Guía de Laboratorio – SciPy para IA
Curso de Inteligencia Artificial

Docente: Carlos R. P. Tovar

Objetivo de la sesión

Al finalizar la sesión, el alumno será capaz de:

- Utilizar las funciones principales del módulo `scipy` en problemas de IA.
- Resolver problemas de optimización, álgebra lineal y procesamiento de señales.
- Aplicar funciones estadísticas y técnicas de procesamiento digital.
- Implementar soluciones científicas para problemas de inteligencia artificial.

Introducción a SciPy en IA

SciPy es una biblioteca fundamental para computación científica en Python. En inteligencia artificial, se utiliza para:

- Optimización de funciones de costo
- Procesamiento de señales e imágenes
- Álgebra lineal en operaciones de redes neuronales
- Análisis estadístico de datos
- Resolución de ecuaciones diferenciales

Instrucciones

Resuelva los siguientes ejercicios en Python utilizando `scipy`. Cada ejercicio debe implementarse en un script independiente o en un notebook de Jupyter.

Ejercicios Prácticos

Ejercicio 1: Optimización de función de costo
Minimice la función $f(x) = x^2 + 10\sin(x)$ que representa una función de costo en un problema de IA.

```python
from scipy.optimize import minimize
import numpy as np

# Funcion de costo a optimizar
def costo(x):
    return x**2 + 10*np.sin(x)

# Encontrar minimo
resultado = minimize(costo, x0=0, method='BFGS')
print(f"Minimo en x = {resultado.x[0]:.4f}, f(x) = {resultado.fun
    :.4f}")
```

Ejercicio 2: Regresión lineal con SciPy

Implemente una regresión lineal para predecir valores.

```python
from scipy import stats
import numpy as np

# Datos de entrenamiento
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

# Ajustar modelo lineal
pendiente, intercepto, r, p, error = stats.linregress(x, y)

print(f"Modelo: y = {pendiente:.2f}x + {intercepto:.2f}")
print(f"Coeficiente de determinacion R^2: {r**2:.4f}")
```

Ejercicio 3: Procesamiento de señales - Filtrado

Aplique un filtro a una señal ruidosa.

```python
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt

# Generar se al con ruido
t = np.linspace(0, 1, 1000)
senal = np.sin(2*np.pi*5*t) + 0.5*np.random.randn(1000)

# Dise ar filtro Butterworth
b, a = signal.butter(4, 0.1, 'low')
senal_filtrada = signal.filtfilt(b, a, senal)

# Visualizar resultados
plt.figure(figsize=(10, 4))
plt.plot(t, senal, alpha=0.5, label='Se al con ruido')
plt.plot(t, senal_filtrada, 'r-', label='Se al filtrada')
plt.legend()
plt.show()
```

Ejercicio 4: Álgebra lineal - Descomposición SVD

Realice descomposición SVD para reducción de dimensionalidad.

```python
from scipy.linalg import svd
import numpy as np

```

```
# Matriz de datos (ejemplo: 5 caracter sticas, 100 muestras)
X = np.random.randn(100, 5)

# Descomposicion SVD
U, s, Vt = svd(X)

# Reduccion a 2 componentes principales
X_reduced = np.dot(X, Vt[:2].T)

print(f"Forma original: {X.shape}")
print(f"Forma reducida: {X_reduced.shape}")
print(f"Valores singulares: {s}")
```

Ejercicio 5: Transformada de Fourier para análisis de series temporales

Analice una señal en el dominio de la frecuencia.

```
from scipy.fft import fft, fftfreq
import numpy as np
import matplotlib.pyplot as plt

# Generar se al compuesta
t = np.linspace(0, 1, 1000)
senal = np.sin(2*np.pi*5*t) + 0.5*np.sin(2*np.pi*20*t)

# Transformada de Fourier
fft_vals = fft(senal)
frecuencias = fftfreq(len(senal), t[1]-t[0])

# Encontrar frecuencias dominantes
idx = np.argsort(np.abs(fft_vals))[::-1]
frecuencias_dominantes = frecuencias[idx[:4]]

print(f"Frecuencias dominantes: {frecuencias_dominantes} Hz")
```

Ejercicio 6: Resolución de ecuaciones diferenciales

Modele un sistema dinámico simple.

```
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt

# Sistema: crecimiento poblacional con capacidad de carga
def sistema(t, y, r, K):
    return r * y * (1 - y/K)

# Parametros
r, K = 0.5, 100  # Tasa de crecimiento, capacidad de carga
y0 = [10]        # Poblacion inicial

# Resolver
sol = solve_ivp(sistema, [0, 50], y0, args=(r, K),
                t_eval=np.linspace(0, 50, 100))

plt.plot(sol.t, sol.y[0])
plt.xlabel('Tiempo')
plt.ylabel('Poblacion')
plt.title('Modelo de crecimiento logistico')
```

```
plt.show()
```

Ejercicio 7: Interpolación para completar datos faltantes

Complete datos faltantes usando interpolación.

```
from scipy.interpolate import interp1d
import numpy as np

# Datos con valores faltantes
x = np.array([0, 1, 2, 4, 5, 6])  # Falta x=3
y = np.array([0, 1, 4, 16, 25, 36])  # Valores conocidos

# Interpolacion cubica
f = interp1d(x, y, kind='cubic', fill_value='extrapolate')

# Estimar valor en x=3
x_nuevo = 3
y_estimado = f(x_nuevo)

print(f"Valor interpolado en x={x_nuevo}: {y_estimado:.2f}")
```

Ejercicio 8: Análisis estadístico de datos

Realice análisis estadístico de un dataset.

```
from scipy import stats
import numpy as np

# Generar datos normales
datos = np.random.normal(50, 15, 1000)

# Calculos estadisticos
media = np.mean(datos)
mediana = np.median(datos)
desviacion = np.std(datos)
asimetria = stats.skew(datos)
curtosis = stats.kurtosis(datos)

# Test de normalidad
stat, p_valor = stats.normaltest(datos)

print(f"Media: {media:.2f}, Mediana: {mediana:.2f}")
print(f"Asimetria: {asimetria:.2f}, Curtosis: {curtosis:.2f}")
print(f"Normalidad: p-valor = {p_valor:.4f}")
```

Ejercicio 9: Procesamiento de imágenes - Convolución

Aplique filtros a imágenes usando convolución.

```
from scipy import ndimage
import numpy as np
import matplotlib.pyplot as plt
from skimage import data

# Cargar imagen de ejemplo
imagen = data.camera()

# Definir kernel de deteccion de bordes
kernel = np.array([[-1, -1, -1],
```

```
                    [-1,  8, -1],
                    [-1, -1, -1]])

# Aplicar convolucion
imagen_bordes = ndimage.convolve(imagen, kernel)

# Visualizar
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.imshow(imagen, cmap='gray')
ax1.set_title('Imagen original')
ax2.imshow(imagen_bordes, cmap='gray')
ax2.set_title('Bordes detectados')
plt.show()
```

Ejercicio 10: Optimización de hiperparámetros

Optimice hiperparámetros usando técnicas de SciPy.

```
from scipy.optimize import differential_evolution
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Generar datos de ejemplo
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=15, random_state=42)

# Funcion objetivo para optimizar
def objetivo(hiperparams):
    n_estimators = int(hiperparams[0])
    max_depth = int(hiperparams[1]) if hiperparams[1] > 1 else None

    modelo = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        random_state=42
    )

    score = cross_val_score(modelo, X, y, cv=3, scoring='accuracy').mean()
    return -score  # Minimizar negativo de accuracy

# Espacio de busqueda de hiperparametros
limites = [(10, 200),  # n_estimators
           (2, 20)]    # max_depth

# Optimizacion por evolucion diferencial
resultado = differential_evolution(objetivo, limites, maxiter=20, popsize=10,
                                   mutation=(0.5, 1), recombination=0.7)

print(f"Mejores hiperparametros: {resultado.x}")
print(f"Mejor accuracy: {-resultado.fun:.4f}")
```

Recursos Adicionales

- Documentación oficial de SciPy: `https://scipy.org/`
- Tutoriales de SciPy: `https://docs.scipy.org/doc/scipy/tutorial/`
- Ejemplos de aplicaciones en IA: `https://github.com/scipy/scipy/tree/main/doc/source/tutorial`
- Libro: "Python for Data Analysis"by Wes McKinney

Evaluación

- Complete todos los ejercicios prácticos
- Documente el código con comentarios explicativos Realice experimentos adicionales modificando los parámetros
- Prepare un breve reporte de los resultados obtenidos