ILI-286: Computación Científica II
Laboratorio #1
"Computación numérica de vectores propios aplicados a PCA"

Claudio Torres Axel Símonsen Martín Villanueva

23 de octubre 2015

PCA: Principal Component Analysis

Principal Component Analysis tiene por objetivo encontrar una transformación ortogonal que convierte a un conjunto de observaciones (posiblemente correlacionadas), en un conjunto de valores de variables no correlacionadas, llamadas los componentes principales. Considere una matriz de datos $\mathbf{X}_{m\times n}$ que contiene m filas con observaciones de un fenómeno descrito por n variables o características. Lo que se quiere es encontrar una matriz de cambio de base (o rotación) $\mathbf{B_{n\times p}}$ tal que $\mathbf{XB}$ permita expresar la data, con *mínima redundancia* (minimizando correlación entre cada característica) y *máxima señal* (maximizando la varianza de cada característica).

Si se estandariza $\mathbf{X}$ restando a cada característica (columna) su media, no es difícil notar que la matriz de covarianzas entre las características puede ser escrita como

$$\mathbf{\Sigma}_X = \frac{1}{n-1}\mathbf{X}^T\mathbf{X} \tag{1}$$

donde cada elemento Σ_{ij} representa la covarianza entre las características i y j, y lo elementos de la diagonal Σ_{ii} corresponde a la varianza de la característica i. Luego otra manera de plantear el problema es, encontrar una nueva representación $\mathbf{XB} = \mathbf{Y}$ tal que la matriz de covarianza $\mathbf{\Sigma}_Y$ sea diagonal.

Si se tiene en cuenta la factorización SVD de $\mathbf{X} = \mathbf{U\Sigma V}^T$, y se elige como matriz de cambio de base a $\mathbf{V}$, entonces $\mathbf{Y} = \mathbf{XV} = \mathbf{U\Sigma}$. Y luego la matriz de covarianzas cumple

$$\begin{aligned}\mathbf{\Sigma}_Y &= \frac{1}{n-1}\mathbf{Y}^T\mathbf{Y}\\ &= \frac{1}{n-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U\Sigma}\\ &= \frac{1}{n-1}\mathbf{\Sigma}^T\mathbf{\Sigma}\\ &= \frac{1}{n-1}\mathbf{\Sigma}^2\end{aligned}$$

que corresponde a una matriz diagonal. Por lo tanto la matriz de cambio de base con los componentes principales, es precisamente $\mathbf{V}$ de la SVD de $\mathbf{X}$. Adicionalmente si se toma en cuenta lo siguiente

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= \mathbf{V\Sigma}\mathbf{U}^T\mathbf{U\Sigma V}^T\\ &= \mathbf{V\Sigma}^2\mathbf{V}^T\end{aligned}$$

corresponde a la *Eigendecomposition* de $\mathbf{X}^T\mathbf{X}$, vale decir, los vectores singulares derechos $\mathbf{V}$ de $\mathbf{X}$, corresponden a los vectores propios de $\mathbf{X}^T\mathbf{X}$. Por lo tanto ,el cálculo de los componentes principales puede entonces ser abordado de dos maneras: **(1)** Computar la factorización SVD de $\mathbf{X}$ y **(2)** Calcular los vectores propios de $\mathbf{\Sigma}_X$. El problema con el primer enfoque, es que la complejidad computacional del cálculo de la SVD es $O(n^3)$ y que necesariamente calcula todos los componentes principales (usualmente sólo son necesarios los primeros).

Dataset

Se provee del dataset `arcene.npy` en formato binario de `NumPy`. Este consiste de 700 mediciones realizadas a pacientes con cáncer, cada una de 2500 características correspondientes a espectros de masa. Este dataset se ocupa para generar algoritmos de clasificación binaria (saber cuando un paciente tiene cáncer o no). El problema es que el desempeño de estos algoritmos depende en gran medida del número de características, y como es de sospechar, no todas las características son realmente útiles para discriminar entre ambos casos. Por lo tanto son muy útiles los algoritmos de *reducción de dimensionalidad* para abordar estos problemas.

Desarrollo

Para el desarrollo de la experiencia, suponga en todo momento que la matriz de entrada de los algoritmos es simétrica. Esto pues se trabajará con la matriz de covarianzas $\mathbf{\Sigma}_X$, que precisamente tiene esa propiedad.

1. Implemente los algoritmos Power Iteration y Rayleigh Quotient Iteration. Estimar la complejidad computacional en cada caso.

2. Se propone el siguiente algoritmo para el cálculo de los primeros k valores y vectores propios

Algorithm 1 Naive k-first eigen(vector/value) finder

procedure kEigenFinder(A,p,k)
  $\lambda_0 \leftarrow 0$
  $v_0 \leftarrow [0 \cdots 0]^T$
  for $i = 0 : k-1$ **do**
    $A \leftarrow A - \lambda_i v_i v_i^T$
    $(\lambda_{i+1}, v_{i+1}) \leftarrow PowerIteration(A, p)$

   (a) Compruebe teóricamente [1] la correctitud del algoritmo 1.

   (b) Estime su complejidad computacional.

   (c) Implementarlo adecuadamente (instrucciones vectorizadas).

3. Un problema con el algoritmo anterior, es que debe realizar *Power Iteration* para cada cómputo de un nuevo valor y vector propio. El algoritmo 2 se propone como una solución a ese problema

Algorithm 2 Clever k-firsts eigen(vector/value) finder

procedure kEigenFinder++(A,p,k)
  $A^p \leftarrow MatrixPower(A, p)$
  $\lambda_0 \leftarrow 0$
  $v_0 \leftarrow [0 \cdots 0]^T$
  $x_0 \leftarrow RandomVector(n)$
  for $i = 0 : k-1$ **do**
    $A^p \leftarrow A^p - \lambda_i^p v_i v_i^T$
    $v_{i+1} \leftarrow A^p x_0$
    $v_{i+1} \leftarrow v_{i+1}/|v_{i+1}|$
    $\lambda_{i+1} \leftarrow v_{i+1}^T A v_{i+1}$

   (a) Compruebe teóricamente [2] la correctitud del algoritmo 2.

   (b) Estime su complejidad computacional.

   (c) Implementarlo adecuadamente (instrucciones vectorizadas), e introduciendo los cambios que estimen convenientes para sobrellevar los problemas numéricos/computacionales que pudiesen aparecer.

4. Un conocido algoritmo para calcular todos los valores y vectores propios es *Unshifted QR*. Sin embargo en problemas como los que estamos abordando, sólo interesan los primeros k valores y vectores propios.

[1] **Hint:** analice las propiedades $A - \lambda_i v_i v_i^T$ para $Av_i = \lambda_i v_i$ y A simétrica.
[2] **Hint:** analice las propiedades de $(A - \lambda_i v_i v_i^T)^p$ para $p \in \mathbb{N}$, $Av_i = \lambda_i v_i$ y A simétrica.

(a) Indique modificaciones sobre *Unshifted QR* para que calcule sólo los primeros k valores y vectores propios.

(b) Estime su complejidad computacional.

(c) Implementarlo adecuadamente (instrucciones vectorizadas).

5. Ejecute su implementación de *Power Iteration* y *Rayleigh Quotient Iteration* sobre la matriz de covarianzas $\mathbf{\Sigma}_X$, midiendo el tiempo de ejecución y memoria (`timeit` y `memit`). Comparar resultados obtenidos y concluir al respecto.

6. Ejecute su implementación de los algoritmos 1, 2 y *Modified Unshifted QR* sobre la matriz de covarianzas $\mathbf{\Sigma}_X$, midiendo el tiempo de ejecución y memoria (`timeit` y `memit`) para valores $k = \{10, 100, 1000\}$. Concluir al respecto. **Observación:** La elección del parámetro p queda a su criterio. Indicar tal criterio.

7. Elegir el algoritmo que usted más estime conveniente para calcular todos los valores y vectores propios de $\mathbf{\Sigma}_X$ justificando su elección.

(a) Proyecte los datos sobre los dos componentes principales. Realice un gráfico de los datos bajo esta nueva representación, e indique el porcentaje de varianza que explican estos datos [3].

(b) Repita el mismo procedimiento anterior pero con los tres componentes principales.

(c) ¿Cuántos componentes son necesarios para que se explique un 90% de la varianza en los datos?. Concluir al respecto.

[3] Explained Variation, corresponde a varianza total explicada por las componentes, divida en la varianza total de la data: $\sum_{i=1}^{k} \Sigma_{ii} / \sum_{i}^{n} \Sigma_{ii}$, con Σ_{ii} entrada i-ésima en la diagonal de la matriz de covarianza.

Instrucciones:

(a) La estructura del laboratorio es la siguiente:

   (a) Título, nombre(s) de estudiante(s), email(s) y rol(s).
   (b) Introducción.
   (c) Desarrollo y análisis de resultados.
   (d) Conclusiones.
   (e) Referencias.

(b) El laboratorio debe ser realizado en `IPython` notebook (con `Python2` o `Python3`).

(c) Se evaluará la correcta utilización de librerias `NumPy` y `SciPy`, así como la correcta implementación de algoritmos vectorizados cuando se indique.

(d) El archivo de entrega debe denominarse Lab1-apellido1-apellido2.tar.gz, y debe contener un directorio con todos los archivos necesarios para ejecutar el notebook, junto con un archivo README indicando explícitamente la versión utilizada de Python.

(e) El descuento por día de atraso será de 30 puntos, con un máximo de 1 día de atraso. No se recibirán entregas después de este día.

(f) El trabajo es personal o en grupos de a 2, no se permite compartir código, aunque sí se sugiere discutir aspectos generales con sus compañeros. Copias exactas serán sancionadas con nota 0.

(g) El no seguir estas instrucciones, implica descuentos en su nota obtenida.