Carlos Mauricio Arteaga Bermudez
Ángel Pablos

Notebook 1

OBJETIVO

Este código demuestra cómo hacer una selección de características usando el método FDR (Fisher Discriminant Ratio) y cómo aplicar Forward Feature Selection (FFS) para elegir las mejores características en un modelo. Esto se aplica sobre en DataSet Iris

Importación de librerías

Importamos las librerías necesarias: scikit-learn para cargar el dataset IRIS, dividir los datos y entrenar un modelo SVM; numpy para cálculos numéricos; y matplotlib para graficar.

```
from sklearn.datasets import load_iris
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from matplotlib import pyplot as plt
```

Carga del dataset IRIS

```
# Cargamos el dataset IRIS
Data = load_iris()
X = Data.data
y = Data.target

# Usamos sólo dos clases para simplificar (100 muestras)
X = X[:100, :]
y = y[:100]

print("Forma de X:", X.shape, " - Forma de y:", y.shape)
```

El dataset IRIS tiene 150 flores de 3 especies. Aquí usamos solo las dos primeras especies (100 muestras) para simplificar el análisis . X contiene las características (como largo del pétalo), e y contiene la clase (especie).

Función para calcular el FDR (Fisher Discriminant Ratio)

```python
def calFDR(X, y):
    X1 = X[:50,:]   # Separamos las primeras 50 muestras (clase 0)
    X2 = X[50:,:]   # Las otras 50 muestras (clase 1)

    # Calculamos la media de cada característica para cada clase
    m1 = np.mean(X1, axis=0)
    m2 = np.mean(X2, axis=0)

    # Calculamos la desviación estándar de cada característica para cada clase
    s1 = np.std(X1, axis=0)
    s2 = np.std(X2, axis=0)

    # Fórmula del FDR: diferencia de medias al cuadrado / suma de varianzas
    fdr = ((m2 - m1)**2)/(s1**2 + s2**2)

    # Ordenamos las características de mayor a menor FDR
    ind = np.argsort(fdr)[::-1]

    return fdr, ind   # Retornamos los valores FDR y el orden de importancia

# Llamamos la función para obtener el FDR y el orden de características
fdr, ind1 = calFDR(X, y)
```

$$\mathrm{FDR}_j = \frac{(\mu_{1j} - \mu_{2j})^2}{\sigma_{1j}^2 + \sigma_{2j}^2}$$

Esta función calcula el FDR, una métrica que nos dice qué tan bien una característica diferencia entre dos clases. Entre más grande sea el FDR, más útil es esa característica para distinguir entre las clases. Luego se ordenan de mayor a menor.

Aplicar FDR y ordenar características

```
# Llamamos la función para obtener el FDR y el orden de características
fdr, ind1 = calFDR(X, y)

# Imprimimos los valores FDR de cada característica
print("FDR:", fdr)

# Imprimimos el orden de las características, desde la más relevante
print("Orden de características por FDR:", ind1)

# Reordenamos X según el orden de importancia (X1 tiene las columnas reordenadas)
X1 = X[:, ind1]
```

Después de calcular el FDR, ordenamos las características desde la más útil hasta la menos útil. Luego reorganizamos los datos (X1) siguiendo ese orden para que podamos probar con las mejores primero.

Aplicación de Forward Feature Selection (FFS)

```python
# Vamos agregando una característica a la vez (desde la más relevante)
for i in range(X.shape[1]):
    X2 = X1[:, :(i+1)]  # Tomamos las primeras (i+1) características

    # Dividimos el dataset en entrenamiento (70%) y prueba (30%)
    X_train, X_test, y_train, y_test = train_test_split(X2, y, test_size=0.3)

    # Creamos un clasificador SVM con kernel lineal
    clf1 = SVC(kernel='linear')

    # Entrenamos el modelo con los datos de entrenamiento
    clf1.fit(X_train, y_train)

    # Predecimos con los datos de prueba
    y_pred = clf1.predict(X_test)

    # Calculamos la exactitud: porcentaje de aciertos
    acc = np.sum(y_pred == y_test)/y_test.shape[0]
    accuracies.append(acc)  # Guardamos el resultado

# Imprimimos la exactitud obtenida con 1, 2, 3 y 4 características
```

```python
# Imprimimos la exactitud obtenida con 1, 2, 3 y 4 características
print("Exactitud en cada paso de FFS:", accuracies)
```

Aquí imprimimos la exactitud obtenida al usar 1, 2, 3 o 4 características. Así podemos ver si usar más variables realmente mejora el desempeño o si algunas no aportan mucho.

```
Forma de X: (100, 4)  - Forma de y: (100,)
FDR: [ 2.2590031   1.82441976 31.83009969 23.70346285]
Orden de características por FDR: [2 3 0 1]
Exactitud en cada paso de FFS: [1.0, 1.0, 1.0, 1.0]
```

Usamos FFS para probar qué tan bueno es el modelo cuando vamos agregando características una por una, empezando por la más útil. Entrenamos un modelo SVM y medimos la exactitud en cada paso.

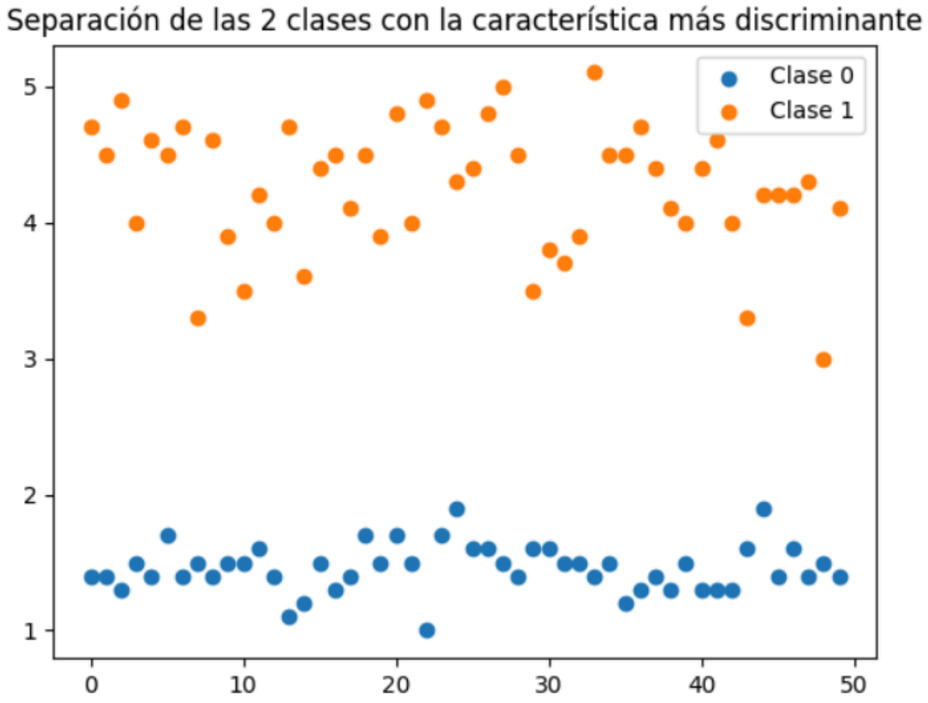
Visualización de la característica más importante

```python
# Seleccionamos la tercera columna de X como ejemplo (puede ser la más relevante)
X_imp = X[:, 2]

# Separamos los valores de esa característica por clase
X_class0 = X_imp[:50]   # Valores para clase 0
X_class1 = X_imp[50:]   # Valores para clase 1

# Creamos un índice para el eje X de la gráfica
idx = np.arange(50)

# Graficamos la característica para cada clase
plt.scatter(idx, X_class0, label='Clase 0')   # Muestra la clase 0
plt.scatter(idx, X_class1, label='Clase 1')   # Muestra la clase 1
plt.title("Separación de las 2 clases con la característica más discriminante")
plt.legend()  # Muestra la leyenda de clases
plt.show()    # Mostramos la gráfica
```



Graficamos la característica más discriminante (la que tiene el FDR más alto) y mostramos cómo se separan visualmente las dos clases. Si se ve separación clara, es una buena señal.

De las cuatro características, la que mejor separa las dos clases de flores es el largo del pétalo, porque tiene el valor de FDR más alto: 31.83. Esto significa que las medias de esta variable entre clases están muy separadas, y además tienen poca variabilidad interna, lo que la hace muy útil para clasificación.

Notebook 2

OBJETIVO

En este notebook se aplica el método RFE (Recursive Feature Elimination) usando una regresión SVM (SVR) sobre el dataset de diabetes.
El propósito es seleccionar las características más importantes para predecir la progresión de la enfermedad.

Importación de librerías

- load_diabetes: carga un dataset real de regresión sobre diabetes (con 10 variables).
- RFE: herramienta de selección de características.
- SVR: Support Vector Regression, sirve como modelo base para que RFE evalúe la importancia de las características.

```
from sklearn.datasets import load_diabetes
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
```

CARGA DEL DATASET

```
# X contiene las variables predictoras (10 características)
X_diab = data_diab.data

# y contiene el valor objetivo: una medida de progresión de la enfermedad
y_diab = data_diab.target

# Definimos un modelo base: SVR (Support Vector Regression) con kernel lineal
model = SVR(kernel="linear")
```

- data_diab.data: contiene las variables predictoras (X), que incluyen edad, IMC, presión arterial, etc.
- data_diab.target: es la variable objetivo (y), que representa una medida de progresión de la enfermedad.

CREACIÓN DEL MODELO BASE (SVR)

```
model = SVR(kernel="linear")
```

- Se define un modelo de regresión SVR con kernel lineal.
- Este modelo será el que use RFE para evaluar qué tan importantes son las características al predecir y.

CREACIÓN DEL MODELO BASE (SVR)

```
# Creamos el selector de características con RFE
# Queremos quedarnos con 5 características al final
# El proceso elimina una característica por vez (step=1)
feat_selector = RFE(model, n_features_to_select=5, step=1)

# Ajustamos el RFE al dataset para que haga la selección
feat_selector.fit(X_diab, y_diab)
```

- RFE(...): va eliminando una característica por vez (step=1), entrenando el modelo y viendo cuál es la menos útil.
- n_features_to_select=5: el proceso para cuando quedan 5 características.
- fit(...): ejecuta el proceso completo sobre los datos.

RESULTADOS: CARACTERÍSTICAS SELECCIONADAS Y RANKING

```
# Mostramos el soporte: qué características fueron seleccionadas (True)
print("Soporte RFE (True=características seleccionadas):")
print(feat_selector.support_)

# Mostramos el ranking de todas las características (1 es más importante)
print("Ranking RFE (1 = más importante):")
print(feat_selector.ranking_)
```

- .support_: lista booleana que indica cuáles características fueron seleccionadas como las 5 mejores (True).
- .ranking_: lista con un número de ranking (1 es mejor). Las características con ranking 1 son las seleccionadas.

```
Soporte RFE (True=características seleccionadas):
[False False  True  True False False  True  True  True False]
Ranking RFE (1 = más importante):
[4 6 1 1 3 5 1 1 1 2]
```

Notebook 3 y 3A

OBJETIVO

Mostrar cómo se puede reconstruir una imagen usando distintas cantidades de componentes principales mediante PCA (1, 10 y 80 componentes), para visualizar cómo mejora la calidad con más componentes.

- Se carga una imagen desde la ruta indicada.
- Se muestra la imagen original para tener un punto de comparación.

```
imagen_path = ('../Datos/IMAGEN_TEST.png')
img1 = plt.imread(imagen_path)
plt.imshow(img1)
plt.title("Imagen original")
plt.show()
```

CONVERTIR A ESCALA DE GRISES

Convertir a escala de grises

- Se define una función que convierte la imagen RGB a escala de grises (fórmula estándar ponderando los canales).
- Esto reduce la complejidad del análisis.

```
def RGBtoGray(img):
    return 0.299*img[:,:,0] + 0.587*img[:,:,1] + 0.114*img[:,:,2]


img_gray = RGBtoGray(img1)
```

PREPARAR LA MATRIZ PARA PCA

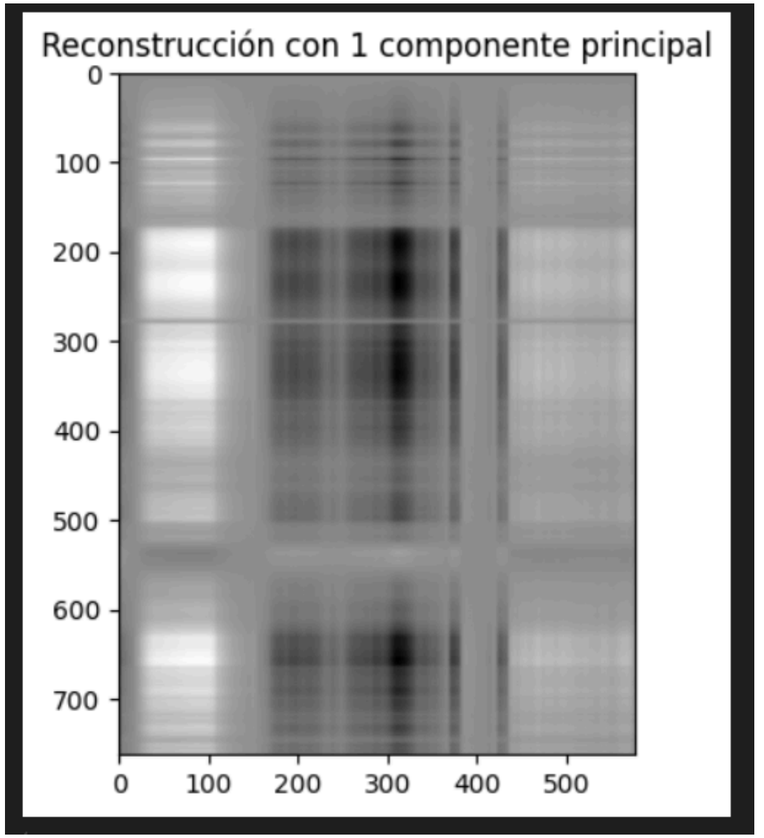
```
X_mean = np.mean(img_gray, axis=1, keepdims=True)  # Promedio por fila
diff = img_gray - X_mean                           # Resta la media (centrado)
cov1 = np.matmul(diff.T, diff)                     # Matriz de covarianza
eigenvalues, eigenvectors = LA.eig(cov1)           # Autovalores y autovectores
```

- Se centra la matriz restando la media.
- Se calcula la matriz de covarianza, que describe cómo varían juntas las columnas (píxeles).
- Se obtiene la descomposición espectral: autovalores y autovectores.
  - Los autovectores son las direcciones principales (componentes).
  - Los autovalores indican cuánta varianza explica cada componente.

RECONSTRUIR IMAGEN CON PCA

Con 1 componente principal

```
T1 = eigenvectors[:, :1]
Transformed = np.matmul(img_gray, T1)
recon = np.matmul(Transformed, T1.T)
recon = np.real(recon)
plt.imshow(recon, cmap='gray')
plt.title("Reconstrucción con 1 componente principal")
plt.show()
```

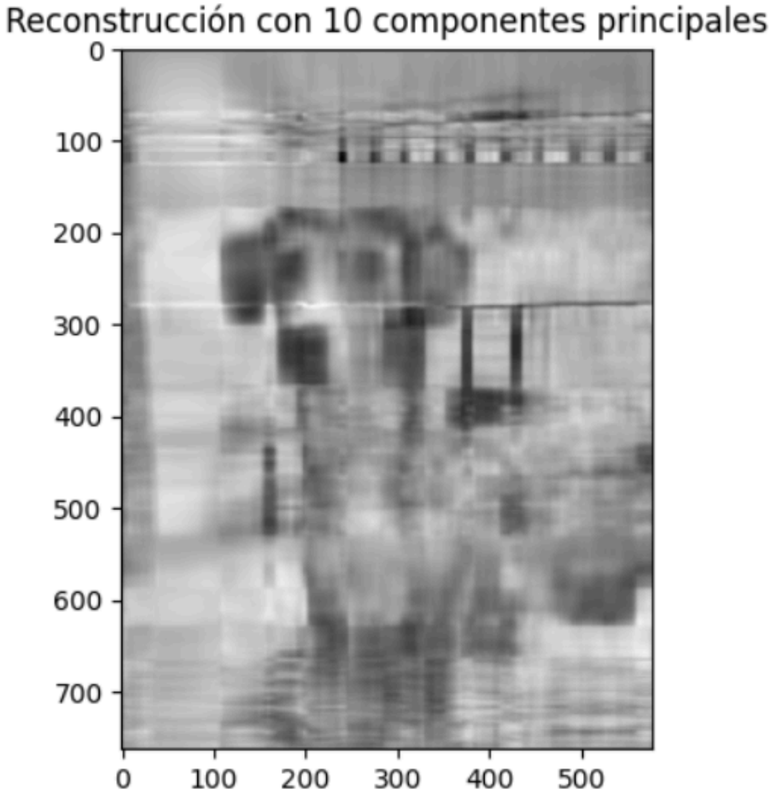


Se proyecta la imagen sobre solo el primer componente.
Se reconstruye desde esa versión comprimida.
La calidad es muy baja, pero es una representación muy comprimida.

RECONSTRUIR IMAGEN CON PCA

Con 10 componentes principales

```
T10 = eigenvectors[:, :10]
Transformed10 = np.matmul(img_gray, T10)
recon10 = np.matmul(Transformed10, T10.T)
recon10 = np.real(recon10)
plt.imshow(recon10, cmap='gray')
plt.title("Reconstrucción con 10 componentes principales")
plt.show()
```



La calidad mejora, se ve más estructura en la imagen.

RECONSTRUIR IMAGEN CON PCA

Con 80 componentes principales

```
T80 = eigenvectors[:, :80]
Transformed80 = np.matmul(img_gray, T80)
recon80 = np.matmul(Transformed80, T80.T)
recon80 = np.real(recon80)
plt.imshow(recon80, cmap='gray')
plt.title("Reconstrucción con 80 componentes principales")
plt.show()
```

- Mucha mejor calidad visual, se acerca mucho a la imagen original.
- Pero también se requiere más información (menos compresión).