Universidad de Valladolid

Facultad de Ciencias

TRABAJO DE FIN DE GRADO

Grado en MATEMÁTICAS

MÉTODOS DE APROXIMACIÓN, COMPLETADO Y SEPARACIÓN DE MATRICES EN INTELIGENCIA ARTIFICIAL

Autor: Sarah Miguel Cournane

Tutor: Eustasio del Barrio Tellado

Julio 2020

Índice

1. Introducción

Si Alan Turing pudiese ver la enorme revolución que ha habido en el mundo tras la máquina Turing que él mismo ideó, no sabemos si se sorprendería. Ya en 1950, (ver [1]), defendía que la computación podría imitar el pensamiento humano. Tampoco sabemos si se sorprenderían John McCarthy, Marvin Minsky y Claude Shannon, tres científicos destacados de la época, quienes en 1956 dieron la definición de Inteligencia Artificial: "*la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cálculo inteligentes*". Estos visionarios ya eran conscientes del enorme potencial que presentaban los ordenadores y de hecho, predijeron que la sociedad no tardaría en vivir rodeada de máquinas. Sin embargo, en su época se encontraron con varias barreras para poder desarrollar la Inteligencia Artificial. Para empezar, los ordenadores eran bastante menos eficientes; aunque parezcan mucho las 40 mil operaciones por segundo que eran capaces de hacer los ordenadores en los años 50, en la actualidad pueden hacer 100 millones por segundo. Por otra parte, la capacidad de almacenar datos era prácticamente nula, todo en papel, mientras que ahora en dos centímetros cúbicos tenemos un USB de 128 Gigabytes. Además, la tecnología no era económica; comparando únicamente este almacenamiento de datos vemos la gran diferencia de precios. En 2020 almacenar un giga de datos cuesta menos de dos céntimos mientras que en 1956 salía a 9,2 millones de dólares. La figura (1) representa el estudio llevado a cabo por John C. McCallum (ver[2]) de la evolución de los precios de distintas unidades a lo largo de los años.

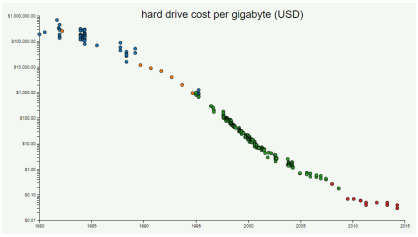


Figura 1: Evolución del precio de almacenamiento

El crecimiento de la Inteligencia Artificial ha sido entonces gradual. De hecho, los logros en esta disciplina siempre han ido de la mano de los avances en el mundo de las TIC (*Tecnología de la Información y de las Comunicaciones)*. Uno de los hitos más importantes fue el ordenador Deep Blue de IBM que consiguió vencer al ajedrez al mejor jugador del momento, el ruso Gary Kaspárov. Esta "simple" partida de ajedrez, en 1997, a las puertas de la era digital, hizo ver a las empresas tecnológicas que la Inteligencia Artificial podría analizar y procesar la enorme cantidad de datos que se avecinaba en un tiempo razonable.

Hoy por hoy, aunque las TIC se siguen desarrollando (la fibra óptica, el 5G...), la Inteligencia Artificial ya puede desarrollarse exponencialmente como disciplina. Cuenta con la infrastructura necesaria para ello; la nube almacena lo que necesitemos, los ordenadores son cada vez más eficientes, económicamente todo es más accesible, la cantidad de datos que poder estudiar es incalculable (en un segundo se han realizado 2.300 llamadas por Skype, 60.000 búsquedas de Google, se han enviado dos millones y medio de emails), la velocidad de

Internet, el poco espacio físico que un ordenador ocupa...

La Inteligencia Artificial cuenta con la base por lo que es el momento de desarrollar sus capacidades. Las aplicaciones son inmensas; tanto en el mundo del ocio y comercial, como en temas de salud, deporte, publicidad.

Los datos son el motor de esta revolución, pero son dados en bruto, hay que *pulirlos* para poder usarlos y son las Matemáticas las que se encargan de esto. Dos desafíos bastante comunes (que no únicos) dentro de de la Inteligencia Artificial son la enorme dimensión de los datos y la presencia de ruido en ellos. Las grandes dimensiones suponen pérdida de eficacia y aumento del tiempo a la hora de hacer operaciones. Es verdad que eficacia ha aumentado, pero es que las exigencias de la sociedad también se han acelerado: un banquero quiere estar segure, en menos de un segundo, de que la huella dactilar de un nuevo cliente no pertenece ya a otro cliente, lo que supondría un fruade. Se quiere comprar por Internet al instante, a la máxima calidad y al mejor precio comparado. Se exige al navegador que al meter el destino en menos de 10 segundos nos proponga la ruta óptima. Por tanto la Inteligencia Artificial no se puede permitir perder el tiempo en operaciones innecesarias. Además, la capacidad de almacenamiento, aunque elevada, es limitada por lo que no interesa tener datos que no son necesarios. Afortunadamente, lo más frecuente es que los datos observados sigan una serie de patrones que permitan caracterizarlos reduciendo así su dimensión y es a través de las técnicas matemáticas como los encontraremos. El ruido supone otro gran desafío. La presencia de datos corruptos o errores pueden hacer que nuestros algoritmos de Inteligencia Artificial no se ajusten adecuadamente a la realidad. Errores en los datos derivan en errores en las predicciones. Por tanto conviene localizar este ruido, apartar los errores o no tenerlos en cuenta. Diversos matemáticos han dedicado estudios para combatir esta barrera. De hecho, en este año 2020 los Premios Princesa de Asturias de Investigación Científica y Técnica han sido para cuatro matemáticos; uno de ellos, el francés Emmanuel Candès por su estudio de cómo reconstruir señales a partir de mediciones parciales y ruidosas [3] que dio solución a uno de los problemas centrales en las imágenes médicas y en general en todas las áreas del procesamiento de señales.

Tanto la barrera de las grandes dimensiones como la del ruido están presentes en el Problema de Completado y de Separación de matrices. Estos son dos de los grandes objetivos dentro de la Inteligencia Artificial y que han tenido numerosas aplicaciones prácticas en nuestro día a día. El Problema de completado hace referencia al caso en el que se tiene una matriz incompleta y se busca rellenar las posiciones no observadas. Es la base de los sistemas de recomendación: Amazon, Kindle, Booking... buscan, con las puntuaciones de sus clientes sobre ciertos productos, predecir en qué otros podrían estar interesados y aumentar así sus ventas. Este problema dio lugar al Desafío Netflix en el que se otorgaba un premio de un millón de euros al que consiguiese reducir en un 10 % los errores de predicción. El otro objetivo conocido como el Problema de Separación busca descomponer la matriz como suma de unos elementos estructurales y algunas fuentes de ruido concentrado en pocas posiciones. Este ultimo objetivo tiene aplicaciones importantes en la deteccion de movimiento en la videovigilancia, y es la base del reconocimiento facial de los moviles de hoy, entre otras

aplicaciones.

Los dos objetivos se pueden abordar como problemas de aproximación penalizada o regularizada. Los datos disponibles son representados en forma de matriz y con la correcta definición del problema y sus restricciones podemos tanto completar la matriz como separarla en distintos elementos estructurales. En este trabajo no solo estudiaremos los fundamentos matemáticos detrás de estos métodos sino que tambien veremos como los dos grandes desafíos de elevada dimensionalidad y presencia de ruido pueden ser solucionados. Trasladaremos también estos conocimientos a la práctica, creando un sistema en R de recomendación de películas. Antes de abordar estos dos objetivos de la Inteligencia Artificial necesitaremos exponer ciertos preliminares matemáticos. En esta memoria se habla de 'aproximación' regularizada y no de 'estimación' regularizada. Esto es porque los problemas centrales en este trabajo tienen que ver con la descripción simplificada de los datos sin que se plantee realmente un modelo subyacente. Sin embargo, las técnicas que se emplean en estos problemas están fuertemente inspirados por métodos de regularización surgidos en el contexto del problema de regresión, para una estimación apropiada de la función de regresión en presencia de multicolinealidad, lo que necesariamente ocurre en el caso de 'alta dimensión', es decir, si el número de casos es menor que el número de variables explicativas. Esto nos lleva a estudiar brevemente dos técnicas clásicas de regularización estadística: el estimador ridge y el lasso. En el caso de la regresión ridge comprobaremos el efecto del nivel de regularización, que se controla a través de un parametro. Valores altos del parámetros están asociados a una reducción de la varianza, a costa de aumentar el sesgo. Se entiende de esta forma que una elección apropiada del parámetro de regularización puede conducir a un compromiso favorable entre sesgo y varianza y, por lo tanto, a un menor error de predicción. Empleamos este análisis como motivación para introducir un método de regularización más reciente: el lasso. El uso de penalizaciones l_1 tiene la propiedad de conducir a estimaciones 'dispersas', en las que el parámetro estimado puede tener una gran cantidad de coeficientes nulos, lo que será de gran utilidad en las aplicaciones que se estudian posteriormente.

En las técnicas de aprendizaje supervisado en machine learning hay dos aspectos principales de interés. Por un lado está la obtención de garantías estadísticas sobre su funcionamiento, principalmente cotas para el error de generalización promedio o cotas probabilísticas para el error real. El otro aspecto importante es el diseño de procedimientos eficientes para el cálculo efectivo de las reglas. Si este aspecto no está garantizado los resultados estadísticos pueden garantizar un buen rendimiento de un método, pero difícilmente podrá ser implementado en un problema realista. En esta memoria nos hemos centrado en este segundo aspecto y sólo en un par de ocasiones se trata el asunto de las garantías estadísticas (la discusión anteriormente citada sobre la regresión ridge y una pequeña discusión sobre la probabilidad de correcta reconstrucción de una matriz con entradas corruptas). Cabe destacar que aunque hay ejemplos que a la vez presentan buenas garantías estadísticas y disponen de algoritmos adecuados de cálculo (el ejemplo principal es el lasso), hay menos resultados en este sentido en los problemas de aproximación, descomposición y completado de

matrices, que son el tema central de esta memoria.

La característica común de todas los algoritmos de cálculo de metodos de aprendizaje que se incluyen en esta memoria es que se basan en la minimización de un funcional criterio convexo sobre un subconjunto convexo del espacio Euclídeo. Por esta razón se dedica un capítulo de la memoria a exponer algunos resultados generales de convexidad y de optimización convexa. Después se presentan algunos grupos de algoritmos de optimización. El elemento principal en las técnicas más habituales es el algoritmo de descenso de gradiente, con variantes. Este método de primer orden puede parecer poco eficiente frente a métodos tipo Newton. Esto es en parte verdad, pero las necesidades de memoria asociadas al almacenamiento de la matriz Hessiana en problemas grandes hacen que el descenso de gradiente sea el preferido en muchos casos. Además se puede adaptar fácilmente a muchos problemas no diferenciables, lo que es esencial en las aplicaciones que se tratan aquí.

En la parte final del trabajo se presentan los dos problemas principales. El completado de matrices con posiciones no observadas se puede tratar con distintos métodos relacionados con la descomposición de valor singular. La regularización por norma nuclear que tratamos aquí conduce a un problema de optimización convexa, para el que se proporcionan garantías de convergencia. Finalmente se trata el problema de descomposición aproximada de una matriz como suma de una parte de rango bajo y otra dispersa. Esto sirve en situaciones en las que se pretende separar una parte permanente de otra variable pero concentrada en pocas posiciones de la matriz. En este caso emplearemos una combinación de penalizaciones para conseguir este objetivo y mostraremos como resolver el problema resultante mediante un algoritmo de proyecciones alternadas.

2. Técnicas de reducción de la dimensión

El mundo de la Inteligencia Artificial necesita de una gran cantidad de datos; para poder predecir, calcular, clasificar... son imprescindibles. De hecho, hasta para comprobar que los algoritmos que hemos creado están bien, necesitamos de más datos todavía. Toda esta información se traduce en matrices de grandes dimensiones, tanto por el número de individuos de la muestra como por todas las características que se miden de cada individuo. Pero, en este mundo del Machine Learning, más no siempre significa mejor. Reducir la dimensionalidad significa aumentar el rendimiento computacional, lo que supone un ahorro en coste y en tiempo. Además, indentificar y eliminar las variables irrelevantes, facilita a la gente la comprensión del modelo y sus resultados.

Reducir la dimensión entonces está bien, pero hay que hacerlo con el menor coste de información posible. Por tanto, dado un conjunto de datos, estamos buscando la combinación de las variables originales que mejor expliquen nuestro conjunto de datos. Para elegir qué variables van a formar parte de nuestro subconjunto tenemos dos criterios que parecen razonables: la correlación y la redundancia. Nos quedaremos con las variables que mayor correlación o asocia-

ción tengan con la variable a predecir y si en nuestro conjunto hubiera variables redundantes que tuvieran una correlación entre ellas suficientemente fuerte, en nuestro subconjunto solo incluiríamos una de ellas.

Entonces estamos ante una matriz de datos $X \in \mathcal{M}_{m\times n}(\mathbb{R})$. Interpretaremos que la fila i-ésima de X recoge la información del caso i y que la columna j corresponde al atributo o *variable* j medida en cada uno de los casos y usaremos la notación

$$X = [x_{ij}]_{1\leq i\leq m, 1\leq j\leq n} = [\mathbf{x}^{(1)} \cdots \mathbf{x}^{(n)}] = \begin{bmatrix} \mathbf{x}_1^T \\ \cdots \\ \mathbf{x}_m^T \end{bmatrix}.$$

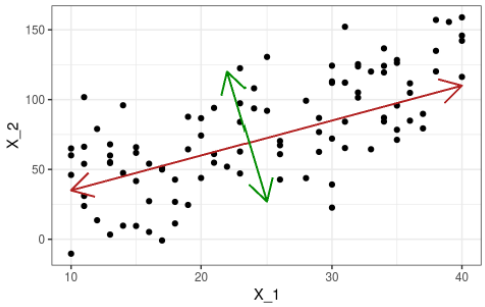
Nuestro objetivo es entonces encontrar un subespacio $r < n$ con el menor coste de información posible. Una manera de elegir las variables más útiles siguiendo estos dos criterios estaría guiado por el conocimiento del campo de donde han salido los datos. Por ejemplo, un veterinario, para determinar de qué raza es un perro, medir su altura o la longitud de sus patas le pueden pueden parecer variables redundantes, dado que un perro alto tendrá patas altas. Sin embargo, a lo mejor la diferencia entre ellas, es decir, la altura del tronco, es una variable clave para determinar de qué raza es un perro. Por tanto no nos podemos fiar siempre de la intuición, a lo mejor hay correlaciones en las variables que desconocemos y nos llevamos alguna sorpresa. Es mejor ser previsores, no dar nada por supuesto y asegurarnos a través de métodos matemáticos de las variables que entran en nuestro subconjunto óptimo. Por ello vamos a introducir ahora dos técnicas de reducción de la dimensión: el análisis de los componentes principales y la descomposición en valores singulares. Se trata de dos técnicas antiguas, ambas nacieron sobre el siglo XIX; el británico K. Pearson [4] fue quien estableció las bases de las componentes principales y el italiano E. Beltrami, 1873 [5], quien primero contribuyó en el estudio de la descomposición en valores singulares. En [6] se puede encontrar una descripción más detallada de los trabajos pioneros en este campo. A pesar de su antigüedad, siguen siendo de las más utilizadas, especialmente a partir de la aparición de los ordenadores que facilitaban los cálculos y ahorraban en tiempo. Ambas se adaptan perfectamente a los datos con los que estamos trabajando, ya sean datos para determinar razas de perros o predecir qué película gustará más. PCA y SVD proporcionan una manera sistemática para reducir la dimensión y además sirven de base para desarrollar nuevas técnicas de reducción adaptadas a problemas de aprendizaje automático.

2.1. Análisis de componentes principales (PCA)

Este procedimiento matemático, en inglés conocido como Principal Component Analysis (PCA), es una técnica que permite simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conserva su información. La base de esta técnica es transformar un conjunto de variables correlacionadas en un conjunto menor de variables no correlacionadas, denominadas componentes principales. Estas variables nuevas serán elegidas de manera que recojan la mayor variabilidad de los datos posible.

Entonces, dada nuestra matriz de datos X, PCA permite encontrar un número de factores subyacentes ($r < n$), obtenidos como combinación lineal de las n variables originales y que explican aproximadamente lo mismo que éstas. Donde antes se necesitaban n valores para caracterizar a cada individuo, ahora bastan r valores.

Para tener una idea intuitiva del proceso de PCA veamos, a través de un ejemplo, las componentes principales desde un punto de vista geométrico. Imaginemos que tenemos un conjunto de observaciones que representamos a través de la matriz X. Esta matriz tiene dos columnas formadas por las variables (X_1, X_2) que constituyen el espacio muestral, es decir, en nuestro ejemplo $n = 2$. m será igual al número de individuos al que se le haya tomado la muestra. El vector que define la primera componente principal (Z_1) sigue la dirección en la que las observaciones varían más (línea roja de la figura 2). La proyección de cada observación sobre esa dirección equivale al valor de la primera componente para dicha observación (*principal component scores,* z_{i1}).



La segunda componente (Z_2) sigue la segunda dirección en la que los datos muestran mayor varianza y que no está correlacionada con la primera componente. En la figura (2) viene representado por la línea verde. La condición de no correlación entre componentes principales equivale a decir que sus direcciones son ortogonales.

Figura 2: Componentes Principales

Cada componente principal (Z_i) se obtiene por combinación lineal de las variables originales

$$Z_i = \phi_{1i}X_1 + \phi_{2i}X_2 + ... + \phi_{ni}X_n$$

Los términos $\phi_{1i}, \ldots, \phi_{ni}$ reciben en el nombre de *loadings* o pesos y son los que definen a la componente. ϕ_{11} es el peso de la variable X_1 de la primera componente principal. Los pesos pueden interpretarse como la importancia que tiene cada variable en cada componente y, por lo tanto, ayudan a conocer que tipo de información recoge cada una de las componentes. De esta manera, PCA construye una transformación lineal que da lugar a un nuevo sistema de coordenadas para el conjunto original de datos.

Antes de empezar a calcular esta transformación, es conveniente asumir que las variables de la matriz X están centradas. Denotando $\mathbf{1} = [1^{\overset{m)}{\cdots}}1]^T$ esto significa que $\mathbf{1} \cdot \mathbf{x}^{(j)} = 0$ para cada j o, equivalentemente, que

$$\mathbf{1}^T X = 0. \tag{1}$$

Necesitamos también varianza igual a uno. Esto se debe a que el proceso de PCA identifica aquellas direcciones en las que la varianza es mayor. Como la varianza de una variable se mide en su misma escala elevada al cuadrado, si antes de calcular las componentes no se estandarizan, aquellas variables cuya escala sea mayor dominarán sobre el resto. Por tanto es conveniente en principio estandarizar los datos.

Buscamos comprimir la información en X buscando un subespacio r dimensional de $\mathbb{R}^n$, $r < n$, sobre el que proyectaremos los puntos $\mathbf{x}_i$ (las filas de X). Asumiremos, sin pérdida de generalidad, que el subespacio está generado por la base ortonormal $\mathbf{b}_1, \ldots, \mathbf{b_k}$ (con $\mathbf{b}_j \in \mathbb{R}^p$, $j = 1, \ldots, r$). Denotaremos por B_r la matriz con columnas $\mathbf{b}_j \in \mathbb{R}^n$, de forma que $B_r = [\mathbf{b}_1, \ldots, \mathbf{b_r}] \in \mathcal{M}_{n\times r}(\mathbb{R})$ satisface $B_r^T B_r = I_r$. La matriz de proyección sobre el subespacio es, por lo tanto $B_r(B_r^T B_r)^{-1} B_r^T = B_r B_r^T$. La proyección de $\mathbf{x}_i$ es $B_r B_r^T \mathbf{x}_i = \sum_{j=1}^r \langle \mathbf{x}_i, \mathbf{v}_j \rangle \mathbf{v}_j = \mathrm{Pr}_{B_r}(\mathbf{x}_i)$. El *error de reconstrucción* es la diferencia entre $\mathbf{x}_i$ y su versión proyectada, $\mathrm{Pr}_{B_r}(\mathbf{x}_i)$. Podemos medir la magnitud del error global de reconstrucción mediante el error cuadrático medio,

$$\mathrm{ECM}(X; B_r) := \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_i - \mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2. \tag{2}$$

Observamos que, por la ortogonalidad entre $\mathrm{Pr}_{B_r}(\mathbf{x}_i)$ y $\mathbf{x}_i - \mathrm{Pr}_{B_r}(\mathbf{x}_i)$ se tiene que

$$\|\mathbf{x}_i\|^2 = \|\mathbf{x}_i - \mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2 + \|\mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2.$$

Sumando en i concluimos

$$\frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_i\|^2 = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_i - \mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2 + \frac{1}{m} \sum_{i=1}^{m} \|\mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2. \tag{3}$$

Para una mejor interpretación de la igualdad (3) nos fijamos en la varianza asociada a la nube de puntos original, es decir,

$$\mathrm{Var}(X) := \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2,$$

donde $\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i = \frac{1}{m} \mathbf{1}^T X$. Como asumimos que las columnas de X están centradas tendremos $\bar{\mathbf{x}} = 0$ y $\mathrm{Var}(X) = \frac{1}{m} \sum_{i=1}^m \|\mathbf{x}_i\|^2$. La matriz de datos proyectados es

$$X_{B_r} := X B_r B_r^T.$$

Las columnas de X_{B_r} son obviamente centradas, por lo que $\mathrm{Var}(X_{B_r}) = \frac{1}{m} \sum_{i=1}^m \|\mathrm{Pr}_{B_r}(\mathbf{x}_i)\|^2$. De esta manera podemos reescribir (3) en la forma

$$\mathrm{Var}(X) = \mathrm{ECM}(X; B_r) + \mathrm{Var}(X_{B_r}). \tag{4}$$

La descomposición (4) nos da un mensaje fundamental: el subespacio que captura más variabilidad de los datos originales (el que maximiza $\mathrm{Var}(X_{B_r})$) es el

que da menor error de reconstrucción (el que minimiza $\mathrm{ECM}(X; B_r)$). De entre todos los subespacios r-dimensionales el subespacio $\hat{B}_r$ que resuelve los dos problemas equivalentes anteriores es el que mejor resume la información contenida en la matriz X.

Para obtener una descripción adecuada de $\hat{B}_r$ nos fijamos en el problema de maximizar $\mathrm{Var}(X_{B_r})$. Observamos que, para una matriz X centrada general,

$$\mathrm{Var}(X) = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{x}_i\|^2 = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Tr}(\mathbf{x}_i^T \mathbf{x}_i) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Tr}(\mathbf{x}_i \mathbf{x}_i^T) = \frac{1}{m}\mathrm{Tr}(X^T X).$$

En el caso de la matriz $\mathrm{Var}(X_{B_r})$ obtenemos

$$\begin{aligned} \mathrm{Var}(X_{B_r}) &= \frac{1}{m}\mathrm{Tr}(X_{B_r}^T X_{B_r}) = \frac{1}{m}\mathrm{Tr}(B_r B_r^T X^T X B_r B_r^T) \\ &= \frac{1}{m}\mathrm{Tr}(B_r^T X^T X B_r). \end{aligned} \tag{5}$$

Para analizar mejor el lado derecho de (5) observamos que la matriz $X^T X$ es simétrica y semidefinida positiva y, por lo tanto, la podemos expresar en la forma $X^T X = V \Lambda V^T$ con $V = [\mathbf{v}_1 \cdots \mathbf{v}_n] \in \mathcal{M}_{n\times n}(\mathbb{R})$ ortogonal, es decir, $V^T V = V V^T = I_n$ y $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ con $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ (equivalentemente, $\mathbf{v}_1, \ldots, \mathbf{v}_n$ es una base ortonormal de $\mathbb{R}^n$ formada por autovectores de $X^T X$: $X^T X \mathbf{v}_j = \lambda_j \mathbf{v}_j$; los valores λ_j son los autovalores de $X^T X$). De la igualdad $B_r B_r^T = \sum_{j=1}^{r} \mathbf{b}_j \mathbf{b}_j^T$ se deduce que

$$\mathrm{Tr}(B_r^T X^T X B_r) = \mathrm{Tr}(X^T X B_r B_r^T) = \sum_{j=1}^{r} \mathrm{Tr}(X^T X \mathbf{b}_j \mathbf{b}_j^T) = \sum_{j=1}^{k} \mathbf{b}_j^T X^T X \mathbf{b}_j. \tag{6}$$

Para cada vector $\mathbf{b}_j$ se tiene $\mathbf{b}_j^T X^T X \mathbf{b}_j = \sum_{l=1}^{n} \lambda_l \langle \mathbf{v}_l, \mathbf{b}_j \rangle^2$. Además, la ortonormalidad de las columnas de B_r se traduce en que $\sum_{l=1}^{n} \langle \mathbf{v}_l, \mathbf{b}_j \rangle^2 = 1$ y $\sum_{l=1}^{n} \langle \mathbf{v}_l, \mathbf{b}_i \rangle \langle \mathbf{v}_l, \mathbf{b}_j \rangle = 0$ si $i \neq j$. A la vista de (6) vemos que

$$\mathrm{Tr}(B_r^T X^T X B_r) = \sum_{l=1}^{n} \lambda_l \Big(\sum_{j=1}^{r} \langle \mathbf{v}_l, \mathbf{b}_j \rangle^2 \Big) = \sum_{l=1}^{n} \lambda_l \|\mathrm{Pr}_{B_r}(\mathbf{v}_l)\|^2. \tag{7}$$

Como $\|\mathbf{v}_l\|^2 = 1$, necesariamente, $x_l := \|\mathrm{Pr}_{B_r}(\mathbf{v}_l)\|^2 \leq 1$. Por otra parte

$$\sum_{l=1}^{n} x_l = \sum_{l=1}^{n} \Big(\sum_{j=1}^{r} \langle \mathbf{v}_l, \mathbf{b}_j \rangle^2 \Big) = \sum_{j=1}^{r} \Big(\sum_{l=1}^{n} \langle \mathbf{v}_l, \mathbf{b}_j \rangle^2 \Big) = \sum_{j=1}^{r} \|\mathbf{b}_j\|^2 = r,$$

donde hemos usado que $\mathbf{v}_1, \ldots, \mathbf{v}_n$ son una base ortonormal de $\mathbb{R}^r$. Por lo tanto, la varianza recogida por el subespacio B_r no puede ser mayor que

$$\frac{1}{m} \,\mathrm{máx}\Big\{ \sum_{l=1}^{n} \lambda_l x_l : \, 0 \leq x_l \leq 1, \, \sum_{l=1}^{n} x_l = r \Big\} = \frac{1}{m} \sum_{l=1}^{r} \lambda_l.$$

Esa máxima varianza se recoge si el espacio generado por las columnas de B_r es el generado por $\mathbf{v}_1, \ldots, \mathbf{v}_r$. Recogemos los resultados obtenidos en el siguiente Teorema.

Teorema 2.1 *Si $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ es una matriz centrada por columnas y $X^T X = V\Lambda V^T$ con con $V = [\mathbf{v}_1 \cdots \mathbf{v}_n] \in \mathcal{M}_{n\times n}(\mathbb{R})$ ortogonal, $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ y $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$, entonces el subespacio generado por las columnas de $\hat{V}_r := [\mathbf{v}_1 \cdots \mathbf{v}_r]$ es, de entre todos los subespacios de $\mathbb{R}^n$ de dimensión r el que minimiza el error de reconstrucción (equivalentemente, el que maximiza la varianza proyectada).*

A las direcciones $\mathbf{v}_1, \ldots, \mathbf{v}_n$ se las conoce como *componentes principales* de la matriz X (o de la nube de puntos $\mathbf{x}_1 \cdots \mathbf{x}_m$). Una consecuencia interesante del Teorema 2.1 es que las proyecciones óptimas sobre subespacios r dimensionales se pueden calcular de forma secuencial. La mejor proyección en un espacio unidimensional se obtiene proyectando sobre $\mathbf{v}_1$. Para obtener la mejor proyección sobre un subespacio bidimensional tenemos que añadir las proyecciones sobre $\mathbf{v}_2$ y así sucesivamente.

Otro aspecto interesante en el análisis de componentes principales es la forma tan simple en la que se puede evaluar el error relativo de reconstrucción o, de forma equivalente, la proporción de de varianza que retiene la proyección óptima. En la igualdad (4) podemos tomar $B_r = \hat{V}_r$ y obtener así

$$\mathrm{Var}(X) = \mathrm{ECM}(X; \hat{V}_r) + \mathrm{Var}(X_{\hat{V}_r}).$$

Dividiendo ambos lados por $\mathrm{Var}(X)$ obtenemos

$$1 = \frac{\mathrm{ECM}(X; \hat{V}_r)}{\mathrm{Var}(X)} + \frac{\mathrm{Var}(X_{\hat{V}_r})}{\mathrm{Var}(X)}.$$

El cociente

$$\frac{\mathrm{Var}(X_{\hat{V}_r})}{\mathrm{Var}(X)} \in [0, 1]$$

es la proporción de varianza que puede conservarse proyectando la nube de puntos sobre un subespacio de dimensión r. Por los cálculos anteriores

$$\frac{\mathrm{Var}(X_{\hat{V}_r})}{\mathrm{Var}(X)} = \frac{\mathrm{Tr}(X_{\hat{V}_r}^T X_{\hat{V}_r})}{\mathrm{Tr}(X^T X)}.$$

Obviamente $\mathrm{Tr}(X^T X) = \sum_{l=1}^{n} \lambda_l$ y con el análisis anterior hemos comprobado que $\mathrm{Tr}(X_{\hat{V}_r}^T X_{\hat{V}_r}) = \sum_{l=1}^{r} \lambda_l$. Por lo tanto la proporción de varianza que puede conservarse proyectando la nube de puntos sobre un subespacio de dimensión r es

$$\frac{\sum_{l=1}^{r} \lambda_l}{\sum_{l=1}^{n} \lambda_l}$$

y se obtiene proyectando sobre las primeras r componentes principales. A la hora de comprimir la información presente en la matriz X podemos tener en cuenta

este cociente y también la proporción de ahorro de almacenamiento. La matriz X está formada por $m \times n$ entradas (supongamos que números reales de precisión doble en el ordenador). Los datos proyectados (sobre el espacio r-dimensional óptimo) son $\mathbf{x}_{i;r} := \sum_{j=1}^{r} \langle \mathbf{x}_i, \mathbf{v}_j \rangle \mathbf{v}_j$. Para poder calcular estos puntos proyectados o reconstruidos se necesita por lo tanto disponer de la matriz $\hat{V}_r \in \mathcal{M}_{m \times n}(\mathbb{R})$ y de la matriz de coeficientes $\langle \mathbf{x}_i, \mathbf{v}_j \rangle$. Esa matriz es $X\hat{V}_r \in \mathcal{M}_{m \times r}(\mathbb{R})$. Por lo tanto la versión comprimida de X se puede almacenar guardando $r \times (m+n)$ entradas, frente a las $m \times n$ de la versión completa. Así que en función de la exactitud que buscamos y de la complejidad que queremos en nuestro modelo, elegiremos más o menos componentes.

2.2. Descomposición en valores singulares

Otra técnica de reducción de la dimensión, muy relacionada con el PCA es la *descomposición en valores singulares* (SVD). Esta es una factorización

$$X = UDV^T \tag{8}$$

donde $X \in \mathcal{M}_{m \times n}(\mathbb{R})$, $U \in \mathcal{M}_{m \times n}(\mathbb{R})$ tiene columnas ortonormales (es decir, $U^T U = I_n$), $D \in \mathcal{M}_{n \times n}(\mathbb{R}) = \text{diag}(d_1, \ldots, d_n)$ con $d_1 \geq \cdots \geq d_n \geq 0$ y $V \in \mathcal{M}_{n \times n}(\mathbb{R})$ es ortogonal ($V^T V = V V^T = I_n$).

Para entender lo que estamos consiguiendo con esta descomposición veamos varios ejemplos. Empezamos con la matriz X de tamaño 7×5 que representa las puntuaciones que han dado siete usuarios a cinco películas. La puntución varía entre 1 y 5, siendo uno la puntuación más baja y cinco la más alta. Las películas son *"Matrix",“ Alien", "Serenity", "Casablanca" y "Amelie"*. No han sido elegidas al azar sino que hemos buscado tres películas de ciencia ficción y dos románticas para entender mejor las nuevas bases que crea la descomposición en valores singulares.

$$X = \left[\begin{array}{ccccc} \textit{Matrix} & \textit{Alien} & \textit{Serenity} & \textit{Casablanca} & \textit{Amelie} \\ \hline \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{array}\right]$$

Vemos por ejemplo que al usuario tres le han gustado las tres primeras películas pero que no ha visto las dos últimas. En cambio al primero no le hacen gracia las películas de ciencia-ficción. Si observamos la matriz X podemos separar a ojo dos tipos de películas: las de ciencia ficción y las románticas. Del mismo modo podemos separar a los usuarios a los que les gusta las películas románticas o a los que les gusta las de ciencia ficción. Pues de esto se encarga SVD, tanto en esta pequeña matriz como en matrices de grandes dimensiones; es capaz de encontrar los géneros (vectores) que mejor explican la variabilidad de nuestro

conjunto de datos. Implementamos en R la descomposición en valores singulares y obtenemos las siguientes matrices.

$$
\underset{U}{\begin{bmatrix} \mathbf{0{,}13} & 0{,}02 & -0{,}01 \\ \mathbf{0{,}41} & 0{,}07 & -0{,}03 \\ \mathbf{0{,}55} & 0{,}09 & -0{,}04 \\ \mathbf{0{,}68} & 0{,}11 & -0{,}05 \\ 0{,}15 & \mathbf{-0{,}59} & 0{,}65 \\ 0{,}07 & \mathbf{-0{,}73} & -0{,}67 \\ 0{,}07 & \mathbf{-0{,}29} & 0{,}32 \end{bmatrix}} \times \underset{D}{\begin{bmatrix} \mathbf{12{,}4} & 0 & 0 \\ 0 & 9{,}5 & 0 \\ 0 & 0 & 1{,}3 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} \mathbf{0{,}56} & 0{,}12 & 0{,}40 \\ \mathbf{0{,}59} & -0{,}02 & -0{,}80 \\ \mathbf{0{,}56} & 0{,}12 & 0{,}40 \\ 0{,}09 & -0{,}69 & 0{,}09 \\ 0{,}09 & -0{,}69 & 0{,}09 \end{bmatrix}}
$$

La matriz U se puede entender como la matriz que relaciona usuarios con géneros. Cada columna representa un género; la primera el de ciencia ficción, la segunda refleja el romántico. Al primer usuario, al que no le gustaban las películas de ciencia-ficción, tiene un valor próximo a cero (0,13), en la dirección de este género mientras que los tres siguientes tienen valores altos. Los tres últimos usuarios presentan, como era de esperar, valores absolutos más elevados en la segunda columna. Lo que ha ocurrido es que la SVD ha creado una nueva base para los usuarios en los que los autovectores son los géneros (ciencia-ficción, romance...) y las columnas de U que nos indican dado el usuario representado en la fila i cuánto se corresponde su gusto con el estilo de la columna j.

Veamos lo que representa la matriz diagonal. Cada una de las entradas de la diagonal podemos entenderlas como el peso que tiene cada uno de los géneros para explicar la muestra; por ejemplo, (12,4) representa el peso del género ciencia-ficción. Las entradas de la diagonal están ordenadas de mayor a menor. De esta manera podemos ver cuáles son los estilos que mejor clasifican a nuestro usuarios y películas.

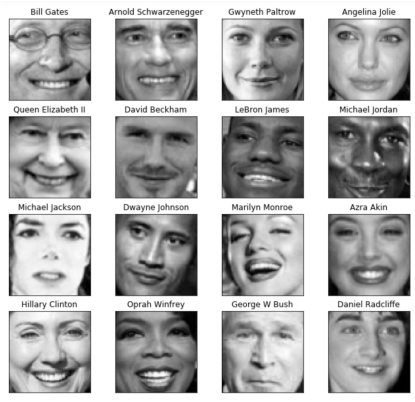


Figura 3: Autovectores de caras

La matriz V muestra relación entre cada película con cada uno de los géneros. Cada columna representa un género y cada fila una película; por tanto cada entrada nos indica cúanto de un género presenta una película.

Otro ejemplo recurrente es el de la clasificación de caras; aquí la matriz X tiene como columnas las imágenes vectorizadas de caras de diferentes personas. La matriz U de la descomposición tiene como columnas los "autovectores de caras"mientras que la matriz V^T tiene como columnas cuánto de cada autovector tiene cada cara x_j.

La figura (4) muestra tanto las cuatro primeras componentes principales como los coeficientes de dicha cara en esta nueva base. Vemos que la primera componente coge los rasgos principales de la cara, con esta componente tendríamos

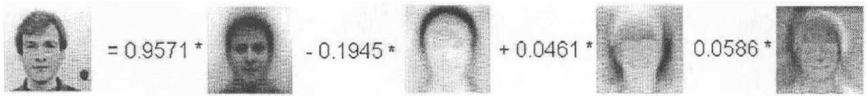


Figura 4: Componentes principales de caras

la base. Las tres siguientes son más los detalles. De hecho si nos fijamos, las tres primeras componentes se diría que son hombres mientras que en la tercera parecen más los rasgos de una mujer.

Otro ejemplo muy presente en física o en ingeniería es cómo evoluciona la energía de un sistema a lo largo del tiempo. Podemos expresarlo en forma de matriz colocando en cada columna cómo está un sistema en un instante determinado; de esta forma si tenemos m columnas quiere decir que sabemos cómo se encuentra el sistema en m momentos distintos. De esta manera la matriz representa cómo evoluciona un sistema a lo largo del tiempo. Aquí las columnas del vector U son los vectores que mejor explican el movimiento de energía del sistema mientras que las columnas de V^T nos indican en cada paso de tiempo cúanto se mueve nuetro sistema en cada una de los autovectores de U.

Vemos aquí una ventaja de la descomposición en valores singulares y es que se adapta perfectamente a la estructura de los datos; en ejemplos muy diversos podemos ver la idea intuitiva que se esconde detrás de esta descomposición. Vamos a probar a continuación que cualquier matriz se puede factorizar de esta manera y que, además, la factorización se obtiene a partir de las componentes principales de X.

Teorema 2.2 *Si $m \geq n$, toda matriz $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ admite una factorización en la forma* (8)*. Si X admite la factorización* (8) *entonces $X^TX = VD^2V^T$. Recíprocamente, $X^TX = V\Lambda V^T$ con $V \in \mathcal{M}_{n\times n}(\mathbb{R})$ ortogonal, $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ y $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ entonces, si $\mathbf{u}_j = \frac{1}{\sqrt{\lambda_j}} X\mathbf{v}_j$ para cada $\lambda_j > 0$ entonces los $\mathbf{u}_j$ correspondientes son ortonormales y si $U = [\mathbf{u}_1 \cdots \mathbf{u}_n]$ se obtiene completando, si fuera necesario, con columnas ortonormales, entonces*

$$X = UDV^T$$

con $D = \operatorname{diag}(d_1, \ldots, d_n)$ y $d_j = \sqrt{\lambda_j}$.

Demostración. Obviamente, si $X = UDV^T$ con U, D y V como en (8) entonces $X^TX = VDU^TUDV^T = VD^2V^T$. Supongamos ahora que $X^TX = V\Lambda V^T$ con V y Λ como en el enunciado. Tendremos que $\langle X\mathbf{v}_i, X\mathbf{v}_i\rangle = \mathbf{v}_i^T X^T X\mathbf{v}_i = \lambda_i$ y $\langle X\mathbf{v}_i, X\mathbf{v}_j\rangle = \mathbf{v}_i^T X^T X\mathbf{v}_j = \lambda_j\langle \mathbf{v}_i, \mathbf{v}_j\rangle = 0$ si $i \neq j$. Por lo tanto los vectores $X\mathbf{v}_i$ son ortogonales entre sí. Si $X = 0$ el resultado es trivial. Si no habrá un $n_0 \in \{1, \ldots, n\}$ de forma que $\lambda_j > 0$ si $j \leq n_0$ y $\lambda_j = 0$ si $j > n_0$. Sea $\mathbf{u}_j = \frac{1}{\sqrt{\lambda_j}} X\mathbf{v}_j$ si $j \leq n_0$. Los vectores $\mathbf{u}_1, \ldots, \mathbf{u}_{n_0}$ son ortonormales. Sea

$U = [\mathbf{u}_1 \cdots \mathbf{u}_{n_0} \cdots \mathbf{u}_n]$ con columnas ortonormales y $D = \Lambda^{1/2}$. Entonces

$$UD = [\sqrt{\lambda_1}\mathbf{u}_1 \cdots \sqrt{\lambda_{n_0}}\mathbf{u}_{n_0} \cdots \sqrt{\lambda_n}\mathbf{u}_n] = [X\mathbf{v}_1 \cdots X\mathbf{v}_{n_0} \cdots \mathbf{0}].$$

Como $X^T X\mathbf{v}_j = \mathbf{0}$ si $j > n_0$, tenemos que $\|X\mathbf{v}_j\|^2 = \mathbf{v}_j^T X^T X\mathbf{v}_j = 0$ y, por lo tanto, $X\mathbf{v}_j = \mathbf{0}$ si $j > n_0$. Concluimos que

$$UD = [X\mathbf{v}_1 \cdots X\mathbf{v}_n] = XV.$$

Como $VV^T = I_n$, de aquí se deduce que

$$X = UDV^T.$$

□

A los valores de la matriz D se les llama *valores singulares* de X. El Teorema 2.2 demuestra que los valores singulares son las raíces cuadradas positivas de los autovalores de $X^T X$. Encontramos así una relación con la norma espectral:

$$\sigma(X) = \max_{\mathbf{x}:\, \|\mathbf{x}\|=1} |X\mathbf{x}| = \sqrt{\max_{\mathbf{x}:\, \|\mathbf{x}\|=1} \mathbf{x}^T X^T X\mathbf{x}} = \sqrt{\lambda_1} = d_1.$$

Es fácil comprobar que σ define una norma en el espacio de matrices reales con m filas y n columnas. Vemos que la norma espectral es igual al mayor valor singular de una matriz.

En el Teorema 2.2 la condición $m \geq n$ es necesaria para que pueda existir una matriz U con n columnas ortonormales. Ésta es una limitación importante, porque muy frecuentemente nos encontraremos en la situación $m \ll n$. Además la factorización incluye posiblemente valores innecesarios: las columnas de U correspondientes a valores singulares nulos se pueden elegir de forma arbitraria. Se puede modificar la representación anterior para eliminar esa información redundante. Supongamos que $\text{rank}(X) = k$. Obviamente $r \leq \text{mín}(m, n)$. Además k es igual al número de autovalores $\lambda_j > 0$ de $X^T X$. La factorización (8) se puede reescribir en la forma

$$X = \sum_{j=1}^{n} \sqrt{\lambda_j}\mathbf{u}_j \mathbf{v}_j^T.$$

Claramente, nada cambia si eliminamos los sumandos con índice $j > k$, de forma que

$$X = \sum_{j=1}^{k} \sqrt{\lambda_j}\mathbf{u}_j \mathbf{v}_j^T = \tilde{U}\tilde{D}\tilde{V}^T, \tag{9}$$

donde $\tilde{U} = [\mathbf{u}_1 \cdots \mathbf{u}_k] \in \mathcal{M}_{n\times k}(\mathbb{R})$, $\tilde{V} = [\mathbf{v}_1 \cdots \mathbf{v}_k] \in \mathcal{M}_{n\times r}(\mathbb{R})$ y $\tilde{D}$ es la matriz diagonal $\tilde{D} = \text{diag}(d_1, \ldots, d_k) \in \mathcal{M}_{k\times k}(\mathbb{R})$. La factorización (9) es válida para cuaquier matriz con m filas y n columnas si $m \geq n$. Pero si $m < n$ entonces podemos factorizar X^T de esta forma y, trasponiendo vemos que la factorización es válida para cualquier matriz, incluso si $n > m$. Lo recogemos en el siguiente Teorema.

Teorema 2.3 *Toda matriz $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ admite una factorización*

$$X = \tilde{U}\tilde{D}\tilde{V}^T, \tag{10}$$

en la que $\tilde{U} \in \mathcal{M}_{m\times k}(\mathbb{R})$ y $\tilde{V} \in \mathcal{M}_{m\times n}(\mathbb{R})$ tienen columnas ortonormales, $\tilde{D} = \operatorname{diag}(d_1, \ldots, d_k) \in \mathcal{M}_{k\times k}(\mathbb{R})$, $d_1 \geq \cdots d_k > 0$ y $k = \operatorname{rank}(X)$. Los valores d_j^2 son los autovalores no nulos de la matriz X^TX.

La factorización SVD está asociada, de forma parecida a PCA, a algunas formas de aproximación óptima. Partiendo de la factorización (10), para $r \leq k$ denotamos $\tilde{D}_r = \operatorname{diag}(d_1, \ldots, d_r, 0, \ldots, 0)$ y

$$X_r := \tilde{U}\tilde{D}_r\tilde{V}^T. \tag{11}$$

La matriz $X_r \in \mathcal{M}_{m\times n}(\mathbb{R})$ tiene rango r. Probamos a continuación que X_r es la mejor aproximación a X, tanto en norma espectral como en norma de Frobenius, en el conjunto de matrices de rango $\leq r$ en $\mathcal{M}_{m\times n}(\mathbb{R})$. Recordamos que la norma de Frobenius de X, que denotaremos $\|X\|_F$ está dada por $\|X\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n x_{ij}^2 = \operatorname{Tr}(X^TX)$. La primera versión de este resultado para la norma de Frobenius es debida a Eckart y Young (ver [7]). La demostración que se presenta aquí está inspirada en el capítulo III de [22].

Teorema 2.4 (Eckart-Young) *La matriz X_r definida en* (11) *satisface*

$$\sigma(X - X_r) = \min_{L\in\mathcal{M}_{n\times p}(\mathbb{R}):\, \operatorname{rank}(L)\leq r} \sigma(X - L).$$

y

$$\|X - X_r\|_F = \min_{L\in\mathcal{M}_{m\times n}(\mathbb{R}):\, \operatorname{rank}(L)\leq r} \|X - L\|_F.$$

Demostración. La descomposición (10) nos dice que $X = \sum_{j=1}^k d_j \mathbf{u}_j \mathbf{v}_j^T$. Es fácil ver que $X^TX = \sum_{j=1}^k d_j^2 v_j v_j^T$ y, como consecuencia, $\|X\|_F^2 = \sum_{j=1}^k d_j^2$. De forma similar se comprueba que

$$\|X - X_k\|_F^2 = \sum_{j=r+1}^{r} d_j^2.$$

Supongamos que $L \in \mathcal{M}_{m\times n}(\mathbb{R})$ tiene rango r. Entonces $\dim(\ker(L)) = n - r$. Sea $V_{r+1} = [\mathbf{v}_1, \ldots, \mathbf{v}_{r+1}]$ (la matriz formada por las primeras $r+1$ columnas de $\tilde{V}$). La ortonormalidad de las columnas implica que $\dim(\operatorname{Im}(V_{r+1})) = r + 1$. Por lo tanto $\dim(\operatorname{Im}(V_{r+1})) + \dim(\ker(L)) = m + 1$, lo que implica que existe un vector $\mathbf{a}$ unitario (con $\|\mathbf{a}\| = 1$) en $\operatorname{Im}(V_{r+1}) \cap \ker(L)$. Por lo tanto

$$\begin{aligned}
\sigma(X - L)^2 &\geq \|(X - L)\mathbf{a}\|^2 = \|X\mathbf{a}\|^2 = \sum_{j=1}^{k} d_j^2 \langle \mathbf{a}, \mathbf{v}_j\rangle^2 \\
&= \sum_{j=1}^{r+1} d_j^2 \langle \mathbf{a}, \mathbf{v}_j\rangle^2 \geq d_{r+1}^2 = \sigma(X - X_r)^2.
\end{aligned}$$

Esto prueba la primera afirmación. Para la segunda, observamos que, por el Teorema de Weyl (Teorema 7.1 en el Apéndice), si $A, B \in \mathcal{M}_{m\times n}(\mathbb{R})$ y $d_j(C)$ denota el valor singular j-ésimo de C entonces

$$d_{i+j-1}(A+B) \leq d_i(A) + d_j(B).$$

De aquí se deduce que

$$d_{i+r}(X) \leq d_i(X-L) + d_{r+1}(L) = d_i(X-L),$$

puesto que L tiene rango r. Como consecuencia

$$\|X-L\|_F^2 = \sum_{i=1}^{\text{mín}(m,n)} d_i^2(X-L) \geq \sum_{j=r+1}^{k} d_j(X)^2 = \|X-X_r\|_F^2.$$

y esto prueba la segunda afirmación. □

Desde el punto de vista teórico hemos visto que SVD y PCA responden, básicamente, a distintas formas de abordar un problema común. PCA históricamente vino antes y desde la aparición de los ordenadores se ha convertido en una práctica habitual dentro del mundo de la estadística. Sin embargo, en este trabajo, para los algoritmos que vamos a desarrollar es la SVD la técnica que más utilizaremos.

Con SVD o PCA se consigue el objetivo de comprimir la información presente en el conjunto de datos. Es más, podemos controlar de forma simple el porcentaje de variabilidad que retiene nuestra reconstrucción y podemos buscar un equilibrio entre la variabilidad retenida y el tamaño de almacenamiento. Sin embargo, SVD (y, por lo tanto, PCA) tiene ciertas limitaciones que nos llevarán a desarrollar una variante *robusta* en un capítulo posterior.

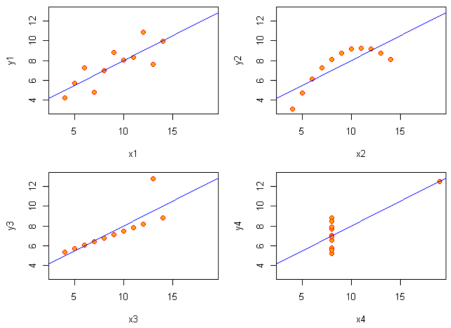


Figura 5: Cuarteto de Anscombe

Una desventaja de SVD y PCA es que son muy sensibles a la presencia de valores atípicos o *outliers*. En el caso de datos bidimensionales la detección es fácil atendiendo a la representación gráfica. Un claro ejemplo es el **Cuarteto de Anscombe**; se trata de cuatro conjuntos de datos que presentan las mismas propiedades estadísticas, pero son claramente distintos cuando observamos sus gráficos

Hemos representado los cuatro conjuntos con su primera componente principal; observamos que aunque la componente principal coincide, los conjuntos de datos son completamente diferentes. En este caso es fácil detectar los valores atípicos puesto que tanto como el conjunto de datos como sus dimensiones son pequeños. En cambio, cuando se trata de múltiples dimensiones el

proceso se complica. No podemos pasar por alto estos valores atípicos; errores de redondeo, una pequeña cantidad de ruido en las mediciones... están presentes en los datos que vamos a tratar y pueden dar lugar a un enorme error de la estimación. Esta inestabilidad –conocida como "mal acondicionamiento"– presenta una gran inconveniencia puesto que en la práctica todas las predicciones generadas tendrán algún tipo de error, que en muchas ocasiones no podremos cuantificar. El desarrollo de técnicas de detección de outliers en datos en alta dimensión es un problema en sí mismo, al que no me he dedicado en este trabajo. Con el objetivo del completado y separado de matrices, la técnica que nos interesa es la *regularización*.

3. Penalización y regularización en aprendizaje automático

Hemos visto que las técnicas de reducción de la dimensión , necesarias en el contexto de alta dimensión, se ven afectadas por la presencia de *outliers.* Desafortunadamente, en el ámbito de Machine Learning, la presencia de este tipo de datos es muy frecuente por lo que no podemos pasarlo por alto. Además, en diversos ámbitos de la ciencia (genética, bioinformática, econometría...) es cada vez más frecuente encontrarse en el contexto de alta dimensión donde $n \gg m$. Al calcular la inversa de la matriz $\mathbf{X}^T\mathbf{X}$, el rango de la matriz $\mathbf{X}$ será como mucho m. Por tanto, la matriz $\mathbf{X}^T\mathbf{X}$ de tamaño $n \times n$ no es inversible. Además, si la matriz $\mathbf{X}^T\mathbf{X}$ es cercana a ser singular, por la existencia de al menos una variable que pudiese ser expresada como combinación lineal del resto, conduce a que el determinante de $(\mathbf{X}^T\mathbf{X})$ sea nulo y por tanto no exista su inversa. Pero, aunque no haya variables que sean combinación lineal de otras basta que algunas estén fuertemente correlacionadas para provocar inestabilidad en la solución del estimador dado que el determinante de $\mathbf{X}^T\mathbf{X}$ será muy cercano a cero. En este contexto de alta dimensión se desarrollaron los métodos de regularización. El objetivo de la regularización era, principalmente, solucionar el problema de la *multicolinealidad*: la estimación de mínimos cuadrados en regresión lineal conduce a estimadores con varianza muy grande cuando m se acerca a n. Para visualizar mejor las barreras que surgen y cómo la regularización los soluciona usaremos un ejemplo clásico de este ámbito: el método de regresión por mínimos cuadrados en altas dimensiones.

En la regresión de mínimos cuadrados tenemos un vector y columna y una matriz X y buscamos entre todas las combinaciones lineales de las columnas de X las que estén más cerca de y en el sentido de mínimos cuadrados

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j X_{ij} \right)^2 \right\} \tag{12}$$

Derivando e igualando a cero, en el caso de que las columnas de X formen un conjunto linealmente independiente, al ser la función objetivo convexa y

diferenciable, la solución es única y está dada por:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{13}$$

Este estimador es insesgado, de hecho es el estimador de mínima varianza de β de la clase de estimadores lineales e insesgados (estimador BLUE por sus siglas en inglés). El modelo asume es $Y = X\beta + \varepsilon$, con Y, $m \times 1$, X matriz $m \times n$ (determinista), β $n \times 1$ y ε (normal) con componentes i.i.d. con media cero y varianza σ^2. Un estimador lineal tiene la forma $\tilde{\beta} = AY$ (A matriz $n \times m$). El estimador es insesgado si $AX\beta = \beta$, as decir, si $(AX - I)\beta = 0$ para todo β. Esto es equivalente a que $AX = I$ (no se puede decir que $A = X^{-1}$ porque X no es cuadrada). Entonces el error cuadratico medio del estimador es

$$E\|\tilde{\beta} - \beta\|^2 = E\|(AX - I)\beta + A\varepsilon\|^2 = E\|A\varepsilon\|^2 = E(\varepsilon^T A^T A\varepsilon) = \sigma^2 \mathrm{Tr}(A^T A).$$

Tenemos que ver ahora que el mínimo de $\mathrm{Tr}(A^T A)$ sujeto a la restricción $AX = I$ es $A = (X^T X)^{-1}X^T$. Si reescribimos A como $A = (X^T X)^{-1}X^T + D$ con $DX = 0$. Entonces

$$\mathrm{Tr}(A^T A) = \mathrm{Tr}((X^T X)^{-1}) + \mathrm{Tr}(D^T D)$$

y el último término es positivo. Luego $A = (X^T X)^{-1}X^T$ es el mínimo.

Este estimador insesgado nos muestra los problemas que surgen con el cálculo de la inversa y la presencia de outliers, por lo que es necesario introducir la regularización. Para ello introduciremos una función de penalización $\Phi(\cdot)$. Una familia de funciones de penalización muy utilizada es la correspondiente a la norma-l_q, dada por

$$\mathbf{\Phi}_\lambda(\beta) = \lambda\left(\|\beta\|_q\right)^q = \lambda\sum_{j=1}^{p}|\beta_j|^q, \quad q > 0 \tag{14}$$

λ es un parámetro de regularización que elegiremos por validación cruzada y que controla la magnitud de la regularización. Dentro de esta familia de funciones, las que utilizaremos a lo largo del trabajo son la regresión Ridge, con norma l_2, y la regresión lasso, con norma l_1. En la primera, Ridge, se ve más fácil cómo la regularización aleja $\mathbf{X}^T\mathbf{X}$ de la singularidad y reduce el sobreajuste, disminuyendo la varianza aunque aumentando el sesgo. Así que veremos las ventajas de la regularización a través de la técnica Ridge. Sin embargo, a efectos prácticos, la que más utilizaremos será la segunda, lasso, puesto que lleva a cabo al mismo tiempo la regularización de nuestro problema y la selección de variables. En los siguientes apartados, para concretar ideas, estudiaremos ambas también desde el contexto de la regresión por mínimos cuadrados. Más adelante veremos el papel del parámetro λ en el caso de la SVD regularizado con la norma nuclear.

3.1. Regularización Ridge

Esta técnica fue propuesta por Hoerl y Kennard, [9], en los años 70 y como hemos visto, se trata de un método de regularización que consiste en añadir una penalización de norma l_2. Para ver cómo este método reduce la varianza y asegura el cálculo de la inversa de la matriz $\mathbf{X}^T\mathbf{X}$ volvamos a nuestro ejemplo de la regresión por mínimos cuadrados.

Los coeficientes estimados por Ridge, $\hat{\beta}^{Ridge}$, son los valores que minimizan

$$\|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2 \tag{15}$$

donde $\lambda \geq 0$ es el parámetro de contracción que determinaremos por validación cruzada.

Si recordamos la solución de la regresión (13), al añadirle la penalización de norma l_2 la solución tendrá la siguiente forma:

$$\hat{\beta}^{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y} \tag{16}$$

La inestabilidad de $\hat{\beta}$ se alivia sumando esta pequeña constante $\lambda > 0$ a cada elemento de la diagonal de $\mathbf{X}^T\mathbf{X}$ antes de invertirla, asegurándonos el rango correcto. Cuanto mayor es λ menor es la varianza del estimador, aunque el sesgo será más grande. Tras unas cuentas, esto es fácil de ver. La SVD de $\mathbf{X}$ tiene la forma UDV^T. Luego $\mathbf{X}^T\mathbf{X} = VD^2V^T$. Con esta reescritura obtenemos la siguiente cadena de igualdades:

$$\begin{aligned}\mathbf{X}^T\mathbf{X} + \lambda I &= VD^2V^T + \lambda VV^T I \\ &= V\left(D^2 + \lambda I\right)V^T\end{aligned}$$

Cuando calculamos la inversa tenemos entonces

$$\begin{aligned}\left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1} &= \left(VD^2V^T + \lambda VV^T I\right)^{-1} \\ &= V\left(D^2 + \lambda I\right)^{-1}V^T\end{aligned}$$

Entre paréntesis tenemos la inversa de una matriz diagonal, que es igual a la inversa de los elementos de la diagonal, $\left(D^2 + \lambda I\right)^{-1} = \sum_{i=1}^{p} \frac{d_i^2}{d_i^2+\lambda}$. Veamos como afecta a la varianza. Tenemos $Y = \beta + \epsilon$

$$\begin{aligned}\left(\mathbf{X}^T\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^T Y = V\left(D^2 + \lambda I\right)V^T \cdot VDU^T \cdot UDV^T\beta + \\ V\left(D^2 + \lambda I\right)V^T \cdot VDU^T\epsilon\end{aligned}$$

El primer término del sumando es fijo, no hay términos que varíen. En cambio el segundo es aleatorio debido a ϵ, que tiene esperanza cero. La esperanza del estimador Ridge es entonces el primer sumando $V\left(D^2 + \lambda I\right)D^2V^T\beta$ y la varianza viene dada por la matriz de covarianzas

$$\left(\sigma^2 V\left(D^2 + \lambda I\right)^{-1}DU^T\right) \cdot \left(V\left(D^2 + \lambda I\right)^{-1}DU^T\right)^T$$

$$= \sigma^2 V \left(D^2 + \lambda I\right)^{-1} D^2 \sigma^2 V \left(D^2 + \lambda I\right)^{-1}$$

Teniendo en cuenta que las matrices diagonales conmutan tenemos que la ecuación anterior es igual a

$$\sigma^2 V[\left(D^2 + \lambda I\right)^{-2} D^2] V^T$$

Entre corchetes tenemos una matriz diagonal que nos va a facilitar el cálculo de la varianza. La varianza del estimador Ridge viene dada por la traza de la matriz de covarianzas y gracias a su conmutatividad tenemos que

$$\begin{aligned}\sum_{j=1}^{p} Var(\beta_j^{Ridge}) &= \sigma^2 Tr\left(V\left(D + \lambda I\right)^{-2} D^2 V^T\right) \\ &= \sigma^2 Tr\left(\left(D + \lambda I\right)^{-2} D^2\right) \\ &= \sigma^2 \sum_{i=1}^{p} \frac{d_i^2}{(d_i^2 + \lambda)^2}\end{aligned}$$

El sesgo de $\hat{\beta}^{Ridge}$ es

$$E(\hat{\beta}^{Ridge}) - \beta = -\lambda V (D^2 + \lambda I)^{-1} V^T \beta$$

De aquí se deduce que

$$\|E(\hat{\beta}^{Ridge}) - \beta\|^2 = \lambda^2 \beta^T V (D^2 + \lambda I)^{-2} V^T \beta$$

y usando la expresión anterior para la varianza, se concluye que el error cuadrático medio del estimador de Ridge se puede escribir en la forma

$$E\|\hat{\beta}^{Ridge} - \beta\|^2 = \sum_{j=1}^{p} a_j \frac{\lambda^2}{(d_j^2 + \lambda)^2} + \sigma^2 \sum_{j=1}^{2} \frac{d_j^2}{(d_j^2 + \lambda)^2},$$

donde a_j es el elemento (j, j) de la matriz $V^T \beta \beta^T V$. Esto permite enteder el papel del λ. En el caso $\lambda = 0$ corresponde a sesgo 0, pero aumentar λ puede disminuir el error cuadrático medio, si la disminución en varianza compensa al aumento en sesgo.

Uno de los inconvenientes de este método es que contrae todos los coeficientes hacia cero, pero sin conseguir anular ninguno de ellos. Por tanto, no se produce selección de variables, permaneciendo en el modelo todas las variables. Este hecho resulta un inconveniente en aquellos estudios que tienen un elevado número n de variables explicativas o predictores. Para evitar este inconveniente se propuso la regresión lasso, incluida en el siguiente apartado.

3.2. Regularización lasso

Tibshirani, [10], propuso la técnica lasso (*least absolute shrinkage and selection operator*), una técnica de regularización, como Ridge, pero con norma

l_1. Esta diferencia en la norma trae consecuencias importantes. Lasso consigue reducir la variabilidad de las estimaciones, reduciendo a la vez los coeficientes, incluso igualándolos a cero. Por tanto lleva a cabo tanto la regularización como una selección de variables, gracias a la norma l_1. De hecho, el auge en los últimos años en la investigación y aplicación de técnicas Lasso se debe, principalmente, a la existencia de problemas donde $m \gg n$, (Tibshirani, [11]) que era uno de los problemas que encontrábamos al calcular la inversa. Ridge sí que reduce los coeficientes, acercándolos a cero, pero sin llegar a igualarlo. Esta es la principal diferencia entre ambas técnicas de regularización. Como nosotros en este trabajo estamos frecuentemente ante matrices con mayor número de predictores que observaciones, lasso va a ser utilizado bastante en nuestro algoritmos.

Si los datos están estandarizados, Lasso resuelve el problema de mínimos cuadrados con restricción de norma l_1 sobre vector de coeficientes:

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j| \tag{17}$$

siendo s y $\lambda \geq 0$ los respectivos parámetros de penalización. Al igual que antes, para buscar el mejor valor del parámetro λ utilizaremos el método de validación cruzada.

El estimador $\hat{\beta}^{\text{lasso}}$ no es lineal en el vector respuesta Y, y no existe una expresión "cerrada" como ocurre con (13) o (16), salvo en el caso de un diseño ortogonal como en el que nos encontramos ($\mathbf{X}^T\mathbf{X}$). La solución a esta regularización es entonces el operador del valor umbral, *soft-thresholding* en inglés. Demostrémoslo. En primer lugar tenemos que tener en cuenta que el paso de regularización implica una minimización escalar, así que analicemos caso escalar, consideremos

$$\min_{x_i}(y_i - x_i)^2 + \lambda|x_i| \quad \lambda > 0 \tag{18}$$

Dividimos en dos casos en función del signo de w_i

Caso 1 $x_i \geq 0$

Calculamos la derivada y la igualamos a cero

$$\begin{aligned}\frac{\partial}{\partial w_i}\left\{(y_i - x_i)^2 + \lambda x_i\right\} &= 0, & \text{x}_i \geq 0\\ -2(y_i - x_i)^2 + \lambda x_i &= 0, & \text{x}_i \geq 0\end{aligned}$$

Y obtenemos como solución

$$x_i = y_i - \frac{\lambda}{2}, \quad x_i \geq 0 \tag{19}$$

Por tanto, las soluciones de w_i son

$$x_i = \begin{cases} x_i = y_i - \frac{\lambda}{2} & \text{si } y_i > \frac{\lambda}{2} \\ x_i = 0 & \text{si } y_i < \frac{\lambda}{2} \end{cases} \tag{20}$$

Lo que se resume en $x_i = \left(y_i - \frac{\lambda}{2}\right)_+$

Caso 2 $x_i \leq 0$

De forma equivalente llegamos a la solución $x_i = \left(y_i - \frac{\lambda}{2}\right)_-$

Por tanto, la solución del operador umbral de valor λ es

$$x_i = \begin{cases} y_i + \frac{\lambda}{2} & \text{si } y_i < -\frac{\lambda}{2} \\ 0 & \text{si } -\frac{\lambda}{2} \leq y_i \leq \frac{\lambda}{2} \\ y_i - \frac{\lambda}{2} & \text{si } y_i > \frac{\lambda}{2} \end{cases} \tag{21}$$

Equivalentemente, $x_i = signo(y_i)\left(|y_i| - \frac{\lambda}{2}\right)$. Esta función es conocida como el operador de valor umbral *Soft-thresholding* en inglés. Se trata de una función continua, con derivada como máximo 1. Por tanto pequeño cambios en y_i no dan lugar a grandes cambios en x_i, es decir, es más estable en presencia de outliers. Reduce la dimensión, puesto que si y_i está entre $[-\frac{\lambda}{2}, \frac{\lambda}{2}]$, *Soft-Thresholding* lo reduce a cero.

Si comparamos ambas técnicas de regresión, no hay un método que siempre domine al otro. En general, los modelos generalizados lasso son mucho mas fáciles de interpretar que los obtenidos mediante Ridge ya que elige un subconjunto de predictores. Así que Lasso sería mejor en un entorno donde en el que un número relativamente pequeño de predictores tiene coeficientes altos y los predictores restantes tienen coeficientes muy pequeños o iguales a cero.

La regresión Ridge obtiene mejores resultados cuando la respuesta es una función de muchos factores predictivos, todos con coeficientes de aproximadamente el mismo tamaño. En este trabajo, según el algoritmo o el contexto, utilizaremos uno o otro.

4. Métodos de optimización en Machine Learning

En este trabajo buscábamos resolver dos objetivos de la Inteligencia Artificial, el Completado y la Separación de matrices. Con PCA y SVD y la regularización hemos reducido y pulido los datos. Ahora lo que queremos saber es qué procedimientos usaremos para resolverlos. Tal como hemos visto en lasso o Ridge, la mayor parte de problemas implican la solución de un determinado problema de optimización. Intentaremos que estos problemas sean convexos ya que admiten un tratamiento numérico más satisfactorio gracias a sus garantías de convergencia óptima global.

Nos centraremos en procedimientos de primer orden por cuestiones de espacio. El método de Newton por ejemplo, que es de segundo orden implicaría el cálculo de la matriz Hessiana, lo que se traduce en una matriz de un millón de entradas en cuanto tengamos 1000 variables. Además necesitamos de algoritmos cuya convergencia no sea sensible a la dimensión ya que la cantidad de datos con los que tratamos son enormes. Por tanto, hemos elegido los métodos derivados

del descenso de gradiente ya que son sencillos de implementar e independientes de la dimensión en su orden de convergencia. Es verdad que no obtenemos una solución exacta, sino estimada, pero siempre que usemos un número suficiente de iteraciones y un ratio de aprendizaje adecuado obtendremos una solución muy cercana al mínimo teórico. Además, dado que ciertos problemas gozan de separabilidad (como es el caso de lasso) hablaremos también de los métodos de proyecciones alternadas. Y dado que no siempre estaremos en el contexto de problemas convexos presentaremos los algoritmos de minorización-mayorización.

Antes de empezar a describirlos, presentaremos conceptos básicos de optimización; daremos definiciones de conjuntos convexos, funciones de coste y restricciones diferenciables, introduciremos el concepto de subgradiente y veremos el uso de los multiplicadores de Lagrange. Una vez visto esto, pasaremos a los diferentes algoritmos iterativos. Así, tras esta sección tendremos los métodos para poder resolver nuestros dos objetivos. La fuente para este capítulo ha sido [23].

4.1. Optimización convexa

A la hora de aplicar el descenso de gradiente necesitamos funciones diferenciables. Sin embargo no todas las funciones a las que nos enfrentamos presentan esta propiedad. Para conseguir generalizar tanto el método de descenso de gradiente como sus variantes introduciremos entonces el concepto de subgradiente. De esta manera, tendremos la base para describir los algoritmos que posteriormente utilizaremos. Empecemos pues definiendo estos conceptos.

4.1.1. Definiciones

- Conjunto Convexo

  Un conjunto $\mathcal{C} \subseteq \Re$ es convexo si para todo $\beta, \beta' \in \mathcal{C}$ y cualquier escalar $s \in [0,1]$, todo vector de la forma $\beta(s) = s\beta + (1-s)\beta'$ también pertenece a $\mathcal{C}$.

- Función convexa

  Una función $f : \Re^p \to \Re$ es convexa si para cualesquiera β, β' del dominio de f y para cualquier escalar $s \in (0,1)$, tenemos que:

  $$f(\beta(s)) = f(s\beta + (1-s)\beta') \leq sf(\beta) + (1-s)f(\beta') \tag{22}$$

  En términos geométricos, esta desigualdad implica que la cuerda que une $f(\beta)$ con $f(\beta')$ se encuentra sobre el grafo de f y garantiza que si una función convexa presenta un mínimo local entonces este mínimo también es global.

- Subgradiente

  Dada una función convexa $f : \Re^p \to \Re$, un vector $z \in \Re^p$ se dice que es un *subgradiente* de f en β si

  $$f(\beta') \geq f(\beta) + \langle z, \beta' - \beta \rangle \text{ para todo } \beta' \in \Re^p.$$

El conjunto de todos los subgradientes de f en β se llama *subdiferencial* y se denota como $\partial f(\beta)$. Esta idea viene de las funciones convexas diferenciables. Éstas tienen la propiedad de que la aproximación de primer orden a la tangente constituye un límite inferior de la función. La noción de subgradiente se basa en generalizar esta propiedad. En los puntos en los que f es diferenciable, el subdiferencial se reduce a un único vector, $\partial f(\beta) = \{\nabla f(\beta)\}$. En los puntos en los que no es diferenciable, el subdiferencial es un conjunto convexo que contiene todos los posibles subgradientes, cada uno de ellos determina un plano tangente que constituye un límite inferior para f.

4.1.2. Optimización en problemas diferenciables

Una vez vistos los conceptos básicos, consideramos el problema de optimización con restricciones

$$\min_{\beta \in \mathcal{C}} \ f(\beta) \text{ sujeto a } \beta \in \mathcal{C} \tag{23}$$

donde $f : \Re^p \to \Re$ es la función convexa que queremos minimizar y $\mathcal{C} \subset \Re^p$ es un conjunto de restricciones convexo. Cuando la funcion f es diferenciable, entonces una condición necesaria y suficiente para que un vector $\beta^* \in \mathcal{C}$ sea el mínimo global óptimo es que

$$\langle \nabla f(\beta^*), \beta - \beta^* \rangle \geq 0 \tag{24}$$

para todo $\beta \in \mathcal{C}$. La condición suficiente es sencilla de ver; para todo $\beta \in \mathcal{C}$, tenemos que

$$f(\beta) \underset{i)}{\geq} f(\beta^*) + \langle \nabla f(\beta^*), \beta - \beta^* \rangle \underset{ii)}{\geq} f(\beta^*), \tag{25}$$

donde la desigualdad (i) se deduce de la convexidad de f, y la desigualdad (ii) de la condición de óptimo.

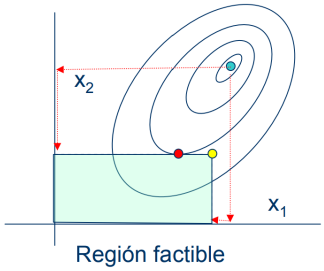


Figura 6: Región factible

Si estamos en el caso donde $\mathcal{C} = \Re^p$ el problema (24) se convierte en un problema sin restricciones y por tanto las condiciones de primer orden se reducen a la clásica condición de gradiente cero $\nabla f(\beta^*) = 0$. En cambio, si $\mathcal{C} \subseteq \Re^p$, se reduce el espacio de búsqueda pero, al mismo tiempo, dificulta el encontrar la solución óptima porque se pierde la condición de que el gradiente es nulo en el óptimo. Una idea que se nos puede ocurrir es calcular el óptimo sin restricciones y luego calcular su proyección a la región factible. Sin embargo, frecuentemente lleva a soluciones incorrectas. Así que tenemos que enfocarlo de otra manera.

Sabemos que, sin restricciones, somos capaces de encontrar el mínimo de una función convexa. Por lo que el objetivo va a ser transformar nuestro problema con restricciones a uno sin ellas para así poder utilizar la condición de gradiente cero $\nabla f(\beta^*) = 0$.

Para ello, primero tenemos que reformular nuestro conjunto de restricción $\mathcal{C}$. Para cualquier función convexa $g : \Re^p \to \Re$, se sigue de la definición (22) que el conjunto subnivel $\{\beta \in \Re^p | g(\beta) \geq 0\}$ es un conjunto convexo (Lema 7.3 en el Apéndice). Entonces reformulamos $\mathcal{C}$ como conjuntos subnivel definidos por funciones de restricción convexas. De esta forma, nuestro problema de optimización restringido a un conjunto se puede escribir como

$$\min_{\beta \in \Re^p} \ f(\beta) \text{ tal que } g_j(\beta) \leq 0 \ para \ j = 1, \ldots, m. \tag{26}$$

donde g_j, $\mathrm{j} = 1, \ldots, \mathrm{m}$ son funciones convexas que expresan las restricciones que deben ser satisfechas. Llamaremos f^* el valor óptimo de este problema de optimización (26).

Ahora que tenemos, $\mathcal{C}$ reformulado utilizamos el *Método de los multiplicadores de Lagrange*, llamados así en honor a Joseph Louis Lagrange, 1797 [12]. Se trata de un procedimiento para encontrar los máximos y mínimos de funciones de múltiples variables sujetas a restricciones. Los problemas convexos, gracias al Lagrangiano, pasan de ser problemas con restricciones a ser resueltos mediante un problema equivalente sin restricciones.

El Lagrangiano asociado al problema (26) se define como $L : \Re^p \times \Re^m_+ \to \Re$, tal que

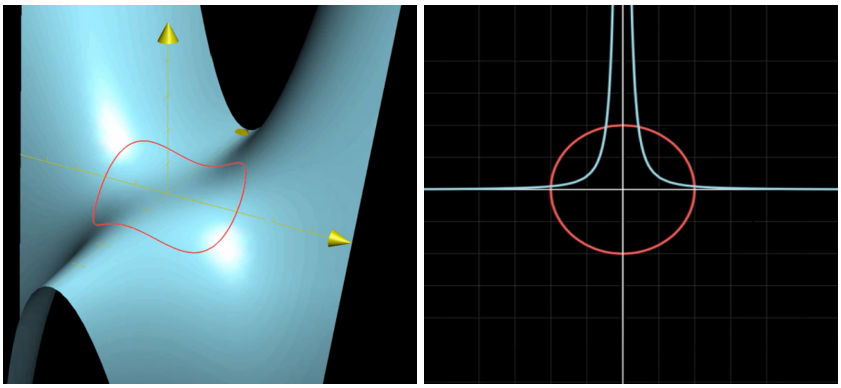
$$L(\beta; \lambda) = f(\beta) + \sum_{j=i}^{m} \lambda_j g_j(\beta). \tag{27}$$

Todo valor óptimo β^* del problema (23), además de satisfacer las restricciones $g_j(\beta^*) \leq 0$, también debe ser un punto de gradiente cero para el Lagrangiano, y por tanto satisfacer esta ecuación

$$0 = \nabla_\beta L(\beta^*; \lambda^*) = \nabla f(\beta^*) + \sum_{j=1}^{m} \lambda_j^* \nabla g_j(\beta^*) \tag{28}$$

Cuando hay una sola función de restricción, la condición se reduce a $\nabla f(\beta^*) = -\lambda \nabla g(\beta^*)$ y tiene una interpretación geométrica intuitiva; en el valor óptimo β^*, el vector normal $\nabla f(\beta^*)$ a la línea de contorno de los puntos de f se encuentra en dirección opuesta al vector normal de la curva de restricción $g(\beta) = 0$. Equivalentemente, el vector normal al contorno de f forma un ángulo recto con el vector tangente de la restricción. En consecuencia, si empezamos en el valor óptimo β^* y nos movemos a lo largo de la tangente en $g(\beta) = 0$, no podremos reducir el valor de $f(\beta)$. La idea central es encontrar puntos en donde los vectores de los gradientes de estas dos funciones sean paralelos, es decir, donde el gradiente del Lagrangiano sea igual a cero. Lo vemos en la figura (7).

En la figura (7a) vemos la función $f = x^2 y$ que es la que queremos minimizar junto con la restricción $x^2 + y^2 = 1$ a la que está sujeta. Esta restricción podemos



(a) Representación gráfica de f y g (b) Curvas de nivel

Figura 7: Lagrangiano asociado a f sujeto a g

verla como la curva de nivel de valor 1 de la función $g = x^2 + y^2$. Los valores máximo y mínimo de f, sujetos a la restricción corresponden a las curvas de nivel de f que son tangentes a la curva de nivel que representa $g(x, y) = 1$. En la figura (7b) el círculo rojo representa la restricción g. Tenemos que ir buscando las curvas de nivel de f que sean tangentes a la de g.

Los pesos no negativos $\lambda \geq 0$, se llaman *multiplicadores Lagrangianos*; estos multiplicadores sirven para imponer una penalización cuando la restricción $g_j(\beta) \leq 0$ no se cumpla. Se puede probar esto a través de la función dual del Lagrangiano como se describe en [13].

Este método de los multiplicadores de Lagrange se aplica a problemas con restricciones de igualdad. Con el fin de generalizarlo y así poder incluir las restricciones de desigualdad surgieron las condiciones de Karush-Kuhn-Tucker (KKT), [14] [15].

4.1.3. Optimización en problemas no diferenciables

Hasta ahora se ha hablado de funciones tanto convexas como diferenciables. Sin embargo, en la práctica, muchos problemas de optimización que encontramos en estadística involucran funciones de coste convexas pero no diferenciables. Por ejemplo, la norma-l_1 $g(\beta) = \sum_{j=1}^{p} |\beta_j|$ es una función convexa, sin embargo no es diferenciable en ningún punto donde al menos una de las coordenadas β_j es igual a cero. Para estos problemas, las condiciones de optimalidad que hemos desarrollado – en particular, la condición de primer orden (23) y la condición del Lagrangiano (28) – no son directamente aplicables puesto que se necesita el gradiente tanto de las funciones de restricción como de coste. Por ello hemos introducido la definición de subgradiente dando lugar a una teoría de optimización más general. En los puntos en los que no es diferenciable, el subdiferencial es un conjunto convexo que contiene todos los posibles subgradientes, cada uno de ellos determina un plano tangente que constituye un límite inferior para f.

Veamos varios ejemplos de esta definición

- Función valor absoluto

  Esta es igual a la la norma l_1 $g(\beta) = \sum_{j=1}^{p} |\beta_j|$ de la que hablábamos antes pero en una sola variable: $f(\beta) = |\beta|$. Se trata de una función convexa, diferenciable en cualquier punto menos en $\beta = 0$. El subgradiente es entonces:

$$\partial f(\beta) = \begin{cases} \{+1\} & si\ \beta > 0 \\ \{-1\} & si\ \beta < 0 \\ [-1, 1] & si\ \beta = 0 \end{cases}$$

- Subgradiente de la norma Euclidea

  Si tomamos la norma Euclidea o también llamada norma l_2 $\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$ tenemos que para cualquier $\beta \neq 0$ la norma $g(\beta) = \|\beta\|_2$ es diferenciable

$$\begin{aligned} \nabla g(\beta) &= \left(\frac{\partial g(\beta)}{\partial \beta_1}, \ldots, \frac{\partial g(\beta)}{\partial \beta_p} \right) \\ &= \left(\frac{\partial \sqrt{\beta_1^2}}{\partial \beta_1}, \ldots, \frac{\partial \sqrt{\beta_p^2}}{\partial \beta_p} \right) \\ &= \left(\frac{1}{2} \left(\beta_1^2\right)^{\frac{1}{2}} 2\beta_1, \ldots, \frac{1}{2} \left(\beta_p^2\right)^{\frac{1}{2}} 2\beta_p \right) \\ &= \left(\frac{\beta_1}{\|\beta_1\|}, \ldots, \frac{\beta_p}{\|\beta_p\|} \right) = \frac{\beta}{\|\beta\|} \end{aligned}$$

  Para $\beta = 0$, por definición de subgradiente tenemos que

$$1 = \nabla g(\beta') \geq \frac{g(\beta') - g(0)}{\beta' - 0}$$

  Por tanto, todo $\hat{s}$ tal que $\|\hat{s}\|_2 \leq 1$ pertenece al subgradiente.

¿Por qué necesitamos el subgradiente? Si recordamos el problema (26) y asumimos que las funciones $\{f, g_j\}$ son convexas pero no diferenciables, entonces el Lagrangiano aquí no tiene sentido. Sin embargo, bajo determinadas condiciones débiles, podemos generalizar la teoría de los multiplicadores y así aplicar el Lagrangiano con ciertas modificaciones

$$0 \in \partial f(\beta^*) + \sum_{j=1}^{*} \lambda_j^* \partial g_j(\beta^*), \tag{29}$$

donde hemos reemplazado los gradientes por los subdiferenciales. Dado que el subdiferencial es un conjunto, la ecuación (29) significa que el vector cero pertenece a la suma de los diferenciales.

Gracias al subgradiente, podemos aplicar el método de los multiplicadores de Lagrange al Lasso. La no diferenciabilidad de la norma l_1, es salvada por el subgradiente. Lasso se puede traducir como un problema de la forma (26) con una función de coste convexa y diferenciable y una única restricción $g(\beta) = \sum_{j=1}^{p} |\beta_j| - R$ para una constante positiva R. Pedir que la restricción cumpla $g(\beta) \leq 0$ equivale a que β pertenezca a la bola-l_1 de radio R. La base estándar es ortonormal, luego la solución de este problema es la que vimos en el apartado de la regresión Lasso. Acabamos de ver el subgradiente de la función valor absoluto, por tanto la condición (29) se convierte en

$$\nabla f(\beta^*) + \lambda^* z^* = 0,$$

donde el vector subgradiente satisface $z_j^* \in sign(\beta_j^*)$ para cada $j = 1, \ldots, p$. Esto se ve recordando la solución de

Una vez definida la teoría, ya podemos introducir varios métodos iterativos para resolver problemas de optimización que tienen como base el descenso de gradiente. .

4.2. Descenso de gradiente

El método de descenso de gradiente es uno de los algoritmos de optimización más populares en aprendizaje automático. Se trata de un método general de minimización para cualquier función f. El gradiente es la generalización vectorial de la derivada, es un vector de tantas dimensiones como la función y cada dimensión contiene la derivada parcial en dicha dimensión:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots \frac{\partial f}{\partial x_n} \right]$$

El gradiente ∇f_x es el vector que contiene la información de cuanto crece la función en un punto especifico x por cada dimensión de nuestra función de forma independiente. De esta manera podemos determinar la dirección de mayor descenso y así acercarnos al mínimo. La versión original del descenso de gradiente es lenta pero versátil, sobretodo en casos como en el que nos encontramos, con funciones multidimensionales. Por ello, en esta sección estudiaremos el descenso de gradiente y algunas de sus alternativas para convertirlo en un algoritmo más eficiente.

4.2.1. Descenso de gradiente sin restricciones

Empezamos por el caso más sencillo: minimizar la función $f : \Re^p \to \Re$ diferenciable y sin restricciones. Asumiendo que se alcanza el mínimo global, que el gradiente sea cero $\nabla f(\beta^*) = 0$ es una condición necesaria y suficiente para encontrar el valor óptimo $\beta^* \in \Re^p$. El descenso de gradiente es un algoritmo iterativo para resolver este problema de punto fijo: genera una secuencia de iterantes $\{\beta^t\}_{t=0}^{\infty}$ gracias a:

$$\beta^{t+1} = \beta^t - s^t \nabla f(\beta^t), \ para \ t = 0, 1, 2, \ldots \tag{30}$$

donde $s^t > 0$ es el parámetro de paso. Geométricamente cuando calculamos el gradiente, determinamos la dirección del descenso más profundo $-f(\beta^t)$, y después nos movemos en esta dirección cierta cantidad determinada por el tamaño de paso s^t. En general, los distintos métodos de descenso se basan en elegir una dirección $\Delta^t \in \Re^p$ tal que $\langle \nabla f(\beta^t), \Delta^t \rangle < 0$, y después llevar a cabo la actualización

$$\beta^{t+1} = \beta^t + s^t \Delta^t \; para \; t = 0, 1, 2, \ldots \tag{31}$$

En términos geométricos, $\langle \nabla f(\beta^t), \Delta^t \rangle < 0$ significa que la dirección elegida Δ^t forma un ángulo de menos de 90^o con la dirección más profunda de descenso. La actualización de descenso de gradiente (30) es un caso especial con $\Delta^t = -\nabla f(\beta^t)$.

Cuando utilizamos el descenso de gradiente, tenemos que elegir el tamaño del paso s^t. Por lo general, no es suficiente con elegir un paso tal que $f(\beta^{t+1}) < f(\beta^t)$; si no lo elegimos bien, podemos hacer que el algoritmo converja a un punto no estacionario. Afortunadamente, existen varias reglas relativamente sencillas para elegir el paso de gradiente que además llevan asociadas garantías de convergencia.

- Regla de la minimización limitada: se elige el paso $s^t = \arg\min_{s \in 0,1} f(\beta^t + s\Delta^t)$. Aunque esta opción es muy intuitiva, requiere resolver un problema de optimización unidimensional en cada paso.

- Regla de Armijo: Dados los parámetros $\alpha \in (0, 0{,}5)$ y $\gamma \in (0, 1)$ y un paso inicial $s = 1$, realiza la reducción $s \leftarrow s\gamma$ hasta que la condición de descenso

$$f(\beta^t + s\Delta^t) \leq f(\beta^t) + \alpha s \langle \nabla f(\beta^t), \Delta^t \rangle \tag{32}$$

se consigue. En la práctica, se suelen elegir $\alpha = 0{,}5$ y $\gamma = 0{,}8$. La condición (32) se puede interpretar como que aceptamos una fracción α de descenso en $f(\beta)$. Como ejemplo tenemos la figura (8). Buscamos el mínimo de la función f representada con la línea azul y vamos reduciendo s a través de γ hasta que se cumpla (8), es decir, que la línea discontinua quede por encima de la función f.

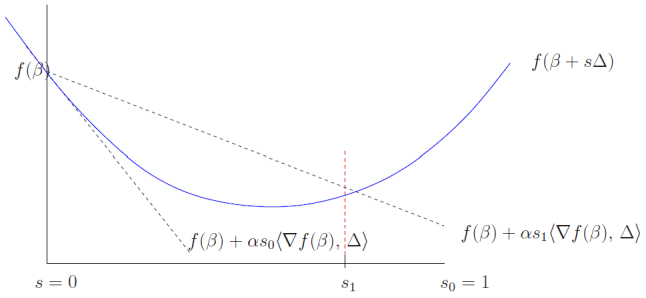


Figura 8: Regla de Armijo

Para funciones convexas, ambas opciones combinadas con una elección de la dirección $\{\nabla^t\}_{t=0}^{\infty}$ adecuada da lugar a algoritmos que garantizan la convergencia al mínimo global de la función convexa.

Otra opción sería elegir un paso constante; sin embargo, solo se asegura la convergencia cuando el problema presenta una estructura especial como es el caso, por ejemplo, de la función cuadrática sin $f(\beta) = \frac{1}{2}\beta^T Q\beta - \langle \beta, b\rangle$ donde $Q \succ 0$ es una matriz simétrica definida positiva, y $b \in \Re^p$

Veamos que la solución existe y es única. Sabemos que se debe cumplir la condición necesaria que $\nabla f(x) = 0$. Tenemos que $\partial f(x) = \underbrace{f'(x)}_{\nabla f(x)^T dx}$. Entonces

$$\begin{aligned}
\partial f &= \frac{1}{2}(\partial x^T Qx + x^T Q dx) + b^T dx \\
&= \frac{1}{2}(x^T Q^T \partial x + x^T Q dx) + b^T dx \\
&= x^T\Big[\frac{Q^T + Q}{2}\Big]dx + b^T dx \\
&= [x^T\Big[\frac{Q^T + Q}{2}\Big] + b^T\Big]dx
\end{aligned}$$

Luego $\nabla f(x) = \frac{Q^T+Q}{2}x + b$ y como $Q = Q^T$, entonces $\nabla f(x) = Qx + b$. Y dado que $\nabla f(x) = 0 \Longrightarrow Qx = -b$

Para probar que hay un un único punto donde se alcanza el mínimo, primero reescribimos la función,

$$\begin{aligned}
f(\beta) &= \frac{1}{2}\beta^T Q\beta - \beta^T b \\
&= \frac{1}{2}\Big[\beta^T Q\beta - 2\beta^T Q(Q^{-1}b)\Big] \\
&= \frac{1}{2}\Big[\beta^T Q\beta - 2\beta^T Q(Q^{-1}b) + b^T Q^{-1}QQ^{-1}b\Big] - \frac{1}{2}b^T Q^{-1}QQ^{-1}b \\
&= \frac{1}{2}(\beta - Q^{-1}b)^T Q(\beta - Q^{-1}b) - \frac{1}{2}b^T Q^{-1}b.
\end{aligned}$$

Si tomamos $x = \beta$ y $y = Q^{-1}b$ tenemos que la igualdad anterior es simplemente una reescritura de la igualdad $\|x - y\|^2 = \|x^2\| + \|y\|^2 - 2\langle x, y\rangle$, válida en cualquier espacio de Hilbert.

Con la igualdad anterior es evidente que la función es un término constante (no dependiente de β) más una forma cuadrática en $\beta - Q^{-1}b$. Como Q es definida positiva la forma cuadrática es positiva y alcanza su mínimo valor posible (cero) si y sólo si $\beta = Q^{-1}b$.

El algoritmo de descenso de gradiente con paso s constante parte de un valor inicial β_0 y luego calcula

$$\beta_{t+1} = \beta_t - s\nabla f(\beta_t) = \beta_t - s(Q\beta_t - b) = (I - sQ)\beta_t + sb, \quad t \geq 1.$$

Estudiemos la convergencia de la iteración

$$\begin{aligned}\beta_{t+1} &= (I-sQ)\beta_t + sb = (I-sQ)((I-sQ)\beta_{t-1}+sb)+sb \\ &= (I-sQ)^2\beta_{t-1} + s[I+(I-sQ)]b \\ &= (I-sQ)^3\beta_{t-2} + s[I+(I-sQ)+(I-sQ)^2]b \\ &\vdots \\ &= (I-sQ)^{t+1}\beta_0 + s\Big[\sum_{k=0}^{t}(I-sQ)^k\Big]b.\end{aligned}$$

Consideremos la norma de operador, es decir,

$$\|A\| = \sup_{x:\,\|x\|=1} \|Ax\|.$$

Esta norma es submultiplicativa, es decir, $\|AB\| \leq \|A\|\|B\|$ (como demostramos en el lema 7.5 del Apéndice). Entonces, si $\|I-sQ\| < 1$ se tendrá que $(I-sQ)^{t+1} \to 0$ (en norma de operador) y por lo tanto $(I-sQ)^{t+1}\beta_0 \to 0$. Además también tendremos que la serie matricial $\sum_{k=0}^{\infty}(I-sQ)^k$ es convergente y

$$\sum_{k=0}^{\infty}(I-sQ)^k = \frac{1}{s}Q^{-1}$$

Conclusión: si $\|I-sQ\| < 1$ entonces la iteración de descenso de gradiente converge a $Q^{-1}b$, que es el único minimizador de f.

¿Qué debe cumplir c para que $\|I-sQ\| < 1$? La respuesta es fácil si se usa que para A simétrica $\|A\|$ es el máximo de los valores absolutos de los autovalores de A. Entonces $\|I-sQ\| < 1$ si y sólo si los autovalores de $I-sQ$ están todos entre -1 y 1. Como Q es definida positiva sus autovalores son positivos. Los autovalores de $I-sQ$ son $1-s\lambda$, con λ autovalor de Q. Entonces la condición que se tiene que cumplir es $1-s\lambda > -1$, es decir, $s\lambda < 2$ para todos los autovalores λ. Equivalentemente, $s < c := \frac{2}{\text{máx}\,\lambda} = \frac{2}{\|Q\|}$. Como resumen, si $0 < s < \frac{2}{\|Q\|}$ entonces la iteración de descenso de gradiente es convergente.

Hasta ahora hemos visto algoritmos iterativos para problemas sin restricciones. Sin embargo, los problemas a los que nos enfrentamos en este trabajo sí que presentan restricciones. Por ello, ahora introduciremos algoritmos para ellos.

Acabamos de ver diferentes algoritmos iterativos para hacer del descenso de gradiente un método eficiente. Sin embargo, los problemas a los que nos enfrentamos están sujetos a restricciones. Por ello, en la siguiente sección buscaremos generalizar el descenso a problemas sujetos a restricciones.

4.2.2. Descenso proximal

Los problemas de completado y separado de matrices de la Inteligencia Artificial están sujetas a restricciones. Como el descenso de gradiente sin restricciones

tiene la condición necesaria y suficiente de que el gradiente sea cero para determinar el mínimo, a través del Lagrangiano hemos conseguido transformar nuestro problema con restricciones a uno sin ellas. Ahora toca hacer el algoritmo más eficiente y versátil tanto para funciones diferenciables como no diferenciables y para ello veremos diferentes métodos. Primero, para tener una idea intuitiva de lo que hacen, hablaremos del método de proyección de gradiente, que es un caso especial dentro de los métodos de gradiente proximal que será el segundo que veamos. Este método nos indicará cómo resolver el los problemas de mínimos no diferenciables. Por último veremos el método de aceleración de gradiente, que consiguen que el algoritmo sea más rápido.

Métodos de proyección de gradiente

La proyección de gradiente es uno de los llamados métodos de penalización. Estos métodos consisten, grosso modo, en la sustitución del problema de mínimos con restricciones (26) por otro problema sin restricciones donde se ha añadido una función (la función de penalización). Esta penalización fuerza a que el mínimo del problema sin restricciones se alcance "cerca" del conjunto $\mathcal{C}$ y que, en el límite, se acerque al mínimo del problema con restricciones. Para tener una idea geométrica de lo que hacen estos métodos es útil tener en cuenta que el paso de gradiente (30) tiene la representación alternativa

$$\beta^{t+1} = arg\min_{\beta\in\Re^p}\{f(\beta^t) + \langle\nabla f(\beta^t), \beta - \beta^t\rangle + \frac{1}{2s^t}\|\beta - \beta^t\|_2^2\} \tag{33}$$

Se puede ver como que se está minimizando una linealización de f cerca del presente iterante, pero con una penalización añadida, la distancia Euclídea.

Al añadirle la penalización (33), llegamos de manera natural al método de proyección de descenso de gradiente, que nos permite minimizar un problema restringudo a un conjunto $\beta \in C$

$$\beta^{t+1} = arg\min_{\beta\in C}\{f(\beta^t) + \langle\nabla f(\beta^t), \beta - \beta^t\rangle + \frac{1}{2s^t}\|\beta - \beta^t\|_2^2\} \tag{34}$$

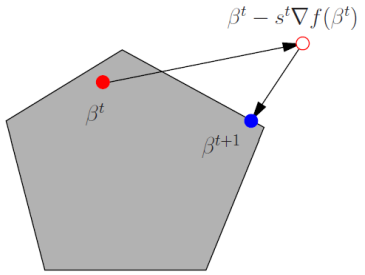


Figura 9: Proyección de gradiente

Este algoritmo corresponde a tomar un paso de gradiente $\beta^t - s\nabla f(\beta^t)$, y después proyectar el resultado al conjunto convexo de restricción, como se muestra en la figura (9). Este algoritmo es eficiente siempre que sea computacionalmente factible.

Métodos de Gradiente Próximal

Hablaremos ahora de una clase de métodos que incluye la proyección de descenso de gradiente como un caso especial. Algunas funciones objetivo f no diferenciables pueden ser descompuestas como suma de dos funciones $f = g + h$, donde g es convexa y diferenciable y h es convexa pero no diferenciable. Supongamos que queremos minimizar una de estas funciones objetivo mediante un algoritmo de tipo gradiente. ¿Cómo podemos trabajar con la no diferenciabilidad de la componente h? Antes de responder, recordemos que un paso de gradiente típico puede ser visto como minimizar la combinación de una aproximación local lineal de f junto con un término cuadrático de regularización, como se ve en la ecuación (33). Nuestra estrategia se basa en esta idea: formar una aproximación lineal de f linearizando la componente diferenciable g y mantener la componente no diferenciable h fija. La actualización del gradiente dada por

$$\beta^{t+1} = arg \min_{\beta \in \Re^p} \{g(\beta^t) + \langle \nabla g(\beta^t), \beta - \beta^t \rangle + \frac{1}{2s^t} \|\beta - \beta^t\|_2^2 + h(\beta)\} \tag{35}$$

La actualización se relaciona con la del método de proyección de gradiente (34); de hecho, se puede ver como su análogo Lagrangiano. Para ver esta conexión, definimos el mapa proximal de una función convexa h:

$$prox_h(z) := \underset{\theta \in \Re^p}{arg\,\min} \left\{ \frac{1}{2}\|z - \theta\|_2^2 + h(\theta) \right\}. \tag{36}$$

De esta definición podemos deducir las siguientes relaciones:

- $prox_{sh}(z) := arg\,\min_{\theta \in \Re^p} \left\{ \frac{1}{2s}\|z - \theta\|_2^2 + h(\theta) \right\}$
- Cuando

  $$h(\theta) = I_C(\theta) = \partial f(\beta) = \begin{cases} 0 & si \;\; \theta \in \mathcal{C} \\ +\infty & sino \end{cases}$$

  tenemos que $prox_h(z) = arg\,\min_{\theta \in \mathcal{C}} \|z - \theta\|_2^2$, correspondiente a la proyección usual de Euclides sobre el conjunto $\mathcal{C}$.
- Si $h(z) = \lambda\|\theta\|_1$, entonces $prox_h(z) = \mathcal{S}_\lambda(z)$

Veamos que el paso de actualización (35) es equivalente al paso

$$\beta_{t+1} = \mathbf{prox}_{sh}(\beta_t - s\nabla g(\beta_t)), \tag{37}$$

Para ver la equivalencia entre (35) y (37) observamos que

$$\mathbf{prox}_{sh}(\beta_t - s\nabla g(\beta_t)) \quad = \quad \text{argmin}_\beta \left\{ \frac{1}{2s}\|\beta - \beta_t + s\nabla g(\beta_t)\|_2^2 + h(\beta) \right\}.$$

Usamos la identidad $\|x+y\|^2 = \|x\|^2 + \|y\|^2 - 2\langle x, y\rangle$ para comprobar que

$$\frac{1}{2s}\|\beta - \beta_t + s\nabla g(\beta_t)\|_2^2 + h(\beta) = \langle \nabla g(\beta_t), \beta - \beta_t\rangle + \frac{1}{2s}\|\beta - \beta_t\|_2^2 + h(\beta) + C.$$

El término C es 'constante', es decir, no depende de β. Sólo falta darse cuenta que la función a minimizar en (35) y la que se minimiza en (37) se diferencian únicamente en un término constante entonces el punto en el que se alcanza el mínimo es el mismo para las dos funciones y por eso los pasos (35) y (37) son equivalentes.

Del mismo modo vemos que el paso de actualización del gradiente proximal

$$\beta_{t+1} = \mathbf{prox}_{I_C}(\beta_t - s\nabla g(\beta_t)), \tag{38}$$

es exactamente el paso de proyección de gradiente.

La actualización (36) se ajusta mejor a los problemas estadísticos que imponen regularización a través de una penalización como pueden ser el *lasso* o el *ridge*, Siempre que el mapa proximal sea fácil de programar, estas actualizaciones serán computacionalmente eficientes. Si en cambio la regularización viene dada por una restricción de la forma $h(\theta) \leq \mathcal{R}$, la actualización (37) es más adecuada.

Como ya hemos comentado, la regularización Lasso será muy utilizada en los algoritmos para resolver nuestro dos objetivos dentro de la Inteligencia Artificial. Así que veamos cómo el método del gradiente proximal funciona con nuestro regularizador. Primero veamos cómo se aplica e método cuando la componente no diferenciable es la norma l_1, $h(\theta) = \lambda\|\theta\|_1$. Con esta elección de h, el descenso de gradiente proximal de paso s^t en la iteración t consiste en dos sencillos pasos:

1. Primero, tomamos un paso de gradiente $z = \beta^t - s^t\nabla g(\beta^t)$.
2. En segundo lugar, aplicar el operador de valor umbral a cada uno de los elementos $\beta^{t+1} = \mathcal{S}_{s^t\lambda}(z)$.

Esto se debe a que si nos fijamos más en el mapa proximal (37)

$$\begin{aligned} prox_{sh}(z) \quad &= \quad \underset{\theta\in\Re^p}{arg\,mín} \left\{ \frac{1}{2s}\|z-\theta\|_2^2 + \lambda\|\theta\|_1 \right\} \\ &= \quad \underset{\theta\in\Re^p}{arg\,mín} \left\{ \frac{1}{2}\|z-\theta\|_2^2 + s\lambda\|\theta\|_1 \right\} \end{aligned}$$

Este problema de optimización tiene una solución explícita cerrada en la que en vez de resolver un problema p-dimensional podemos resolver cada uno de los problemas univariantes puesto que es de variables separadas.

$$\frac{1}{2}\|z-\theta\|_2^2 + s\lambda\|\theta\|_1 = \sum_{j=1}^{p}\left\{\frac{1}{2s}(z-\theta)_2^2 + \lambda|\theta|_1\ \right\},$$

Para el lasso tenemos que

$$g(\beta) = \frac{1}{2N}\|y - X\beta\|_2^2 \ \ y\ h(\beta) = \lambda\|\beta\|_1,$$

de tal manera que el la actualización de gradiente (37) tiene la siguiente forma

$$\beta^{t+1} = \mathcal{S}_{s^t\lambda}\left(\beta^t - s^t\frac{1}{N}X^t(y - X\beta^t)\right) \tag{39}$$

Hemos visto cómo el método de gradiente proximal resuelve el lasso. Pero, ¿está garantizada la convergencia del método? Nesterov (2007) [16], dio condiciones suficientes para la convergencia de las actualizaciones (37) cuando se aplican a una función objetivo compuesta $f = g + h$.

Teorema 4.1 (Nesterov) *Dada una función objetivo compuesta $f = g + h$, donde g es una función convexa y diferenciable y h es convexa, diferenciable o no. Supongamos que la componente g es continuamente diferenciable con un gradiente de Lipschitz, es decir, que existe cierta constante L tal que*

$$\|\nabla g(\beta) - \nabla g(\beta')\|_2 \leq L\|\beta - \beta'\|_2 \text{ para todo } \beta, \beta' \in \Re^p. \tag{40}$$

Bajo esta condición y bajo cierto paso de gradiente constante $s^t = s \in (0, 1/L]$ se puede demostrar que existe una constante C, independiente de la iteración, tal que la actualización (37) satisgafa

$$f(\beta^t) - f(\beta^*) \leq \frac{C}{t+1}\|\beta^t - \beta^*\|_2 \text{ para todo } t = 1, 2, \ldots, \tag{41}$$

donde β^* es la solución óptima. En otras palabras, la diferencia entre la iteración t^{th} y el valor óptimo $f(\beta^*)$ decrece a una velocidad $O(1/t)$.

Este ratio se conoce como convergencia sublineal y está garantizada para cualquier paso de gradiente fijo del intervalo $(0, 1/L]$. La elección del paso requiere una cota superior de la constante de Lipschitz L, que puede estar disponible o no. Si utilizamos en cambio la regla de Armijo, también obtenemos el mismo ratio.

Si la función objetivo tiene cierta estructura adicional, podemos encontrar mejores cotas. Pero no van a ser utilizadas en el TFG por lo que no entraremos en detalles.

Métodos de aceleración de gradiente

Este nuevo método de penalización nos sirve para acelerar el proceso de descenso de gradiente cuando estamos ante una función obejetivo f diferenciable

y convexa. El paso de gradiente estándar (30) puede dar lugar entre paso y paso a un movimiento en "zig-zag"no deseado ya que ralentiza la convergencia. Con el objetivo de aliviar este inconveniente, Nesterov (2007), [16] propuso una clase de métodos de aceleración de gradiente que usaba combinaciones ponderadas tanto de la actual como la anterior dirección de gradiente. Este tipo de método necesita entonces de dos secuencias $\{\beta^t\}_{t=0}^{\infty}$ y $\{\theta^t\}_{t=0}^{\infty}$ y la condición inicial $\beta^0 = \theta^0$. Para las iteraciones $t = 0, 1, 2, \ldots$, la secuencias siguen la siguiente actualización:

$$\beta^{t+1} = \theta^t - s^t \nabla f(\theta^t),\ y \tag{42}$$

$$\theta^{t+1} = \beta^{t+1} + \frac{t}{t+3}(\beta^{t+1} - \beta^t) \tag{43}$$

Para funciones f no diferenciables, que admiten la descomposición en suma de diferenciable y no diferenciable, $g + h$, como es el caso del lasso, el esquema de aceleración de Nesterov se puede combinar con la actualización de gradiente proximal: si sustituimos el paso de gradiente ordinal (42) con la actualización

$$\beta^{t+1} = prox_{s^t h}(\theta^t - s^t \nabla g(\theta^t)). \tag{44}$$

Tenemos entonces el método de gradiente acelerado a (39) donde el par tendrá la forma

$$\beta^{t+1} = \mathcal{S}_{s^t \lambda}\left(\theta^t + s^t \frac{1}{N} X^t (y - \theta^t)\right) \tag{45}$$

$$\theta^{t+1} = \beta^{t+1} + \frac{t}{t+3}\left(\beta^{t+1} - \beta^t\right) \tag{46}$$

A nivel de computación, las actualizaciones (42) y (44) solo suponen un poco más de trabajo que la actualización de gradiente ordinario. Sin embargo, Nesterov (2007) probó que este cambio da lugar a una mejora significativa en la velocidad de convergencia.

$$f(\beta^t) - f(\beta^*) \leq \frac{C}{(t+1)^2} \|\beta^0 - \beta^*\|_2.$$

En consecuencia, el error $f(\beta^t) - f(\beta^*)$ decrece a una velocidad $\mathcal{O}(1/t^2)$, obviamente más rápida que $\mathcal{O}(1/t)$ del método no acelerado.

4.2.3. Descenso por coordenadas

Hemos visto el descenso de gradiente, como adaptarlo a los problemas con restricciones. Con los métodos de penalización hemos superado la barrera de la no diferenciabilidad y conseguimos mayor rapidez a través de la aceleración de gradiente. De cada uno de los métodos hemos visto cómo se aplica a nuestro regularizador Lasso. Pero precisamente este regularizador y sus variantes, gozan de una característica que todavía no hemos usado y que permitiría desarrollar algoritmos más eficientes: la separabilidad. Esta propiedad puede ser aprovechada a través de los algoritmos de optimización por coordenadas. El *descenso*

por coordenadas es un algoritmo iterativo que para pasar de β^t a β^{t+1} elige una única coordenada para actualizar y lleva a cabo una minimización univariante sobre ella, es decir, si la coordenada k es elegida en la iteración t, entonces la actualización viene dada por

$$\beta_k^{t+1} = \underset{\beta_k}{arg\,\text{mín}} f(\beta_1^t, \beta_2^t, \ldots, \beta_{k-1}^t, \beta_k, \beta_{k+1}^t, \ldots, \beta_p^t), \tag{47}$$

y $\beta_k^{t+1} = \beta_k^t \ \forall j \neq k$. Normalmente, el algoritmo va de manera cíclica por todas las coordenadas siguiendo un orden fijado. Este método también se puede llevar a cabo por bloques; en este caso dividimos las coordenadas en partes que no se solapan y llevamos a cabo en cada ronda una minimización sobre cada bloque.

¿Cúando podemos asegurar que este proceso converge al mínimo global de una función convexa? Una condición suficiente es la de pedir que f sea continuamente diferenciable y fuertemente convexa sobre cada coordenada. Pero es demasiado restrictiva. Sin ir más lejos, el Lasso no es diferenciable. Así que tenemos que buscar otras condiciones no tan restrictivas que aseguren la convergencia de estos métodos.

Nos ponemos primero en contexto. Supongamos que la función de coste f tiene la descomposición aditiva

$$f(\beta_1, \ldots, \beta_p) = g(\beta_1, \ldots, \beta_p) + \sum_{j=1}^{p} h_j(\beta_j), \tag{48}$$

donde $g : \Re^p \to \Re$ es una función diferenciable y convexa, y donde las funciones univariables $h_j : \Re :\to \Re$ son convexas (aunque no necesariamente diferenciables). Este es el caso del Lasso por ejemplo, con $g(\beta) = \frac{1}{2N}\|y - X\beta\|_2^2$ y $h_j(\beta_j) = \lambda \cdot |\beta_j|$.

Tseng (1988,2001), [17], [18] demostró que para cualquier función de coste convexa con estructura separable (48), el algoritmo de descenso coordinado asegura la convergencia al mínimo global. La propiedad clave que subyace este resultado es la separabilidad de la componente no diferenciable $h(\beta) = \sum_{j=1}^{p} h_j(\beta_j)$ como suma de funciones de coordenadas individuales. De hecho, cuando la componente no diferenciable h no es separable, no tenemos garantías sobre la convergencia y es posible que el algoritmo se atasque en un punto y no consiga llegar al mínimo global.

Un ejemplo donde vemos que falla el descenso por coordenadas es el siguiente. Tomamos el *fussed lasso*. Este regularizador tiene dos penalizaciones: la primera se corresponde a la norma l_1 característica mientras que la segunda fuerza a que los coeficientes seguidos sean similares. Este regularizador es útil cuando procesamos el ruido de los píxeles de una imagen: forzamos que píxeles vecinos tengan valores similares. O también es muy utilizado en estudios genómicos ya que más que genes individuales, cuando se lleva a cabo una replicación se utiliza un segmento. El regularizador explota esta estructura de similitud a través de

la siguiente fórmula

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^N}{\text{minimize}} \left\{ \frac{1}{2}\sum_{i=1}^{N}(y_i-\beta_i)^2 + \lambda_1\sum_{i=1}^{N}|\beta_i| + \lambda_2\sum_{i=2}^{N}|\beta_i-\beta_{i-1}| \right\} \tag{49}$$

Para ver cómo falla la no separabilidad de la componente no diferenciable tomamos este operador con 100 parámetros. Aquí la componente no diferenciable toma la forma $h(\beta) = \sum_{j=1}^{p}|\beta_j - \beta_{j-1}|$, por tanto no separable. Las soluciones de dos de sus parámetros son $\beta_{63} = \beta_{64} \approx -1$. Los dos primeros gráficos de la figura (10) representan la función f en función de cada uno de los parámetros β_{63} y β_{64}, con el valor óptimo del resto de parámetros. El descenso por coordenadas minimiza a ambos por separado a -0.69. La última gráfica que representa ambos parámetros en dos dimensiones nos muestra como el descenso coordinado se queda atascado en el punto (-0.69, -0.69). A pesar de ser estrictamente convexa, la superficie presenta esquinas por lo que el descenso por coordenadas se quedará atascado a no ser que movamos a la vez β_{63} y β_{64}.

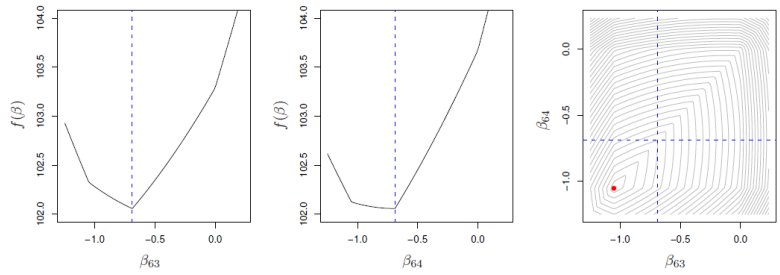


Figura 10: Fussed lasso

Para evitar este tipo de situaciones, Tseng (2001), [18] dio una condición más general e intuitiva sobre la convergencia del descenso por coordenadas, una que dependiese del comportamiento de las derivadas direcionales de la función de coste f. Dada una dirección $\Delta \in \Re$, la menor derivada direccional en β es dada por

$$f'(\beta;\Delta) := \liminf_{s\downarrow 0}\frac{f(\beta+s\Delta)-f(\beta)}{s}. \tag{50}$$

En términos generales, el algoritmo de descenso por coordenadas solo gana información sobre las direcciones con la forma $e^j = (0, 0, \ldots 0, e_j, 0, \ldots, 0)$ para algún $e_j \in \Re$. Por lo tanto, supongamos que el algoritmo de descenso coordinado alcanza un punto β donde

$$f'(\beta; e^j) \geq 0 \;\forall\; j = 1, \ldots, p, \text{ y vectores coordenados } e^j. \tag{51}$$

En dicho punto, no existirá una dirección e_j en la que disminuya el valor de la función. Por lo tanto necesitamos que dado β que satisfaga esta condición

(51), también cumpla $f'(\beta;\Delta) \geq 0$ para toda dirección $\Delta \in \Re^p$. Tseng llamó *regularidad* a esta condición. Observemos que la separabilidad de la componente no diferenciable de la función objetivo implica regularidad.

Recordamos el estimador lasso. Como ya hemos visto, las condiciones de optimalidad para este problema son son

$$-\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)x_{ij} + \lambda s_j = 0 \tag{52}$$

donde $s_j \in signo(\beta_j)$ para $j =, \ldots, p$. El descenso por coordenadas resuelve estas ecuaciones de manera cíclica, iterando sobre $j = 1, 2, \ldots, p, 1, 2, \ldots$.

Para simplificar, definimos el *residuo parcial* $r_i^{(j)} = y_i - \sum_{k\neq j} x_{ik}\hat{\beta}_k$, que elimina de la variable respuesta el ajuste actual de todos los predictores excepto el $j - ésimo$. En este caso la solución de $\hat{\beta}_j$ satisface

$$\hat{\beta}_j = \frac{\mathcal{S}_\lambda\left(\frac{1}{N}\sum_{i=1}^{N} r_i^{(j)} x_{ij}\right)}{\frac{1}{N}\sum_{i=1}^{N} x_{ij}^2} \tag{53}$$

donde $\mathcal{S}_\lambda = (\theta)(|\theta| - \lambda)_+$ es el operador del valor umbral. Si además de centrar las variables, también las estandarizamos (lo cual es una buena idea especialmente cuando tratamos con variables en diferentes unidades), la actualización presenta la siguiente forma

$$\hat{\beta}_j = \mathcal{S}_\lambda(\tilde{\beta}_j) \tag{54}$$

donde $\tilde{\beta}_j$ es el coeficiente de la regresión lineal simple del residuo parcial de la variable j.

Podemos seguir varias estrategias para hacer eficientes estas operaciones. Para aliviar la notación asumiremos que los predictores están estandarizados y que tienen media cero y varianza uno; si los datos no están estandarizados sería similiar.

- Residuos parciales

  Reescribimos $r_i^{(j)} = y_i - \sum_{k\neq j} x_{ik}\hat{\beta}_k = r_i^{(j)} = r_i - x_{ij}\hat{\beta}_j$ donde r_i es el residuo parcial en la obervación i. Teniendo en cuenta que las variables están estandarizadas$\{x_j\}_{j=1}^{p}$ tenemos

  $$\frac{1}{N}\sum_{i=1}^{N} x_{ij} r_i^{(j)} = \frac{1}{N}\sum_{i=1}^{N} x_{ij} r_i + \widehat{\beta}_j \tag{55}$$

  Esta manera de escribirlo refleja la eficiencia computacional del descenso de gradiente; muchos coeficientes son iguales a cero y mantienen este valor tras pasar por el operador umbral, por lo que nada tiene que cambiar. Computacionalmente, el mayor coste vienen dado por la suma de la ecuación (55), que requiere $\mathcal{O}$(N) operaciones. Por otro lado, si algún

coeficiente sí que cambia al aplicarle el operador del valor umbral, entonces r_i también cambia en $\mathcal{O}(\mathrm{N})$ y entonces el coste de las operaciones sería $\mathcal{O}(2\mathrm{N})$. Una vuelta entera sobre todas las variables p costaría $\mathcal{O}(p\mathrm{N})$. Friedman et al. (2010), [19] llamó *Actualización naïve* a este método dado que trabaja directamente con los productos internos de los datos.

- Actualización por covarianza

  Este tipo de actualización es todavía más eficiente que la anterior cuando $N \gg p$ y N es grande. Salvo por el factor $\frac{1}{N}$, podemos escribir el primer término de la derecha de la expresión (55) como

$$\sum_{i=1}^{N} x_{ij} r_i = \langle \mathbf{x}_j, \mathbf{y} \rangle - \sum_{k||\widehat{\beta}_k|>0} \langle \mathbf{x}_j, \mathbf{x}_k \rangle \, \widehat{\beta}_k \tag{56}$$

  Visto así, podemos calcular los productos internos de cada variable con y inicialmente, y después cada vez que una nueva variable x_j entra en el modelo por primera vez, calculamos el producto interno de dicha variable con el resto de variables que se encuentren activas, un total de $\mathcal{O}(\mathrm{p})$ operaciones. También almacenamos los p gradientes (56). Si alguno de los coeficientes del modelo cambia podemos actualizar cada uno de los gradientes con $\mathcal{O}(N)$ operaciones. Con k componentes distintos de cero en el modelo, una vuelta completa supone un coste de $\mathcal{O}(pk)$ operaciones si ninguna variable nueva deja de ser cero y $\mathcal{O}(Np)$ por cada una que entre nueva. Pero lo que es más importante, ninguna de los pasos necesita de $\mathcal{O}(N)$ operaciones.

- Buen arranque

  Normalmente se busca un conjunto de soluciones *lasso* para una secuencia decreciente de valores $\{\lambda_l\}_0^L$. Tenemos que el máximo valor de λ que podemos considerar es

$$\lambda_0 = \frac{1}{\mathrm{N}} \max_j |\langle \mathrm{x}_j, \mathrm{y} \rangle| \tag{57}$$

  ya que un valor mayor daría lugar a un modelo vacío. Una estrategia, empleada por el paquete de R, glmnet, es la de crear una secuencia de valores $\{\lambda_l\}_{l=0}^L$ decreciente de λ_0 hasta $\lambda_L \approx \epsilon\lambda_0$ en escala logarítmica. El número de coeficientes distintos de cero tiende a aumentar lentamente con l, siendo cero cuando $l = 0$. Para calcular $\hat{\beta}(\lambda_{l+1})$, si tomamos como valor inicial $\hat{\beta}(\lambda_l)$ el algoritmo se vuelve más eficiente. De esta manera, doblar el número $\mathrm{L} = 100$ no dobla el tiempo de computación, sino que los valores iniciales son cada vez mejores y necesitan de menos iteraciones en cada ronda para llegar al óptimo.

- Convergencia del conjunto activo

  Después de una iteración sobre el conjunto de variables p con un nuevo valor de λ_l, empezando desde $\hat{\beta}(\lambda_{l-1})$, definimos como conjunto activo $\mathcal{A}$ al

conjunto que guarda el índice de todas aquellas variables cuyo coeficiente es distinto de cero. La idea es realizar iteraciones del algoritmo únicamente sobre las variables que están en $\mathcal{A}$. Cuando el algoritmo converge, hacemos un test de exclusión a estas variables $\frac{1}{\mathrm{N}}|\langle \mathbf{x}_j, \mathbf{r}\rangle| < \lambda_l$, donde $\mathbf{r}$ es el residuo actual. Si todas pasan este test entonces habremos llegado a la solución para el conjunto entero de las p variables. Si hay alguna variable que no pasa este test, entonces se la incluye en el conjunto activo $\mathcal{A}$ y se repite el proceso. En la práctica se suele hacer el conjunto del revés, es decir, en $\mathcal{A}$ incluimos las variables que tienen coeficiente cero hasta el momento.

- Convergencia del conjunto fuerte

  Similar al anterior, elegimos un subconjunto de variables que parecen ser buenos candidatos para estar en el conjunto activo. Sea $\mathbf{r}$ el residuo en $\hat{\beta}(\lambda_{l-1})$, y queremos la solución en λ_l. Definimos el conjunto fuerte $\mathcal{S}$ como

$$\mathcal{S} = \{j \mid |\frac{1}{N}\langle \mathbf{x}_j, \boldsymbol{r}\rangle| > \lambda_\ell - (\lambda_{\ell-1} - \lambda_\ell)\} \tag{58}$$

  Ahora buscamos la solución restringiéndonos únicamente a las variables que se encuentran en $\mathcal{S}$. Salvo raras excepciones, el conjunto fuerte cubre el conjunto activo óptimo. Las reglas fuertes son muy útiles, especialmente cuando p es elevada.

- Dispersión

  Computacionalmente, la principal operación que se lleva a cabo en todas las anteriores es el producto interno de vectores de dimensión N, donde al menos uno de ellos es una columna de la matriz $\mathbf{X}$. Si $\mathbf{X}$ es una matriz prácticamente vacía, entonces estos productos internos son eficientes computacionalmente. Un ejemplo es la clasificación de documentos, donde cada variable sigue el modelo "saco de palabras". Cada documento se puntúa según la presencia o ausencia de cada una de estas palabras del diccionario que consideremos. Dado que la mayoría de las palabras estarán ausentes, el vector característica de cada documento será practicamente el vector cero y por tanto la matriz también. Dichas matrices pueden ser almacenadas eficientemente si nos quedamos únicamente con los coeficientes distintos de cero y la posición que ocupa cada uno de ellos. Ahora cuando calculamos los productos internos, tenemos en cuenta únicamente los que son distintos de cero.

- Penalización fuerte

  Por defecto, se aplica a todas las variables el mismo parámetro de penalización. Sin embargo, habrá variables que tenemos claro que están correlados con nuestra predicción, por tanto, no es lógico penalizarlos. Para evitar esto, es sencillo añadir una penalización de fuerza $\gamma_j \geq 0$ por variable,

haciendo que la penalización global sea

$$\lambda \sum_{j=1}^{p} \gamma_j P_\alpha(\beta_j).$$

De esta manera permitimos que algunos γ_j sean iguales a cero, lo que significa que dichas variables estarán siempre en el modelo, sin penalización.

- Límites para los parámetros El descenso coordinado también permite poner cotas tanto inferiores como superiores sobre cada parámetro:

$$\mathcal{L}_j \leq \beta_j \leq \mathcal{U}_j$$

Normalmente, $-\infty \leq \mathcal{L}_j \leq \beta_j \leq \mathcal{U}_j \leq \infty$. Uno de los límites más frecuentes es el de los coeficientes no negativos.

4.3. Regresión del ángulo mínimo

Nos centramos ahora en el caso del lasso. Hemos sabido aprovechar la separabilidad que presenta este regularizador y acabamos de ver diferentes estrategias que podemos seguir para hacer más eficiente su solución. Los problemas de Inteligencia Artificial a los que nos enfrentamos se basan en la resolución de un problema de optimización regularizado. Como vimos en el capítulo 3, esta regularización viene acompañada de un parámetro λ. Cambiar el valor del parámetro λ significa cambiar el problema y por tanto tener que empezar de cero cada vez para resolverlo. Con el objetivo de hacer más eficiente el cálculo de la solución para diferentes valores de λ nació la regresión del ángulo mínimo (LAR). No solo busca resolver el problema regularizado para un λ fijo, sino que intenta de forma eficiente encontrar todas las soluciones para cada λ en un intervalo sin tener que repetir el algoritmo para cada valor de λ. No surgió específicamente para el lasso pero una pequeña modificación en uno de sus pasos vale para calcular $\hat{\beta}(\lambda)$ para todo λ que nos interese. Este algoritmo es una versión más democrática de la regresión progresiva (*forward stepwise regression).*

La regresión progresiva crea un modelo de manera secuencial; en cada paso, identifica la variable que mejor explica la variabiliad del modelo y la incluye en el conjunto activo. El modelo LAR sigue una estrategia similar, aunque solo incluye tanto de cada predictor como merece. Son necesarios m pasos, donde m es el número de variables. Tal como la regresión gradual, empezamos con todos los coeficientes igualados a cero y buscamos la variable con mayor correlación con la variable respuesta. En lugar de ajustar esta variable completamente, nos movemos hacia ella de forma continua de tal modo que el error cuadrático medio va disminuyendo (al igual que su correlación con el residuo). Cuando otra variable tiene tanta correlación con el residuo actual como la primera, se para el proceso. La segunda variable entra entonces en el conjunto activo. Y es ahora donde vemos la diferencia entre los dos métodos; la regresión de ángulo mínimo no sigue la dirección de la variable que mayor correlación tenga con el residuo, sino que se mueve de manera equiangular entre las dos variables. Continúa así,

haciendo decrecer la correlación de ambas variables con el residuo hasta que otra tercera variable equipara esta correlación y se mueven las tres a lo largo de la "*dirección de ángulo mínimo*". Así sucesivamente hasta que todas las variables estén dentro del modelo.

Para ver cómo las correlaciones entre cada variable con el residuo van disminuyendo de la misma manera en todas las variables a la vez consideremos el siguiente ejemplo. Tomamos una regresión en la que tanto regresores como respuesta tienen media cero y desviación típica uno.

$$y, x_1, \ldots, x_p \in \mathbb{R}^N; \quad y_1 \perp 1, x_i \perp 1, \text{ para todo } i = 1, \ldots, p_i,$$

$$\frac{1}{N}\sum_{i=1}^{N} y^2 = \frac{\|y_1\|^2}{N} = \frac{\|x_j\|^2}{N} = 1$$

Supongamos que todas las variables presentan la misma correlación con la respuesta, es decir,

$$\frac{1}{N}\left|\langle \mathbf{x}_j, \mathbf{y}\rangle\right| = \lambda, \quad \text{para todo } j = 1, \ldots, p \tag{59}$$

Sea $\widehat{\beta}$ el coeficiente de mínimos cuadrados de $\mathbf{y}$ en $\mathbf{X}$, que asumimos único. Siendo $\mathbf{X} = [x_1, \ldots, x_p]$ la matriz de variables, tenemos que la solución de nuestro modelo de regresión debe cumplir que $\mathbf{X}^\top \mathbf{X}\hat{\beta} = \mathbf{X}^\top y$. Llamamos $\mathbf{u} = \mathbf{X}\hat{\beta}$.

Con el objetivo de ver cómo decrece la correlación consideremos la siguiente función $\mathbf{u}(\alpha) = \alpha \mathbf{X}\hat{\beta}$ para $\alpha \in [0, 1]$. Esta función hace que el vector que se mueva una fracción α hacia el mejor ajuste $\mathbf{u}$ de nuestro problema de regresión. Veamos cómo van cambiando las correlaciones de cada variable $\mathbf{x}_j$ con el residuo a medida que nos acercamos a $\mathbf{u}$. Sea x_j una de las variables tenemos que

$$\begin{aligned}
\left\langle x_j, y - \alpha x\hat{\beta} \right\rangle &= \left\langle x_j, y - x\hat{p} + x\hat{\beta} - \alpha x\hat{\beta} \right\rangle \\
&= \left\langle x_j, y - x\hat{\beta} \right\rangle + \left\langle x_j, (1-\alpha)\cdot x \cdot \hat{\beta} \right\rangle \\
&= (1-\alpha)\left\langle x_j, x\hat{\beta} \right\rangle = (1-\alpha)\left\langle x_j, y \right\rangle
\end{aligned}$$

Luego $\frac{1}{N}\left\langle x_j, y - \alpha x\hat{\beta} \right\rangle = (1-\alpha)\lambda$ para todo $j = 1, \ldots, p$ por lo que las correlaciones de todas las variables $\mathbf{x}_j$ con el residuo se mantienen iguales a medida que nos acercamos a $\mathbf{u}$. Denotamos $RSS = \|y - \mathbf{X}\widehat{\beta}\|_2^2$, al residuo del mejor ajuste, de acuerdo con el criterio de mínimo error cuadrático y tenemos que

$$\begin{aligned}
\|y - \mathbf{u}(\alpha)\|\|^2 &= \|y - \mathbf{X}\hat{\beta} + (1-\lambda)\cdot \mathbf{X}\hat{\beta}\|^2 \\
&= \|y - \mathbf{X}\hat{\beta}\|^2 + (1-\lambda)\cdot \|\mathbf{X}\hat{\beta}\|^2 \quad \text{por ortogonalidad} \\
&= RSS + (1-\alpha)^2 \cdot \|\mathbf{X}\hat{\beta}\|^2
\end{aligned}$$

Por hipótesis, $\|y\|^2 = N$; además $\|y\|^2 = \|y - \mathbf{X}\hat{\beta}\|^2 + \|\mathbf{X}\hat{\beta}\|^2$ también por ortogonalidad. Por lo que deducimos que $\|\mathbf{X}\hat{\beta}\|^2 = N - RSS$. De aquí obtenemos que

$$\|y - \mathbf{u}(\alpha)\|^2 = \left(1 - (1-\alpha)^2\right) RSS + N(1-\alpha)^2 = \alpha(2-\alpha) \cdot RSS + N(1-\alpha)$$

Gracias a todas estas observaciones podemos escribir la correlación entre el residuo y las variables como

$$\begin{aligned}
\lambda(\alpha) &= \frac{\sum_{i=1}^N (x_i - \bar{x}) \cdot (y_i - \mathbf{u}(\alpha))}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^N (y_i - \mathbf{u}(\alpha))^2}} \\
&= \frac{\sum_{i=1}^N (x_i) \cdot (y_i - \mathbf{u}(\alpha))}{\sqrt{\sum_{i=1}^N (x_i)^2} \cdot \sqrt{\sum_{i=1}^N (y_i - \mathbf{u}(\alpha))^2}} \\
&= \frac{\left|\frac{1}{N} \cdot \langle x_j, y - \mathbf{u}(\alpha)\rangle\right|}{\underbrace{\sqrt{\frac{1}{N} \cdot \|x_j\|^2}}_{1} \sqrt{\frac{1}{N}||y - \mathbf{u}(\alpha)||^2}} = \frac{(1-\alpha)\lambda}{\sqrt{(1-\alpha)^2 + \frac{\alpha(2-x)}{N} \cdot RSS}} \\
&= \frac{\lambda}{\sqrt{1 + \frac{\alpha(2-\alpha)}{(1-\alpha)^2} \cdot \frac{RSS}{N}}}
\end{aligned}$$

Además, esta correlación decrece de manera monótona a cero. Esto es lo mismo que probar que $g(\alpha) = \frac{\alpha(2-\alpha)}{(1-\alpha)^2} = \frac{1-(1-\alpha)^2}{(1-\alpha)^2} = \left(\frac{1}{1-\alpha}\right)^2 - 1$ es creceiente con α que es obvio a partir de la última igualdad con $\alpha \in (0, 1)$.

Una vez vista la idea que hay detrás de este algoritmo y cómo decrecen las correlaciones de las variables con respecto al residuo, describamos el algoritmo

Algoritmo: Regresión de ángulo mínimo.

1. Estandarizar los predictores. Empezar con el residuo $\boldsymbol{r}_0 = \mathbf{y} - \overline{\mathbf{y}}, \beta^0 = (\beta_1, \beta_2, \ldots, \beta_p) = \mathbf{0}$
2. Encontrar el predictor $\mathbf{x}_j$ más correlado con el residuo r_0; es decir, el que mayor valor de $|\langle \mathbf{x}_j, \boldsymbol{r}_0 \rangle|$ presente. Llamaremos a este valor λ_0, definimos el conjunto activo $\mathcal{A} = \{j\}$, y $\mathbf{X}_\mathcal{A}$, la matriz formada por esta única variable.
3. Para $k = 1, 2, \ldots, K = \text{mín}(N-1, p)$ hacer:
   a) Definir la dirección de mínimos cuadrados $\delta = \frac{1}{\lambda_{k-1}} \left(\mathbf{X}_\mathcal{A}^T \mathbf{X}_\mathcal{A}\right)^{-1} \mathbf{X}_\mathcal{A}^T \boldsymbol{r}_{k-1}$, y definir el vector p dimensional Δ tal que $\Delta_\mathcal{A} = \delta$, y el resto de elementos están igualados a cero.
   b) Mover los coeficientes β de β^{k-1} en la dirección Δ hacia la solución de mínimos cuadrados de $\mathbf{X}_\mathcal{A}$: $\beta(\lambda) = \beta^{k-1} + (\lambda_{k-1} - \lambda)\Delta$ para $0 < \lambda \leq \lambda_{k-1}$, haciendo un seguimiento de los residuos $\boldsymbol{r}(\lambda) = \mathbf{y} - \mathbf{X}\beta(\lambda) = \boldsymbol{r}_{k-1} - (\lambda_{k-1} - \lambda)\mathbf{X}_\mathcal{A}\delta$.
   c) Hacer un seguimiento de $|\langle \mathbf{x}_l, \boldsymbol{r}(\lambda) \rangle|$ para $l \notin \mathcal{A}$, identificar el mayor valor de λ para el cual la variable entraría en el conjunto activo; si la variable tiene índice j, eso significa que $|\langle \mathbf{x}_j, \boldsymbol{r}(\lambda) \rangle| = \lambda$. Esto define el siguiente "nodo" λ_j.
   d) Actualizamos $\mathcal{A} = \mathcal{A} \cup j$, $\beta^k = \beta(\lambda_k) = \beta^{k-1} + (\lambda_{k-1} - \lambda_k)\Delta$ y $r_k = \mathbf{y} - \mathbf{X}\beta^k$
4. Devolver la secuencia $\left\{\lambda_k, \beta^k\right\}_0^K$.

Para entender mejor el código de este algoritmo vamos a hacer un par de observaciones. Para empezar, el número de veces K que repetimos el bucle es $\text{mín}(N-1, p)$. Esto se debe a que si $p > N-1$, la solución de LAR llega a correlación cero con el residuo después de $N-1$ pasos (el -1 viene de que el modelo presenta un intercepto por lo que hemos tenido que centrar los datos). Una vez que estamos dentro del bucle, el primer paso (a) nos dice que la dirección de mínimos cuadrados $\delta = \frac{1}{\lambda_{k-1}} \left(\mathbf{X}_\mathcal{A}^T \mathbf{X}_\mathcal{A}\right)^{-1} \mathbf{X}_\mathcal{A}^T \boldsymbol{r}_{k-1}$. Veamos porqué; supongamos que estamos en la primera fase del bucle, $k = 1$. Luego el conjunto activo está formado por una única variable x_1, (si no, reordenamos índices). Proyectamos el residuo sobre $\mathbf{X}_\mathcal{A}$ y teniendo en cuenta que en esta ronda $\mathcal{A}$ está formado por una única variable tenemos que

$$\delta = \left(\mathbf{X}_\mathcal{A}^T \mathbf{X}_\mathcal{A}\right)^{-1} \mathbf{X}_\mathcal{A}^T \boldsymbol{r}_0 = \left(x_1^T x_1\right)^{-1} x_1^T \boldsymbol{r}_0 = \left(x_1^T x_1\right)^{-1} \lambda_0 = \lambda_0 \|x_1\|^2$$

Ahora creamos el vector p-dimensional Δ que tiene como coordenada de x_1 el valor δ que acabamos de calcular y las otras $p-1$ coordenadas igualadas a cero. Sin embargo, lo que buscamos es desplazarnos en esta dirección de manera continua para así ir reduciendo la correlación de esta variable con el residuo hasta que nos encontremos con la siguiente variable que tenga tanta correlación con el

residuo como el conjunto activo actual. En consecuencia, buscamos únicamente la dirección para ir disminuyendo la correlación, es por esto que dividimos δ por la actual correlación, en este caso λ_0, y obtenemos así tanto el δ como el vector $p-$dimensional Δ que estamos buscando. Por tanto es claro para $k = 1$.

Lo mismo ocurre en la ronda k. Ahora el conjunto activo $\mathbf{X}_{\mathcal{A}}$ está formado por las primeras k variables $x_1, \ldots, x_k$, salvo reordenación de índices. La última en entrar ha sido x_k con correlación con el residuo λ_k. Como todas las variables presentan la misma correlación con el residuo, a la hora de calcular la dirección de mínimos cuadrados volvemos a tener un escalar, λ_k tal que

$$\delta = \left(\mathbf{X}_{\mathcal{A}}^T \mathbf{X}_{\mathcal{A}}\right)^{-1} \mathbf{X}_{\mathcal{A}}^T \boldsymbol{r}_k = \left(\mathbf{X}_{\mathcal{A}}^T \mathbf{X}_{\mathcal{A}}\right)^{-1} \langle \mathbf{X}_{\mathcal{A}}^T, \boldsymbol{r}_k \rangle = \lambda_k \left(\mathbf{X}_{\mathcal{A}}^T \mathbf{X}_{\mathcal{A}}\right)^{-1}$$

por lo que necesitamos dividir δ por λ_k y redefinir el vector p dimensional Δ tal que $\Delta_{\mathcal{A}} = \delta$, y el resto de elementos igualados a cero. De esta manera, el conjunto activo se moverá en la dirección de mínimos cuadrados hasta que otra variable que no esté en el conjunto presente la misma correlación.

¿Cómo sabremos cúal es la siguiente variable en entrar al conjunto activo? ¿Cuál será el siguiente nodo? Intuitivamente, nos movemos en la dirección que disminuye la correlación con el residuo Δ y a la vez vamos calculando la correlación de todas las variables que no están en el conjunto activo. En cuanto veamos que una iguala su correlación con el conjunto, la incorporamos y tendremos el nodo, calculamos la siguiente dirección y repetimos. En cambio, parece computacionalmente costoso ir preguntando en cada paso tantas correlaciones. Con el objetivo de automatizar el proceso, en el apartado 2 de [20], aprovechando que en cada ronda la dirección Δ que va a seguir nuestro modelo es conocida, además del punto de partida, β_k y la correlación de todas las variables con el modelo $c_l = \langle x_l, \mathbf{r}_{k-i} \rangle$ para todo l, consigue determinar cúal es la nueva variable que entra en el modelo sin calcular la correlación en cada paso.

El algoritmo LAR es bastante eficiente y consigue reducir la correlación de las variables con el residuo de manera continua. Además con un simple cambio en el Paso 3.c podemos calcular la solución de lasso para diferentes valores de λ como veremos en el próximo capítulo en el algoritmo `Soft-Impute` .Aprovecha las estrategias del conjunto activo, utiliza la actualización por covarianza, la elección de los valores iniciales que toma a cada paso hacen que el algoritmo converja más rápido... Sin embargo, a gran escala no es eficiente. Es por esto que introducimos en el siguiente apartado una nueva clase de problemas que serán interesantes puesto que permiten dividir el problema inicial en problemas más pequeños, luego las grandes dimensiones no suponen un inconveniente.

4.4. Método de los multiplicadores de Lagrange con direcciones alternadas

El método de los multiplicadores de Lagrange con direcciones alternadas (ADMM) se trata de un algoritmo basado en el Lagrangiano muy útil para problemas a gran escala. Estamos ante un problema con restricciones cuya función de coste es separable, por lo que podemos escribirlo como

$$\min_{\beta\in\mathbb{R}^m,\theta\in\mathbb{R}^n} f(\beta)+g(\theta) \quad \text{subject to } \mathbf{A}\beta+\mathbf{B}\theta=c \tag{60}$$

donde $f:\Re^m\to\Re$ y $g:\Re^n\to\Re$ son funciones convexas, y $\mathbf{A}\in\Re^{n\times d}$ y $\mathbf{B}\in\Re^{n\times d}$ son las matrices restricción y $c\in\Re^d$ es el vector restricción. Para resolverlo, introducimos el Lagrangiano aumentado asociado

$$L_\rho(\beta,\theta,\mu):=f(\beta)+g(\theta)+\langle\mu,\mathbf{A}\beta+\mathbf{B}\theta-c\rangle+\frac{\rho}{2}\|\mathbf{A}\beta+\mathbf{B}\theta-c\|_2^2 \tag{61}$$

donde $\mu\in\Re^d$ es el vector de los multiplicadores de Lagrange y $\rho>0$ es un parámetro pequeño. El algoritmo ADMM se basa en en minimizar el Lagrangiano aumentado (61) sucesivamente sobre β y θ y después alplicar una actualización sobre μ para que cumpla la restricción asociada a ρ. Las actualizaciones entonces serían

$$\begin{aligned}\beta^{t+1}&=\arg\min_{\beta\in\mathbb{R}^m} L_\rho\left(\beta,\theta^t,\mu^t\right)\\ \theta^{t+1}&=\arg\min_{\theta\in\mathbb{R}^n} L_\rho\left(\beta^{t+1},\theta,\mu^t\right)\\ \mu^{t+1}&=\mu^t+\rho\left(\mathbf{A}\beta^{t+1}+\mathbf{B}\theta^{t+1}-c\right)\end{aligned} \tag{62}$$

para las iteraciones $t=0,1,2,\ldots$ La elección del parámetro ρ tiene gran influencia sobre la convergencia de ADMM; si elegimos ρ demasiado grande no se dará tanta importancia a la minimización de $f+g$, en cambio si lo elegimos demasiado pequeño entonces estará poco regularizado. Boyd et al. (2010) propuso una estrategia para calcular ρ, que funciona bien en la práctica pero que carece de garantías de convergencia.

El algoritmo ADMM, presenta grandes ventajas, por ejemplo, en problemas convexos pero con restricciones no diferenciables; podemos encontrar la solución mediante una separación de los parámetros entre β y θ. Lo veremos en el siguiente ejemplo aplicando ADMM a *lasso*. Otra ventaja es que puede dividir un problema de grandes dimensiones en varios problemas pequeños, más fáciles de resolver. Estamos en el caso donde la función objetivo es seperable.

Si escribimos el problema *lasso* en el formato ADMM tenemos que

$$\begin{array}{ll}\text{minimize} & f(x)+g(z)\\ \text{subject to} & x-z=0\end{array} \tag{63}$$

donde $f(x)=(1/2)\|Ax-b\|_2^2$ y $g(z)=\lambda\|z\|_1$. Las acualizaciones del algoritmo son entonces

$$\begin{aligned}x^{k+1}&=\left(A^TA+\rho I\right)^{-1}\left(A^Tb+\rho\left(z^k-u^k\right)\right)\\ z^{k+1}&=S_{\lambda/\rho}\left(x^{k+1}+u^k\right)\\ u^{k+1}&=u^k+x^{k+1}-z^{k+1}\end{aligned}$$

Se necesita entonces la regresión ridge para β , un paso por el valor umbral para θ y después una simple actulización lineal para μ. El primer paso es el más

costoso, puesto que es necesario una descomposición en valores singulares de $\mathbf{X}$ que necesita $\mathcal{O}(p^3)$ operaciones. Sin embargo el resto de iteraciones son rápidas, de coste $\mathcal{O}(\mathrm{N}p)$. Por tanto, después de la primera fase, el coste por iteración es similar al descenso coordinado o método de gradiente compuesto.

4.5. Algoritmos de Minorización-Mayorización

Hasta ahora hemos visto cómo tratar con problemas convexos, cómo hacer los algoritmos más eficientes y asegurándonos siempre de su convergencia. Sin embargo, a la hora en el mundo de la Inteligencia Artificial en el que nos movemos, los problemas no suelen ser convexos. Presentan saltos, irregularidades... que hacen que nuestros algoritmos no aseguren el óptimo global. Por eso, presentamos ahora una nueva clase de algoritmos para la optimización de problemas convexos. Se conocen como los algoritmos de minorización-mayorización (MM) y se basan en introducir una variable extra que tiene como objetivo establecer una cota superior a la función objetivo que queremos minimizar. Daremos una breve descripción de esta clase de algoritmos aunque luego no serán más utilizados a lo largo del trabajo.

Estos métodos se suelan aplicar generalmente a problemas sujetos a restricciones, pero para describirlos consideremos que no las presenta. Tomamos el problema $\mín_{\beta\in\Re^p} f(\beta)$, donde $f:\Re^p\to\Re$ puede ser convexa o no.

Una función $\Psi:\Re^p\times\Re^p\to\Re^1$ mayoriza la función f en el punto $\beta\in\Re^p$ si

$$f(\beta)\leq\Psi(\beta,\theta)\quad\text{para todo }\theta\in\Re^p \tag{64}$$

La igualdad se alcanza cuando $\beta=\theta$. La correspondiente definicion de función minorizante tiene la igualdad en sentido contrario. El algoritmo MM para llevar a cabo la minimización lo que hace es inicializar con un valor β y después llevar a cabo la minimización

$$\beta^{t+1}=\arg\min_{\beta\in\mathbb{R}^p}\Psi\left(\beta,\beta^t\right)\qquad\text{for } t=0,1,2,\ldots \tag{65}$$

Por la propiedad (64), este esquema genera una secuencia para la función de coste no creciente dado que

$$f\left(\beta^t\right)=\Psi\left(\beta^t,\beta^t\right)\geq\Psi\left(\beta^{t+1},\beta^t\right)\geq f\left(\beta^{t+1}\right) \tag{66}$$

donde la primera desigualdad utiliza el hecho que β^{t+1} es un minimizador de la función $\beta\mapsto\Psi(\beta,\beta^t)$, y la segunda desigualdad utiliza la propiedad de mayoración (64). Si la función original f es estrictamente convexa, se puede probar que el algoritmo MM converge a la solución global. Hay diferentes clases de funciones de mayorización para este tipo de problemas. En general, se considera que una función de mayorización es buena si es relativamente sencilla de programar y permite aún así minimizar f.

Por ejemplo, si recordamos el algoritmo de gradiente proximal, éste lo aplicábamos a funciones de coste f que se descomponen como suma de funciones $f=g+h$ donde g es una función convexa y diferenciable y h es convexa y probablemente

no diferenciable. Aplicando la serie de Taylor de segundo orden con respecto a g obtenemos

$$\begin{aligned} f(\beta) &= g(\beta) + h(\beta) \\ &= g(\theta) + \langle \nabla g(\theta), \theta - \beta \rangle + \frac{1}{2} \left\langle \theta - \beta, \nabla^2 g\left(\beta'\right)(\theta - \beta) \right\rangle + h(\beta) \end{aligned}$$

donde $\beta' = s\beta + (1-s)\theta$ para cierta $s \in [0,1]$. Nuestro objetivo es buscar una función que acote a f y que nos sirva a su vez para minimizarla. Si recordamos la condición de Lipschitz sobre el gradiente ∇g (**??**) tenemos una cota superior del Hessiano, es decir, $\nabla^2 g(\beta') \preceq L\mathbf{I}_{p\times p}$, por lo que podemos obtener la desigualdad

$$f(\beta) \leq \underbrace{g(\theta) + \langle \nabla g(\theta), \theta - \beta \rangle + \frac{L}{2}\|\theta - \beta\|_2^2 + h(\beta)}_{\Psi(\beta,\theta)}$$

y se da la igualdad cuando $\beta = \theta$. De esta manera vemos que el gradiente proximal puede ser visto como un tipo de algoritmo de mayorización.

5. Aproximación y completado de matrices

Nos movemos ahora al mundo de la Inteligencia Artificial. La base de muchos algoritmos de machine learning es encontrar los parámetros del modelo que minimizan una función de coste dados los datos de entrenamiento. Acabamos de ver en la seción anterior los problemas convexos, diferenciables o no, sin restricciones o con ellas, hemos definido el Lagrangiano, introducido diferentes algoritmos iterativos para resolverlo de manera eficiente, hemos visto distintas formas de aprovehar la separabilidad, regularidad y otras propiedades de las funciones y también hemos hablado de varias estrategias a seguir (residuos parciales, conjuntos activos, penalizaciones...) con el fin de elaborar algoritmos más eficientes,y en general menos costosos y más rápidos. Si un problema de optimización es convexo sabemos que si existe un mínimo local este será un mínimo global. Por tanto, si los problemas presentes en Machine Learning son convexos, tenemos garantizada su solución. Sin embargo este no siempre es el caso; los problemas presentan irregularidades que hacen que el proceso de optimización se complique. Pero podemos adaptarnos; en lugar de trabajar con este problema podemos utilizar como función de coste su envolvente convexa, añadr penalizaciones, restricciones... y así llevar el problema a nuestro terreno.

Por tanto, en esta sección trataremos con conjuntos de datos en forma de matriz $\mathbf{X} = x_{ij}$ de tamaño $m \times n$ y buscamos una matriz $\hat{\mathbf{X}}$ que la aproxime. Generalmente, nuestra aproximación tendrá la siguiente estructura

$$\hat{X} = \arg\min_{M \in \Re} \|X - M\|_F^2 \ \textit{sujeto a}\ \Phi(M) \leq c \tag{67}$$

donde $\|\cdot\|$ es la norma de Frobenius de una matriz, y $\Phi(\cdot)$ es la función de restricción. La manera en la que elegimos esta restricción derivará en una gran cantidad de procedimientos.

El objetivo de este trabajo era el completado y separado de las matrices, luego las restricciones que elegiremos serán de acuerdo a estos objetivos. Primero empezaremos exponiendo el completado de matrices; introduciremos el problema, elegiremos la restricción conveniente, estableceremos las condiciones en las que se puede completar la matriz e introduciremos dos algoritmos diferentes para conseguir nuestro objetivo. Para poner en práctica todo esto, expondremos el Desafío Netflix en el que también se reflejará la necesidad de la reducción de la dimensionalidad y de la regularización que se vieron en las dos primeras secciones. Después, nos centraremos en resolver el segundo objetivo, el separado de matrices. Este apartado sigue una estructura similar al anterior, introduciendo el problema y las restricciones elegidas, también hablaremos de cúando es posible este separado de matrices y de las numerosas aplicaciones que presenta.

5.1. Completado de matrices

El análisis de las componentes principales nos da gran información sobre cómo están organizados los datos de nuestra matriz X, la regularización evita los problemas que presenta este análisis y los algoritmos de la sección anterior ecuentran los mínimos de nuestras funciones de coste. Pero ¿qué ocurre si algunas de las entradas de la matriz están vacías? Este problema de rellenar los valores que faltan a una matriz se conoce como *completado de matrices* (Laurent 2001, ver [21]).

Un ejemplo en el que aparece este tipo de problema es en los filtros colaborativos que dan lugar a los sistemas de recomendación. Estos filtros toman una matriz de datos, donde las filas son los usuarios y las columnas los productos y cada entrada de la matriz es la puntuación del usuario i al producto j. Tomamos el sistema de recomendación de Netflix como ejemplo. Mientras que antes, para llevar a cabo el completado se daba más importancia a las características de las películas (género, año, actores, reseñas...) y también de los usuarios (edad, preferencias...), estos filtros se basan más en las puntuaciones de los usuarios que en esta información externa. De esta manera, los usuarios "colaboran" a la hora de que el sistema recomiende tanto a ellos mismos como a usuarios con gustos similares. Sin embargo, lo normal será que los usuarios valoren una cantidad pequeña de películas, con lo que la mayor parte de las entradas de la matriz de valoraciones estará vacía.

Este es el problema en el que no todas las entradas de la matriz $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ están disponibles, sino que únicamente se dispone de las entradas x_{ij} con $(i,j) \in \Omega \subset \{1,\dots,m\} \times \{1,\dots,n\}$. En esta situación podríamos plantearnos completar la matriz buscando una matriz de rango bajo que esté próxima a la parte observada de X. Esto parece obvio puesto que, por ejemplo, cuando vamos a elegir qué película ver, con que nos la haya recomendado un amigo, saber que está nuestro actor favorito o que sea un musical que nos encanta, nos vale para saber que nos va a gustar. De acuerdo con resultados anteriores, la mejor aproximación (tanto en norma de Frobenius como en norma espectral) a una matriz X por matrices de rango a lo sumo k se calcula de forma directa a partir de la descomposición de valor singular. Esta aproximación de rango bajo

es uno de los elementos básicos en muchas técnicas de completado de matrices. Denotando $\|A\|_{F,\Omega}^2 = \sum_{(i,j)\in\Omega} a_{ij}^2$, la reconstrucción estaría dada por

$$\hat{X}_k = \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R}):\,\mathrm{rank}(L)\leq k}{\operatorname{argmin}} \|X - L\|_{F,\Omega}. \tag{68}$$

El problema de minimización (68) no es convexo y, desafortunadamente, no hay un algoritmo eficiente para el cálculo aproximado de la solución. Por esta razón resulta más conveniente considerar relajaciones convexas del problema de aproximación de rango bajo. De esta manera podremos aprovechar todo lo que hemos hablado a lo largo de este trabajo sobre problemas convexos, algoritmos para resolverlos, subdiferenciales, lagrangianos... y así resolver este problema a través de una de las estrategias más exitosas dentro del completado de matrices: el completado usando la *norma nuclear* (o trace norm, ver [22], p. 91), dada por

$$\|X\|_* = \sum_{j=1}^{\min(m,n)} d_j(X),$$

donde $d_j(X)$ son los valores singulares de X. El uso de la norma nuclear surge al observar que el rango de una matriz coincide con su número de valores singulares no nulos. Por tanto, resolver el problema (68) es equivalente a minimizar la cantidad de valores singulares no nulos. Siguiendo esta idea, pero a la vez buscando un problema convexo, surge el uso de la norma nuclear que admite la representación dada por el siguiente resultado:

Teorema 5.1 *Si $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ entonces*

$$\|X\|_* = \max_{Y\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(Y)\leq 1} \mathrm{Tr}(Y^T X)$$

Demostración. Los valores singulares no nulos de X son las raíces cuadradas de los autovalores no nulos de $X^T X$ (o, equivalentemente, las raíces cuadradas de los autovalores no nulos de XX^T). Esto implica que en la descomposición de valor singular $X = UDV^T$ tendremos $\|X\|_* = \|D\|_*$. Por otro lado

$$\max_{Y\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(Y)\leq 1} \mathrm{Tr}(Y^T X) = \max_{Y\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(Y)\leq 1} \mathrm{Tr}((U^T Y V)^T D).$$

Si denotamos $\tilde{Y} = U^T Y V$ entonces $\sigma(Y) = \sigma(\tilde{Y})$, de donde se deduce que

$$\max_{Y\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(Y)\leq 1} \mathrm{Tr}(Y^T X) = \max_{\tilde{Y}\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(\tilde{Y})\leq 1} \mathrm{Tr}(\tilde{Y}^T D).$$

Esto demuestra que basta considerar el caso $m = n$ y X diagonal con entradas no negativas ordenadas de forma decreciente. En este caso podemos tomar $Y = I$ para comprobar que

$$\max_{Y\in\mathcal{M}_{m\times n}(\mathbb{R}):\sigma(Y)\leq 1} \mathrm{Tr}(Y^T X) \geq \mathrm{Tr}(X) = \sum_{j=1}^{r} d_j(X).$$

Recíprocamente, si e_j, $j = 1, \ldots, m$ son los vectores de la base canónica de $\mathbb{R}^m$ entonces $X = \sum_{j=1}^m d_j(X) E_j$ con $E_j = e_j e_j^*$. De aquí

$$\begin{aligned} \max_{Y \in \mathcal{M}_{m\times m}(\mathbb{R}):\sigma(Y)\leq 1} \operatorname{Tr}(Y^T D) &\leq \sum_{j=1}^m d_j(X) \max_{Y \in \mathcal{M}_{m\times m}(\mathbb{R}):\sigma(Y)\leq 1} \operatorname{Tr}(Y^T E_j) \\ &= \sum_{j=1}^m d_j(X) \max_{Y \in \mathcal{M}_{m\times m}(\mathbb{R}):\sigma(Y)\leq 1} e_j^T Y^T e_j. \end{aligned}$$

Por la desigualdad de Cauchy-Schwarz tenemos que

$$e_j^T Y^T e_j \leq \|Y^T e_j\| \leq \|e_j\| = 1$$

y esto, unido a la cota anterior demuestra que

$$\max_{Y \in \mathcal{M}_{m\times m}(\mathbb{R}):\sigma(Y)\leq 1} \operatorname{Tr}(Y^T D) \leq \sum_{j=1}^m d_j(X).$$

Esto completa la demostración (de hecho prueba que el máximo se alcanza para $Y = I$). □

El Teorema (5.1) justifica que efectivamente $\|\cdot\|_*$ es una norma. Es evidente que $\|X\|_* \geq 0$ y que $\|X\|_* = 0$ si y sólo si $X = 0$. Tampoco es difícil ver a partir de la descomposión SVD que $\|\lambda X\|_* = |\lambda| \|X\|_*$, para $\lambda \in \mathbb{R}$. La subaditividad es, en principio, menos directa, pero con la representación dada por el Teorema (5.1) es claro que

$$\|X + Y\|_* \leq \|X\|_* + \|Y\|_*,$$

con lo que comprobamos que $\|\cdot\|_*$ es una norma. En particular es una función convexa de X. El Teorema (5.1) nos está diciendo también que la norma nuclear es la norma dual asociada a la norma espectral. Esta relación entre norma nuclear y norma espectral es simétrica, es decir, la norma espectral también satisface

$$\sigma(Y) = \max_{X \in \mathcal{M}_{m\times n}(\mathbb{R}):\|X\|_*=1} \operatorname{Tr}(Y^T X) = \max_{X \in \mathcal{M}_{m\times n}(\mathbb{R}):\|X\|_*\leq 1} \operatorname{Tr}(Y^T X). \quad (69)$$

Para comprobarlo, observamos que el Teorema 5.1 nos dice que

$$\operatorname{Tr}(Y^T X) \leq \|X\|_* \sigma(Y)$$

para cada par de matrices X e Y, por lo que

$$\operatorname{Tr}(Y^T X) \leq \sigma(Y)$$

para cada X con $\|X\|_* = 1$ (por lo tanto también para cada X con $\|X\|_* \leq 1$). Si $Y = UDV^T$ se ve fácilmente que la igualdad se alcanza tomando $X = uv^T$ si u y v son las primeras columnas de U y V, respectivamente. Esto demuestra (69).

Para el uso de algoritmos de optimización convexa en en problemas en los que intervenga la norma nuclear es necesario conocer de forma apropiada sus propiedades de diferenciabilidad o subdiferenciabilidad. Este es el contenido del siguiente Teorema (Theorem 2, Example 2 en [25]).

Teorema 5.2 *Si $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ tiene rango r y descomposición de valor singular reducida $X = UDV^T$, con $U \in \mathcal{M}_{m\times r}(\mathbb{R})$ tal que $U^TU = I_r$, $V \in \mathcal{M}_{n\times r}(\mathbb{R})$ tal que $V^TV = I_r$ y $D = \mathrm{diag}(d_1(X), \ldots, d_r(X))$ con $d_1(X) \geq \ldots \geq d_r(X) > 0$ entonces*

$$\partial\|X\|_* \quad = \quad \{UV^T + W : U^TW = 0,\, WV = 0,\, \sigma(W) \leq 1,\}.$$

Con ayuda del Teorema 5.2 podemos, en principio, plantearnos el diseño de algoritmos eficientes para el cálculo aproximado de

$$\hat{X} = \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R})}{\mathrm{argmin}} \frac{1}{2}\|X - L\|_F^2 + \lambda\|L\|_*. \tag{70}$$

Vamos a ver que, en realidad, este problema admite una solución explícita en términos de la descomposición de valor singular. La función objetivo en (70) es de la forma $f = g + h$ con g convexa y diferenciable (la parte correspondiente a la norma de Frobenius) y h convexa (el término asociado a la norma nuclear). Por el Teorema 5.2 el subgradiente de f en L está dado por

$$\partial f(L)_* \quad = \quad \{L - X + \lambda(\tilde{U}\tilde{V}^T + \tilde{W}) : \tilde{U}^T\tilde{W} = 0,\, \tilde{W}\tilde{V} = 0,\, \sigma(\tilde{W}) \leq 1,\}$$

si $L = \tilde{U}\tilde{D}\tilde{V}^T$ es la descomposición de valor singular reducida de L. La matriz $\hat{X}$ es un minimizador de f si y sólo si $0 \in \partial f(\hat{X})$, es decir, si y sólo si

$$L = X - \lambda(\tilde{U}\tilde{V}^T + \tilde{W}) \tag{71}$$

con $L = \tilde{U}\tilde{D}\tilde{V}^T$ y $\tilde{W}$ como antes. Supongamos $X = UDV^T$ con $d_j(X) > \lambda$ para $j = 1, \ldots, r_0$, $d_j(X) \leq \lambda$, $j = r_0+1, \ldots, r$. Denotamos por U_1 la matriz formada por las primeras r_0 columnas de U, U_2 para la formada por las siguiente $r - r_0$ y de forma similar V_1 y V_2. Tomamos

$$\begin{aligned}
\tilde{W} &= \frac{1}{\lambda}U_2\mathrm{diag}(d_{r_0+1}(X), \ldots, d_r(X))V_2^T, \\
L &= U_1\mathrm{diag}((d_1 - \lambda)(X), \ldots, (d_{r_0}(X) - \lambda))V_1^T, \\
\tilde{U} &= U_1, \quad \tilde{V} = V_2.
\end{aligned}$$

Con esta elección se tiene (71), lo que demuestra que esta elección de L es una solución del problema (70). Recogemos este hecho en el siguiente enunciado.

Teorema 5.3 *Si $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ tiene descomposición de valor singular $X = UDV^T$ con $D = \mathrm{diag}(d_1, \ldots, d_r)$ y*

$$\hat{X} := U\mathrm{diag}(\mathcal{S}_\lambda(d_1), \ldots, \mathcal{S}_\lambda(d_r))V^T,$$

donde $\mathcal{S}_\lambda$ denota el operador soft-thresholding, entonces $\hat{X}$ es un minimizador de

$$\frac{1}{2}\|X - L\|_F^2 + \lambda\|L\|_*$$

en el conjunto de todas las matrices $L \in \mathcal{M}_{m\times n}(\mathbb{R})$.

Vemos de esta forma que la regularización por norma nuclear convierte el problema no convexo

$$\hat{X}_k = \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R}):\,\mathrm{rank}(L)\leq k}{\operatorname{argmin}} \|X - L\|_F^2.$$

en el problema convexo (70). Este problema regularizado tiene un caracter parecido. La solución óptima en ambos casos es una aproximación de rango bajo a X. En el problema (70) el rango de la solución depende de λ. Valores grandes de λ conducen a la anulación de más valores singulares y por lo tanto a una aproximación por una matriz de rango menor.

Volviendo al objetivo inicial de esta sección podemos ahora sustituir el problema (68) por la versión regularizada

$$\min_{L\in\mathcal{M}_{m\times n}(\mathbb{R})} \frac{1}{2}\|X - L\|_{F,\Omega}^2 + \lambda\|L\|_*. \tag{72}$$

Usando la notación $\mathcal{P}_\Omega(X)$ para la matriz que tiene las mismas entradas que X en las posiciones de Ω y 0 en el resto, la función objetivo en (72) se reescribe $f(L) = \frac{1}{2}\|\mathcal{P}_\Omega(X) - \mathcal{P}_\Omega(L)\|_F^2 + \lambda\|L\|_* =: g(L) + h(L)$, de nuevo con g convexa y diferenciable y h convexa (y fácil de manejar). El gradiente de g es

$$\nabla g(L) = \mathcal{P}_\Omega(L) - \mathcal{P}_\Omega(X).$$

Claramente este gradiente es una función de Lipschitz (con constante 1 si tomamos la norma de Frobenius).

El algoritmo de descenso de gradiente proximal parte de un iterante inicial, X_0, y en etapas sucesivas calcula

$$X_{n+1} = \mathrm{prox}_{sh}(X_n - s\nabla g(X_n)),$$

donde $s > 0$ es la *amplitud de paso* y prox_h, el operador proximal, asociado a h, está dado por

$$\mathrm{prox}_h(Y) = \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R})}{\operatorname{argmin}} \Big[\frac{1}{2}\|Y - L\|_F^2 + \lambda\|L\|_*\Big].$$

Si tomamos $s = 1$ tendremos que

$$L - \nabla g(L) = \mathcal{P}_{\Omega^C}(L) + \mathcal{P}_\Omega(X)$$

por lo tanto

$$\begin{aligned}\mathrm{prox}_h(X_n - \nabla g(X_n)) &= \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R})}{\operatorname{argmin}} \Big[\frac{1}{2}\|X_n - \nabla g(X_n) - L\|_F^2 + \lambda\|L\|_*\Big] \\ &= \underset{L\in\mathcal{M}_{m\times n}(\mathbb{R})}{\operatorname{argmin}} \Big[\frac{1}{2}\|\mathcal{P}_\Omega(X_n) + \mathcal{P}_{\Omega^C}(X) - L\|_F^2 + \lambda\|L\|_*\Big].\end{aligned}$$

Esto, unido al Teorema 5.3, demuestra que el algoritmo de descenso proximal aplicado al problema (72) está asociado a la iteración

$$X_{n+1} = U_n S_\lambda(D_n) V_n^T, \tag{73}$$

donde $U_n D_n V_n^T$ es la descomposición de valor singular asociada a $\tilde{X}_n = \mathcal{P}_\Omega(X_n) + \mathcal{P}_{\Omega^C}(X)$ y $S_\lambda(D_n)$ es la matriz diagonal que se obtiene aplicando el operador $\mathcal{S}_\lambda$ a cada uno de los elementos en la matriz diagonal D_n. Esta es precisamente la iteración principal en el algoritmo `SoftImpute`. Como consecuencia de esta última discusión obtenemos un resultado de convergencia para el algoritmo SoftImpute.

Teorema 5.4 *Si* $f(L) = \frac{1}{2}\|\mathcal{P}_\Omega(X) - \mathcal{P}_\Omega(L)\|_F^2 + \lambda\|L\|_*$, X_{opt} *es un minimizador de* f *y* X_n *una sucesión de matrices obtenidas según la iteración* (73) *entonces existe una constante* $C > 0$ *tal que*

$$f(X_n) - f(X_{\text{opt}}) \leq \frac{C}{n+1}\|X_1 - X_{\text{opt}}\|_F^2.$$

Demostración. Puesto que la iteración (73) es la iteración de descenso de gradiente proximal para una función 1-Lipschitz, el resultado es un caso particular del Teorema de Nesterov (Teorema 4.1). □

Algoritmo: SOFT-IMPUTE para el completado de matrices.

Paso 1 Inicializamos $\mathbf{Z}^{\text{old}} = \mathbf{0}$ y creamos la sucesión decreciente $\lambda_1 > \ldots > \lambda_K$

Paso 2 Para cada $k = 1, \ldots, K$, set $\lambda = \lambda_k$ e iteramos hasta llegar a la convergencia:

(a) Calculamos $\widehat{\mathbf{Z}}_\lambda \leftarrow \mathcal{S}_\lambda\left(P_\Omega(\mathbf{Z}) + P_\Omega^\perp\left(\mathbf{Z}^{\text{old}}\right)\right)$

(b) Actualizamos $\mathbf{Z}^{\text{old}} \leftarrow \widehat{\mathbf{Z}}_\lambda$

Paso 3 Escribimos la secuencia de soluciones $\widehat{\mathbf{Z}}_{\lambda_1}, \ldots, \widehat{\mathbf{Z}}_{\lambda_K}$

El algoritmo `SoftImpute` fue propuesto en [24], planteado como un tipo de algoritmo EM. En este trabajo se demuestra que la iteración (73) es de descenso (es decir, que la función objetivo disminuye a cada paso), algo que no se ha probado aquí. También se da un resultado de convergencia parecido al Teorema (5.4), con una demostración diferente. Que la iteración (73) es un caso de descenso de gradiente proximal y que por lo tanto se puede aplicar el resultado general de Nesterov es algo observado en [23], con demostración propuesta a través de una serie de ejercicios. La versión dada aquí es una solución a esa colección de ejercicios.

Acabamos de regularizar nuestro problema para hacerlo más robusto frente a observaciones corruptas; sin embargo, el problema que estamos resolviendo no es el problema inicial de completado de la matriz sujeta a rango, sino su componente convexa. Por tanto, parece obvio preguntarse cuándo la solución de nuestro problema relajado (68) vale también para nuestro problema original (**??**). Esto es de lo que hablaremos en lel siguiente apartado, de la completabilidad de las matrices.

5.1.1. Completabilidad de la matriz

Con el problema convexo (72) sabemos que llegamos a una solución única, pero para que esta solución sea la misma que la de nuestro problema inicial (68), necesitamos ciertas garantías. Si la matriz a completar es de rango bajo, entonces la aproximación será bastante precisa: por ejemplo, si tenemos una matriz de cinco filas donde todas son múltiplos de la primera, vemos que aunque nos falten valores en alguna fila estos son simplemente un múltiplo de los de otras filas. Por tanto, matrices de bajo rango serán más fáciles de completar.

Con respecto al número de entradas, también necesitamos ciertas garantías. Supongamos que tenemos una muestra de tamaño N que rellena de manera uniformemente aleatoria una matriz $p \times p$. Es obvio que el completado es posible cuando $N \geq p^2$, sin embargo nuestro objetivo es ver cuando podemos garantizar la recuperación de nuestra matriz en el caso donde $N \ll p^2$. Si el tamaño de nuestra muestra es muy pequeña y p muy grande entonces no la podremos completar por muy eficientes que sean los algoritmos; si $|N|$ es 1 y la matriz tiene un millón de entradas, vemos que es imposible. Por tanto nos preguntamos, ¿Cúal es el mínimo valor que debería tomar N, en función de la dimensión p y del rango r de la matriz, para que la norma de relajación nuclear (72) sea capaz de recuperar la matriz de manera exacta?

Empecemos por el caso más simple, supongamos que no hay ruido. Una primera observación que podemos hacer es que si una fila o columna está completamente vacía, recuperar la matriz de manera exacta es tarea imposible, incluso aunque sea de rango uno. Es necesario que haya al menos una observación por fila (o columna). Este fenómeno es conocido como el *Problema del coleccionista de cupones* solo que en lugar de determinar el número medio de estampas para completar una colección formada por N estampas diferentes, lo llamamos el número medio de observaciones para que haya al menos una observación en cada fila N diferente.

Pero también tenemos que poner restricciones al rango; si todas las entradas de la matriz no vacías se concentran en unas pocas líneas entonces el completado también es imposible. Necesitamos que tenga $\mathcal{O}(rp)$ parámetros para especificar una matriz arbitraria $p \times p$ de rango r, puesto que tiene $\mathcal{O}(r)$ vectores singulares, cada uno con p componentes. Bajo ciertas restricciones de"coherencia"de la matriz que definiremos ahora, la regularización de la norma nuclear consigue recuperar la matriz con un tamaño de la muestra de solo un factor logarítmico mayor.

Introducimos ahora la coherencia de una matriz; este término define cómo

están de alineados los vectores de los valores singulares con respecto a la base estándar. Para ver la importancia que tienen estas restricciones coherentes, consideremos las dos matrices de rango uno: $\mathbf{X} = \mathbf{e_1 e_1^T}$, con un único uno en la esquina superior derecha y $\mathbf{X'} = \mathbf{e_1 v^T}$ donde $\mathbf{v} \in \Re^p$ es un vector p arbitrario.

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \; y \; \mathbf{X'} = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{74}$$

Tanto la matriz $\mathbf{X}$ como $\mathbf{X'}$ tienen máxima coherencia con la base estándar de $\Re^4$, lo que significa que algún subconjunto de sus vectores singulares tanto de la derecha como de la izquierda está perfectamente alineado con cierto vector de la base estándar. Analicemos la matriz $\mathbf{X}$. Si observamos únicamente $N \ll p^2$ entradas de esta matriz, con dichas N observaciones elegidas de manera aleatoria, entonces con probabilidad muy alta no observaremos el único valor distinto de cero y por tanto no la distinguiremos de la matriz nula. Lo mismo ocurre con la matriz $\mathbf{X'}$. En este caso $\mathbf{X'} = \mathbf{e}_1 \mathbf{v}^T$, donde $\mathbf{v} \in \Re^p$ es un vector de tamaño p arbitrario. Por tanto, para garantizar la eficiencia de la regularización con la norma nuclear tenemos que evitar estos casos patológicos.

Una manera de excluir estas matrices problemáticas es la de crear matrices a partir de un conjunto aleatorio; por ejemplo, podemos contruir una matriz aleatoria de la forma $X = \sum_{j=1}^{r} \mathbf{a_j b_j^T}$ donde los vectores $\mathbf{a}_j \sim N(0, I_p)$ y $\mathbf{b}_j \sim N(0, I_p)$ son independientes. Tales matrices aleatorias presentan poca probabilidad de poseer vectores singulares que sean coherentes con los vectores de la base estándar. Para este ejemplo, Gross (2011, ver [26]) mostró que garantizando la aleatoriedad del conjunto y de la muestra, la norma de regularización nuclear garantiza la recuperación exacta si el tamaño de la muestra cumple

$$N \leq Crp \log p, \tag{75}$$

donde $C > 0$ es una constante universal fijada. Es posible dar más garantías para la recuperación exacta de la matriz cuando el factor C depende en la incoherencia de los vectores singulares. En los artículos de de Candes y Retch (2009) [27], Gross (2011) [26] y Retch (2011) [28] se detallan estas garnatías.

Nos movemos ahora al caso en el que las entradas observadas de la matriz presentan ruido; ¿Qué cantidad N de observaciones con respecto a la dimensión de la matriz deberíamos tener para poder completar la matriz? Pues bien, recuperar la matriz "exacta" es prácticamente imposible, así que es más razonable suponer que las entradas observadas tienen cierto ruido adicional, $X = L^* + W$, donde L^* tiene rango r. En esta situación, aunque el completado exacto no es posible, sí que podría interesarnos cúanto podemos acercarnos a esta matriz L^* de bajo rango. Para ello utilizaremos la descomposición (70). Las condiciones de incoherencia para los vectores singulares son los menos apropiados para observaciones ruidosas, dado que no son robustas frente a pequeñas perturbaciones. Veamos un ejemplo, supongamos que empezamos con una matriz B que tiene rango $r-1$, con norma de Frobenius uno, y que es *maximalmente incoherente*, es decir, que todos los vectores singulares son ortogonales a los de la base estándar. Recordando las matrices problemáticas (74), consideremos ahora $L^* = B + \delta \mathbf{X}$

para cierto $\delta > 0$. La matriz L^* siempre tendrá rango r, y no importa lo pequeño que sea el parámetro δ, siempre será maximalmente incoherente puesto que uno de sus vectores singulares es el vector de la base estándar $\mathbf{e_1} \in \Re^{\mathbf{p}}$. Por tanto la incoherencia de la matriz no nos vale puesto que es demasiado sensible a pequeñas perturbaciones.

Un criterio que nos podría valer está basado en la uniformidad del ratio de la matriz (Negahban y Wainwright 2012, [29]); para cualquier matriz no nula $L \in \Re^{p \times p}$, definimos el ratio de la matriz $\alpha_{sp}(L) = \frac{p\|L\|_\infty}{\|L\|_F}$, donde $\|L\|_\infty$ es el elemento con mayor valor absoluto de las entradas de la matriz. Este ratio es una medida de la uniformidad en la que se encuentran las entradas por toda la matriz; varía entre 1 y p. Por ejemplo, cualquier matrix L con todas las entradas iguales tiene $\alpha_{sp}(L) = 1$, el valor más pequeño posible, mientras que la matriz más irregular posible es la matriz X de la ecuación (74) que obtiene el máximo ratio $\alpha_{sp}(L) = p$. Al contrario que la incoherencia de los vectores singulares, el ratio de uniformidad involucra tanto a los vectores como a los valores singulares. Por tanto $L^* = B + \delta X$ tendrá un ratio de uniformidad bajo siempre que la perturbación $\delta > 0$ sea lo suficientemente pequeña. En el caso del estimador (70) que utiliza la norma muclear como regularizador, si le añadimos una restricción al ratio de uniformidad, Negahban y Wainwright demostraron que dicho estimador satisaface la siguiente cota

$$\frac{\|\hat{L} - L^*\|_F^2}{\|L^*\|_F^2} \leq C \,\text{máx}\{\sigma^2, \alpha_{sp}^2(L^*)\} \frac{rp \log p}{N} \tag{76}$$

5.1.2. Margen máximo de factorización y métodos relacionados

Estamos ante el problema de completado de la matriz, en el contexto del filtro colaborativo. Es decir, tenemos una matriz de puntuaciones de usuarios a productos que queremos completar. Los algoritmos de completado de la matriz suponen que hay pocos factores que determinan las preferencias de los usuarios. Luego buscan una aproximación de rango bajo; con la regularización de norma nuclear tratamos con un problema convexo y limitando el número de valores singulares que puede tener nuestra matriz de aproximación estamos limitando el número de factores que van a determinar las preferencias de los usuarios. Los algoritmos de aproximación de rango bajo de la matriz restringen entonces la dimensión.

En busca de otras restricciones para completar mejor la matriz se optó por restringir no los factores pero sí la variabilidad explicada o la no negatividad, como sugirieron Lee y Seung (1999) (véase [30]). Pero esto derivaba en problemas de optimización no convexos.

No fue hasta 2005 cuando Rennie y Srebo (véase [31]) dieron con otra solución. En lugar de restringir la dimensión, propusieron restringir la norma. Si lo miramos desde el punto de vista de filtro colaborativo significa que no se reducen el número de factores, sino su fuerza; puede haber un enorme número de factores que determinen las preferencias de los usuarios pero solo deja que algunos de ellos sean importantes. Por ejemplo, en el caso de la matriz de Netflix, puede

haber un factor importante que es el de la violencia en las películas (ya que se considera más determinante), y otros factores de menor relevancia que se corresponden al género dramático o la comedia. Los procedimientos que siguen esta idea se conocen como *métodos de factorización de margen máximo de matrices* (MMMF), y, aunque con alguna modificación, siguen el siguiente modelo para aproximar la matriz X: consideramos una matriz de factorización con la forma $M = AB^T$, donde A y B son de dimensión $m \times r$ y $n \times r$, respectivamente. Se trata entonces de resolver el problema de optimización

$$\underset{\substack{A\in\Re^{n\times r}\\ B\in\Re^{m\times r}}}{minimizar} \|\mathcal{P}_\Omega(X) - \mathcal{P}_\Omega(AB^T)\|_F^2 + \lambda(\|A\|_F^2 + \|B\|_F^2) \tag{77}$$

Este problema no es convexo en A y en B, pero sí biconvexo. Para poder resolverlo se utilizan los algoritmos de minimización alternados que vimos en la sección anterior. Si fijamos A, y buscamos resolver (77) para B, se puede ver que el problema se descompone en n regresiones de Ridge separadas, con cada columna X_j de Z como vector respuesta y las r-columnas de como predictores. Dado que algunas columnas de X están vacías, se ignoran y las correspondientes filas de A se eliminan de la j-ésima regresión. Por tanto estamos ante regresiones de Ridge completamente separadas y con matrices de regresión distintas, aunque deriven todas de A. Por simetría, fijamos B y resolvemos A para m regresiones de Rigde diferentes.

La solución no tiene por qué ser única. Vemos que la restricción depende de la norma de Frobenius, la cual es invariante a multiplicaciones por matrices unitarias. Luego si multiplicamos A o B por cualquier matriz unitaria también nos vale. De hecho, gracias a esta invariabilidad podemos ver la relación que existe entre MMMF y el completado de la norma nuclear

$$\|M\|_\star = \min_{\substack{A\in\Re^{n\times r}, B\in\Re^{m\times r}\\ M=AB^T}} \frac{1}{2}\big(\|A\|_F^2 + \|B\|_F^2\big) \tag{78}$$

Para ello, tomamos la descomposición en valores singulares de $M = UDV^T$ y multiplicamos a la izquierda y la derecha del problema por U^T y V respectivamente y nos quedamos con la matriz diagonal. El problema queda entonces como

$$\|D\|_\star = \min_{\substack{A\in\Re^{n\times r}, B\in\Re^{m\times r}\\ D=AB^T}} \frac{1}{2}\big(\|A\|_F^2 + \|B\|_F^2\big) \tag{79}$$

Buscamos entonces dos matrices A y B diagonales (ya que si presentan valores fuera de la diagonal solo haría aumentar la norma de Frobenius) y de rango como mucho r. Reescribiendo tenemos que

$$\begin{aligned}\frac{1}{2}\big(\|A\|_F^2 + \|B\|_F^2\big) &= \frac{1}{2}\big(\text{traza}(A'A) + \text{traza}(B'B)\big)\\ &= \frac{1}{2}\left(\sum_{i=1}^{r}\sigma_{\mathcal{A}i}^2 + \sum_{i=1}^{r}\sigma_{\mathcal{B}i}^2\right)\end{aligned}$$

Con esta reescritura, sujeta a reestricciones tenemos que la solución al problema de minimización será

$$\widehat{A} = \mathcal{S}_\lambda (D_r)^{\frac{1}{2}}$$
$$\widehat{B} = \mathcal{S}_\lambda (D_r)^{\frac{1}{2}}$$

Esta realción entre los dos problemas implica que la familia de soluciones $\hat{M} = \hat{A}\hat{B}^T$ de los problemas biconvexos (77) para $r \leq \text{mín}(m, n)$ son los mismos que para aquellas de la famlia de problemas convexos de (72). Para ser más específicos, tenemos el siguiente teorema:

Teorema 5.5 *Sea X una matriz $m \times n$ que tiene como entradas observadas las indexadas por el subconjunto Ω:*

1. *Las soluciones de MMMF (78) con $r = min\{m, n\}$ coinciden para todo $\lambda \geq 0$ con las soluciones de la regularización de norma nuclear.*

2. *Para cierto $\lambda^* > 0$ fijado, supongamos que la función objetivo (72) tiene como solución óptima una matriz de rango r^*. Entonces, para cualquier solución óptima $(\hat{A}, \hat{B})$ del problema (77) con $r \leq r^*$ y $\lambda = \lambda^*$, la matriz $\hat{M} = \hat{A}\hat{B}^T$ es la solución óptima del problema (72). Por tanto, el espacio de soluciones de (72) está contenido en el de (77).*

Los métodos MMMF definen una familia bidimensional de modelos indexados por el par (r, λ) , mientras que el criterio `Soft-Impute` (72) define una familia unidimensional. Gracias al Teorema 1 vemos que esta última familia es un subconjunto de la bidimensional, de hecho marca un camino en la red de soluciones del par $\left(\hat{A}_{(r,\lambda)}, \hat{B}_{(r,\lambda)}\right)$.

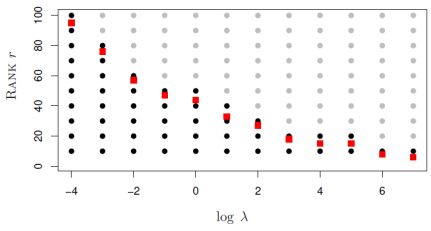


Figura 11: MMMF y `Soft-Impute`

Cualquier modelo MMMF con una combinación de parámetros por encima de los puntos rojos es redundante, puesto que ajusta igual que los situados en los puntos rojos. Sin embargo, en la práctica, los MMMF no conocen estos puntos rojos ni el rango donde se da la solución óptima. Sería necesaria una ortogonalización de $\hat{A}$ y de $\hat{B}$ para revelar el rango, e incluso en este caso sería una aproximación (dependiendo de el criterio de convergencia del algoritmo MMMF). En resumen, la formulación (72) es preferible por dos razones: es convexa y lleva a cabo al mismo tiempo la reducción de rango y la regularización. En cambio, usando (77) necesitamos elegir el rango de la aproximación y el parámetro de la regularización.

Otro enfoque relacionado es el propuesto por Keshavan et al. en 2010 (ver [32]), que sigue la siguiente ecuación

$$\|\mathcal{P}_\Omega(X) - \mathcal{P}_\Omega(USV^T)\|_F^2 + \lambda\|S\|_F^2 \tag{80}$$

Se minimiza sobre el triplete (U, V, S), donde $U^TU = V^TV = I_r$ y S es una matriz $r \times r$. Fijando el rango r, para minimizar el criterio (80) se utiliza el descenso de gradiente. Este criterio es similar a MMMF (77) pero sobre la descomposición en valores singulares, añadiendo una restricción de ortonormalidad a las matrices U, V, de esta manera la señal y correspondiente regularización se aplican a la matriz S. Al igual que en MMMF, el problema es no convexo por lo que el descenso de gradiente no asegura la convergencia al global óptimo; además, tiene que ser resuelto de manera independiente cada vez que cambiamos el rango r.

En Keshavan et al. (2010) [32] también se describe la teoría asintótica del estimador (80) cuando se aplica al completado de una matriz con ruido, usando una escala en el que la razón m/n converge a una constante $\alpha \in (0, 1)$. No vamos a entrar en detalles sobre esta convergencia pero en líneas generales toma una matriz X de dimensión $m \times n$ que puede ser escrita como suma de la forma $X = U\Sigma V + W$, donde $\Sigma \in \Re^{r\times r}$ es una matriz diagonal. Aquí el término W es una matriz aleatoria con entradas i.i.d., cada una de media cero y varianza $\sigma^2\sqrt{mn}$. Asumimos que cada entrada de la matriz X es observada de manera independiente con probabilidad ρ. Sea $\hat{X}$ el estimador obtenido al minimizar el criterio (80) usando el valor óptimo para λ. En esta situación, el error relativo $\|\hat{X} - X\|_F^2/\|X\|_F^2$ converge en probabilidad a $1 - c(\rho)$ cuando $m/n \to \alpha \in (0, 1)$. La contante $c(\rho)$ es cero si $\sigma^2/\rho \geq \max_{jj} \Sigma_{jj}$ y distinto de cero sino. Esto muestra que el estimador es sometido a una fase de transición: si el ruido y la probabilidad de falta de datos son bajos en comparación con la intesidad de señal, entonces los valores faltantes pueden ser recuperados con éxito. En caso contrario, el estimador es prácticamente inútil a la hora de recuperar los datos que faltan. Más detalles sobre la convergencia y su demostración pueden ser encontrados en Keshavan et al. 2009 [33] y en Keshavan et al. 2010, [32].

5.1.3. Desafío Netflix

Los sistemas de recomendación tienen un papel crucial en el comercio electrónico y en los servicios online como Amazon, Youtube o Netflix. Haciendo buenas recomendaciones al usuario de productos, música o películas consigues que se quede y que consuma, lo que se traduce en aumento en ventas y ganancias. Compañías que compiten por la fidelidad de sus clientes invierten enormes cantidades en sistemas capaces de atraer, analizar a sus usuarios, y ofrecerles productos que probablemente compren. El objetivo primordial de estos sistemas de recomendación es ayudar a los usuarios a encontrar lo que buscan basándose en sus preferencias y sus interacciones previas, y predecir la puntuación que daría el usuario a un nuevo producto. Y aquí es donde entra en juego todo lo que hemos hablado sobre el completado de matrices en las secciones anteriores; ningún

usuario se ha visto todas las películas, escuchado todas las canciones y obviamente no se habrá comprado todos los productos, por lo que estas compañías están ante una matriz incompleta. La base de estos sistemas de recomendación está en el completado de matrices. Veamos un caso práctico de cómo aplicar nuestro completado de matrices: **El Desafío Netflix**.

Éste es uno de los principales ejemplos sobre el completado de matrices, descrito en Bennet and Lanning (2007) [34]. Netflix en un principio era una compañía de aquiler de películas online que con el objetivo de mejorar su sistema de recomendación lanzó una competición en 2006 con el premio de 1 millón de dólares para el ganador. El dataset de Netflix estaba organizado en $n = 17{,}770$ películas (columnas) y $m = 480{,}189$ clientes (filas). Los clientes habían valorado algunas películas siguiendo una escala del 1 al 5, donde uno es el peor y cinco el mejor. La matriz de este conjunto de datos era muy dispersa; contaba con 100 millones de puntuaciones que aunque parezcan muchas, solo representan un 1 % del total.

El objetivo era predecir las valoraciones de películas sin puntuación, para así mejorar el sistema de recomendación. En 2006, Netflix contaba con un algoritmo de *Cinematch* que tenía un error cuadrático medio del 0.9525 sobre el *test set*. La competición la ganaría el primer algoritmo que consiguiese mejorar el RMSE en al menos un 10 %. Un millón de dólares estaban en juego, investigadores de diferentes países participaron. Tres años se tardó en que apareciese un ganador; en 2009 un grupo de investigadores bajo el nombre de "Bellkor's Pragmatic Chaos" lo consiguió. El algoritmo ganador usaba una combinación de técnicas estadísticas, pero como muchos otros algoritmos de la competición, la SVD jugaba un papel protagonista. Un modelo de bajo rango proporciona un buen método para evaluar las películas; en particular, supongamos que queremos predecir la puntuación del usuario i para la película j con el modelo:

$$z_{ij} = \sum_{l=1}^{r} c_{ij} g_{jl} + w_{ij}, \tag{81}$$

o escrito en forma matricial $Z = CG^T + W$, donde $C \in \Re^{m \times r}$ y $G \in \Re^n$. Vemos que hay r tipos de películas y a cada una le corresponde un grupo (*click*) de gente que le gusta. Aquí cada espectador i tiene un peso c_{il} para el grupo l, y el género asociado a ese grupo tiene una puntuación g_{jl} para la película j. La media de la puntuación se obtiene sumando estos productos sobre los r géneros y después se añade el ruido w_{ij}.

Esta es la base, sin embargo, con esta simple clasificación no se ganó el millón de euros; hay que tener en cuenta posibles *outliers*, datos corruptos, películas que jamás han sido votadas, la tendencia de cada usuario a votar a la alta o a la baja... Vemos que el completar la matriz se va complicando a medida que vemos los factores que pueden afectarla. Así que vamos a crear nuestro propio sistema de recomendación, desde el principio, y viendo cómo se podrían ir solventando los diferentes problemas que nos aparecen.

Los datos de Netflix no están disponibles, pero el laboratorio GroupLens generó un conjunto de datos con más de 20 millones de valoraciones sobre 27.000

películas y 138.000 usuarios. Nosotros usaremos un subconjunto de este con el fin de no saturar el ordenador; 100.000 puntuaciones sobre más de 1.600 películas y casi 1.000 usuarios. Cada usuario ha votado al menos 20 películas y cada película ha sido valorada al menos una vez.

Empezaremos analizando las variables, creando gráficos, tablas, estadístcas... con el fin de entender mejor nuestro conjunto datos y ver qué características son importantes a la hora de predecir las puntuaciones. De hecho la base de los sistemas de recomendación está en identificar cuáles son los regresores que mejor explican nuestra variable respuesta.

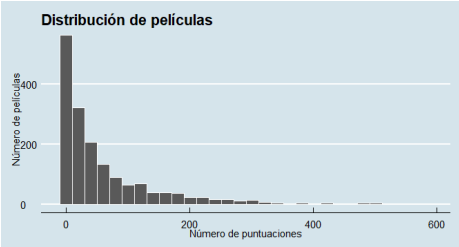


Figura 12: Distribución de películas

Si analizamos las puntuaciones vemos que hay una pequeña diferencia con respecto a Netflix y es que las películas pueden tener medias estrellas, es decir entre 0.5 hasta 5.0, con un total de 10 posibles valores. Las películas de nuestro dataset no han sido evaluadas el mismo número de veces; es obvio que los llamados *taquillazos* tengan más puntuaciones que las menos conocidas. Ésto nos muestra la figura (12) que representa un histograma del número de votos que tiene cada película.

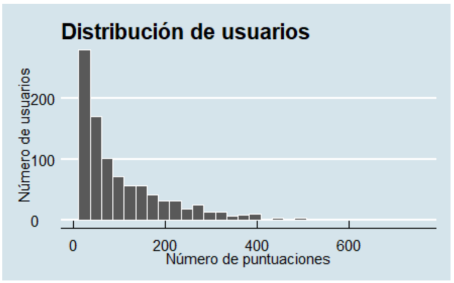


Figura 13: Distribución de usuarios

Hacemos lo mismo con los usuarios y vemos que los usuarios incluidos han valorado al menos 20 películas.

Veamos la matriz usuario vs película. Se trata de una matriz dispersa, imcompleta prácticamente. Se observa que hay varias películas que han sido votadas más veces y que algún usuario es más activo a la hora de votar. Esta es la matriz que tenemos que completar.

Dividimos nuestro dataset en dos subconjuntos: el de entrenamiento, *training set*, en el que probaremos nuestro modelos y el *test set*, para hacer la prueba final y ver que los buenos resultados no son fruto del sobreajuste. Para evitar que haya en el conjunto de prueba usuarios y películas que no están en nuestro conjunto de entrenamiento, utilizamos la función *semi-join()*. Esto permite eliminar el caso de las matrices (74) en las que había filas o columnas nulas.

Empezamos con un modelo lineal. Este modelo predice que todos los usuarios darán la misma puntuación a todas las películas y asume que la variación de película a película viene dada por un error aleatorio. La teoría estadística nos dice que el valor que menor error cuadrático medio da según este modelo es la

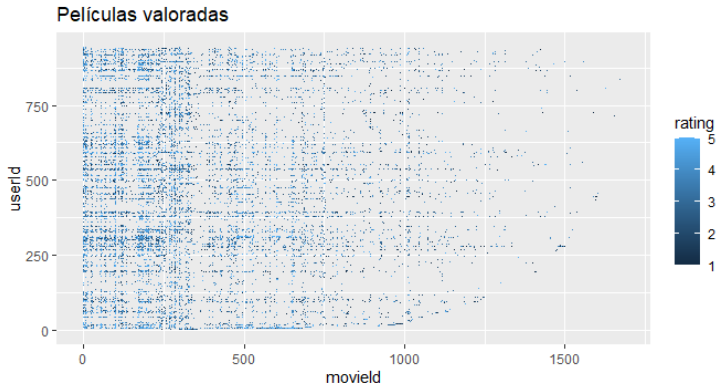


Figura 14: Matriz a completar

media muestral

$$\hat{Y}_{u,i} = \hat{\mu} + \epsilon_{i,u} \tag{82}$$

Donde $\hat{Y}_{u,i}$ es la variable respuesta (nuestra predicción), μ es la media muestral y $\epsilon_{i,u}$ el error de distribución.

Pero este modelo es demasiado sencillo, así que para mejorarlo tenemos que ver dónde se producen los mayores errores. Parte de la variabilidad entre las puntuaciones de las diferentes películas puede ser explicada por la actitud de los usuarios: puede haber dos usuarios a los que les gusta el mismo tipo de películas, en cambio, uno tiende a puntuar a la baja y otro a la alta, por lo que tres estrellas del primero es lo mismo que cuatro estrellas del segundo. Usuarios similares presentan entonces diferentes distribuciones o patrones. Este efecto de los usuarios se puede calcular con

$$\hat{b}_u = \frac{1}{N} \sum_{i=1}^{N} (y_{u,i} - \hat{\mu}) \tag{83}$$

El modelo de predicción que incluye este efecto del usuario se escribe como

$$\hat{Y}_{u,i} = \hat{\mu} + \hat{b}_u + \epsilon_{i,u} \tag{84}$$

Si lo implementamos en R, junto con la factorización `Soft-Impute` de la que hablaremos más adelante, vemos que no obtenemos un error por debajo del 0.90 para ningún rango de la matriz aproximada, así que tenemos que continuar mejorando.

De la misma manera que hemos calculado la tendencia de cada usuario, veamos la tendencia de las películas. Parte de la variabilidad entre las puntuaciones de las diferentes películas puede ser explicada con que dstintas películas presentan diferentes distribuciones a la hora de ser valoradas: solemos ser más optimistas a lahora de valorar *taquillazos*, las películas con actores como Brad

Pitt o Leonardo di Caprio suelen tener puntuaciones por encima de la media... Este efecto por película b_i se suma al modelo con la siguiente fórmula

$$\hat{Y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \epsilon_{i,u} \tag{85}$$

El efecto de cada película se puede calcular como la media entre la diferencia del valor observado y y la media μ y sin olvidarnos del efecto usuario:

$$\hat{b}_i = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{b}_i - \hat{\mu} \right) \tag{86}$$

El modelo lineal proporciona una buena estimación de las valoraciones, pero no es suficiente. El error sigue siendo elevado, así que intentemos buscar qué es lo que hace aumentar el RMSE. Al mirar las películas con mayor y menor puntuación vemos que han sido votadas muy pocas veces, en ocasiones una única vez, por lo que tenemos poca certeza sobre si estas valoraciones de tan poca genta vale para todos los usuarios. La incertidumbre hace aumentar el RMSE, por lo que no podemos fiarnos de estas entradas ruidosas; es mejor ser conservadores cuando no estamos seguros. Podríamos calcular desviaciones típicas para obtener un intervalo de confianza, pero cuando estamos ante una predicción no queremos un intervalo queremos un valor. Es por esto que incorporamos una regularización. La regularización nos permite penalizar estimaciones que se han obtenido a partir de muestras pequeñas. La idea que hay detrás de esta penalización es reducir la variabilidad sobre el efecto del tamaño de la muestra. Veamos un ejemplo. Supongamos que tenemos una película a con 100 puntuaciones de diferentes usuarios y una película b con una única puntuación. Intentamos ajustar el modelo (86). Suponemos que $\mu = 3$. Si utilizamos el error cuadrático medio, el estimador para la primera película b_a sobre los 100 usuarios sería $1/100 \sum_{i=1}^{100} (Y_{i,1} - \mu)$, que suponemos que es bastante preciso. Sin embargo, las estimaciones sobre la película b serán la diferencia de la media con respecto a la única puntuación que ha tenido dicha película, $\hat{b}_i = Y_{u,i} - \hat{\mu}$. Se trata de un estimador basado en una única puntuación por lo que no será para nada preciso y dará lugar a sobreajuste. De hecho, ignorando las películas que han sido puntuadas muy pocas veces y simplemente dar como valor de predicción la media de las películas ($b_i = 0$) es probable que tengamos una mejor estimación. La idea entonces de regresión penalizada aquí es controlar la variabilidad total de los efectos de las películas: $\sum_{i=1}^{5} b_i^2$. Específicamente, en lugar de minimizar la ecuación de mínimos cuadrados, minimizamos

$$\frac{1}{N} \sum_{u,i} \left(y_{u,i} - \mu - b_i \right)^2 + \lambda \sum_i b_i^2 \tag{87}$$

El primer término es igual, en cambio el segundo aumenta cuando los b_i aumentan. Si reescribimos esta ecuación obtenemos

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu}) \tag{88}$$

donde n_i es el número de puntuaciones hechas que ha tenido la película i. De esta manera, cuando el tamaño de nuestra muestra n_i es grande, la penalización λ es ignorada ya que $n_i + \lambda \approx n_i$. Sin embargo, cuando n_i es pequeño, la estimación $\hat{\beta}_i(\lambda)$ lo acerca a cero. Cuanto más grande , más lo acerca.

El mismo procedimiento se aplica a los usuarios. Si juntamos ambas penalizaciones tendremos

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right) \tag{89}$$

Para elegir el λ que mejor se ajusta a nuestro modelo hemos creado una función que calcula el error del modelo para cierta cantidad λ. Hacemos pasar esta función por un bucle para diferentes valores de λ y vemos que el que menor error cuadrático se consigue cuando $\lambda = 2{,}75$, como vemos en la figura (15). Una vez que tenemos los datos centrados y regularizados los hacemos pasar por la función `Soft-Impute` como hemos ido haciendo hasta ahora, y esta vez sí, obtenemos un error cuadrático medio de 0.9.

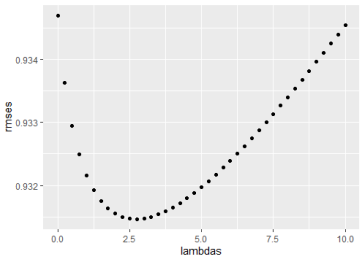


Figura 15: Elección de λ

Hemos ido mejorando paso por paso; primero hemos considerado el efecto usuario, luego incluido el efecto película y por último, hemos regularizado ambos efectos. De esta manera los datos han sido centrados y se ha reducido el peso de posibles *outliers* que aparecen por pocas valoraciones para poder calcular una media. Cada vez que íbamos mejorando los datos, los hacíamos pasar por una función llamada `soft-calculo` que a su vez llamaba a la función principal `Soft-Impute`. Esta última función pertenece a una librería con el mismo nombre, creada por Hastie y Mazumder (ver [35] para la descripción de la librería). Se trata de una librería que tiene como objetivo el completado de matrices utilizando la norma nuclear como regularizador; además de algoritmos iterativos para resolver este problema, esta librería también cuenta con funciones para calcular SVD de bajo rango o incluso para llevar a cabo la centralización que acabamos de hacer tanto de filas como de columnas (`biScale`). Esta última función no la hemos utilizado dado que el número de entradas vacías era demasiado elevado.

La función que hemos desarrollado `Soft-Calculo` tiene como objetivo calcular la descomposición en valores singulares de una matriz regularizada con la norma nuclear y calcular el error de las predicciones. Además se calcula el tiempo que tarda en llevar a cabo esta descomposición. La función recibe como argumentos

1. Los conjuntos de datos centralizados tanto de entrenamiento como de prueba

2. El rango máximo de SVD que permitimos

3. El valor de λ que es el parámetro de nuestro regularizador

4. El número máximo de iteraciones para evitar bucles infinitos

5. El tipo de algoritmo iterativo que usaremos para `Soft-Impute`

Lo primero que hace es transformar los datos de entrenamiento en una matriz. Para ello utilizamos la función `Incomplete` que viene incluida en la librería. Esta función toma como argumentos la fila i (usuario), la columna j (película) y la puntuación correspondiente. La matriz se representa como una matriz de clase dispersa, donde las entradas no observadas toman el valor cero. Una vez que tenemos nuestra, iniciamos el cronómetro y utilizamos la función `Soft-Impute` para completarla. Cuando termina, paramos el cronómetro e introducimos en el data-set el error cuadrático medio de las predicciones del conjunto prueba y los argumentos que le hemos pasado a la función para así poder compararlo después.

Una vez entendida la función `Soft-Calculo` que hemos creado, pasemos al verdadero corazón del programa: la función `Soft-Impute`. Esta función calcula una aproximación de bajo rango a la matriz incompleta anterior usando como regularizador la norma nuclear.

$$\text{minimizar} \frac{1}{2} \sum_{(i,j)\in\Omega} (X_{ij} - Z_{ij})^2 + \lambda \|Z\|_* \tag{90}$$

Los argumentos que toma esta función son muy similares a los de `Soft-Calculo:`

- **Matriz incompleta** que queremos aproximar

- **Rango máximo**: Este valor numérico restringe el rango de la solución, es decir, el número de valores singulares distintos de cero será como mucho este valor que pasamos por argumento. El valor máximo que puede tomar es mín$(dim(X)) - 1$. Si es lo suficientemente elevado, y usamos el tipo '*svd*, `Soft-Impute` resolverá el problema convexo. Si en cambio, el rango máximo exigido es demasiado bajo, no tenemos garantizada la solución. Aunque sí un mínimo local bastante exacto.

- **Landa** λ: Éste es el parámetro de la regularización. Si $\lambda = 0$, entonces el problema a minimizar carece de regularización, se convierte en el llamado algoritmo `Hard-Impute` que da lugar a sobreajuste, además de converger más lentamente a un mínimo local. El valor de λ ideal sería aquel en el que la solución de la matriz tuviese rango poco por debajo del máximo permitido.

- **Número máximo de iteraciones** para evitar entrar en bucles infinitos.

- **Tipo**: Hay dos tipos de algoritmos implementados: tipo *svd* o el tipo por defecto *als*. Ambos representan diferentes maneras de calcular a la solución. Veamos ambas más detenidamente.

El algoritmo *svd* hace referencia al algoritmo `Soft-Impute` del que hablábamos en la sección anterior. La solución de este problema viene dada por $\hat{Z} = \mathbf{S}_\lambda(Z)$ donde

$$\mathbf{S}_\lambda(Z) \equiv UD_\lambda V' \quad \text{con} \quad D_\lambda = \text{diag}\left[(d_1 - \lambda)_+, \ldots, (d_r - \lambda)_+\right]$$

UDV' es la SVD de W, $D = \text{diag}\left[d_1, \ldots, d_r\right]$, y $t_+ = \text{máx}(t, 0)$.

Vimos una manera de aprovechar la dispersión de la matriz para guardarla y poder operar con ella de manera eficiente. Además podíamos explotar la estructura de las componentes para hacer las multiplicaciones más rápidas tanto por la izquierda como por la derecha. El rango máximo que pasamos por argumento condiciona en cierta manera la salida del bucle; si el más pequeño de los valores singulares es menor que λ, entonces estamos ante la solución deseada. Este tipo de algoritmo funciona bastante bien, aunque en cada paso se tiene que calcular la SVD la cual es costosa. Para hacerlo más eficiente, aprovecha la solución de la anterior ronda del bucle como valor inicial para agilizar los cálculos. De hecho, a medida que nos acercamos a la solución, estos valores iniciales tienden a ser mejores, por lo que las últimas iteraciones suelen ser más rápidas.

Como vimos en el apartado anterior, Rennie y Srebo (2005) (ver [31]) propusieron el criterio biconvexo para resolver el completado de la matriz. El margen máximo de factorización descomponía el problema en n regresiones de Ridge separadas, donde cada columna X_j de X era el vector respuesta y las $r-$columnas de A los predictores. Y lo mismo con B. Hemos visto además la relación que hay entre ambas soluciones a través del Teorema 5.5. Y es de esta relación de donde nace el otro tipo de algoritmo de `Soft-Impute`: *als.* Este tipo mezcla pasos de ambos algoritmos para hacerse más eficiente. Recordemos el problema a minimizar en MMMF:

$$\underset{\substack{\mathbf{A}\in\Re^{n\times r}\\ \mathbf{B}\in\Re^{m\times r}}}{minimizar} \|\mathcal{P}_\Omega(\mathbf{Z}) - \mathcal{P}_\Omega(\mathbf{AB^T})\|_F^2 + \lambda(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2)$$

Con el objetivo de explotar la estructura y reducir los costes de almacenamiento de la matriz, reescribimos:

$$P_\Omega\left(X - AB^T\right) = P_\Omega(X) - P_\Omega\left(AB^T\right) = P_\Omega(X) + P_\Omega^\perp\left(AB^T\right) - AB^T \quad (91)$$

donde $A_{m\times r}$ y $B_{n\times r}$ tienen como rango máximo $r \leq \text{mín}(m, n)$. El algoritmo tiene la forma:

Algoritmo: SOFT-IMPUTE- ALS

Paso 1 Inicializar $A = UD$ donde $U_{m\times r}$ es una matriz aleatoria con columnas ortonormales, $D = I_r$, la matriz identidad $r \times r$, $B = VD$ con $V = 0$. También se podría haber cogido cualquier solución previa (A=U D and $B = VD$ como buen arranque.

Paso 2 Dadas $A = UD$ y $B = VD$, resolver

$$\text{minimizar}_{\widetilde{B}} \frac{1}{2} \left\| P_\Omega \left(X - A\widetilde{B}^T \right) \right\|_F^T + \frac{\lambda}{2} \|\widetilde{B}\|_F^2$$

para actualizar B. Conseguimos esto mediante los siguientes pasos:

a) Sea $X^* = \left(P_\Omega^\perp(X) - P_\Omega \left(AB^T \right) \right) + AB^T$, guardada como matriz dispersa más otra de rango bajo

b) Calcular

$$\text{minimizar}_{\widetilde{B}} \frac{1}{2} \left\| X^* - A\widetilde{B}^T \right\|_F^2 + \frac{\lambda}{2} \|\widetilde{B}\|_F^2$$

que tiene como solución:

$$\begin{aligned} \widetilde{B}^T &= \left(D^2 + \lambda I \right)^{-1} DU^T X^* \\ &= \left(D^2 + \lambda I \right)^{-1} DU^T \left(P_\Omega(X) - P_\Omega \left(AB^T \right) \right) \\ &\quad + \left(D^2 + \lambda I \right)^{-1} D^2 B^T \end{aligned}$$

c) Tomar esta solución para $\widetilde{B}$ y actualizar V y D :

1) Calcular $\tilde{B}D = \tilde{U}\tilde{D}^2\tilde{V}^T$
2) $V \leftarrow \tilde{U}$, y $D \leftarrow \tilde{D}$

Paso 3 Dada $B = VD$, resolver A. Por simetría, esto es equivalente al paso 2, con X^T reemplazando X, y B y A intercambiados.

Paso 4 Repetir los pasos (3)-(4) hasta convergencia.

Paso 5 Calcular $M = X^*V$, y su descomposición en valores singulares $M = UD_\sigma R^T$. Devolver

$$U, V \leftarrow VR \text{ y } D_{\sigma,\lambda} = \text{diag}\left[(\sigma_1 - \lambda)_+, \ldots, (\sigma_r - \lambda)_+ \right]$$

Este algoritmo es ligeramente distinto al de de MMMF; en cada iteración 2(a) utilizamos la última matriz calculada X^* en vez de X. Los cálculos de 2(b) siguen siendo eficientes ya que usamos la versión actual de A y B para predecir las entradas nos observadas y después se lleva a cabo la multiplicación de una matriz dispersa por una matriz delgada.

El último paso del algoritmo, sirve para dar ceros exactos tras el paso por

el valor umbral. Las regresiones alternadas de Ridge, dado que no reducen los coeficientes a cero, no revelan el verdadero rango de la matriz. Así que tras este paso, valores más pequeños que λ se convierten en ceros.

Vemos que el tipo *als* es una combinación de los algoritmos `Soft-Impute` y MMMF: sigue la estructura del segundo pero combinándolo con la solución del primero. De hecho, mezclando los dos *als* consigue eliminar los problemas que presenta cada uno.

1. `Soft-Impute` gasta demasiado tiempo en calcular la SVD de bajo rango, sobretodo en las primeras iteraciones cuando estamos lejos de la solución óptima. *als* para solucionar esto, rellena la matriz al mismo tiempo que calcula la SVD. De esta manera llega a la solución buscada por `Soft-Impute` pero en menos iteraciones.

2. MMMF resuelve con su algoritmo diferentes problemas de regresión por cada fila/columna, debido a las entradas vacías que presenta la matriz incompleta. En cambio, *als*, al haber primero rellenado todas las entradas de la matriz, lleva a cabo una regresión de Ridge simultánea sobre todas las filas y columnas. Y aunque esta regresión simultánea no es tan fuerte como la de MMMF, la rapidez que se consigue lo compensa.

Por tanto, la ventaja principal que tiene el tipo *als* con respecto a *svd* en la función `Soft-Impute` es la rapidez. Esto se hace obvio con la gráfica de la figura (16). Si recordamos, en nuestra función `SoftCalculo` medíamos cuánto tiempo se tardaba en cada iteración. Para cada valor de λ y de rango máximo r, implementábamos tanto *als* como *svd.* Vemos cómo los puntos azules de la gráfica, correspondientes al tipo *svd*, están prácticamente siempre por encima de los rojos de *als. svd* es por tanto más lento.

Pero no todo iban a ser ventajas. Este tipo de algoritmo *als* converge a un punto estacionario de (77), pero este punto no tiene por qué ser el mínimo del problema convexo. Se tienen que cumplir ciertas condiciones para que dichos puntos estacionarios sean a su vez el mínimo que buscamos. Estas condiciones vienen descritas en [36].

Aquí concluye nuestra larga función `Soft-Calculo`. Tras regularizaciones, diferentes valores de rangos y landas y dos tipos distintos de algoritmos, hemos conseguido reducir el error en un 5 %. No hubiésemos ganado el millón de euros, pero tenemos la base para conseguirlo. ¿Qué es lo que nos faltaría para ser millonarios entonces? Pues bien, en esta sección se ha expuesto cómo convertir nuestro problema a convexo y así aplicar nuestros algoritmos de descenso de gradiente y sus diferentes versiones, también cómo aplicar la descomposición en valores singulares a nuestro problema, la necesidad de las regularizaciones, las condiciones en las que podemos completar y hasta hemos definidos dos algoritmos capaces de resolver nuestro problema. Lo que nos faltan entonces son los detalles que se traducen en factores específicos de Netflix. Es decir, esta es la base para los filtros colaborativos en general, pero como es obvio el filtro de Netflix no vale tal cual para Amazon, sino que hay ciertos factores que se de-

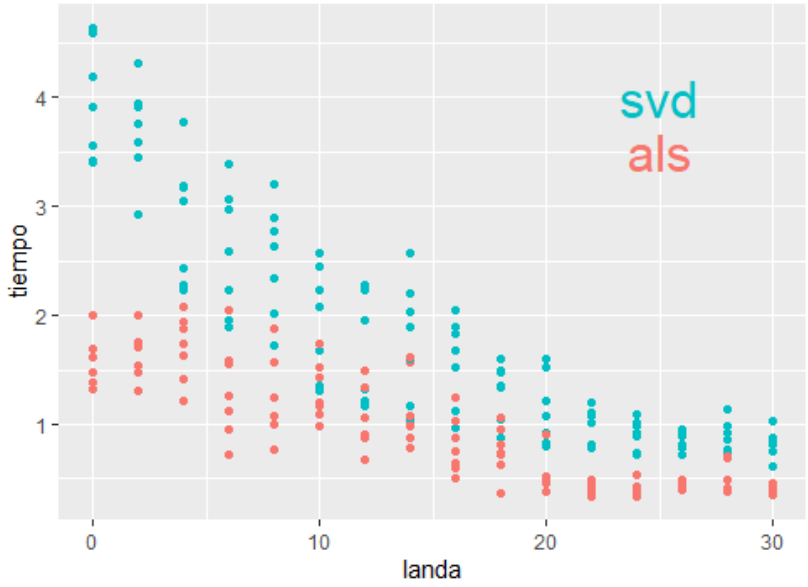


Figura 16: SVD económica

berían incluir y que salen al analizar cada caso en concreto. Por ejemplo, en el caso de Netflix otros predictores que podríamos incluir pueden ser:

- **Tiempo que lleva un usuario utilizando Netflix:** Si lo pensamos, nos volvemos más críticos a lo largo del tiempo; al principio a la hora de votar éramos mucho menos duros que ahora.
- **Tiempo desde que se hizo la primera puntuación a dicha película:** Si un usuario es de las primeras personas en ver la película puede ser que seea un gran fan de esa película o de un actor que venga en ella, por lo que tenderá a dar una puntuación más alta.
- **Cantidad de usuarios que han puntuado una película:** Cuando vemos que una película ha sido muy valorada (con puntuaciones tanto negativas como positivas) quiere decir que ha sido vista por mucha gente, y eso nos da más confianza a la hora de elegir dicha película para verla.
- **Media de puntuaciones:** Si íbamos a dar un 2 a una película pero vemos que la media es 4, solemos hacer un "*ni para ti, ni para mí*" y ponemos un tres.

Estos son algunos de los factores que podemos incluir. Pueden parecer menos importantes, más aleatorios. Pero los ganadores del Desafío Netflix, Bell y Koren dijeron en su solución final que la parte de regularización y la búsqueda de factores que no se ven con tan solo las puntuaciones es igual de importante que toda la parte algorítmica.

5.2. Descomposiciones aditivas de matrices

En el apartado anterior hemos visto cómo completar una matriz; partimos de la base de buscar una matriz de bajo rango, llevamos el problema a nuestro terreno convirtiéndolo en convexo, estudiamos el algoritmo `Soft-Impute`, la clase de MMMF y la relación entre ellos para resolverlo y así resolver el problema de completado. Hasta hemos visto un caso práctico, el desafío Netflix, en el que todo lo hablado sobre la descomposición en valores singulares, la regularización y los diferentes algoritmos se juntan para predecir de manera eficiente los gustos de los usuarios. También hablamos de las garantías necesarias sobre el número de elementos y su distribución para su completado.

Pero, en el mundo de enormes dimensiones de datos que presenta el Inteligencia Artificial, ¿con que sean suficientes y estén bien colocados nos vale? ¿es fiable pensar que todos estos datos son correctos, que no han sido corrompidos ni sometidos a errores? Evidentemente no. Lo más probable es que haya datos incorrectos o incluso filas enteras de datos falsos. Siguiendo el caso de Netflix, por ejemplo, a un actor o compañía de producción le interesa corromper datos para que sus películas sean las mejor votadas y por tanto las más recomendadas. Y fuera del caso Netflix, otro ejemplo llamativo y real ocurrió en 2002 en Amazon: su sistema había sido manipulado por la competencia de tal manera que recomendaba manuales sexuales a los usuarios interesados en la lectura cristiana.

Tenemos entonces un nuevo problema: vemos que aunque estemos en condiciones de completar la matriz, es posible y bastante probable que no estemos completándola con los datos correctos. Entonces lo que nos gustaría es poder separar los datos válidos de los errores o datos corruptos. Es decir, queremos separar nuestra matriz en dos matrices, una que sea de rango bajo que sirva para predecir y la otra que recoja la variabilidad que originan los datos corruptos. Esta separación en dos matrices no es una cosa aislada, sino que es una de las aplicaciones del Problema de Separación. Este problema es uno de los grandes objetivos de Machine Learning. El Problema de Separación busca descomponer la matriz en suma de matrices de estructuras complementarias, como por ejemplo, la que acabamos de ver en una de bajo rango y otra dispersa.

Matemáticamente, podemos describir el problema como dada la matriz X buscamos el par de estimadores (L, S) donde $X = L+S$. Los estimadores hemos visto que tienen estructuras complementarias por lo que cada uno estará sujeto a funciones de penalización diferentes para conseguir la estructura deseada. De esta manera, el problema a resolver por la descomposición en suma de matrices se puede describir como

$$\min_{\substack{\mathbf{L}\in\Re^{m\times n}\\ \mathbf{S}\in\Re^{m\times n}}} \left\{ \frac{1}{2}\|X-(L+S)\|_F^2 + \lambda_1\Phi_1(L) + \lambda_2\Phi_2(S) \right\}, \tag{92}$$

donde Φ_1 y Φ_2 son las funciones de penalización correspondientes. Por ejemplo uno de los casos más recurrentes y que analizaremos nosotros es la descomposición en una matriz dispersa y otra de bajo rango que se consigue a través de

las restricciones $\Phi_1(L) = \|L\|_\star$ y $\Phi_2(S = \|S\|_1$.

Vamos a ver como queda esta descomposición en suma de matrices en el caso de las matrices incompletas. Estamos en el contexto en el que no solo nos faltan datos sino que también parte de los datos que tenemos es probable que estén mal. Así que lo primero que hacemos es tomar la proyección ortogonal de estos elementos sobre $\Omega \subset m \times n$

$$\mathcal{P}_\Omega X = \begin{cases} X_{ij}, & (i,j) \in \Omega \\ 0, & (i,j) \notin \Omega \end{cases}$$

Suponemos ahora que solo tenemos una parte de las entradas de $L_0 + S_0$ disponibles que reescribimos convenientemente de la siguiente manera

$$Y = \mathcal{P}_\Omega (L_0 + S_0) = \mathcal{P}_\Omega L_0 + S_0'$$

Este modelo se resume en recuperar L_0 con tan solo parte de las entradas de L_0 y además parte de estas entradas observadas corruptas. Dentro de esta corrupción podemos distinguir dos casos: podemos suponer que hay ciertas entradas que están mal (sin querer hemos puntuado 5 estrellas pero en verdad queríamos dar a 2) o suponer que filas enteras han sido alteradas (es el caso de la productoras y actores buscando que sus películas sean más vistas). Entonces si buscamos L de rango máximo r y S con k de elementos distintos de cero; en el caso de corrupción son k entradas sueltas distintas de cero, o en el segundo caso, con k como total de filas distintas de cero.

$$\underset{\substack{rank(L)\leq r \\ card(S)\leq k}}{\text{minimizar}} \frac{1}{2}\|X - (L+S)\|_F^2. \tag{93}$$

Este criterio es doblemente no convexo, tanto por la restricción del rango como por la de cardinalidad. Pero siguiendo la estructura de (92) si imponemos las restricciones $\Phi_1(L) = \|L\|_\star$ y $\Phi_2(S) = \sum_{i,j} |s_{ij}|$ tenemos entonces un problema convexo que podemos resolver.

Parece que solo hablamos de completar matrices, filtros colaborativos y Netflix. Pero como hemos dicho antes, esta es solo una de las aplicaciones que tiene el Problema de Separación de matrices dentro del mundo del Inteligencia Artificial. Hay otras aplicaciones importantísimas en las que los datos que se estudian necesitan de esta separación entre una matriz de rango bajo y otra dispersa. De hecho ha sido la base de muchos desafíos dentro de la Inteligencia Artificial en los últimos años. Lo curioso, es que no siempre interesa la el sumando de bajo rango, a veces nuestro objetivo es conseguir la correcta parte dispersa, la del "error". Veamos varios ejemplos tanto donde es más relevante una como otros donde es más relevante la otra:

1. `Videovigilancia`: Dadas secuencias de videos de cámaras de vigilancia, queremos separar el fondo del movimiento. Por ejemplo, por la noche en el museo, no hay actividad. Para vigilarlo, sale caro tener a muchas personas mirando gran cantidad de cámaras en las que prácticamente nunca pasa

nada. Es más rentable que cuando se produzca algún tipo de movimiento, que se traduce en cambios en la matriz de píxeles, se muestre por pantalla y así un único vigilante es capaz de vigilar todo el museo. Se separa entonces el fondo, que no cambia, del moviento del ladrón por ejemplo. En este caso nos interesa la parte dispersa, la del error.

2. `Reconocimiento facial`: Esta capacidad ya presente hasta en algunos móviles ha sido uno de los grandes logros de la Inteligencia Artificial. Con el reconocimiento facial del móvil se pueden hasta hacer pagos que antes solo se podían con tarjeta y PIN. Este reconocimiento se hace con el análisis de las componentes principales de caras que veíamos al principio del trabajo. De hecho, con pocas componentes se representa gran parte de la variabilidad de las caras. Estas pocas componentes se traducen en una matriz de bajo rango, por lo que aquí L es lo que nos interesa. S en este ejemplo recoge las saturaciones por brillo, las sombras creadas por lo propios ojos y nariz, reflejos, gafas...

3. `Clasificación de textos`: Los motores de búsqueda como Google, cuando les das un par de palabras, son capaces de encontrar en menos de un segundo los documentos en la web relacionados. La base de estos motores de búsqueda se encuentra en el Análisis Semántico Latente (*Latent Semantic Analysis*). Este análisis consiste en un matriz donde las columnas corresponden a documentos y las filas a palabras. Cada entrada de la matriz representa el peso de una palabra en un documento. LSA agrupa tanto a los documentos que contienen palabras similares como las palabras que ocurren en documentos similares. En este ejemplo L recogería las palabras comunes de todos los documentos, mientras que S detecta las palabras clave que distinguen un documento de otro.

5.2.1. Separabilidad

Estamos ante el problema

$$\min_{sujeto\ a\ L+S=Z} \|L\|_\star + \lambda\|S\|_1 \tag{94}$$

donde la solución L es de bajo rango y S es dispersa. Tenemos que minimizar entonces la suma de dos funciones convexas, luego el problema es convexo y buscar el mínimo de una función convexa es sinónimo de solución única. A pesar de la ventaja de la convexidad, si nos ponemos a pensar, resolverlo es prácticamente imposible: para empezar, el número de incógnitas entre L y S es el doble que el de la medida de $Z \in \Re^{m\times n}$ y además no sabemos previamente ni qué datos ni cuántos son correctos. Nos hace falta un poco más de información y ciertas garantías porque sino pueden aparecer los varios problemas. Las condiciones que vamos a exponer tienen como fuente [37]

Por ejemplo, recordemos la primera matriz de (74), la matriz de todo ceros excepto un uno en la primera entrada. Dado que X es a la vez dispersa y de bajo rango, ¿cómo elegimos si es L o S? Para solventar esto, tenemos que imponer

que la matriz L es no dispersa. Recuperamos la definición de coherencia que vimos en el apartado de completibilidad de la matriz. Este término nos indica cómo están de alineados los vectores singulares con respecto a la base estándar. Si escribimos entonces la SVD de L tenemos que

$$L = UDV^T = \sum_{i=1}^{r} d_i u_i v_i^T$$

donde r es el rango de la matriz, $d_1, \ldots, d_r$ los valores singulares y $U = [u_1 \ldots, u_r]$ y $V = [v_1, \ldots, v_r]$ son las matrices de la descomposición. La condición de incoherencia se traduce entonces en

$$\max_i \left\|U^T e_i\right\|^2 \leq \frac{\mu r}{n_1}, \quad \max_i \left\|V^T e_i\right\|^2 \leq \frac{\mu r}{n_2} \tag{95}$$

y

$$\left\|UV^T\right\|_\infty \leq \sqrt{\frac{\mu r}{n_1 n_2}} \tag{96}$$

Otro problema puede surgir cuando la matriz S es de bajo rango. Imaginemos, por ejemplo, que las entradas no nulas de S se concentran en una o pocas columnas. Supongamos que todas las columnas de S son cero excepto la primera que es la opuesta de la de L. Entonces, nunca podríamos dar con la solución ya que $X = L + S$, tendría un subespacio de columnas iguales, o incluidas en L. Para evitar esto, asumimos que la variabilidad de la matriz S está distribuida de manera uniformemente aleatoria.

Si L y S cumplen estas condiciones sobre su estructura, entonces tenemos el siguiente teorema que garantiza que podremos obtener la solución única.

Teorema 5.6 *Supongamos que L cumple las condiciones de coherencia (95) y (96) y que el soporte de S está uniformemente distribuido sobre todos los subconjuntos de cardinalidad m. Entonces existe una constante numérica c tal que con probabilidad al menos $1 - cn^{-10}$ el problema (94) con $\lambda = 1/\sqrt{n}$ es exacta, es decir, $\hat{L} = L$ and $\hat{S} = S$, siempre que*

$$\text{rango}\,(L_0) \leq \rho_r n \mu^{-1} (\log n)^{-2} \quad \text{and} \quad m \leq \rho_s n^2$$

Donde ρ_r y ρ_s son constantes numéricas. En general, en el caso rectangular donde $L = m \times n$, denotamos $n_{(1)} = \max(m, n)$ y $n_{(2)} = \min(m, n)$. Tenemos garantizada la solución a (94) y $\lambda = 1/\sqrt{n_{(1)}}$ con probabilidad al menos $1 - cn_{(1)}^{-10}$, siempre que rango$(L_0) \leq \rho_r n_{(2)} \mu^{-1} \left(\log n_{(1)}\right)^{-2}$ y $m \leq \rho_s n_1 n_2$

Este teorema garantiza que si la matriz L cumple las condiciones de coherencia se puede resolver el problema con probabilidad casi uno si los datos corruptos de S se distribuyen de manera uniforme por toda la matriz.

5.2.2. Algoritmos

En estos últimos años, resolver el problema (94) se ha ido haciendo cada vez más eficiente, sobretodo por las mejoras al minimizar la norma l_1 y la nuclear. Vamos a ir viendo este progreso.

Cai et. al. (ver [39]) explotaron la analogía entre los algoritmos iterativos del operador umbral de la minimización l_1 para desarrollar un algoritmo iterativo de minimización de la norma nuclear en el que se van encogiendo repetidamente los valores singulares hasta dar con la matriz de rango deseado. La complejidad de este algoritmo era únicamente el cálculo de la SVD en cada iteración [40]. Sin embargo, para nuestro problema, esto es demasiado lento. Incluyendo las ideas de Nesterov [44], la rapidez aumentó pero todavía no lo suficiente. Basándose en [45], Toh et. al. desarrollaron un tipo de algoritmos de gradiente proximal que llamaron Gradiente Proximal Acelerado (APG), del que hablábamos en la sección 4. Este APG se adaptó a nuestro problema (ver [45]) que hizo que este algoritmo fuese 50 veces más rápido que el anterior y además mantiene la convergencia de ratio $\mathcal{O}(1/k^2)$ propia de esta clase de algoritmos. Sin embargo, nosotros no nos quedaremos con este, sino que en lugar de resolver el problema (94) utilizaremos el método de los multiplicadores de Lagrange con direcciones alternadas, que por lo general, es más eficiente que APG y con menos iteraciones.

El Lagrangiano aumentado de este problema toma la forma

$$l(L,S,Y) = \|L\|_* + \lambda\|S\|_1 + \langle Y, X-L-S\rangle + \frac{\mu}{2}\|X-L-S\|_F^2$$

La estrategia para resolverlo es la siguiente: primero minimizar l con respecto a L fiajando S, después minimizar l con respecto a S fijando L y después actualizar la matriz de multiplicadores de Lagrange según el residuo $Z-L-S$. Resumimos la estrategia con el siguiente algoritmo:

Algoritmo: DIRECCIONES ALTERNADAS DE LAGRANGE

Paso 1 Iniciamos con $S_0 = Y_0 = 0, \mu > 0$

Paso 2 **mientras que** no converge **hacer:**

a) Calcular: $L_{k+1} = \mathcal{D}_\mu\left(M - S_k - \mu^{-1}Y_k\right)$;

b) Calcular: $S_{k+1} = \mathcal{S}_{\lambda\mu}\left(M - L_{k+1} + \mu^{-1}Y_k\right)$;

c) Calcular: $Y_{k+1} = Y_k + \mu\left(M - L_{k+1} - S_{k+1}\right)$

Paso 3 **Escribir:**L, S.

La convergencia de este tipo de algoritmos ya la hemos estudiado en la sección anterior. Funciona sobre gran cantidad de problemas y solo le hacen falta pocas iteraciones para obtener un resultado preciso. El coste más grande de cada iteración viene dado por el cálculo de L_{k+1} ya que supone calcular los vectores singulares de $M - S_k + \mu^{-1}Y_k$ que sobrepasan el umbral marcado por μ^{-1}. La elección de μ y de la condición de parada son detalles importantes de

este algoritmo. Es frecuente elegir $\mu = m \times n/4\|X\|_1$ ([46]) y parar el algoritmo cuando $\|X - L - S\|_F \leq \delta\|X\|_F$, con $\delta < 10^{-7}$.

5.2.3. Aplicación a videovigilancia

Una vez entendida la idea de la descomposición aditiva de matrices y establecidas las garantías que deben cumplirse, vamos a intentar llevarlo a la práctica. Aplicamos las ideas anteriores a la detección de movimiento en las imágenes registradas por una cámara de vigilancia. El vídeo está en un repositorio público de una universidad italiana. Se han extraído del vídeo fotogramas del primer minuto, muestreaNdo cada segundo. En él se ve un campus universitario con tres chicos que aparecen fumando y hablando. Permanecen prácticamente parados, por lo que forma parte de la estructura fija. En los segundos 45, 46 y 47 unas personas cruzan la escena. Nuestro objetivo es detectar esta "intrusión" separando la imagen permanente de lo que cambia de forma rápida.

Empezamos preparando los datos. De los casi cinco minutos de vídeo ya hemos dicho que tomamos fotogramas únicamente del primero. Nos quedamos con un canal puesto que los colores no nos dan información aquí (de RGB elegimos el primero; se ven solo tonos de gris). Por cada pixel tenemos un valor en [0,1] (un tono de gris). Cada imagen se ha convertido en una fila de la matriz X (el primer segundo es la primera fila de pixels, luego el segundo, etc). La matriz X tiene por lo tanto 60 filas (una por fotograma) y 110592 columnas.

Buscamos ahora la descomposición en suma de matrices; una de rango bajo que recogerá el "escenario" y a los tres chicos que no se mueven y la otra matriz dispersa que mostrará a la pareja que cruza la escena rápidamente.

$$X \simeq L + S$$

con L de rango bajo y S dispersa. Una forma de conseguir esta aproximación sería resolver el problema de optimización

$$\min_{L,S} \frac{1}{2}\|X - (L + S)\|_F^2 + \lambda_1\|L\|_* + \lambda_2\|S\|_{\ell_1}.$$

Aquí $\|S\|_{\ell_1} = \sum_{i,j} |s_{ij}|$ es la norma ℓ_1 de la matriz S, la misma que en el lasso.

Para L fijo el problema a resover sería

$$\min_{S} \frac{1}{2}\|(X - L) - S\|_F^2 + \lambda_2\|S\|_{\ell_1} \qquad (A)$$

para el que podemos calcular la solución de forma explícita:

$$\hat{S}_{\lambda_2} = [\mathcal{S}_{\lambda_2}(x_{ij} - l_{ij})]$$

De igual modo, para S fijo el problema a resolver se convierte en

$$\min_{L} \frac{1}{2}\|(X - S) - L)\|_F^2 + \lambda_1\|L\|_*, \qquad (B)$$

que tiene como solución

$$\hat{L}_{\lambda_1} = UD_{\lambda_1}V'$$

si $X - S = UDV'$ (la SVD de $X - S$) y D_{λ_1} se obtiene aplicando el operador de soft-thresholding a los valores diagonales de D. Entonces podemos explotar esta facilidad para la solución de los subproblemas partiendo de iterantes iniciales L_0, S_0 y aplicando después los pasos de actualización

$$S_{k+1} = \text{argmin}_S \frac{1}{2}\|(X - L_k) - S\|_F^2 + \lambda_2\|S\|_{\ell_1}$$

$$L_{k+1} = \text{argmin}_L \frac{1}{2}\|(X - S_{k+1}) - L)\|_F^2 + \lambda_1\|L\|_*,$$

iterando hasta la convergencia. A este método de optimización se le conoce como método de proyecciones alternadas.

Una vez explicado el procemiento, pongámoslo en práctica. Tomamos $\lambda_1 = 9$ y $\lambda_2 = 0{,}1$ e inicializamos tanto la matriz de rango bajo L como S como matrices nulas. Empezamos el bucle *while* permitiendo una tolerancia de 10^{-8} y como máximo 100 iteraciones para evitar blucles infinitos. Llevamos a cabo el algoritmo que descrito y calculamos en cada ronda del bucle el error cuadrático medio (en norma de Frobenius) de los actuales valores de L y S.

Para comprobar la eficiencia del algoritmo tenemos que comprobar los segundos 45, 46 y 47. Esto nos lo muestra la figura (17): en la primera columna están los fotogramas originales de los segundo 45, 46 y 47, la segunda columna tiene los fotogramas correspondientes de la matriz L (la de rango bajo) y la tercera los de la matriz S (la parte dispersa). Se aprecia perfectamente como el algoritmo ha conseguido separar la parte poco varible de la secuencia de imágenes del movimiento rápido de la pareja que cruza la escena.

6. Conclusiones

Sabemos que la Inteligencia Artificial ha revolucionado la sociedad, cambiado nuestra forma de vida y ha dejado huella en prácticamente todas las áreas que podamos imaginar; desde la salud, a la economía, tocando el deporte, la educación, el ocio. Los dos grandes objetivos dentro de la Inteligencia Artificial a los que hemos dedicado este trabajo tienen numerosas aplicaciones en la vida real; hemos hablado de filtros colaborativos, reconocimiento facial, motores de búsqueda... Pero de la misma manera que la Inteligencia Artificial crece exponencialmente, las aplicaciones de nuestros dos objetivos también. Veamos varios ejemplos en los que se han aplicado estos últimos años y qué aplicaciones se predicen que van a surgir a partir de ellos.

Las cámaras de reconocimiento facial son una de las grandes aplicaciones del problema de separado de la matriz. Estas cámaras son capaces de reconocer una cara entre una multitud en cuestión de milisegundos. La base que hay detrás de estos sofisticados sistemas es la combinación de técnicas de videovigilancia y de reconocimiento facial que hemos visto en el capítulo anterior. Una vez dado el

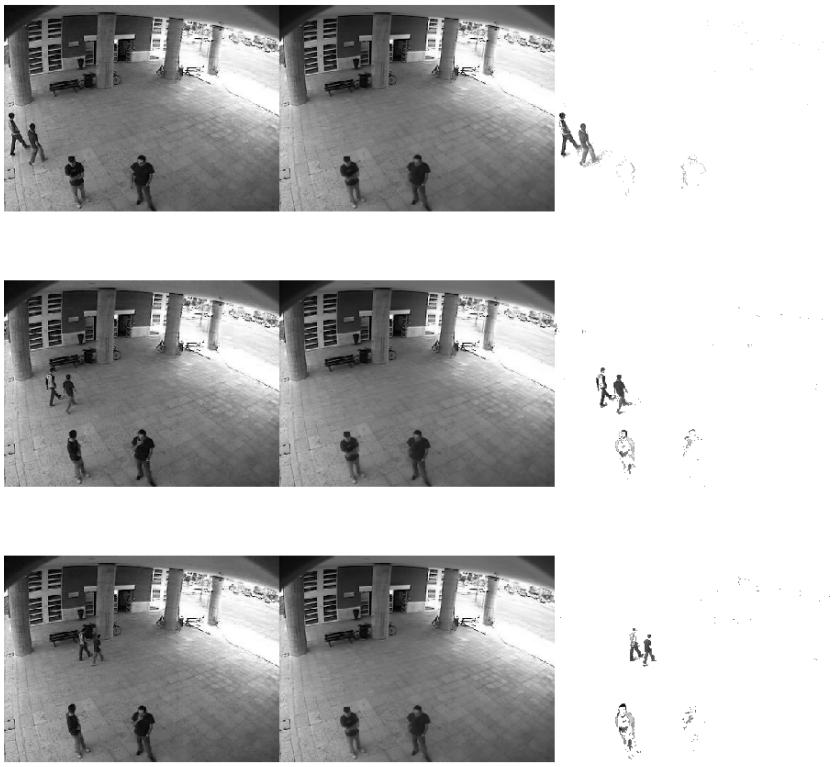


Figura 17: Cámara de videovigilancia

vídeo de una cámara, lo primero que hacen estos sistemas es separar el fondo de las personas (lo estático del ruido). Una vez que se obtiene la matriz dispersa se tienen que identificar las caras aplicando un sistema de reconocimiento facial. Así fue como el Ministerio de Seguridad Pública de China en 2015 encontró a casi la mitad de los 50 fugitivos sospechosos de grandes delitos, desde asesinos a evasores fiscales. Esta operación bautizada como *Cloud Sword* no hubiese sido posible si no es por el uso de Inteligencia Artificial (y de las 300 millones de cámaras de seguridad que vigilan a los ciudadanos chinos).

Siguiendo esta idea de atrapar al ladrón, Mercadona propuso recientemente que quería aplicar un sistema de reconocimiento facial en sus sistemas de videovigilancia; de esta manera, podrían reconocer la cara de toda persona que entrase en uno de sus comercios, compararlo con la base de datos de personas que tienen orden de alejamiento y vetarle de esta manera el acceso. Aunque las imágenes no se graben, este tipo de medidas puede entrar en conflicto con la Ley de Protección de Datos o el Derecho a la Intimidad. Este es uno de los grandes retos de la Inteligencia Artificial; buscar el equilibrio entre funcionalidad y

privacidad.

Otro tipo de aplicaciones no tienen esta controversia. Por ejemplo, los sistemas de reconocimiento facial podrían mejorar la seguridad en las residencias de ancianos, localizándolos en zonas peligrosas para ellos y generando alarmas. Se podría aplicar también a cárceles para que ciertos presos no entren en contacto unos con otros, controlar que estén en el módulo correcto...

En la sección de completado de matrices nos hemos centrado principalmente en el filtro colaborativo. Son muchas las compañías que invierten grandes cantidades de dinero cada año en Inteligencia Artificial, con el objetivo de conocer mejor a sus clientes aumentando así sus ventas. Estos últimos años, se está desarrollando conjuntamente a los filtros colaborativos, el llamado *Social Listening*. Ya no se tienen en cuenta solo las puntuaciones de usuarios a productos, sino que se incluyen nuevas variables sobre las satisfacción que muestra un usuario en redes sociales sobre dicho producto. Un *tweet* comentando lo que te ha emocionado cierta película, *influencers* vendiendo determinada marca, comentarios sobre un producto en Amazon... son factores que repercuten a los usuarios a la hora de elegir el producto.

El mundo de los eventos también está cambiando gracias a la Inteligencia Artificial. El sector MICE (Meetings, Incentives, Conventions and Exhibitions) reúne miles de personas en cada evento que realiza. Estos eventos son cada vez más sofisticados, grandes y complejos. Es imposible que una persona pueda asistir a todo lo que en un gran evento se ofrece. Sin embargo, cada uno de los miles de asistentes que van a estos eventos anuales genera información y datos, (visita *stands*, asiste a conferencias, consulta páginas web en su teléfono, publica en redes sociales, aumenta el número de empresas a las que sigue...). Con algoritmos de completado de matrices se podrán generar servicios de recomendación para los asistentes a eventos a partir de los datos generados.

Pero el completado de matrices no tiene única y exclusivamente fin económico. Desde el año 2015, la ONU se propone desarrollar los Objetivos de Desarrollo Sostenible. Se trata de iniciativas que hacen un llamamiento a todos los agentes políticos, sociales y empresariales a contribuir con sus actuaciones a la causa de la sostenibilidad. Los ODS son 17 objetivos que pretenden guiar de manera coordinada medidas a nivel mundial para acabar con los grandes problemas del planeta y la humanidad: fin de la pobreza y desigualdad, proteger el medio ambiente, favorecer el desarrollo sostenible...

Las grandes empresas están reportando voluntariamente sus actuaciones con objeto de cumplir con estos 17 objetivos. Se están generando informes de manera anual y mundial con multitud de actuaciones. Es necesario el uso inteligente de motores de búsqueda para llegar a valorar si hay controversias entre los informes de las compañías y las noticias publicadas en prensa y en boletines oficiales a nivel diario y mundial. Thomson Reuters ya utiliza estos datos para intentar valorar las compañías, el riesgo de operar con ellas e incluso el cumplimiento que hacen de las normas. Será necesario aplicar algoritmos de completado y separado de matrices para generar nuevos servicios, ya que se genera información de manera diaria y anual de más de 6000 compañías y más de 400 métricas diferentes relacionados con indicadores de desarrollo sostenible.

Figura 18: ODS

Es bien sabido que al Inteligencia Artificial está teniendo un crecimiento exponencial. Tanto el completado como el separado de matrices van a ser necesarios para generar los nuevos servicios que se van a desarrollar. Pero estos objetivos también van a tener que aprender a adaptarse, sabemos que van a aparecer datos que todavía hoy no existen; no sabemos por ejemplo si en el futuro habrá bases de datos fisiológicas, (sangre, exhalación de aire, sudor...), nuevas redes sociales o profesionales... Prueba de esta evolución de datos es que hace 5 años el reconomiento de huellas dactilares era lo último, ahora es el reconocimiento facial lo que se está explotando y quién dice que dentro de 5 años no será el aire que exhalamos lo que desbloquee nuestro móvil. Alan Turing no sabemos si se sorprendería de este crecimiento, pero nosotros estamos avisados, y en nuestra mano está el formar parte de esta revolución.

7. Apéndice

Para empezar, demostraremos el teorema de Weyl que da cotas superiores para los valores singulares de la suma de dos matrices en función de valores singulares para cada uno de los sumandos. Aquí los valores singulares de la matriz $X \in \mathcal{M}_{m\times n}(\mathbb{R})$ ordenados de forma decreciente se denotan $d_j(X)$. El enunciado del teorema es una adaptación del III.6.5 en [22]. La demostración que se presenta aquí es la solución (parcial) a ese problema.

Si H y L son subespacios vectoriales de $\mathbb{R}^p$ entonces

$$\dim(H \cap L) = \dim(H) + \dim(L) - \dim(H + L) \geq \dim(H) + \dim(L) - n.$$

En el caso de tres subespacios, H, L y M, podemos decir que

$$\dim(H\cap L\cap M) \geq \dim(H\cap L)+\dim(M)-n \geq \dim(H)+\dim(L)+\dim(M)-2n. \tag{97}$$

Esta observación será útil en la siguiente demostración.

Teorema 7.1 (Weyl) *Si $X, Y \in \mathcal{M}_{m\times n}(\mathbb{R})$, $1 \leq i \leq j \leq p$ y $1 \leq j-i+1 \leq n$ entonces*

$$d_j(X + Y) \leq d_i(X) + d_{j-i+1}(X).$$

Demostración. Denotamos por $\mathbf{w}_1 \cdots \mathbf{w}_n$, $\mathbf{u}_1 \cdots \mathbf{u}_n$ y $\mathbf{v}_1 \cdots \mathbf{v}_n$ bases ortonormales de autovectores para $(X + Y)^T(X + Y)$, X^TX e Y^TY, respectivamente. Denotamos por H, L y M los subespacios generados por $\mathbf{w}_1 \cdots \mathbf{w}_j$, $\mathbf{u}_i \cdots \mathbf{u}_n$ y $\mathbf{v}_{j-i+1} \cdots \mathbf{v}_n$, respectivamente. Entonces $\dim(H)+\dim(L)+\dim(M) = j+(p-i+1)+(n-j+i) = 2n+1$. Por (97) podemos tomar $\mathbf{h} \in H \cap L \cap M$ con $\|\mathbf{h}\| = 1$. Ahora, como $\mathbf{h} \in H$,

$$\begin{aligned}\|(X + Y)\mathbf{h}\|^2 &= \mathbf{h}^T(X + Y)^T(X + Y)\mathbf{h} = \sum_{r=1}^{j} d_r(X + Y)^2\langle \mathbf{h}, \mathbf{w}_r\rangle^2 \\ &\geq d_j(X + Y)^2 \sum_{r=1}^{j}\langle \mathbf{h}, \mathbf{w}_r\rangle^2 = d_j(X + Y)^2.\end{aligned}$$

De forma similar comprobamos que $\|X\mathbf{h}\| \leq d_i(X)$ y $\|Y\mathbf{h}\| \leq d_{j-i+1}(X)$. Combinando las tres desigualdades se deduce

$$d_j(X + Y) \leq \|(X + Y)\mathbf{h}\| \leq \|X\mathbf{h}\| + \|Y\mathbf{h}\| \leq d_i(X) + d_{j-i+1}(X).$$

□

Lemma 7.2 *Dada una función f convexa, todo subnivel de f es convexo.*

Demostración. Sea $S_\alpha = \{x \in \mathit{dominio}(f) \mid f(x) \leq \alpha\}$. Tomamos $x, y \in S_\alpha$ y $\lambda \in [0, 1]$. Por la convexidad de f tenemos que

$$\begin{aligned}f(\lambda x + (1 - \lambda)y) &\leq \lambda f(x) + (1 - \lambda)y \\ &\leq \lambda\alpha + (1 - \alpha)\lambda = \alpha\end{aligned}$$

Luego $\lambda x + (1-\alpha) \in S_\alpha$ y concluimos que f es convexo en S_α.

□

Lemma 7.3 *Dada una función f convexa, todo subnivel de f es convexo.*

Demostración. Sea $S_\alpha = \{x \in dominio(f) \mid f(x) \leq \alpha\}$. Tomamos $x, y \in S_\alpha$ y $\lambda \in [0,1]$. Por la convexidad de f tenemos que

$$\begin{aligned} f(\lambda x + (1-\lambda)y) &\leq \lambda f(x) + (1-\lambda)y \\ &\leq \lambda\alpha + (1-\alpha)\lambda = \alpha \end{aligned}$$

Luego $\lambda x + (1-\alpha) \in S_\alpha$ y concluimos que f es convexo en S_α. □

Lemma 7.4 *La norma de operador*

$$\|A\| = \sup_{x:\, \|x\|=1} \|Ax\|$$

es submultiplicativa, es decir, $\|AB\| \leq \|A\|\|B\|$

Demostración. Consideremos la norma de operador, es decir,

$$\|A\| = \sup_{x:\, \|x\|=1} \|Ax\|.$$

Esta norma es submultiplicativa, es decir, $\|AB\| \leq \|A\|\|B\|$, veámoslo: [esto al apéndice]

$$\|AB\| = \sup_{x:\, \|x\|=1} \|ABx\| \leq \sup_{x:\, \|x\|=1} \|Ax\| \sup_{x:\, \|x\|=1} \|Bx\| \leq \|A\|\|B\|.$$

□

En el apartado 5.1.1 hablamos del posible completado de matrices estableciendo ciertas garantías para poder llegar a la solución óptima y no un mínimo local. Una de las condiciones es el número de entradas observadas de la matriz y hablamos del fenómeno conocido como *Problema coleccionista de cupones*. Demostraremos aquí el número medio de observaciones para que haya al menos una observación en cada fila N diferente. Estamos en el caso en el que se toman N muestras con reemplazamiento sobre las entradas de la matriz. Sea $\mathcal{F}$ el suceso en el que hay al menos una fila con ninguna observación. Para cada fila $j = 1, \ldots, p$, sea Z_j la variable binaria que indica que ninguna entrada de la fila j ha sido observada y definimos $Z = \sum_{j=1}^{n} Z_j$. Tenemos entonces que $\mathbb{P}[\mathcal{F}] = \mathbb{P}[Z > 0]$. Vamos a buscar una cota inferior a esta igualdad. Teniendo en cuenta la desigualdad de Cauchy-Schwarz:

$$\mathbb{P}[Z > 0] = \mathbb{E}\left[I_{Z>0}\right] \geq \mathbb{E}\left[I_{Z>0}^2\right] \geq \frac{(\mathbb{E}[Z \cdot I_{Z>0}])^2}{\mathbb{E}\left[Z^2\right]} = \frac{(\mathbb{E}[Z])^2}{\mathbb{E}\left[Z^2\right]} \tag{98}$$

Calculemos el numerador de la fracción:

$$\mathbb{E}[Z] = \mathbb{E}[\sum_{i=1}^{p} Z_i] = p\mathbb{E}[Z_i]$$

La esperanza de cada Z_i es igual a

$$\mathbb{E}[Z_i] = 1 \cdot \mathbb{P}(Z_i = 1) + 0 \cdot \mathbb{P}(Z_i = 0) = \mathbb{P}(Z_i = 1) = \left(\frac{p \times p - p}{p \times p}\right)^N = \left(1 - \frac{1}{p}\right)^N$$

Con esta igualdad tenemos entonces que $\mathbb{E}[Z] = p\left(1 - \frac{1}{p}\right)^N$

Ahora calculamos el denominador.

$$\mathbb{E}\left[Z^2\right] = \mathbb{E}\left[\sum_{i=1}^{p} Z_i \sum_{j=1}^{p} Z_j\right] = \mathbb{E}\left[\sum_{i,j=1}^{p} Z_i Z_j\right] = \sum_{i,j=1}^{p} \mathbb{E}\left[Z_i Z_j\right]$$

Cuando $i \neq j$ tenemos que

$$\begin{aligned}
\mathbb{E}\left[Z_i Z_j\right] &= \mathbb{P}(Z_i = 1, Z_j = 1) = \left(\frac{p \times p - 2p}{p \times p}\right)^N = \left(1 - \frac{2}{p}\right)^N \\
&\leq \left(1 + \frac{1}{p^2} - \frac{2}{p}\right)^N = \left(1 - \frac{1}{p}\right)^{2N} \\
&= \mathbb{E}\big[Z_i\big]\mathbb{E}\big[Z_j\big]
\end{aligned}$$

Cuando $i = j$, dado que la variable Z es binaria tenemos que $\mathbb{E}\left[Z_i^2\right] = \mathbb{E}\left[Z_i\right]$. Luego el denominador

$$\begin{aligned}
\mathbb{E}\left[Z^2\right] &= \sum_{i,j=1}^{p} \mathbb{E}\left[Z_i Z_j\right] = \sum_{i}^{p} \mathbb{E}\left[Z_i^2\right] + \sum_{i \neq j}^{p} \mathbb{E}\left[Z_i Z_j\right] \\
&\leq p\mathbb{E}\left[Z_i^2\right] + (p-1)^2 \mathbb{E}\left[Z_i Z_j\right] \\
&\leq p\left(1 - \frac{1}{p}\right)^N + p^2\left(1 - \frac{1}{p}\right)^{2N}
\end{aligned}$$

Si retomamos la desigualdad (98) tenemos que la cota inferior

$$\begin{aligned}
\mathbb{P}[Z > 0] &\geq \frac{p^2\left(1 - \frac{1}{p}\right)^{2N}}{p\left(1 - \frac{1}{p}\right)^N + p^2\left(1 - \frac{1}{p}\right)^{2N}} \\
&\geq \frac{p\left(1 - \frac{1}{p}\right)^N}{1 + p\left(1 - \frac{1}{p}\right)^N}
\end{aligned}$$

Esta probabilidad tiende a cero cuando $1 \geq p\left(1-\frac{1}{p}\right)^N$. Tomando logaritmos a cada lado de la inecuación y con p suficientemente grande tenemos entonces que $\mathbb{P}[Z > 0] < 0$ siempre que $N > p \log p$. Tenemos entonces una cota sobre el número de elementos que debe presentar la matriz a completar en función de la dimensión.

A lo largo del trabajo tratamos con la norma de Frobenius y utilizamos que es invariante bajo la multiplicación por matrices unitarias. Vemos esto con el siguiente lema.

Lemma 7.5 Sea $A \in \mathcal{M}_{m\times n}(\mathbb{C})$

Si $U \in \mathcal{M}_m(\mathbb{C})$ *y* $U^*U = I_m$, *entonces*

$$\|UA\|_F = \|A\|_F$$

Si $V \in \mathcal{M}_n(\mathbb{C})$ *y* $V^*V = I_n$, *entonces*

$$\|AV\|_F = \|A\|_F$$

Demostración.

$\|UA\|_F^2 = \operatorname{tr}((UA)^*(UA)) = \operatorname{tr}(A^*U^*UA) = \operatorname{tr}(A^*A) = \|A\|_F^2$
$\|AV\|_F^2 = \operatorname{tr}((AV)^*(AV)) = \operatorname{tr}(V^*A^*AV) = \operatorname{tr}(V^*VA^*A) = \operatorname{tr}(A^*A) = \|A\|_F^2$

□

Referencias

[1] A.M.Touring, (1950). Computing Machinery and Intelligence. Mind 49: 433-460.

[2] Matt Komorowski, (2014). A history of storage cost, *mkomo.com*.

[3] Emmanuel J. Candès, Xiaodong Li, Yi Ma and John Wright4. *Robust Principal Component Analysis?*.

[4] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series*, **6**, 2(11):559–572.

[5] Beltrami, E. (1873). Sulle Funzioni Bilineare. *Giornale di Matematiche*, volume XI, 98–106.

[6] Reris, R. and Brooks, J. P. (2015). Principal Component Analysis and Optimization: A Tutorial. *14th INFORMS Computing Society Conference*, 212–225.

[7] Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, **1**, 211–218.

[8] Bhatia, R. (1997). *Matrix Analysis.* Springer.

[9] Hoerl, A.E. y Kennard, R.W. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12(1): págs. 5567. 1970.

[10] Tibshirani,R. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society.* Series B (Methodological), 58(1): págs. 267288. 1996.

[11] Tibshirani, R. (2011). Regression shrinkage and selection via the lasso. *A retrospective, Journal of the Royal Statistical Society: Series B (Methodological)*, 73(3): págs. 273282.

[12] J.L. Lagrange, (1797) Théorie des fonctions analytiques contenant les principes du calcul différentiel. *Imprimerie de la République.*

[13] David Knowles (2010). Lagrangian Duality for Dummies

[14] Kuhn, H. W.; Tucker, A. W., (1991). Nonlinear programming. *Proceedings of 2nd Berkeley Symposium.* Berkeley: University of California Press. pp. 481–492.

[15] W. Karush, (1939). Minima of Functions of Several Variables with Inequalities as Side Constraints (M.Sc. thesis). Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois.

[16] Nesterov, Y. (2007), Gradient methods for minimizing composite objective function, *Technical Report 76,* Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL).

[17] Tseng, P. (1988), Coordinate ascent for maximizing nondifferentiable concave functions, *Technical Report LIDS-P* ; 1840, Massachusetts Institute of Technology. Laboratory for Information and Decision Systems.

[18] Tseng, P. (2001), Convergence of block coordinate descent method for nondifferentiable maximization, *Journal of Optimization Theory and Applications* 109(3), 474–494.

[19] Friedman, J., Hastie, T. and Tibshirani, R. (2010b), Regularization paths for generalized linear models via coordinate descent, *Journal of Statistical Software* 33(1), 1–22.

[20] Efron, Bradley; Hastie, Trevor; Johnstone, Iain; Tibshirani, Robert. Least angle regression. *Ann. Statist. 32* (2004), no. 2, 407–499.

[21] Laurent, M. (2001), Matrix completion problems, *The Encyclopedia of Optimization,* Kluwer Academic, pp. 221–229.

[22] Bhatia, R. (1997). *Matrix Analysis.* Springer.

[23] Hastie, T., Tibshirani, R. y Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations.* CRC Press.

[24] Mazumder, R., Hastie, T. y Tibshirani, R. (2010). Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, **11**, 2287–2322.

[25] Watson, G.A. (1992). Characterization of the subdifferential of some matrix norms. *Linear Algebra and its Applications*, **170**, 33–45.

[26] Gross, D. (2011), Recovering low-rank matrices from few coefficients in any basis, *IEEE Transactions on Information Theory* 57(3), 1548–1566.

[27] Cand'es, E. and Recht, B. (2009), Exact matrix completion via convex optimization, *Foundation for Computational Mathematics* 9(6), 717–772.

[28] Recht, B. (2011), A simpler approach to matrix completion, *Journal of Machine Learning Research 12,* 3413–3430.

[29] Negahban, S. and Wainwright, M. J. (2012), Restricted strong convexity and (weighted) matrix completion: Optimal bounds with noise, *Journal of Machine Learning Research 13,* 1665–1697.

[30] Lee, D., Seung, H. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401, 788– 791

[31] Rennie, J. D. M., Srebro, N. (2005). Loss functions for preference levels: Regression with discrete ordered labels. *Proceedings of the IJCAI Multidisciplinary Workshop on Advances in Preference Handling*

[32] Keshavan, R. H., Montanari, A. and Oh, S. (2010), Matrix completion from noisy entries, *Journal of Machine Learning Research 11,* 2057–2078.

[33] Keshavan, R. H., Oh, S. and Montanari, A. (2009), Matrix completion from a few entries, *IEEE Transactions on Information Theory 56(6),* 2980– 2998.

[34] Bennett, J. and Lanning, S. (2007), The netflix prize, in *Proceedings of KDD Cup and Workshop in conjunction with KDD.*

[35] Mazumder, Rahul Hastie, Trevor Tibshirani, Robert. (2010). Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of machine learning research : JMLR. 11.* 2287-2322.

[36] Srebro, Nathan Rennie, Jason Jaakkola, Tommi. (2004). Maximum-Margin Matrix Factorization.. *Adv. Neural Inf. Process. Syst.. 17.*

[37] Emmanuel J. Candès and Xiaodong Li and Yi Ma and John Wright,(2009). Robust Principal Component Analysis?.

[38] Li, L., Huang, W., Gu, I. Y. and Tian, Q. (2004), Statistical modeling of complex backgrounds for foreground object detection, IEEE Transactions on Image Processing 13(11), 1459–1472.

[39] W. Yin, E. Hale, and Y. Zhang. Fixed-point continuation for l_1 - minimization: *Methodology and convergence.* preprint, 2008.

[40] W. Yin, S. Osher, D. Goldfarb, and J. Darbon. Bregman iterative algorithms for $l^1$1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences,* 1(1):143–168, 2008.

[41] J. Cai, E. J. Cand‘es, and Z. Shen. A singular value thresholding algorithm for matrix completion. preprint, 2008.

[42] D. Goldfarb and S. Ma. Convergence of fixed point continuation algorithms for matrix rank minimization. preprint, 2009.

[43] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k2)$. *Soviet Mathematics Doklady,* 27(2):372–376, 1983.

[44] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming,* 103(1), 2005.

[45] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences,* 2(1):183–202, Mar 2009.

[46] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. preprint, 2009.