FACTORIZACIÓN DE MATRICES BINARIAS VIA APRENDIZAJE DE DICCIONARIOS

Ignacio Ramírez
nacho@fing.edu.uy

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Introducción

- Factorización de matrices: una forma de analizar datos
- Factorización de matrices (MF): problema muy viejo (Pearson, 1901)
  - Análisis de correlaciones/covarianza (PCA)
  - Ajuste de distribuciones (MVN)
  - Aproximación de funciones
  - Compresión de datos (KLT)
- Factorización de matrices binarias (BMF): problema viejo (M8, 196?)
  - Análisis de asociaciones / data mining
  - Bi-clustering
  - Aproximación de funciones binarias
  - Compresión de datos
  - No hay soluciones exactas

Problema

Factorización de matrices binarias (BMF)

- **En general** se busca aproximar una matriz binaria X como el producto de dos matrices binarias U y V: $X \approx UV^T$
- Según el álgebra que se utilice, existen dos variantes: factorización Booleana o factorización módulo 2.
- Ambas tienen sus ventajas y desventajas.
- Nosotros nos enfocaremos en la factorización módulo 2, pero mucho de lo que presentaremos se puede modificar fácilmente para el caso Booleano

Antecedentes

Reseña sobre BMF

- Registro más antiguo citado, un tal algoritmo 8M del paquete estadístico BMDP; ni idea cómo es.
- Gran interés con el auge de Data Mining en computación
- Métodos como ASSO y Proximus (principios de S. XXI) muy extendidos
- Cientos de métodos distintos; todos heurísticos

Motivación

BMF y Big Data

- Hasta donde sabemos, todos los métodos existentes son *off-line*
- Muchos son extremadamente costosos incluso para problemas pequeños (ej., métodos basados en relajaciones convexas)
- Otros son eficientes y producen buenos resultados a pequeña escala, pero escalan muy mal con la cantidad de muestras (ej., ASSO)
- Otros, como Proximus, escalan bien, pero se limitan a soluciones de bajo rango (1)
- Todo esto representa serios problemas a la hora de trabajar en un contexto de *big data*

Contribución

BMF y Big Data

Desarrollar y aplicar métodos de BMF que puedan aplicarse a problemas de tipo *big data*. En particular:

- Eficientes y paralelizables
- Escala lineal, sobre todo con la cantidad de muestras
- Aplicables on-line

Para lograr lo anterior, adaptamos técnicas de *dictionary learning* al caso binario, varias de las cuales que cumplen con todos estos requisitos.

Dictionary learning

- En lugar de $X \approx UV^T$, usaremos $X \approx DA$ donde...
- $D \in \{0,1\}^{m \times p}$ es un *diccionario*: una colección de p patrones representativos de las columnas de $X \in \{0,1\}^{m \times n}$
- $A \in \{0,1\}^{p \times n}$ es una matriz de coeficientes: su rol es expresar las columnas de X como combinación lineal de las columnas de D.
- La interpretación de los *factores* D y A es muy distinta al caso general de BMF
- Además, se supone que $n \gg p$ y $n \gg m$; no se supone que los modelos sean de *bajo rango*, es decir, que $p < m$ (puede ocurrir $p > m$).
- Métodos desarrollados en el ámbito de *procesamiento de señales*, en particular, de imágenes (Olshausen, 1997)
- Motivación original: representación adaptiva de datos; ir más allá de Fourier o Wavelets; lograr modelos *esparsos*

Aprendizaje de diccionarios

Idea general

- Problema no convexo; sólo se puede aproximar su solución
- Casi todos los métodos existentes aproximan la solución iterando dos etapas hasta converger a un mínimo local:

$$\begin{aligned} A^{(t+1)} &= \arg\min_A \{f(D^{(t)}, A) + g(A)\} \\ D^{(t+1)} &= \arg\min_D \{f(D, A^{(t+1)}) + g(A^{(t+1)})\} \end{aligned}$$

- La primera se llama *actualización de coeficientes* y la segunda *actualización de diccionario* (aunque a veces también modifica A)
- Lo usual es buscar coeficientes A que sean *esparsos*, es decir, que tengan pocos elementos no nulos. Esto es una forma de *regularización*.
- Ahora veremos uno de los más simples.

MP: Matching Pursuit

Representación de datos en términos de un diccionario

Data: vector to encode x, dictionary D, maximum residual norm ϵ, maximum coefficients weight $h_{\text{máx}}$

Result: Coefficients vector a

Set iteration $t \leftarrow 0$, residual $r^{(0)} \leftarrow x$, initial Set $g^{(0)} \leftarrow D^{\intercal} r^{(0)}$, $G \leftarrow D^{\intercal} D$;

while $\left\| r^{(t)} \right\| \geq \epsilon$ *and* $h(a) < h_{\text{máx}}$ **do**

$i = \arg\text{máx} \left\{ g^{(t)} \right\}$;

$\Delta \leftarrow D_{:i}^{\intercal} r^{(t)}$;

$a_i^{(t+1)} \leftarrow a_i^{(t)} + \Delta$;

$r^{(t+1)} \leftarrow r^{(t)} - \Delta D_{:i}$;

$g^{(t+1)} \leftarrow g^{(t)} - \Delta G_{:i}$;

$t \leftarrow t + 1$;

end

return $a \leftarrow a^{(t)}$;

Aprendizaje de diccionarios binarios

- Ahora veremos cómo adaptar métodos como el anterior al caso binario
- También veremos dos métodos distintos de actualización de diccionarios, inspirados en algoritmos existentes para el caso real.
- Pero antes, un poco de sopa ...

Notación

- $h(X) = \sum_{ij} X_{ij}$ es el peso de Hamming de X
- Si a y b son dos operandos binarios tenemos que $a \vee b$, $a \oplus b$ y $a \wedge b$ son la suma Booleana (OR), la suma modulo 2 (XOR) y el producto (AND) (tanto en álgebra de Boole como modulo 2)
- para dos vectores x e y y dos matrices A y B, los productos interno, externo, matriz-vector y matriz-matriz son los usuales, $a^\intercal b$, $ab^\intercal$, Ax, etc.
- El producto interno *booleano* $x^\intercal \circ y = \bigvee x_i \wedge y_i$. El elemento C_{ij} del producto booleano de matrices es $C = A \circ B$ el producto booleano entre la fila i-ésima de A y la columna j-ésima de B.
- El producto interno modulo-2 vectores es $x \otimes y = (x + y) \mod 2$. El producto entre matrices se define de manera análoga al Booleano.

BMP: Binary Matching Pursuit

Data: sample x, dict. D, initial coeffs. $a^{(0)}$, $h_{\text{máx}}$, $w_{\text{máx}}$.
Result: Coefficients a such that $Da \approx x$
Set iteration $t = 0$, coefficients $a^{(0)} = a_0$,residual $r^{(0)} = x \oplus D \otimes a^{(0)}$;
Set $G \leftarrow D^{\intercal} \otimes D$, $g^{(0)} \leftarrow D^{\intercal} r^{(0)}$;
while $h(r^{(t)}) \geq w_{\text{máx}}$ **and** $h(a^{(t)}) < h_{\text{máx}}$ **do**
    $k \leftarrow \arg\text{máx}_l \{ \, |g_l^{(t)}| \, / \, \|D_{:l}\|_0 \, \}$;
    if $g_k^{(t)} = 0$ **then**
        return $a \leftarrow a^{(t)}$;
    end
    $r^{(t+1)} \leftarrow r^{(t)} \oplus D_{:k}$;
    if $h(r^{(t+1)}) \geq h(r^{(t)})$ **then**
        return $a \leftarrow a^{(t)}$;
    end
    $a_k^{(t+1)} \leftarrow 1 - a_k^{(t)}$, $\Delta \leftarrow a_k^{(t+1)} - a_k^{(t)}$;
    $g^{(t+1)} \leftarrow g^{(t)} - \Delta G_{:k}$;
end
return $a \leftarrow a^{(t)}$;

MOB: Method Of Binary Directions

MOD: Method of Optimal Directions:

Ejecuta un paso de gradiente proyectado del problema

$$\min_{D} = \|X - DA\|_F^2, \quad \text{s.t.} \quad \|D_{:j}\|_2 \leq 1$$

para A fijo

$$\begin{aligned} D_r^{(t+1)} &= U_r / \max\{1, \|U_r\|_2\}, \\ U &= X(A^{(t+1)})^\intercal \left(A^{(t+1)}(A^{(t+1)})^\intercal \right)^{-1} \end{aligned}$$

MOB: Method of Binary Directions:

Ejecuta un paso de máximo descenso del problema

$$\min_{D} = h(X - DA)$$

para A fijo:

$$D_{ir}^{(t+1)} = \mathbf{1}\left(\frac{h(EA_{r:}^\intercal)}{h(A_{r:})} > \frac{1}{2} \right).$$

PROXIMUS

aproximación de rango 1 de una matriz binaria X

Data: matrix $X \in \{0,1\}^{m \times n}$, $u^{(0)} \in \{0,1\}^m$, $v^{(0)} \in \{0,1\}^n$
Result: Vectors u, v so that $X \approx uv^\intercal$
Set iteration $k = 0$;
repeat

$$u_i^{(t+1)} \leftarrow \mathbf{1}\left(X_{i:}v^{(t)} > h(v^{(t)})/2\right),\ i = 1, \ldots, m\ ;$$

$$v_j^{(t+1)} \leftarrow \mathbf{1}\left(X_{:j}^\intercal u^{(t+1)} > h(u^{(t+1)})/2\right), j = 1, \ldots, n\ ;$$

$$k \leftarrow k + 1\ ;$$

until $u^{(t+1)}(v^{(t+1)})^\intercal = u^{(t)}(v^{(t)})^\intercal$;

Theorem

The output (u, v) of the Proximus Algorithm is a local optimum of the problem $\min h(X - uv^\intercal)$.

K-SVD vs. K-PROX

PROXIMUS como proxy de SVD binaria

Update de K-SVD:

for $r = 1, \ldots, p$ **do**

$J \leftarrow \{j : A^{(t)}_{rj} \neq 0\}$;

$R \leftarrow X_{:J} - D^{(t)} A^{(t)}_{:J} + D^{(t)}_{:r} A^{(t)}_{rJ}$;

$U \Sigma V^{\intercal} \leftarrow \mathrm{SVD}(R)$;

$(D^{(t+1)}_{:r}, A^{(t+1)}_{rJ}) \leftarrow (U_{:1}, V_{:1})$;

end

Update de K-PROX:

for $r = 1, \ldots, p$ **do**

$J \leftarrow \{j : A^{(t)}_{rj} \neq 0\}$;

$R \leftarrow X_{:J} \oplus D^{(t)} A^{(t)}_{:J} \oplus D^{(t)}_{:r} A^{(t)}_{rJ}$;

$(D^{(t+1)}_{:j}, A^{(t+1)}_{j:}) \leftarrow \mathrm{Proximus}(R)$;

end

Selección del mejor modelo

En qué se parecen el modelo verdadero y Superman?

- Ahora un poco de evangelización
- Es muy común es asumir un *modelo subyacente* a los datos como un postulado de la *verdad*; ese dogma suele dar lugar a confusiones
- Salvo en casos muy particulares, el modelo real *no existe* o no es computable
- Seamos más pragmáticos: los modelos son herramientas, nos sirven para analizar los datos; no son ni ciertos ni falsos
- Son más o menos útiles, según el problema, según los datos

Minimum Description Length

A buen entendedor pocas palabras

- La premisa que usaremos es: el mejor modelo es aquel que logra extraer la mayor cantidad de redundancia de los datos
- Esto se inspira en el concepto de complejidad de Kolmogorov (pero no voy a entrar en eso)
- Una forma de medir esa propiedad es a través de la *compresión*: mientras más redundancia le quitamos a los datos, menos símbolos (o palabras) tenemos que usar para describirlos.
- La implementación formal del concepto anterior se denomina Minimum Description Length principle (for model selection)

MDL en la práctica

Cómo implementar MDL eficientemente?

Hay que resolver dos cosas:

- medir el largo de la descripción comprimida de P dado un modelo
- hacer una búsqueda eficiente de los posibles modelos candidato

Cálculo de largo de código

Enumerative coding

- Para lograr una descripciòn completa, debe transmitirse la representación de los datos dado el modelo M; esto cuesta $L(X|M)$ bits
- También debe transmitirse al propio modelo M, usando $L(M)$ bits
- en ambos casos la compresiòn se simula; sólo se calculan los largos en bits
- Para eso se usa la denominada "codificación de Shannon" $L(X|M) = -\log P(X|M)$ y $L(M) = -\log_2 Q(M)$ bajo ciertos modelo probabilísticos P, M.
- Resta ver cómo evaluar $-\log P()$ y $-\log Q()$ eficientemente
- En el caso binario es fácil. Para un vector binario cualquiera x, de largo m calculamos $L(x) = \lceil \log_2 m \rceil + \log_2 \binom{m}{r}$
- Esto lo aplicamos a cada fila de A, cada columna de D, y cada fila de E por separado. Describiendo esas tres matrices describimos completamente $X = D \otimes A \oplus E$.

Buscar candidatos

Forward selection

- Tenemos $L(X) = L(A) + L(D) + L(E)$
- Cómo buscamos los mejores A, D?
- Para p fijo, utilizamos alguno de los algoritmos vistos (MOB, K-PROX)
- Podríamos aprender un diccionario para cada p posible, pero eso no es práctico ni rápido . . .
- Lo que hacemos es comenzar con un p pequeño arbitrario, aprender A y D para ese p, incrementar p, y así hasta que $L(X)$ no aumente
- En general esto funciona muy bien, especialmente con *warm restarts*

Resultados y conclusiones

Vemos en el paper...

Gracias

Preguntas?