SPEC: Un Módulo para Transformaciones Espectrales de Imágenes en Google Earth Engine

José Luis Silván Cárdenas

Enero 2025

Abstract

Las transformaciones espectrales son una herramienta básica de procesamiento de imágenes de percepción remota en la que los píxeles de la imagen son tratados como vectores y las transformaciones como matrices. Dichas transformaciones se basan en la estadística de la imagen, emplean el algebra lineal y la descomposición en eigenvectores y eigenvalores. SPEC es un módulo que fue desarrollado en el CentroGeo para la transformación de imágenes multiespectrales en Google Earth Engine (GEE). Este manual describe su filosofía, implementación y uso en códigos de GEE. El módulo implementa transformaciones para reducir redundancia y el ruido en las bandas, pero también para realzar los cambios entre dos imágenes y reducir el espacio de características de una clasificación para aumentar la separabilidad entre clases.

1 Introducción

A diferencia de la transformada de Fourier o la transformada Wavelets, que operan sobre las coordenadas espaciales, las transformaciones espectrales operan a lo largo de la coordenada espectral de las imágenes (bandas espectrales). En ese sentido, los píxeles de una imagen multiespectral se consideran como vectores multidimensionales, los cuales pueden ser transformados por traslación, rotación y escalamiento en el espacio multidimensional. Dichas transformaciones se denominan transformaciones espectrales lineales porque son representadas por matrices, donde para estimar sus coeficientes se emplea la optimización y el muestreo estadístico.

El objetivo del módulo SPEC es precisamente la aplicación de este tipo de transformaciones en el procesamiento de imágenes de percepción remota en la nube. Fue desarrollado con propósitos de enseñanza y, como tal, se pone a disposición de alumnos y académicos interesados en el procesamiento de datos de percepción remota disponibles en GEE.

SPEC incluye actualmente las transformaciones de Análisis de Componentes Principales (Principal Component Analysis [10]), Mínima Fracción de Ruido (Minimum Noise Fraction [5]), Detección de Alteración Multivariada (Multivariate Alteration Detection [8]), tanto en su versión original, como la versión reponderada iterativamente [9], y el Análisis de Discriminantes Lineales (Linear Discriminant Analysis [3]). También se hacen disponibles una serie de funciones básicas que permiten complementar el análisis, incluyendo la reconstrucción de imágenes a partir de los componentes seleccionados para aplicaciones de realce, filtrado y calibración relativa de imágenes, entre otros.

1.1 Uso del Modulo

Para usar SPEC es necesario contar con una cuenta de GEE y acceder al editor de códigos en `https://code.earthengine.google.com/`. En su programa de GEE incluya la siguiente línea al principio[1]:

```
var spec = require("users/jsilvan/modules/spec");
```

Lo siguiente es establecer la muestra aleatoria para el cálculo de las estadísticas de primer y segundo orden en la que se basan las transformaciones. Para ello se usa la función `.SetSample(region,`*numPixels,scale)*, como en el siguiente ejemplo[2]:

```
spec.SetSample(aoi,1000);
```

En este ejemplo `aoi` es una geometría que define la región de interés dentro de la cual se seleccionan aleatoriamente 1000 píxeles para el cálculo del vector de medias y la matriz de covarianza.

En general, la geometría se puede especificar como objetos tipo `Geometry`, `ee.Geometry` o `ee.FeatureCollection` que intersectan la imagen a procesar. En el caso de LDA, que usa regiones clasificadas para indicar la muestra de entrenamiento, el parámetro `region` debe ser un `ee.FeatureCollection`. Para los otros casos, la región es opcional, ya que por defecto se considera la cobertura de la imagen de entrada como

[1] Aquí el nombre de la variable `spec` es arbitrario, pero conviene usar un nombre que recuerde a qué módulo hace referencia la variable.

[2] En todos los ejemplos del documento se resaltan las funciones del módulo en tono café.

Función	Descripción
`.SetSample(region,`*`numPixels,scale`*`)`	Define el muestreo de la imagen
`region`	región de muestreo
`numPixels`	tamaño de la muestra
`scale`	tamaño de la unidad de muestreo
`.CenterBands()`	Centra las bandas en media nula
`.PCA(image,`*`minCPV`*`)`	Análisis de Componentes Principales
`image`	imagen de entrada
`minCPV`	porcentaje de varianza acumulada
`.MNF(image,`*`minSNR,Nregion`*`)`	Fracción de Ruido Mínima
`image`	imagen de entrada
`minSNR`	relación señal a ruido mínima
`Nregion`	región de muestreo del ruido
`.MAD(xImage,yImage,`*`term,minCorr`*`)`	Detección de Alteración Multivariada
`xImage`	imagen de entrada X
`yImage`	imagen de entrada Y
`term`	término a calcular "U","V","M" o "Cal"
`minCorr`	correlación mínima entre componentes
`.WMAD(xImage,yImage,wImage,...` *`term,minCorr`*`)`	MAD ponderada
`wImage`	probabilidad de no cambio
`.IRMAD(numIter,xImage,yImage,...` *`term,minCorr`*`)`	MAD iterativamente reponderada
`numIter`	número de iteraciones
`.LDA(image,`*`minSep`*`)`	Análisis de Discriminantes Lineales
`image`	imagen de entrada
`minSep`	separabilidad mínima de clases

Table 1: Lista de funciones públicas con argumentos (formato de *italicas* significa opcional)

región de muestreo, y un tamaño de muestra de 1000 píxeles de 100 metros de resolución. En la práctica se recomienda usar la resolución nativa de la imagen en lugar de la resolución por defecto. Para determinar la resolución nativa use la siguiente línea:

```
var scale = band.projection().nominalScale();
```

1.2 Filosofía

La filosofía del módulo se basa en el encadenamiento de resultados mediante el operador "." de Java y a partir de unas cuantas funciones públicas. Las funciones que se acceden como `spec.`*`func`* se denominan funciones públicas y se listan en la Tabla 1. Éstas incluyen las transformaciones espectrales y un par de funciones inicializadoras.

Por otro lado, las funciones básicas sólo se hacen disponibles a través del resultado de una función pública y siguiendo ciertos pasos lógicos de procesamiento. El listado completo de las funciones básicas se presenta en la Tabla 2.

Por ejemplo, para poder calcular los eigenvalores y eigenvectores, antes es necesario calcular la matriz de covarianza, lo que a su vez requiere centrar las bandas en media nula, de tal manera que resulta necesario encadenar las llamadas de funciones como sigue:

```
var centeredImage = spec.CenterBands(image);
var imageWithCov = centeredImage.setCovMatrix();
var imageWithEigs = ImageWithCov.setEigens();
```

Si los resultados intermedios no son necesarios, podemos emplear la forma compacta:

```
var imageWithEigs = spec.CenterBands(image)
  .setCovMatrix().setEigens();
```

Así, las funciones básicas proporcionan un mecanismo flexible para explorar resultados intermedios o para implementar otras transformaciones. En el resto del documento se describen brevemente la teoría detrás de cada transformación espectral, su implementación y ejemplos de aplicación en procesamiento de imágenes multiespectrales.

Función	Descripción
`.setCovMatrix`	agrega la matriz de covarianza
`.setCovMatrixN`	agrega la matriz de covarianza del ruido
`.setCovMatrixC`	agrega la matriz de covarianza de clases
`.setMADMatrices`	agrega las matrices de bloque para MAD
`.setEigens`	agrega eigenvalores y eigenvectores
`.setCPV`	agrega vector de porcentaje de varianza acumulada
`.setSNR`	agrega vector de relación señal a ruido
`.setCorr`	agrega vector de correlación
`.setSep`	agrega vector de separabilidad
`.getSep`	obtiene separabilidad de clases en componentes LDA
`.getImSep`	obtiene separabilidad en bandas de la imagen
`.getMD`	obtiene la matriz de distancia de Mahalanobis
`.getBD`	obtiene la matriz de distancia de Bhattacharyya
`.getJD`	obtiene la matriz de distancia de Jeffries-Matusita
`.getInvMatrix`	obtiene la matriz de reconstrucción
`.rotateBands`	aplica la matriz de transformación
`.rotateBandsMAD`	aplica la transformación MAD
`.iterateMAD`	recalcula los componentes MAD normalizados
`.asCentered`	considera como imagen centrada
`.normalizePCs`	componentes con variazas unitarias
`.denormalizePCs`	restaura las varianzas originales
`.addChiSqrBand`	agrega bandas `ChiSqr` y `noChangeProb`
`.replacePC`	reemplaza componente normalizada
`.restoreIm`	reconstruye imagen a partir de los componentes

Table 2: Lista de funciones básicas

2 Análisis de Componentes Principales

El análisis de componentes principales, o PCA por sus siglas en inglés, se puede formular desde el punto de vista estadístico como sigue [10, 6]:

Dada k variables aleatorias correlacionadas $\mathbf{X} = [X_1, X_2, ..., X_k]^T$ (e.g., las bandas de una imagen multiespectral) tales que:

$$E\{\mathbf{X}\} = \mathbf{0} \tag{1}$$

$$Cov\{\mathbf{X}\} = \mathbf{\Sigma_x} \tag{2}$$

Se requiere encontrar la matriz $\mathbf{A}$ de la transformación $\mathbf{Z} = \mathbf{A}^T\mathbf{X}$, tal que las variables $\mathbf{Z} = [Z_1, Z_2, ..., Z_k]^T$ no estén correlacionadas entre sí. En otras palabras, que $Cov\{\mathbf{Z}\} = \mathbf{D}$ sea una matriz diagonal, cuyos elementos diagonales correspondan a las varianzas de las componentes transformadas.

El problema de PCA equivale a encontrar los eigenvalores ($\mathbf{diag}(\mathbf{D})$) y eigenvectores ($\mathbf{A}$) de la matriz de covarianza $\mathbf{\Sigma_x}$, o en términos matemáticos, resolver la ecuación característica:

$$\mathbf{\Sigma_x A} = \mathbf{AD} \tag{3}$$

La solución de dicha ecuación en GEE se realiza mediante la función `.eigen()`.

2.1 Implementación en SPEC

Para calcular PCA en SPEC se emplea la función `.PCA(image,`*`minCPV`*`)` que acepta opcionalmente un umbral de porcentaje de varianza acumulada mínima. La imagen de entrada es primero convertida a una imagen de array centrada respecto a su media y luego se rota aplicando la matriz de eigenvectores. El umbral de varianza determinará el número de componentes (bandas) que tendrá la imagen de salida, de tal forma que el porcentaje de varianza acumulada será al menos el valor indicado por `minCPV`.

Por ejemplo:

```
var PC90 = spec.PCA(image,90);
```

retiene las componentes que explican al menos el 90% de la varianza en la imagen de entrada. Para retener todos los componentes se puede omitir el segundo argumento, ya que su valor por defecto es 100%.

Para visualizar un compuesto RGB de componentes, es recomendable reescalarlos. Una opción es normalizarlos mediante la función `.normalizePCs`, como en el siguiente ejemplo:

```
var PC90normalized = PC90.normalizePCs();
var visParams = {min:-5,max:5,bands:["PC1","PC2","PC3"]};
Map.addLayer(PC90normalized,visParams,"PC1,PC2,PC3");
```

Atributo	Descripción	Función
`meanVector`	vector de medias	`.CenterBands()`
`covMatrix`	matriz de covarianza	`.setCovMatrix()`
`eigenValues`	eigen valores	`.setEigens()`
`eigenVectors`	eigen vectores	`.setEigens()`
`compStdVector`	desviación estandar de componentes	`.setEigens()`
`CPV`	porcetaje de variance acumulada	`.setCPV`
`numBands`	número de bandas de entrada	`.rotateBands`
`numComps`	número de componentes de salida	`.rotateBands`

Table 3: Lista de propiedades que se agregan a la imagen PCA

Una representación de la imagen en términos de las componentes principales se obtiene con la función `.restoreIm(`*`bandNames`*`)`. Por ejemplo, la siguiente línea de código genera una aproximación con los componentes del ejemplo previo:

```
var img90 = PC90.restoreIm();
```

La imagen PC cuenta con toda la información requerida para recuperar la imagen original, salvo los nombres de las bandas originales, lo cual se puede especificar como argumento de `.restoreIm`.

La Tabla 3 describe las propiedades que se acumulan en la imagen de componentes de PCA y que pueden consultarse con la función `.get` de GEE. También se indica la función que genera dicha propiedad (vea la Tabla 2 para el listado de funciones), mismas que pueden emplearse para calcular la descomposición paso a paso.

Por ejemplo, para obtener todas las componentes normalizadas[3] uno puede ejecutar los siguientes pasos:

```
var PC = spec.CenterBands(image) // imagen con media nula
  .setCovMatrix()  // calcula CovMatrix
  .setEigens() // calcula los eigens de covMatrix
  .rotateBands() // aplica matrix de rotacion
  .normalizePCs(); // varianzas unitarias
```

Las funciones básicas también se pueden usar para calcular directamente alguno de los parámetros de la Tabla 3. Por ejemplo, para calcular los CPV directamente sin calcular la transformación, usamos:

```
var CPV = spec.CenterBands(image) // imagen con media nula
  .setCovMatrix() // calcula covMatrix
  .setEigens() // calcula los eigens de covMatrix
  .setCPV() // calcula CPV
  .get('CPV'); // extrae CPV
```

2.2 Aplicación en Fusión Pansharpening

Una aplicación típica de PCA es la fusión de una imagen multiespectral con una banda pancromática de mayor resolución espacial. El resultado es una imagen multiespectral a la misma resolución que la pancromática.

El siguiente código implementa la fusión *pansharpen* entre la imagen multiespectral `image_ms` y la pancromática `image_pan`:

```
var image_fused = spec.PCA(image_ms) // componentes ms
  .normalizePCs() // varianzas unitarias
  .replacePC(1,image_pan,true) // reemplaza componente PC1
  .denormalizePCs() // restaura varianzas
  .restoreIm(bandNames); // reconstruye la imagen
```

Para reemplazar el componente 1 con la banda pancromática normalizada se emplea la función `.replacePC(numPC,band,`*`normalize`*`)`. El tercer argumento indica si la banda debe normalizarse, lo cual en este caso es `true`. La normalización de la banda consiste, no solo en sustraer su media y dividir entre su varianza, sino además multiplicar por el signo de la correlación con el componente reemplazado. Esto último es necesario ya que la orientación de los componentes puede estar invertida.

La figura 1 muestra las imágenes relevantes del ejemplo anterior. Es notable que el componente PC1 está invertido respecto a la banda pancromática, lo cual, como ya se dijo, es tomado en cuenta durante el reemplazamiento de la banda. Es visible también la mayor nitidez de la imagen fusionada respecto a la imagen multiespectral.

[3]Las funciones de trasformación (incluyendo `.PCA`) entregan los componentes sin normalizar.

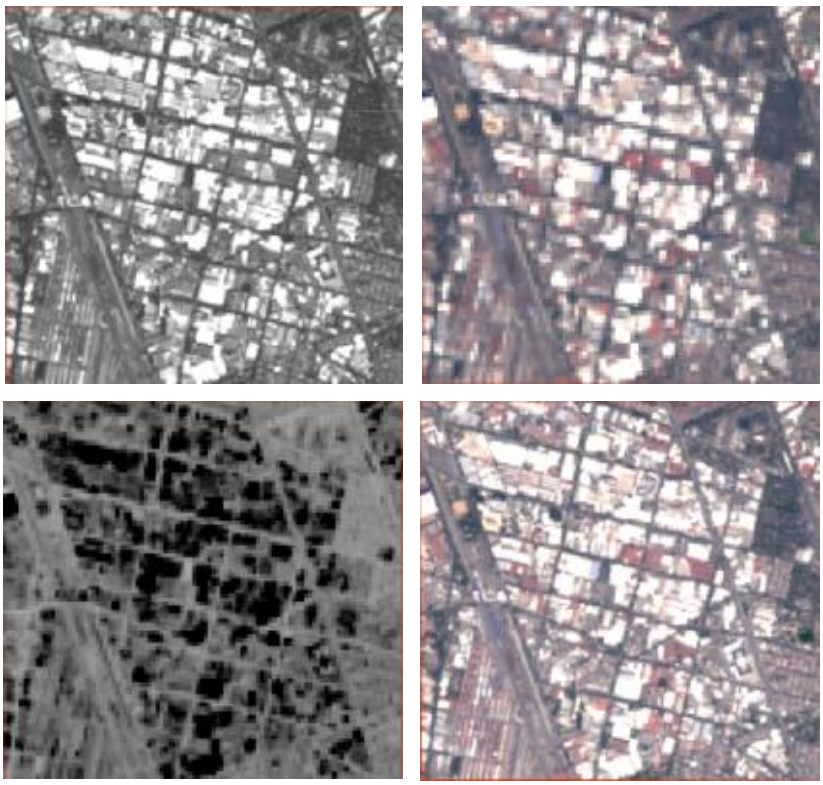

Figure 1: En orden lexicográfico se muestra: imagen pancromática, imagen multiespectral, componente PC1 e imagen fusionada.

3 Mínima Fracción de Ruido

La transformada de mínima fracción de ruido o MNF, por sus siglas en inglés, es similar a PCA pero ahora los componentes se ordenan por mínima fracción de ruido [5].

Los píxeles de la imagen multiespectral se modelan como $\mathbf{X} = \mathbf{S} + \mathbf{N}$, donde $\mathbf{S} = [S_1, S_2, ..., S_k]^T$ y $\mathbf{N} = [N_1, N_2, ..., N_k]^T$ son las componentes de señal y ruido respectivamente, tales que:

$$E\{\mathbf{S}\} = \mathbf{0}, \qquad Cov\{\mathbf{S}\} = \mathbf{\Sigma_S}, \tag{4}$$

$$E\{\mathbf{N}\} = \mathbf{0}, \qquad Cov\{\mathbf{N}\} = \mathbf{\Sigma_N}, \tag{5}$$

$$E\{\mathbf{X}\} = \mathbf{0}, \qquad Cov\{\mathbf{X}\} = \mathbf{\Sigma}, \tag{6}$$

MNF consiste en la transformación $\mathbf{Z} = \mathbf{A}^T\mathbf{X}$ que maximiza la relación señal a ruido:

$$SNR = \frac{Var\{\mathbf{a}^T\mathbf{S}\}}{Var\{\mathbf{a}^T\mathbf{N}\}} = \frac{\mathbf{a}^T\mathbf{\Sigma a}}{\mathbf{a}^T\mathbf{\Sigma_N a}} - 1 \tag{7}$$

lo que equivale a resolver el eigenproblema generalizado:

$$\mathbf{\Sigma_N A} = \mathbf{\Sigma A D} \tag{8}$$

donde la matriz diagonal $\mathbf{D}$ corresponde a los valores de $1+SNR$ cuando se consideran 1,2,3,... componentes.

Para poder calcular la transformación MNF se requiere estimar la covarianza del ruido. La implementación de SPEC considera una aproximación que emplea una imagen de ruido construida a partir del laplaciano de la imagen como en [2], es decir:

$$N \sim X * \begin{bmatrix} 0 & -0.25 & 0 \\ -0.25 & 1 & -0.25 \\ 0 & -0.25 & 0 \end{bmatrix} \tag{9}$$

3.1 Implementación en SPEC

Dado que GEE no cuenta con una solución explícita para el eigenproblema generalizado, la implementación en SPEC resuelve dos veces el problema clásico empleando la función `.eigen()`.

El algoritmo de MNF se basa en [2] y se puede resumir en los siguientes pasos:

Atributo	Descripción	Función
`meanVector`	vector de medias	`.CenterBands`
`covMatrix`	matriz de covarianza	`.setCovMatrix`
`eigenValues`	eigen valores	`.setEigens`
`eigenVectors`	eigen vectores	`.setEigens`
`compStdVector`	vector de desviación estandar	`.setEigens`
`noiseCovMatrix`	matriz de covarianza del ruido	`.MNF`
`noiseEigenValues`	eigen valores del ruido	`.MNF`
`noiseEigenVectors`	eigen vectores del ruido	`.MNF`
`noiseStdVector`	vector de desviación estandar del ruido	`.MNF`
`noiseKernel`	kernel que genera la imagen de ruido	`.setCovMatrix`
`SNR`	relación señal a ruido por componente	`.setSNR`
`numBands`	número de bandas de entrada	`.rotateBands`
`numComps`	número de componentes de salida	`.rotateBands`

Table 4: Lista de propiedades que se agregan a la imagen MNF

1. Calcular la imagen de ruido
2. Calcular la matriz de transformación a paritr de la imagen de ruido
3. Aplicar la trasformación obtenida a la imagen original
4. Normalizar los componentes
5. Calcular PCA de la imagen del paso anterior

Esto se hace mediante la función `.MNF(image,minSNR,Nregion)`, la cual acepta opcionalmente dos argumentos adicionales a la imagen de entrada. El primero (`minSNR`) determina el número de componentes a retener en términos de su contribución a la relación señal a ruido, y el segundo (`Nregion`) es una geometría que define el área a muestrear en la imagen de ruido. Por defecto, el primer argumento se asume[4] 0, y para el segundo se toma la misma geometría de muestreo de la imagen original (indicada a través de `.SetSamp`).

Por ejemplo, para calcular las componentes MNF usando los valores por defecto, escribimos:

```
var MNFcomp = spec.MNF(image);
```

La imagen MNF incluye las propiedades que se muestran en la Tabla 4. Como es de suponerse, los atributos que tienen el prefijo `noise` se basan en la imagen de ruido.

Adicionalmente a las funciones que emplea PCA (Tabla 3), la implementación de MNF usa una función `.setCovMatrixN(Nregion)` para definir la matriz de covarianza del ruido y el kernel. Sin embargo, el resultado se almacena en la propiedad `covMatrix` para poder reusar la función `.setEigens`, como en la siguiente implementación:

```
var MNFcomp = spec.CenterBands(image)
  .setCovMatrixN() // calcula covMatrix del ruido
  .setEigens() // calcula eigenValues y eigenVectors
  .rotateBands() // aplica la rotacion
  .normalizePCs() // normaliza
  .asCentered() // reinicia para segunda vuelta
  .setCovMatrix() // recalcula covMatrix
  .setEigens() // recalcula eigenValues y eigenVectors
  .rotateBands(numComps,"MNF"); // rotacion final
```

Es importante notar que aunque este código entregará una imagen de los componentes como los que se calculan con la función `.MNF`, ésta no contendrá todos los parámetros de la Tabla 4 debido a que algunos de estos (como `covMatrix`) se reemplazan en la segunda llamada de la misma función. Esto no pasa con la implementación `.MNF` ya que se hacen copias renombradas con el prefijo 'noise' después de la primera llamada de la función, además de que emplea `.setSNR` para determinar el número de componentes de salida.

La imagen de componentes generada mediante `.MNF` habilita además la función `.getInvMatrix()` que calcula la matriz de transformación inversa ($\mathbf{A^{-1}}$) empleada por la función `.restoreIm(bandNames)`. Por lo tanto, no es recomendable la implementación paso a paso, a menos que se esté interesado en los resultados intermedios.

3.2 Reducción de ruido

La aplicación obvia del análisis-síntesis MNF es la reducción de ruido en las imágenes multiespectrales, como en el siguiente ejemplo:

```
var MNFcomps = spec.MNF(image,0)
var denoised = MNFcomps.restoreIm();
```

[4]Es posible optener valores de SNR negativos en casos cuando la varianza del ruido es mucho mayor que la de la señal.

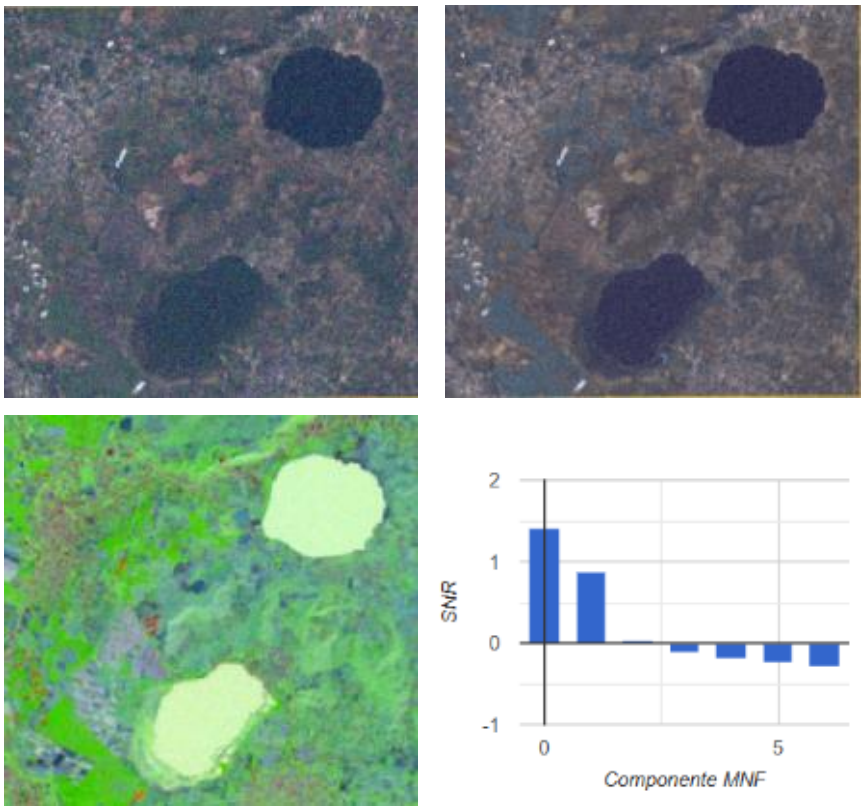

Figure 2: En orden lexicográfico se muestra: imagen original, imagen reconstruida, componentes MNF 1-3 y grafica de SNR por componente.

Para graficar el SNR de los componentes se puede emplear `ui.Chart` como el siguiente ejemplo:

```
var SNR = ee.Array(MNFcomps.get('SNR'));
var chart = ui.Chart.array.values(SNR,0)
  .setChartType('ColumnChart')
  .setOptions({
    vAxis: {title: 'SNR'},
    hAxis: {title: 'Componente MNF'},
  });
print(chart);
```

La Figura 2 muestra el ejemplo de reducción de ruido con la gráfica y una visualización de los componentes MNF de mayor SNR. El cambio más notable entre la imagen original y la reconstruida se puede observar en los cuerpos de agua y zonas de suelo homogéneas.

4 Detección de Alteración Multivariada

La detección de alteración multivariada, o MAD por sus siglas en inglés, es un método de detección de cambios a partir de dos imágenes multiespectrales [8]. Como tal, es insensible a transformaciones lineales de las imágenes de entrada, por lo que compensa algunos de los efectos atmosféricos o de estiramiento de histograma que hayan sido aplicado previamente. También se usa para calibración espectral relativa de una imagen respecto a otra, esto es, para simular las condiciones espectrales de una adquisición en otra [1]. Su formulación es como sigue.

Dadas dos imágenes multiespectrales $\mathbf{X} = [X_1, X_2, ..., X_k]^T$ y $\mathbf{Y} = [Y_1, Y_2, ..., Y_k]^T$, tales que:

$$E\{\begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix}\} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{10}$$

$$Cov\{\begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \end{bmatrix}\} = \begin{bmatrix} \mathbf{\Sigma_X} & \mathbf{\Sigma_{XY}} \\ \mathbf{\Sigma_{YX}} & \mathbf{\Sigma_Y} \end{bmatrix} \tag{11}$$

la transformación MAD consiste en determinar las matrices $\mathbf{A}$ y $\mathbf{B}$ tales que las componentes $\mathbf{U} = \mathbf{A}^T\mathbf{X}$ y $\mathbf{V} = \mathbf{B}^T\mathbf{Y}$ tienen correlación máxima entre bandas coincidentes y nula entre bandas disímiles, es decir:

$$Corr(\mathbf{U}, \mathbf{V}) = \mathbf{D} \tag{12}$$

es una matriz diagonal que contiene los valores de correlación de las componentes $(\mathbf{U}, \mathbf{V})$.

El problema equivale al problema de eigenvalores y eigenvectores generalizados acoplados mediante la matriz diagonal $\mathbf{D}$ y se formula mediante el sistema de ecuaciones:

$$\begin{aligned} \mathbf{\Sigma_{XY}B} &= \mathbf{\Sigma_X AD} \\ \mathbf{\Sigma_{YX}A} &= \mathbf{\Sigma_Y BD} \end{aligned}$$

Alternativamente, empleando matrices en bloques, el mismo se puede escribir como:

$$\begin{bmatrix} \mathbf{0} & \mathbf{\Sigma_{XY}} \\ \mathbf{\Sigma_{YX}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma_X} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma_Y} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \mathbf{D} \tag{13}$$

Mas aún, las componentes MAD para el análisis de cambios consisten en las diferencias

$$\mathbf{M} = \mathbf{U} - \mathbf{V} \tag{14}$$

donde los cambios se detectan mediante el z-score

$$z = \sum_{i=1}^{k} \left(\frac{M_i}{\sigma_{M_i}} \right)^2 \tag{15}$$

y las varianzas de las diferencias σ_{M_i} corresponden a los elementos de la matriz $2(\mathbf{I} - \mathbf{D})$. El z-score sigue una distribución chi-cuadrada de tal forma que la probabilidad de no cambio se expresa como:

$$\mathbf{Pr}(no\,cambio) = 1 - P_{\chi^2,k}(z) \tag{16}$$

Como tal, no es posible reconstruir ambas imágenes a partir de las diferencias $\mathbf{M}$, sólo de los componentes $\mathbf{U}$ y $\mathbf{V}$, es decir, $\mathbf{X} = \mathbf{A}^{-1T}\mathbf{U}$ y $\mathbf{Y} = \mathbf{B}^{-1T}\mathbf{V}$. También es posible recalibrar una imagen (digamos $\mathbf{X}$) en términos de otra de referencia ($\mathbf{Y}$) intercambiando la matriz de reconstrucción. Específicamente:

$$\mathbf{X}_{cal} = \mathbf{B}^{-1}\mathbf{DU} \tag{17}$$

es la versión calibrada de $\mathbf{X}$ relativa a $\mathbf{Y}$. Esto as así porque la matriz $\mathbf{C}$ que minimiza el valor esperado del error $\varepsilon = \mathbf{Y} - \mathbf{C}^T\mathbf{X}$ resulta ser:

$$\mathbf{C}^T = \mathbf{\Sigma_{XY}}^T \mathbf{\Sigma_X}^{-1} = \mathbf{B}^{-1}\mathbf{DA}^T \tag{18}$$

Finalmente, hay que considerar que mientras que para el problema de detección de cambios se deben priorizar los componentes de menor correlación (mayores cambios), para el problema de calibración relativa se priorizan los de mayor correlación (menores cambios). En cualquier caso, la correlación es el criterio para determinar los componentes retenidos.

4.1 Versión reponderada iterativamente

La versión reponderada iterativamente de MAD, o IR-MAD por sus siglas en inglés, es una versión que mejora iterativamente la estimación estadística [9]. Inicialmente, las medias y covarianzas de las imágenes se estiman a partir de un muestreo uniforme sobre la región de interés; sin embargo, una mejor estimación debería ponderar las observaciones en función de las probabilidades de no cambio, ya que los componentes MAD de las observaciones de no cambio siguen una distribución normal, lo que a su vez asegura que el z-score tenga una distribución chi-cuadrada. Por lo tanto, IR-MAD consiste en recalcular los componentes MAD con estadísticas reponderadas por la probabilidad de no cambio de la iteración previa (Ec. 16).

4.2 Implementación en SPEC

A diferencia de MNF, la solución del problema generalizado se implementa en SPEC calculando explícitamente la matriz de transformación por el método descrito en [4], el cual emplea la descomposición del problema generalizado en dos problemas clásicos de eigenvalores y eigenvectores.

La función `.MAD(xImage,yImage,term,minCorr)` permite calcular las componentes $\mathbf{U}, \mathbf{V}$ o $\mathbf{M}$, dependiendo del valor de `term`, el cual es por defecto "M", donde las componentes retenidas cumplen con un valor de correlación mínima `minCorr`. El valor por defecto es 0. La implementación usa la función `.setMADMatrices()` para calcular las matrices de bloque del problema de eigenvalores y eigenvectores generalizado acoplado (Ec. 13), el cual se resuelve mediante el método descrito en [4]. Adicionalmente, calcula las correlaciones de las componentes MAD empleando la función `.setCorr()`. La imagen MAD incluye las propiedades listadas en la Tabla 5.

La implementación por partes de MAD se puede hacer empleando la función `.rotateBandsMAD(numComps, term)`, la cual permite calcular cualquiera de los tres términos. El siguiente ejemplo ilustra el cálculo de los componentes U, paso a paso:

Atributo	Descripción	Función
`meanVector`	vector de medias	`.CenterBands`
`covMatrix`	matrices de convarianza XY	`.setCovMatrix`
`covMatrixA`	covarianzas cruzadas	`.setMADMatrices`
`covMatrixB`	covarianzas no cruzadas	`.setMADMatrices`
`eigenValues`	eigen valores	`.setEigens`
`eigenVectors`	eigen vectores	`.setEigens`
`compStdVector`	vector de desviación estandar	`.setEigens`
`Corr`	correlación de las componentes	`.setCorr`
`numBands`	número de bandas de entrada	`.rotateBandsMAD`
`numComps`	número de componentes de entrada	`.rotateBandsMAD`

Table 5: Lista de propiedades que se agregan a la imagen MAD

```
var images = xImage.addBands(yImage); // apila
var Ucomp = spec.CenterBands(images) // centra en media nula
  .setCovMatrix() // calcula covMatrix de ambas
  .setMADMatrices()// construye las matrices de bloques
  .setEigens() // eigens generalizados
  .rotateBandsMAD(numComps,"U"); // rota solo xImage
```

La versión reponderada iterativamente se calcula empleando la función `.IRMAD(numIter,xImage,yImage,term,minCorr)`, la cual requiere establecer un número de iteraciones `numIter` ≥ 1. En cada iteración, IRMAD calcula la probabilidad de no cambio y la emplea para remuestrear las imágenes en función del valor de la probabilidad, es decir, que la probabilidad de que un píxel sea seleccionado para calcular el vector de medias o la matriz de covarianza es precisamente el valor de `noChangeProb` en ese píxel [5]. Los parámetros `term` y `minCorr` sólo se aplican en la última iteración, es decir, que en las previas siempre se calculan todos los componentes $\mathbf{M}$. La imagen de salida contendrá los mismos parámetros que MAD, salvo un parámetro adicional `Corrs` que corresponde al historial de correlaciones en cada iteración.

El siguiente ejemplo calcula los componentes IRMAD en 10 iteraciones y grafica la evolución de las correlaciones estimadas (vea la Fig. 4):

```
var MADcomps = spec.IRMAD(10,xImage,yImage);
var Corrs = ee.Array(MADcomps.get('Corrs'));
var CorrsPlot = ui.Chart.array.values(Corrs,0)
  .setChartType('LineChart');
print(CorrsPlot);
```

4.3 Detección de cambios y calibración relativa

Para agregar las bandas `ChiSqr` y `noChangeProb` que se emplean en detección de cambio, se usa la función `.addChiSqrBand`, la cual requiere que los componentes hayan sido normalizados, como en el siguiente código:

```
var MADComps = spec.MAD(xImage,yImage)
  .normalizePCs() // normaliza
  .addChiSqrBand(); // agrega ChiSqr y noChangeProb
```

La Figura 3 muestra un ejemplo de detección de cambios ocurridos en la zona de la pista olímpica de Cuemanco entre 1990 y 2010 usando imágenes Landsat TM. La banda `noChangeProb` se muestra con una rampa rojo-blanco para resaltar las áreas de cambio en tonos rosa-rojo, las cuales corresponden a zonas de urbanización, reforestación e inundación de los cuerpos de agua.

Otra aplicación de la función `.MAD` o `.IRMAD` es en la calibración relativa de una imagen $\mathbf{X}$ a partir de una imagen de referencia $\mathbf{Y}$. Esto se logra haciendo `term` = "Cal", el cual es como "U" para el cálculo de los componentes, pero emplea la Ec. (17) para la reconstrucción.

El siguiente código ilustra la calibración de `X` relativa a `Y`:

```
var Xcal = spec.IRMAD(20,X,Y,'Cal') // Componentes U
  .restoreIm(X.bandNames()) // Calibracion MAD
```

Para poder generar un diagrama de dispersión entre el antes y el después de la normalización en la zona de no cambio, se requiere la banda `noChangeProb`, la cual solo puede agregarse a partir de los variantes $\mathbf{M}$ normalizados. Una alternativa al código previo es implementar IRMAD por pasos usando `.iterateMAD(numIters)` para actualizar la probabilidad de no cambio en forma iterativa seguido de `.WMAD` para generar los componentes $\mathbf{U}$ finales, como se muestra a continuación:

```
var MADComps = spec.MAD(X,Y) // primera aproximacion
  .normalizePCs() // normaliza
  .addChiSqrBand() // agrega bandas
```

[5] Para acelerar la convergencia, ningún pixel con probabilidad menor a 0.75 es seleccionado.

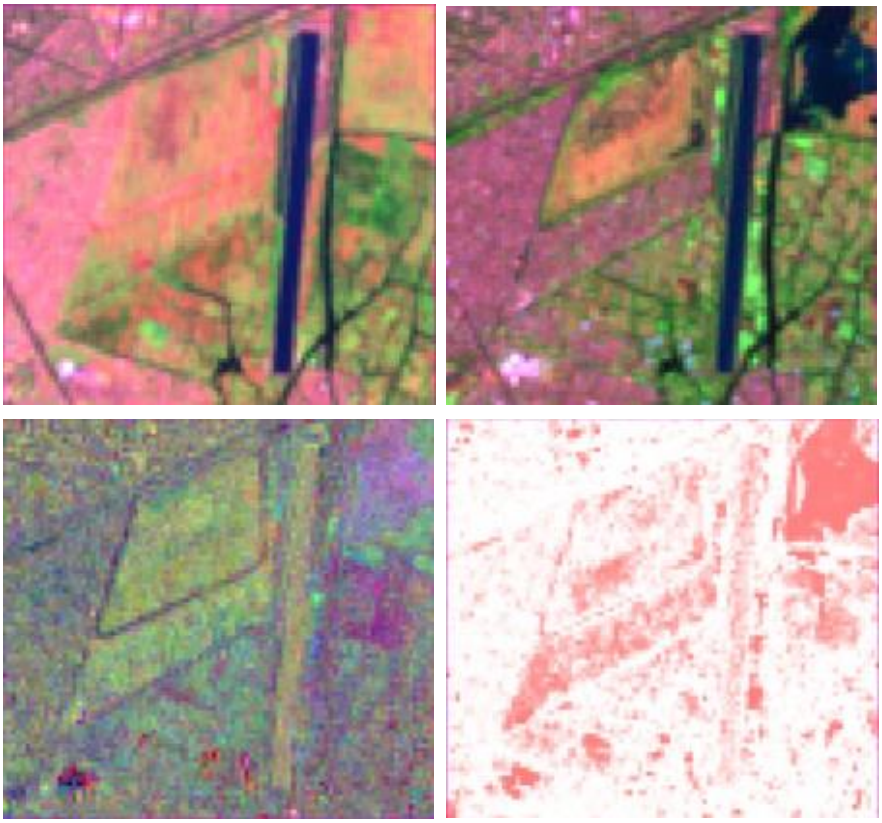

Figure 3: En orden lexicográfico se muestra: imagen de 1990, imagen de 2010, componente MAD4-6 e imagen de probabilidad de cambios.

```
    .iterateMAD(10); // recalcula 10 veces
var noChangeProb = MADComps
    .select('noChangeBands'); // para reponderacion
var XCal = spec.WMAD(X,Y,noChangeProb,'Cal') // U's
    .restoreIm(X.bandNames()); // X calibrada
```

La banda `noChangeProb` se pasa a `.WMAD` para el cálculo de la estadística ponderada por la probabilidad de no cambio. También se puede emplear para seleccionar una muestra de comparación entre X y XCal respecto a Y en la región de no cambio. La Figura 4 muestra un ejemplo de calibración relativa. La calibración IRMAD hace que la imagen original se transforme en una más parecida globalmente a la imagen de referencia. De hecho, la comparación de las bandas en la zona de no cambio se hace prácticamente 1:1 como lo ilustra el diagrama de dispersión de la banda roja. También se incluye el gráfico de las correlaciones en cada iteración, revelando pocos cambios a partir de la iteración 6. Que el algoritmo converja en unas cuantas iteraciones es deseable ya que IRMAD es un algoritmo muy demandante desde el punto de vista computacional.

5 Análisis de Discriminates Lineales

El análisis de discriminantes lineales, o LDA por sus siglas en inglés, se puede ver como un PCA supervisado en el cual los ejes de rotación se orientan en la dirección de máxima separabilidad entre dos o más clases [3, 7]. LDA es útil para reducir la dimensionalidad del espacio de características y aumentar la separabilidad de las clases. El problema se formula como sigue.

Se desea clasificar los píxeles $\mathbf{X} = [X_1, X2, ..., X_k]^T$ de una imagen multiespectral en J clases, de las cuales se conocen sus muestras de entrenamiento. Sean las variables $\mathbf{X}_j$, para $j = 1, 2, \ldots, J$, las muestras de entrenamiento de las J clases con media y covarianzas:

$$E\{\mathbf{X}_j\} = \mu_j, \qquad Cov\{\mathbf{X}_j\} = \mathbf{\Sigma}_j$$

Se define la medida de separabilidad de las clases como:

$$S = \frac{tr\{\mathbf{\Sigma_B}\}}{tr\{\mathbf{\Sigma_W}\}} \tag{19}$$

donde las matrices:

$$\begin{aligned} \mathbf{\Sigma_B} &= \frac{1}{J}\sum_{j=0}^{J}(\mu_j - \mu)(\mu_j - \mu)^T \\ \mathbf{\Sigma_W} &= \frac{1}{J}\sum_{j=0}^{J}\mathbf{\Sigma}_j \end{aligned}$$

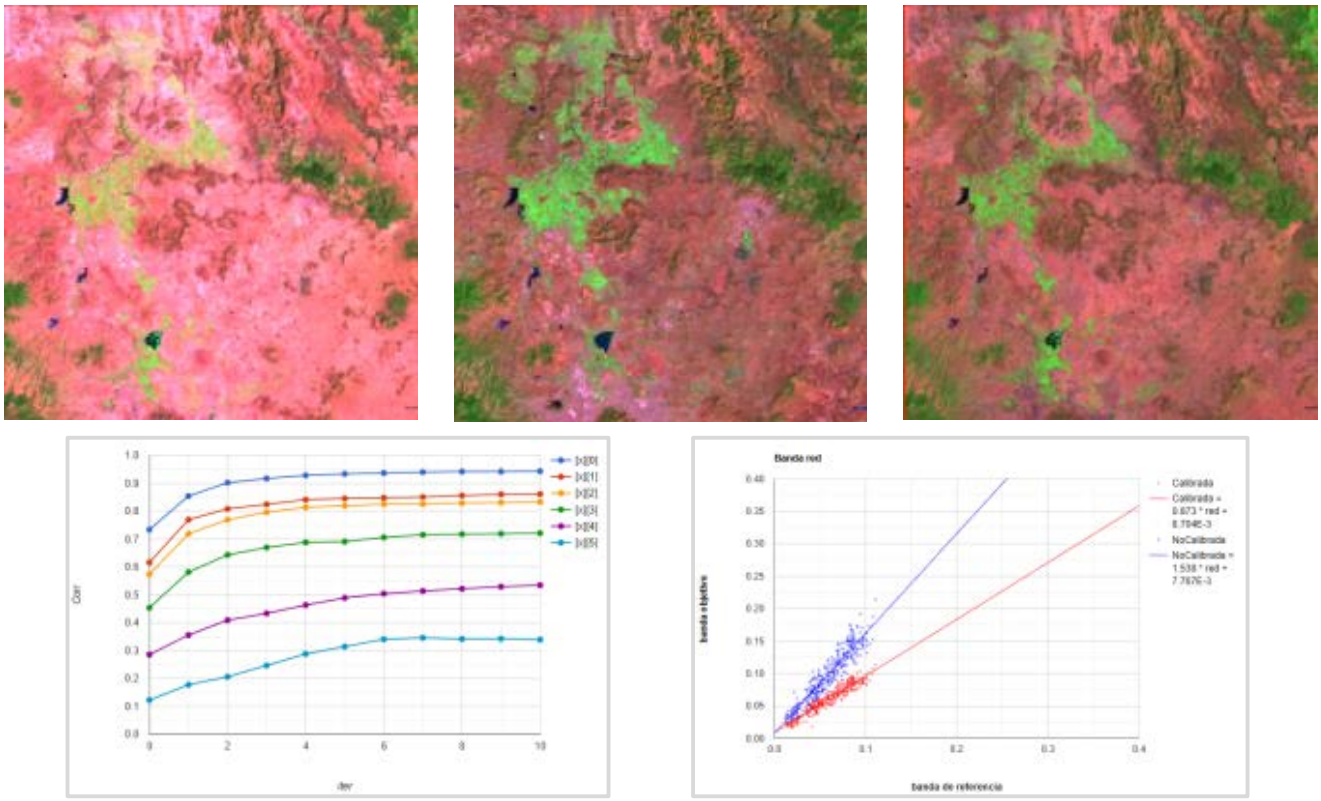

Figure 4: En orden lexicográfico: Imagen objetivo, imagen de referencia, imagen calibrada, correlaciones, y diagramas de dispersión antes y después de la calibración.

son las covarianzas interclase (*between class*) e intraclase (*within class*), respectivamente. La Ecuación (19) no es otra cosa que la distancia entre la media de los elementos de distintas clases a la media global, μ, dividida por el promedio de la distancia de elementos a la media de su clase.

La transformación LDA consiste en encontrar la transformación $\mathbf{Z} = \mathbf{A}^T\mathbf{X}$ que maximiza la separabilidad en el espacio transformado:

$$S(\mathbf{a}) = \frac{\mathbf{a}^T\mathbf{\Sigma_B}\mathbf{a}}{\mathbf{a}^T\mathbf{\Sigma_W}\mathbf{a}} \tag{20}$$

Encontrar la matriz de transformación LDA equivale a resolver el eigenproblema generalizado:

$$\mathbf{\Sigma_B}\mathbf{A} = \mathbf{\Sigma_W}\mathbf{A}\mathbf{D} \tag{21}$$

donde los eigenvalores $\mathbf{D} = \mathbf{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ definen la contribución a la separabilidad de clases en el subespacio de las componentes principales. Específicamente, la separabilidad en el espacio de los n componentes de mayor separabilidad está dada por:

$$S_n = \sum_{i=1}^{n} \frac{\lambda_i}{n} \tag{22}$$

5.1 Implementación en SPEC

La transformación LDA se realiza en SPEC mediante la función `.LDA(image,minSep)` la cual acepta como segundo argumento opcional el umbral de separabilidad mínima para determinar el número de componentes a retener. Los parámetros agregados a la imagen de componentes se muestran en la Tabla 6.

En la práctica, no se conoce el nivel de separabilidad que retiene un número de componentes deseados. Para calcular un número de componentes previamente determinado, se puede aplicar la implementación paso a paso. El siguiente ejemplo ilustra el cálculo de las primeras tres componentes normalizadas:

```
spec.SetSamp(training,1e6,scale); // define muestras de entrenamiento
var LDAComps = spec.CenterBands(image); // centra bandas en media nula
  .setCovMatrixC() // calcula covarianzas CB y CW
  .setEigens() // calcula eigen valores y vectores
  .rotateBands(3) // transformacion de 3 componentes
  .normalizePCs(); // normalizacion
```

Atributo	Descripción	Función
`meanVector`	vector de medias	`.CenterBands`
`classMeans`	número de bandas de entrada	`.setCovMatrixC`
`classCovs`	bandas de la imagen X	`.setCovMatrixC`
`covMatrixA`	covariaza de medias	`.setCovMatrixC`
`covMatrixB`	media de covarianzas	`.setCovMatrixC`
`eigenValues`	eigen valores	`.setEigens`
`eigenVectors`	eigen vectores	`.setEigens`
`compStdVector`	desviación estandar de componentes	`.setEigens`
`Sep`	separabilidad por componente	`.setSep`
`numBands`	número de bandas de entrada	`.rotateBands`
`numComps`	número de componentes de salida	`.rotateBands`

Table 6: Lista de propiedades que se agregan a la imagen LDA

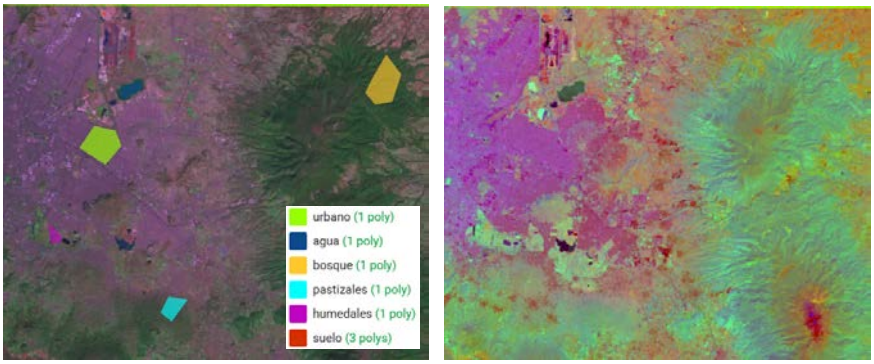

Figure 5: Imagen de entrada con muestras de entrenamiento y compuesto de componentes LDA1-3-2.

La función `.setCovMatrixC()` se usa aquí para calcular las matrices de covarianza intraclase e interclases, las cuales se emplean para resolver el eigenproblema generalizado por `.setEigens()`. Posteriormente, `.rotateBands(3)` aplica toma sólo tres eigenvectores para calcular tres componentes LDA, los cuales se normalizan por su desviación estándar en el último paso. Este último paso no se incluye en la implementación `.LDA()`, pero se emplea `.setSep()` para agregar el vector de separabilidad `Sep`, el cual se usa para determinar el número de componentes que garantizan el nivel de separabilidad solicitado. La ventaja de emplear la función `.LDA()` es que ésta habilita la reconstrucción a partir de los componentes retenidos mediante `.restoreIm(`*`bandNames`*`)`, así como el cálculo de distancias estadísticas entre las clases y la separabilidad global antes (Ec. 19) y después de la rotación (Ec. 22), como se ilustra en la siguiene aplicación.

5.2 Análisis de separabilidad

El siguiente ejemplo calcula los componentes LDA con separabilidad mayor a 6 y muestra la separabilidad de clases para las bandas originales y las componentes LDA:

```
var LDAComps = spec.LDA(image,6); // Componentes LDA
// Muestra separabilidad antes y despues de la rotacion
print("Separabilidad Original",LDAComps.getImSep());
print("Separabilidad LDA",LDAComps.getSep());
```

El resultado del ejemplo anterior se ilustra en la Fig. 5, donde la separabilidad aumenta de 3.18 en las bandas originales a 7.32 en las 3 componentes seleccionadas. Esta mayor separabilidad se manifiesta como un aumento de contraste entre los píxeles de las distintas clases en el compuesto RGB de las componentes (Fig. 5-der).

Finalmente, hay que considerar que la separabilidad global no permite predecir qué clases presentarán una mayor confiabilidad en el producto clasificado. Para ello, es necesario emplear otras métricas de separabilidad por clase, tales como la distancia de Mahalanobis, distancia de Bhattacharya o distancia de Jeffries-Matusita. Todas estas métricas se basan en las matrices de covarianza generadas con `.setCovMatrixC` y pueden ser generadas en SPEC para realizar un análisis de separabilidad más detallado (vea la Tabla 2 para las funciones correspondientes).

6 Comentarios finales

Debe notarse que las transformaciones implementadas en SPEC no restringen su uso a imágenes multi/híperespectrales, y que su aplicación podría extenderse a capas de datos de distinta naturaleza, como elevaciones, temperaturas, humedad del suelo, etc. Incluso, es posible realizar operaciones espaciales, por ejemplo, apilando píxeles de un vecindario en una imagen multibanda y aplicando una transformación espectral a dichos datos. De tal forma que el potencial del módulo no se limita a los ejemplos citados en este documento.

El desarrollo de SPEC ha sido producto del trabajo de enseñanza del curso de Percepción Remota en el CentroGeo que por años ha desempeñado el autor. Está pensado para ser reusable y simple de comprender. No tiene garantía de encontrarse libre de errores, aunque continuamente se actualiza, amplía y mejora. Si lo encuentras útil en tus investigaciones o proyectos académicos, será un placer conocer tu trabajo y recibir cualquier sugerencia de mejoras a través del correo institucional del autor, jsilvan@centrogeo.edu.mx.

References

[1] Morton J Canty and Allan A Nielsen. Automatic radiometric normalization of multitemporal satellite imagery with the iteratively re-weighted mad transformation. *Remote Sensing of Environment*, 112(3):1025–1036, 2008.

[2] Morton John Canty. *Image analysis, classification and change detection in remote sensing: with algorithms for Python*. Crc Press, 2019.

[3] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[4] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley. Eigenvalue and generalized eigenvalue problems: Tutorial. *arXiv preprint arXiv:1903.11240*, 2019.

[5] Andrew A Green, Mark Berman, Paul Switzer, and Maurice D Craig. A transformation for ordering multispectral data in terms of image quality with implications for noise removal. *IEEE Transactions on geoscience and remote sensing*, 26(1):65–74, 1988.

[6] Trevor Hastie. The elements of statistical learning: data mining, inference, and prediction, 2009.

[7] Cheng Li and Bingyu Wang. Fisher linear discriminant analysis. *CCIS Northeastern University*, 6, 2014.

[8] Allan A Nielsen, Knut Conradsen, and James J Simpson. Multivariate alteration detection (mad) and maf postprocessing in multispectral, bitemporal image data: New approaches to change detection studies. *Remote Sensing of Environment*, 64(1):1–19, 1998.

[9] Allan Aasbjerg Nielsen. The regularized iteratively reweighted mad method for change detection in multi-and hyperspectral data. *IEEE Transactions on Image processing*, 16(2):463–478, 2007.

[10] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.