y las columnas de $\mathbf{W}$ los autovectores. Por otro lado, la descomposición en valores singulares del conjunto de datos es $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$. Por tanto

$$\mathbf{X}\mathbf{X}^T = \left(\mathbf{U}\Sigma\mathbf{V}^T\right)\left(\mathbf{U}\Sigma\mathbf{V}^T\right)^T = \left(\mathbf{U}\Sigma\mathbf{V}^T\right)\left(\mathbf{V}\Sigma\mathbf{U}^T\right) = \mathbf{U}\Sigma(\mathbf{V}^T\mathbf{V})\Sigma\mathbf{U}^T$$

En la última igualdad se ha utilizado la propiedad de la traspuesta de la multiplicación de matrices y el hecho de que la matriz Σ es diagonal, y por tanto coincide con su traspuesta. Y recordando que los componentes principales son ortogonales, es decir que $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{1}$,

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\Sigma\Sigma\mathbf{U}^T = \mathbf{U}\Sigma^2\mathbf{U}^T.$$

Y por otro lado

$$\mathbf{K} = \mathbf{X}^T\mathbf{X}/(m-1) = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T,$$

así que, comparando estas dos últimas, la matriz de autovalores de la descomposición de la covarianza es el cuadrado de la matriz de valores singulares del conjunto de datos, salvo por la constante $(m-1)$.

¿Y si en vez de calcular $\mathbf{X}^T\mathbf{X}$ utilizamos un kernel $\mathcal{K}(\mathbf{X}, \mathbf{X}^T)$ que nos devuelva este resultado en otro espacio diferente?

El resultado es un "KernelPCA", con el que podemos obtener componentes principales como si hubiéramos mapeado el conjunto de datos a otro espacio, aunque realmente no lo hayamos hecho (recordar que no tenemos la función de mapeo). La Figura 5 muestra el resultado de utilizar un kernel RBF sobre el conjunto de datos '0' vs. '1' con 2 y 3 componentes principales.

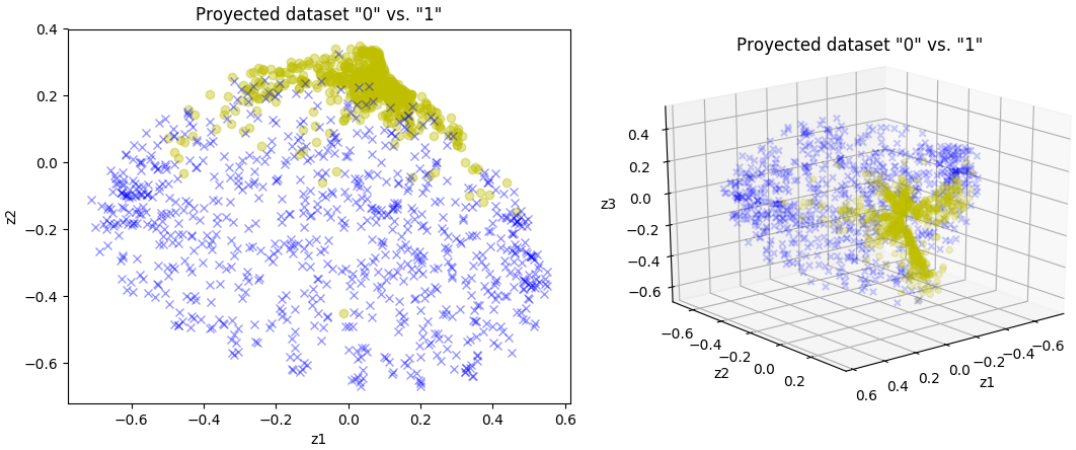


Figura 5: Proyección del conjunto de datos '0' vs. '1' en los 2 y 3 primeros componentes principales con un PCA de kernel RBF, con parámetro 0.01. [Fuente: Original de A. Cuesta]

La reconstrucción del conjunto original ya no es tan sencilla. Cuando se aplica el truco del kernel estamos calculando las distancias en un espacio con más dimensiones, puede que incluso infinito-dimensional. Podemos comprobar con Python que si intentamos reconstruir un conjunto de datos con más componentes principales que dimensiones utilizando PCA no es posible, pero utilizando KPCA sí.

3. Locally Linear Embedding (LLE)

Esta es una técnica de Aprendizaje de Variedades (*Manifold Learning*). Una variedad (*manifold*) es un espacio donde, a escala local, se conservan las propiedades euclídeas. Por ejemplo, sabemos que la tierra es redonda, sin embargo durante años se pensó que era plana porque, a nuestra escala, no apreciamos la curvatura. En general, cualquier espacio que a "pequeña escala", se puede ver y tratar como un espacio de menor dimensión, habitualmente un plano, es una variedad.

Por este motivo, si localmente podemos empotrar o asimilar nuestro espacio n-dimensional a un plano entonces, extendiendo esta vecindad local, podemos construir una "superficie" sobre la que proyectamos el conjunto de datos. Recuperando el ejemplo de la tierra, es lo que hacemos cuando representamos la superficie de la tierra en un mapa.

El procedimiento es el siguiente:

1. Para cada ejemplo i del conjunto de datos, $\mathbf{x}^{(i)}$, el algoritmo identifica los k vecinos más próximos
2. Se reconstruye dicho ejemplo mediante una función lineal de estos ejemplos, es decir:

$$\mathbf{x}^{(i)}_{\texttt{rec}} = \sum_{j=1}^{k} w_{ij}\mathbf{x}^{(j)}.$$

   Naturalmente, trataremos de que $\mathbf{x}^{(i)}$ y $\mathbf{x}^{(i)}_{\texttt{rec}}$ sean lo más parecidos posible, y para ello buscamos los pesos óptimos, es decir la matriz $\mathbf{W}^*$

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i=1}^{m} \left\| \mathbf{x}^{(i)} - \sum_{j=1}^{k} w_{ij}\mathbf{x}^{(j)} \right\|^2$$

   Frecuentemente se normalizan los pesos de manera que $\sum_{j=1}^{m} w_{ij} = 1$ para $i = 1, 2, \ldots, m$.
3. Sea $\mathbf{Z} = \{\mathbf{z}^{(i)}\}$ el mapeo de $\mathbf{X} = \{\mathbf{x}^{(i)}\}$. Como LLE preserva las distancias localmente, para obtener $\mathbf{Z}$ intentaremos minimizar la distancia entre vecinos del conjunto, dada la matriz de pesos $\mathbf{W}^*$ obtenida en el paso anterior. Es decir:

$$\mathbf{Z}^* = \arg\min_{\mathbf{Z}} \sum_{i=1}^{m} \left\| \mathbf{z}^{(i)} - \sum_{j=1}^{k} w^*_{ij}\mathbf{z}^{(j)} \right\|^2$$

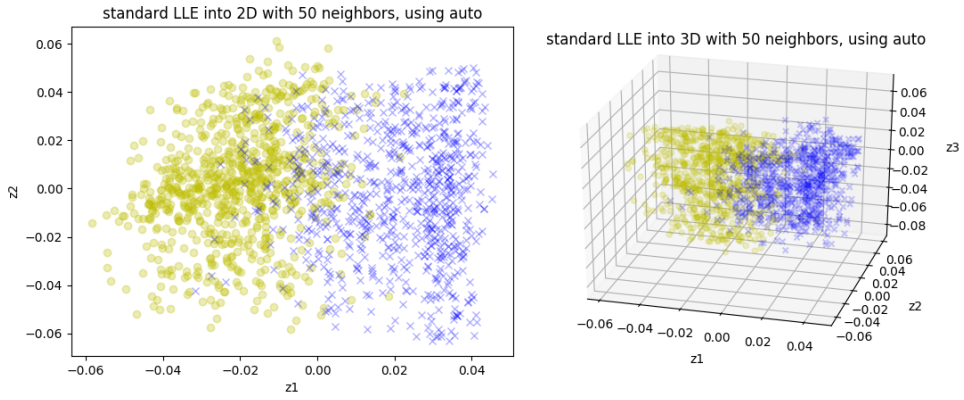


Figura 6: Proyección del conjunto de datos '0' vs. '1' en una variedad 2D (izq.) y en una 3D (der.) mediante LLE estandar, con 50 vecinos y método 'auto'. [Fuente: Original de A. Cuesta]

4. Linear Discriminant Analisis (LDA)

En el tema de modelos generativos aprendimos a clasificar utilizando LDA. Recordar que para ello asumíamos que todas las clases tenían distribuciones MVN y, además, con la misma covarianza. La superficie de decisión que se obtiene es un hiperplano plano que resuelve el compromiso entre las MVN de cada clase.

Pero si cambiamos el punto de vista, y consideramos la proyección del conjunto de datos sobre el hiperplano ortogonal al discriminante, entonces los puntos de dicha proyección tendrán el menor solapamiento posible, como se muestra en el ejemplo de la Figura 7.

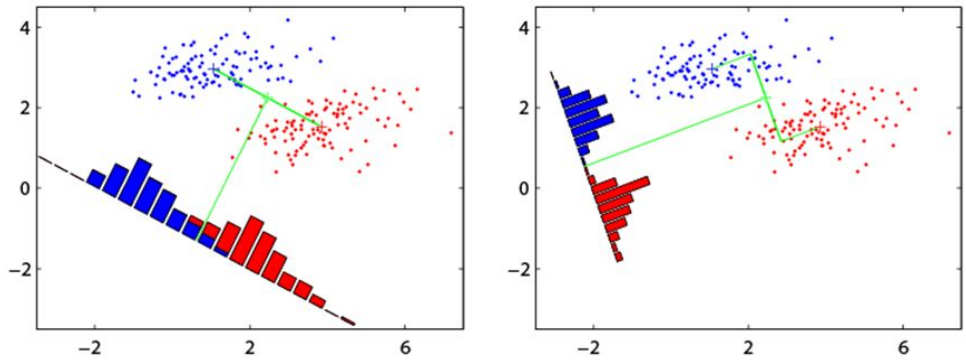


Figura 7: Las proyecciones de las clases se solapan más a la izquierda que a la derecha. LDA nos asegura la máxima separación. [Fuente: "Pattern recognition and machine learning" C.M. Bishop]

Las condiciones de esta técnica son bastante fuertes ya que normalmente los datos de cada clase suelen ser multimodales, e incluso si asumiéramos que se concentran entorno a su media, sería muy raro que cada clase además tuvieran la misma matriz de covarianza.

Por tanto es preferible restringir esta técnica a problemas multiclase donde, además, los ejemplos tengan una distribución MNV.

Índice alfabético