INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

CARRERA

Ingeniería en informática

MATERIA

Fundamentos de Inteligencia Artificial

PRÁCTICA 11

Práctica 12 - Modelo clasificador de imágenes

SECUENCIA

6NM62

INTEGRANTES

- Gonzalez Calzada Maximiliano

PROFESORA

Gonzalez Arroyo Lilia

FECHA

03 - 12 - 2025

Índice

Modelo clasificador de imágenes

1. Introducción

El reconocimiento de dígitos escritos a mano es un problema fundamental en la visión por computadora. En esta práctica, exploramos el uso de Máquinas de Vectores de Soporte (SVM) para clasificar imágenes de dígitos del conjunto de datos de scikit-learn. Este enfoque permite entender cómo los algoritmos de aprendizaje supervisado pueden aprender patrones visuales a partir de datos de píxeles.

2. Objetivos e Hipótesis

2.1. Objetivo general

Implementar un modelo de clasificación de imágenes basado en SVM para identificar dígitos escritos a mano con alta precisión.

2.2. Objetivos específicos

- Cargar y explorar el conjunto de datos de dígitos de `sklearn`.
- Preprocesar las imágenes transformándolas de matrices 2D a vectores 1D.
- Entrenar un clasificador SVM utilizando un kernel lineal.
- Evaluar el desempeño del modelo mediante métricas de precisión, recall y f1--score.
- Visualizar los datos y los resultados para una mejor interpretación.

2.3. Hipótesis

Se plantea que un modelo SVM con kernel lineal será suficiente para clasificar correctamente la mayoría de los dígitos (>95% de precisión), dado que las imágenes de baja resolución (8x8) presentan características que suelen ser linealmente separables en un espacio de alta dimensión.

2.4. Dataset

Se utilizó el conjunto de datos `digits` disponible en la biblioteca `scikit-learn`. Este dataset consta de 1797 imágenes de 8x8 píxeles, donde cada elemento representa

un valor de escala de grises (0-16). Cada imagen está asociada a una etiqueta numérica del 0 al 9.

2.5. Limpieza de datos

El dataset ya se encuentra limpio y normalizado. El paso principal de preprocesamiento consistió en «aplanar» (reshape) las imágenes de 8x8 a vectores de 64 elementos, ya que el algoritmo SVM requiere una entrada unidimensional para cada muestra.

2.6. Procedimiento analítico

1. **Carga de datos**: Importación del dataset y separación en características (X) y etiquetas (y).
2. **Visualización**: Generación de una gráfica con muestras de los dígitos usando el mapa de color "plasma" para resaltar intensidades.
3. **División de datos**: Separación del conjunto en 80% para entrenamiento y 20% para prueba, asegurando reproducibilidad con una semilla aleatoria.
4. **Entrenamiento**: Ajuste del modelo SVM lineal con los datos de entrenamiento.
5. **Evaluación**: Predicción sobre el conjunto de prueba y cálculo de métricas de rendimiento.

3. Análisis y Resultados

3.1. Código fuente

El código fue modificado para incluir comentarios explicativos, mejorar la visualización y guardar automáticamente la figura generada.

```
# Importar las bibliotecas necesarias
# datasets: contiene conjuntos de datos de prueba, incluyendo el de dígitos
# train_test_split: función para dividir los datos en conjuntos de entrenamiento y
prueba
# SVC: Support Vector Classifier, el algoritmo de clasificación que usaremos
# classification_report, accuracy_score: métricas para evaluar el rendimiento del
modelo
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
import os

# Cargar el conjunto de datos de digitos
# Este dataset contiene imágenes de 8x8 píxeles de dígitos escritos a mano
digits = datasets.load_digits()

# Mostrar digitos en forma de arreglos (opcional descomentar para mirarlos)
# print(digits)

# Dividir los datos en características (X) y etiquetas (y)
# X contiene las matrices de imágenes (los datos de entrada)
# y contiene los números reales que representan cada imagen (las etiquetas)
X = digits.images
y = digits.target

# Visualizar una muestra de las imágenes y etiquetas
# Se mostrarán los primeros 5 dígitos del dataset para entender con qué estamos
trabajando
n_muestras = 5
plt.figure(figsize=(10, 2))
for i in range(n_muestras):
    plt.subplot(1, n_muestras, i + 1)
    # Usamos 'plasma' para una mejor visualización de la intensidad de los píxeles
    plt.imshow(X[i], cmap='plasma')
    plt.title(f"Digito {y[i]}")
    plt.axis('off')

# Guardar la figura generada en la carpeta docs/media
# Esto permite incluirla automáticamente en el reporte
output_dir = "../docs/media"
if not os.path.exists(output_dir):
```

```
    os.makedirs(output_dir)
plt.savefig(os.path.join(output_dir, "figure_1.png"))
print(f"Figura guardada en {os.path.join(output_dir, 'figure_1.png')}")
# plt.show() # Comentado para evitar bloqueo en ejecución automática

# Preprocesar las imágenes a formato 1D
# Las imágenes son matrices 2D (8x8), pero el modelo SVM requiere un vector 1D (64
elementos)
# reshape transforma cada imagen de 8x8 a un vector de 64
X = X.reshape([X.shape[0],-1])

# Dividir los datos en conjuntos de entrenamiento y prueba
# Usamos el 20% de los datos para prueba (test_size=0.2) y el 80% para entrenamiento
# random_state=43 asegura que la división sea reproducible
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=43)

# Crear un modelo de SVM con el kernel lineal
# El kernel lineal es adecuado cuando los datos son linealmente separables
modelo = SVC(kernel='linear')

# Entrenar el modelo con los datos de entrenamiento
# El modelo aprende la relación entre los píxeles (X_train) y los dígitos (y_train)
modelo.fit(X_train, y_train)

# Predecir las etiquetas para el conjunto de pruebas
# Usamos el modelo entrenado para predecir los dígitos de las imágenes que no ha visto
(X_test)
y_pred = modelo.predict(X_test)

# Calcular la precisión del modelo
# Comparamos las predicciones (y_pred) con los valores reales (y_test)
precision = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo SVM con kernel lineal: {precision:.2f}")
```

```
# Mostrar un informe de clasificación detallado
# Incluye precision, recall, f1-score para cada dígito
print("Informe de clasificación")
print(classification_report(y_test, y_pred))
```

3.2. Figura generada

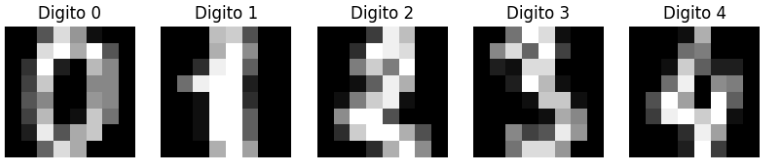

Figura 1: *Figura generada* Muestra de dígitos del dataset con mapa de color "plasma".

3.3. Interpretación de Resultados

El modelo obtuvo una **precisión global del 98%**, lo cual valida nuestra hipótesis inicial.

Del informe de clasificación obtenido en la terminal:

- **Precisión perfecta (1.00)**: Los dígitos 0, 2, 3, 4, 6 y 8 fueron clasificados con muy alta precisión.
- **Áreas de mejora**: El dígito 1 tuvo una precisión de 0.91, lo que indica algunos falsos positivos (otros números clasificados erróneamente como 1). El dígito 9 tuvo un recall de 0.92, indicando que algunos 9 reales no fueron identificados correctamente.
- **F1-Score**: El promedio ponderado del F1-score es 0.98, lo que demuestra un equilibrio excelente entre precisión y exhaustividad.

```
Precisión del modelo SVM con kernel lineal: 0.98
Informe de clasificación
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        42
```

1	0.91	1.00	0.96	32
2	1.00	1.00	1.00	25
3	1.00	0.96	0.98	25
4	1.00	0.98	0.99	44
5	0.94	1.00	0.97	30
6	1.00	1.00	1.00	41
7	0.97	1.00	0.99	39
8	1.00	0.93	0.96	43
9	0.95	0.92	0.94	39
accuracy			0.98	360
macro avg	0.98	0.98	0.98	360
weighted avg	0.98	0.98	0.98	360

4. Conclusiones

1. **Eficacia del SVM**: El algoritmo SVM con kernel lineal es altamente efectivo para el reconocimiento de dígitos en imágenes de baja resolución, logrando un 98% de exactitud con un costo computacional bajo.
2. **Importancia del Preprocesamiento**: La transformación de las imágenes a vectores 1D es un paso crucial que permite al algoritmo procesar la información espacial como características independientes.
3. **Visualización**: El uso de mapas de color como "plasma" ayuda a distinguir mejor los niveles de intensidad en las imágenes, facilitando la inspección visual de los datos.

5. Investigaciones futuras

- Evaluar el rendimiento con kernels no lineales (RBF, polinomial) para ver si se puede alcanzar el 100% de precisión.
- Probar el modelo con imágenes de dígitos escritos a mano por los propios usuarios para verificar su robustez en escenarios reales.