Parcial Segundo Corte

Curso:

Machine Learning

Hecho por:

Andrea Terraza Sequea
Isabella Ardila Martinez
Laura Catalina García González

Ricardo Andrés Fonseca Perdomo

Universidad Sergio Arboleda

Programa de Ciencias de la computación e Inteligencia Artificial
Escuela de Ciencias Exactas e Ingeniería

Bogotá, Colombia

30 de Abril del 2024

Declaración y firma:

"Certifico que todas las soluciones son enteramente de mi autoría y que no he consultado las soluciones de otro estudiante. He dado crédito a todas las fuentes externas que consulté para la solución del ejercicio".

Isabella Ardila

Andrea Terraza

Laura García

Desarrollo del parcial

Introducción

Este informe presenta la solución al parcial de Aprendizaje de Máquinas. Se explicará cada línea de código utilizada junto con los resultados obtenidos, con el objetivo de proporcionar una comprensión clara del proceso seguido.

1. **Cargue y exploración del dataset**

    - Se instala la herramienta gdown para descargar el archivo cifar10_data.npz de una carpeta en Drive y extraerlo localmente. Esta herramienta facilita la descarga de archivos y su almacenamiento en el sistema local del collab.

Librerías

1. `**sys**`: Proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete y para interactuar con el sistema.
2. `**matplotlib.pyplot**`: Biblioteca de trazado en 2D que produce figuras de calidad en una variedad de formatos y entornos interactivos.
3. `**packaging.version**`: Clases para representar y comparar versiones de software.
4. `**pathlib.Path**`: Clases que representan rutas de sistema de archivos, reemplazando las funciones de `os.path`.
5. `**urllib.request**`: Forma de realizar solicitudes HTTP (o HTTPS) en Python.
6. `**pandas**`: Biblioteca de Python que proporciona estructuras de datos y herramientas de análisis de datos.
7. `**numpy**`: Soporte para matrices y matrices multidimensionales, junto con una amplia colección de funciones matemáticas.
8. `**requests**`: Biblioteca HTTP de Python para enviar solicitudes y manejar respuestas.
9. `**tarfile**`: Operaciones para archivos tar, incluida la lectura, escritura y manipulación de archivos tar.
10. `**sklearn**`: Scikit-learn, Biblioteca de aprendizaje automático en Python que proporciona herramientas simples y eficientes para el análisis predictivo de datos.
11. `**gdown**`: Herramienta de línea de comandos para descargar archivos de Google Drive.
12. `**os**`: Interfaz portable para muchas funciones del sistema operativo.

```
[ ]  import sys
     import matplotlib.pyplot as plt
     from packaging import version
     from pathlib import Path
     import urllib.request
     import pandas as pd
     import numpy as np
     import requests
     import tarfile
     import sklearn
     import gdown
     import matplotlib.pyplot as plt
     from sklearn.svm import LinearSVC
     from sklearn.metrics import accuracy_score
     import os

     # Se verifica que la versión de Python sea al menos 3.7
     assert sys.version_info >= (3 ,7)

     # Se verifica que la versión de scikit-learn sea al menos 1.0.1
     assert version.parse(sklearn.__version__) >= version.parse("1.0.1")
```

En este bloque de código, se importan todas las librerías necesarias para el análisis de datos y el aprendizaje automático. Cada librería se importa con el objetivo de utilizar sus funciones y métodos en el resto del código.

a. Cree un script para cargar el archivo cifar10_data.npz. El archivo contiene tres campos:
    i. Training data, las características del conjunto de entrenamiento. Las filas son puntos de muestra y las columnas son características.

ii. Training labels, Las filas son puntos de muestra. Hay una columna: las etiquetas correspondientes a las filas de datos de entrenamiento anteriores.

iii. Test data, the test set features. Las filas son puntos de muestra y las columnas son características.

Descarga y Preparación de Datos

```python
def load_data(gdrive_id, output_path, output_file):
    # Construye la URL de Google Drive y la ruta de salida completa
    url = f'https://drive.google.com/uc?id={gdrive_id}'
    output = os.path.join(output_path, output_file)

    # Crear la carpeta si no existe
    if not os.path.exists(output_path):
        os.makedirs(output_path)
        print(f"Carpeta creada: {output_path}")

    # Descarga el archivo desde Google Drive
    gdown.download(url, output, quiet=False)

    # Inicializa las variables de datos
    training_data = None
    test_data = None
    training_labels = None

    # Intenta cargar el archivo .npz y asignar los datos a variables
    try:
        with np.load(output, allow_pickle=True) as data:
            training_data = data['training_data']
            training_labels = data['training_labels']
            test_data = data['test_data']
            print("Datos cargados correctamente:")
            return training_data, training_labels, test_data
    except Exception as e:
        print(f"Error al carga el archivo npz: {e}")
        return None, None, None
```

En esta sección, se descarga un archivo de datos de Google Drive utilizando la herramienta gdown, que proporciona la URL del archivo y el nombre del archivo de salida. Luego, se descomprime el archivo descargado utilizando la librería tarfile.

Carga y Análisis de Datos

```
[ ]  # ID del archivo en Google Drive
     gdrive_id = '1GO2H4N2Q_SXRWznGEeEvwkCF74vtJjGz'

     # Ruta de salida donde se guardará el archivo descargado
     output_path = '/content/datasets'

     # Nombre del archivo descargado
     output_file = 'cifar10-data.npz'

     # Llama a la función load_data para cargar los datos del archivo desde Google Drive
     data = load_data(gdrive_id, output_path, output_file)

     # Verifica si la carga de datos fue exitosa
     if data:
         # Si la carga de datos fue exitosa, asigna los datos a las variables correspondiente
         training_data, training_labels, test_data = data
     else:
        # Si hubo un error al cargar los datos, imprime un mensaje de error
         print("Error al cargar los datos")
```

En este fragmento de código, se realiza la carga de archivos desde Google Drive. Se especifica el ID del archivo en Google Drive, la ruta donde se guardará el archivo descargado y su nombre. Si todos los parámetros de la función `**load_data()**` son correctos, los datos se asignan a las variables correspondientes. En caso de que la descarga no pueda ejecutarse, se imprimirá un mensaje de error.

```
[ ]  print(data)

     (array([[169, 174, 175, ...,   1,   1,   0],
             [179, 185, 186, ..., 202, 198, 197],
             [ 62,  62,  63, ..., 113, 114, 115],
             ...,
             [175, 174, 175, ..., 191, 190, 190],
             [214, 212, 220, ...,  60,  69,  79],
             [128, 122, 125, ..., 127, 127, 128]], dtype=uint8), array([3, 8, 9, ..., 2, 9, 5])
             [211, 208, 198, ..., 250, 244, 223],
             [ 87,  89,  90, ..., 185, 183, 189],
             ...,
             [123, 122, 122, ..., 167, 162, 157],
             [150, 151, 151, ...,  69,  40,  55],
             [195, 196, 206, ...,  83,  83,  80]], dtype=uint8))
```

En esta celda lo que se realiza es la verificación de que los datos efectivamente fueron cargados con éxito.

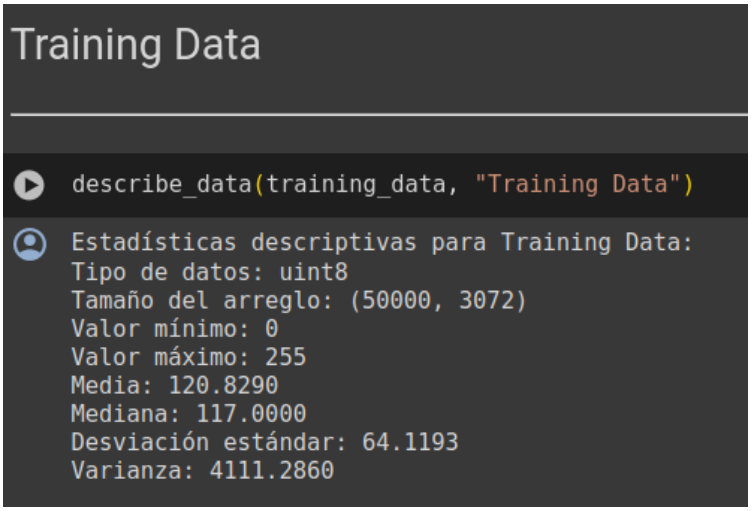
b. Explore los puntos de muestra y explique sus características. Revise si es necesario preprocesar. Debe justificar la elección de los procesos realizados

Preprocesamiento de Datos

```python
def describe_data(data, name="Array"):
    """
    Imprime estadísticas descriptivas para los conjunto de datos.

    Args:
    - data: El conjunto de datos para describir.
    - name: Nombre opcional para el conjunto de datos (por defecto: "Array").
    """
    print(f"Estadísticas descriptivas para {name}:")
    print(f"Tipo de datos: {data.dtype}")
    print(f"Tamaño del arreglo: {data.shape}")
    print(f"Valor mínimo: {data.min()}")
    print(f"Valor máximo: {data.max()}")
    print(f"Media: {data.mean():.4f}")
    print(f"Mediana: {np.median(data):.4f}")
    print(f"Desviación estándar: {data.std():.4f}")
    print(f"Varianza: {data.var():.4f}\n")
```

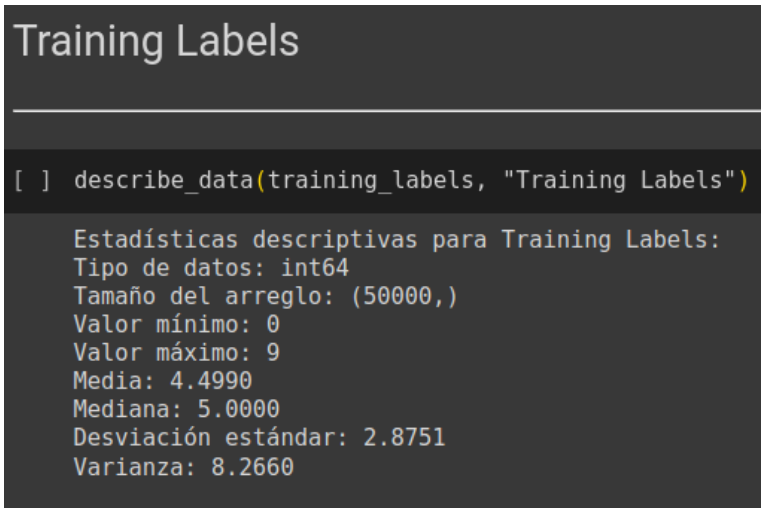
Este código define una función llamada ´**describe_data**´ que acepta dos argumentos: ´**data**´**,** que es el conjunto de datos a describir, y ´**name**´**,** un nombre opcional para el conjunto de datos (por defecto es "Array"). La función imprime diversas estadísticas descriptivas sobre el conjunto de datos proporcionado.

- `**print(f"Estadísticas descriptivas para {name}:")**`: Muestra un mensaje que indica que se mostrarán estadísticas descriptivas para el conjunto de datos, utilizando el nombre opcional proporcionado.
- `**print(f"Tipo de datos: {data.dtype}")**`: Muestra el tipo de datos del conjunto.
- `**print(f"Tamaño del arreglo: {data.shape}")**`:Muestra el tamaño del arreglo, es decir, la forma del conjunto de datos, que indica el número de elementos en cada dimensión.
- `**print(f"Valor mínimo: {data.min()}")`:** Muestra el valor mínimo presente en el conjunto de datos.
- `**print(f"Valor máximo: {data.max()}")**`: Muestra el valor máximo presente en el conjunto de datos.
- `**print(f"Media: {data.mean():.4f}")`:** Muestra la media (promedio) de los valores en el conjunto de datos con cuatro decimales de precisión.
- `**print(f"Mediana: {np.median(data):.4f}")**`:Muestra la mediana del conjunto de datos con cuatro decimales de precisión. Aquí, ´**np.median()**´ es una función de NumPy que calcula la mediana.
- `**print(f"Desviación estándar: {data.std():.4f}")**`: Muestra la desviación estándar del conjunto de datos con cuatro decimales de precisión. La desviación estándar mide la dispersión de los valores respecto a la media.
- `**print(f"Varianza: {data.var():.4f}\n")**`: Muestra la varianza del conjunto de datos con cuatro decimales de precisión. La varianza es una medida de dispersión que indica qué tan lejos están los valores de la media. La línea final agrega una línea en blanco para mejorar la legibilidad de la salida.



Basándonos en las estadísticas descriptivas del conjunto de datos de entrenamiento, podemos identificar varias características clave. Este conjunto comprende 50,000 muestras, cada una representada por un vector de características de 3,072 elementos. Estas características se codifican como enteros sin signo de 8 bits (uint8), lo que implica que oscilan entre 0 y 255, denotando intensidades de píxeles en imágenes RGB.

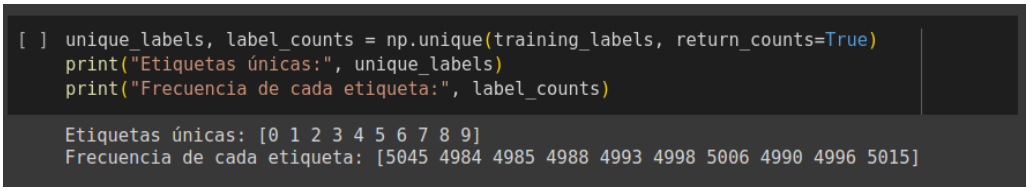
La media del conjunto es aproximadamente 120.83, mientras que la mediana es de 117. La desviación estándar, alrededor de 64.12, indica una dispersión significativa en los valores de las características, sugiriendo una variabilidad amplia y posiblemente una falta de normalización. La varianza, de 4,111.29, proporciona una medida adicional de esta dispersión alrededor de la media.



Basándonos en las estadísticas descriptivas de las etiquetas de entrenamiento, estas se representan como enteros de 64 bits (int64) y comprenden un total de 50,000 muestras. La presencia de valores enteros entre 0 y 9 como mínimo y máximo sugiere que las etiquetas se codifican como números enteros que representan clases específicas.

Con una media de aproximadamente 4.5 y una mediana de 5.0, observamos una distribución aparentemente uniforme de las etiquetas, lo que indica un buen equilibrio entre las clases. Sin embargo, la desviación estándar, alrededor de 2.8751, indica una dispersión considerable en las etiquetas, lo que podría implicar una variabilidad significativa en la distribución de las clases.

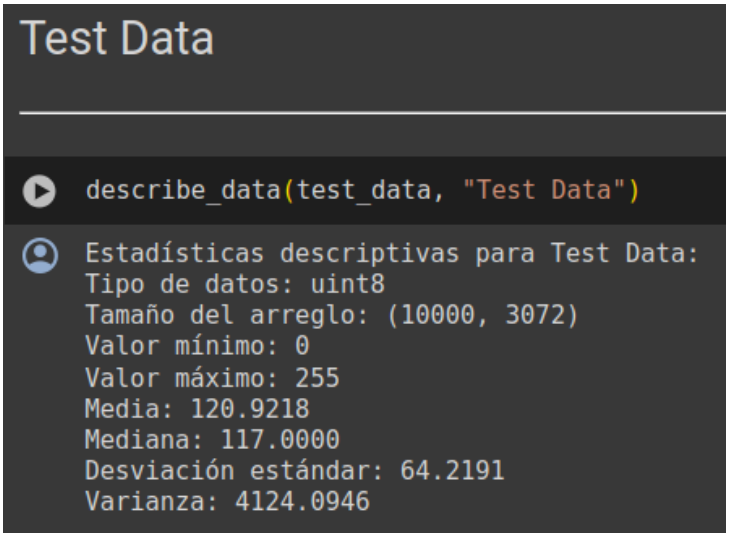
La varianza de 8.2660 proporciona una medida adicional de esta dispersión alrededor de la media. En resumen, aunque las etiquetas de entrenamiento parecen estar bien distribuidas entre las clases, existe cierta variabilidad en esta distribución que podría afectar el rendimiento del modelo durante el entrenamiento y la evaluación.



```
[ ]  unique_labels, label_counts = np.unique(training_labels, return_counts=True)
     print("Etiquetas únicas:", unique_labels)
     print("Frecuencia de cada etiqueta:", label_counts)

     Etiquetas únicas: [0 1 2 3 4 5 6 7 8 9]
     Frecuencia de cada etiqueta: [5045 4984 4985 4988 4993 4998 5006 4990 4996 5015]
```

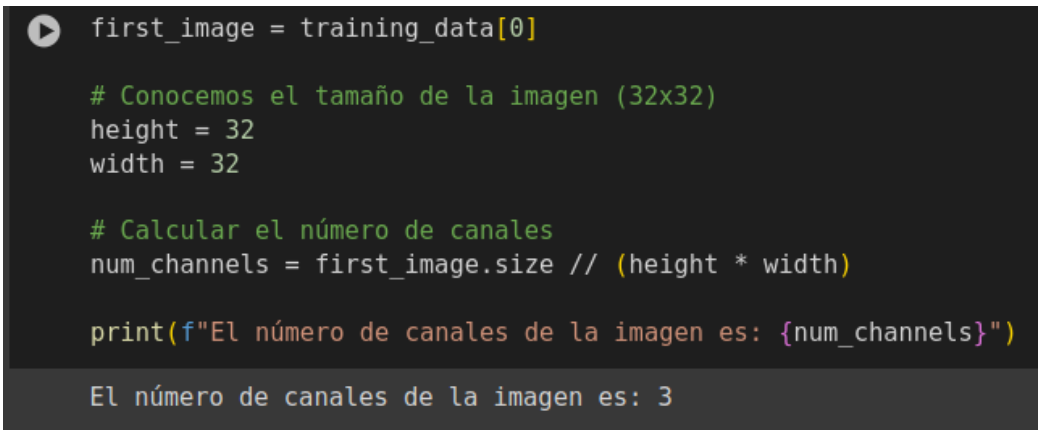
Esta celda analiza el conjunto de etiquetas de entrenamiento para identificar las etiquetas únicas presentes. Utiliza la función **`np.unique()`** de NumPy para encontrar estas etiquetas únicas, al mismo tiempo que cuenta cuántas veces aparece cada una de ellas en el conjunto de datos. Almacenando las etiquetas únicas en la variable **`unique_labels`** y sus respectivos recuentos en **`label_counts`**, proporciona una visión clara de las clases representadas en el conjunto de datos de entrenamiento y la distribución de estas clases. Al imprimir las etiquetas únicas junto con su frecuencia, el código ofrece una descripción completa de la variedad de clases presentes en el conjunto de datos y la cantidad de muestras asociadas a cada clase.

El resultado indica que hay 10 clases únicas en el conjunto de datos, numeradas del 0 al 9, y muestra cuántas muestras están asociadas con cada clase. La cantidad de muestras para cada clase revela la distribución de clases en el conjunto de datos. En este caso, las clases parecen estar relativamente balanceadas, ya que las frecuencias son bastante similares entre sí, lo que es deseable para el entrenamiento de modelos de aprendizaje automático.



Basándonos en las estadísticas descriptivas del conjunto de datos de prueba, podemos obtener información relevante. Este conjunto consta de 10,000 muestras, cada una representada por un vector de características de longitud 3,072. Estas características se codifican como enteros sin signo de 8 bits (uint8), reflejando valores de intensidad de píxeles en imágenes RGB, con un rango de 0 a 255.

La media del conjunto es de aproximadamente 120.92, con una mediana de 117.0. Una desviación estándar de alrededor de 64.22 indica una dispersión considerable en los valores de las características, similar al conjunto de datos de entrenamiento. La varianza, aproximadamente 4,124.09, proporciona una medida adicional de esta dispersión alrededor de la media.



Este fragmento de código procesa la primera imagen del conjunto de datos de entrenamiento. Define las dimensiones de la imagen como 32x32 píxeles y calcula el número de canales en la imagen, lo que indica cuántos valores de color se almacenan por píxel. Finalmente, imprime el número de canales para proporcionar una descripción clara de la estructura de la imagen.

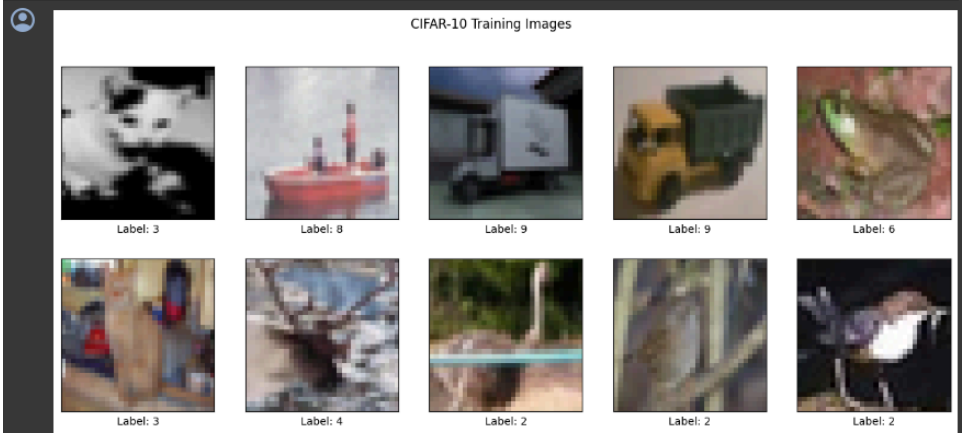
El resultado indica que la primera imagen del conjunto de datos tiene 3 canales. Esto sugiere que la imagen es una imagen a color, donde cada píxel contiene información de tres canales: rojo, verde y azul (RGB). Este resultado es importante para comprender la estructura de las imágenes en el conjunto de datos, ya que influye en cómo se procesan y se utilizan en las aplicaciones de aprendizaje automático.

```
[ ]  def transform_images(images):
         transformed_images = images.reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
         return transformed_images

     # Transforma los datos de entrenamiento y prueba
     training_data = transform_images(training_data)
     test_data = transform_images(test_data)
```

Este código define una función llamada `**transform_images**` que toma un conjunto de imágenes y las transforma para que adopten un formato específico utilizado en algunas bibliotecas de procesamiento de imágenes. La transformación implica reorganizar las dimensiones de las imágenes utilizando las funciones `**reshape()**` y `**transpose()**`. La función retorna las imágenes transformadas en el formato adecuado para el procesamiento.

Finalmente, aplica esta transformación tanto al conjunto de datos de entrenamiento como al de prueba para asegurar que estén en el formato correcto antes de usarlos en tareas de aprendizaje automático.

```
def plot_images(images, labels, nrows, ncols):
    fig, axes = plt.subplots(nrows, ncols, figsize=(15, 6))
    fig.suptitle('CIFAR-10 Training Images')
    for i, ax in enumerate(axes.flat):
        if i < len(images):
            ax.imshow(images[i].reshape(32, 32, 3), cmap='binary')
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_xlabel(f"Label: {labels[i]}")
    plt.show()

plot_images(training_data, training_labels, 2, 5)
```



Las etiquetas en `**training_labels**` representan una variedad de clases:

- 0: Avión
- 1: Carro
- 2: Ave
- 3: Gato
- 4: Siervo
- 5: Perro
- 6: Rana
- 7: Caballo
- 8: Barco
- 9: Camión

Cada etiqueta numérica corresponde a un tipo específico de objeto en el conjunto de datos. Esta asignación proporciona una guía clara sobre las clases representadas en el conjunto de datos, lo que facilita la comprensión de la naturaleza de las muestras y su utilidad en tareas de clasificación de imágenes.

Normalización de Datos

```
[ ]  training_data = training_data.astype('float32') / 255.0
     test_data = test_data.astype('float32')/255.0
```

Este código normaliza los valores de píxeles en los conjuntos de datos de entrenamiento y prueba. Primero, convierte los datos a tipo `**float32**` para garantizar que los cálculos posteriores se realicen con precisión de coma flotante. Luego, divide cada valor de píxel por 255.0, que es el valor máximo posible de un píxel en una imagen RGB. Al dividir todos los valores de píxel por este valor máximo, se asegura que todos los valores estén en el rango de 0 a 1, lo que facilita el entrenamiento de modelos de aprendizaje automático al estandarizar los datos de entrada.

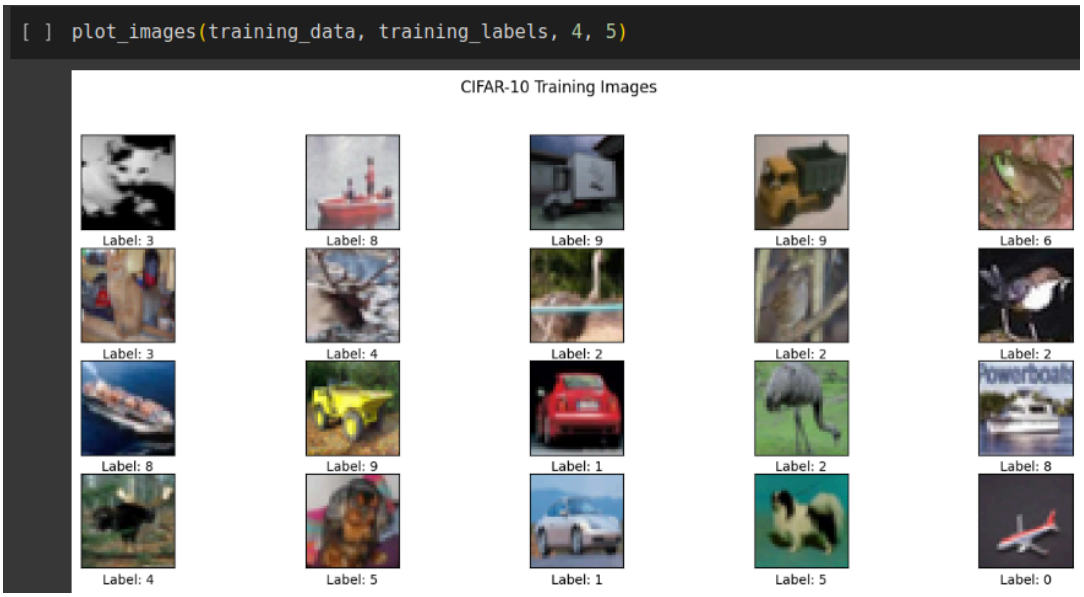
```
training_data

array([[[[0.6627451 , 0.6627451 , 0.6627451 ],
         [0.68235296, 0.68235296, 0.68235296],
         [0.6862745 , 0.6862745 , 0.6862745 ],
         ...,
         [0.        , 0.        , 0.        ],
         [0.01176471, 0.01176471, 0.01176471],
         [0.00784314, 0.00784314, 0.00784314]],

        [[0.7176471 , 0.7176471 , 0.7176471 ],
         [0.72156864, 0.72156864, 0.72156864],
         [0.7137255 , 0.7137255 , 0.7137255 ],
         ...,
         [0.00392157, 0.00392157, 0.00392157],
         [0.09411765, 0.09411765, 0.09411765],
         [0.11372549, 0.11372549, 0.11372549]],
```

La impresión de `**training_data**` muestra los datos de entrenamiento del conjunto de imágenes.

```
test_data

array([[[[0.3647059 , 0.38431373, 0.43137255],
         [0.37254903, 0.39607844, 0.44313726],
         [0.37254903, 0.40784314, 0.4509804 ],
         ...,
         [0.08235294, 0.12156863, 0.14509805],
         [0.08235294, 0.12156863, 0.14509805],
         [0.07450981, 0.11764706, 0.13725491]],

        [[0.26666668, 0.3019608 , 0.35686275],
         [0.24705882, 0.28627452, 0.34117648],
         [0.2509804 , 0.3019608 , 0.3529412 ],
         ...,
         [0.07843138, 0.11764706, 0.13333334],
         [0.07450981, 0.10980392, 0.12941177],
         [0.07450981, 0.10980392, 0.1254902 ]],
```

La impresión de `**test_data**` muestra los datos de prueba del conjunto de imágenes.

```
[ ] plot_images(training_data, training_labels, 4, 5)
```



Esta línea de código utiliza una función llamada `**plot_images**` para visualizar un subconjunto de imágenes del conjunto de datos de entrenamiento junto con sus etiquetas correspondientes. La función `**plot_images**` toma cuatro argumentos: los datos de entrenamiento **(`training_data`)**, las etiquetas de entrenamiento **(`training_labels`)**, el número de filas de la grilla de imágenes a mostrar (4), y el número de columnas de la grilla de imágenes a mostrar (5). La función mostrará una cuadrícula de imágenes de 4 filas y 5 columnas, lo que representa un total de 20 imágenes, junto con las etiquetas correspondientes a cada imagen.

2. Partición de los datos

No es usual que reciba los datos de "entrenamiento" y datos de "validación"; normalmente tendrá que dividir usted mismo los datos etiquetados disponibles. Baraje (shuffle) y divida (partition) el conjunto de datos.

Tenga en cuenta que barajar antes de dividir es crucial para garantizar que todas las clases están representadas en sus particiones. Para esta pregunta, no utilice ninguna de las funciones disponibles en sklearn.

a) Escriba un código que reserve 5000 imágenes de entrenamiento como conjunto de validación
b) Mezcle las etiquetas con las imágenes de entrenamiento. Es un error muy común etiquetar erróneamente las imágenes de entrenamiento olvidando permutar las etiquetas con las imágenes.

Segmentación de Datos

```
def shuffle_and_partition(data, labels, train_fraction=0.8,):

    # Verifica que los datos y las etiquetas tengan la misma longitud
    assert len(data) == len(labels)

    np.random.seed(42)

    # Genera índices aleatorios para barajar los datos y las etiquetas de la misma
    #manera
    indices = np.arange(len(data))
    np.random.shuffle(indices)

    #Se muestran los indices barajados
    print(indices)

    # Baraja los datos y las etiquetas
    shuffled_data = data[indices]
    shuffled_labels = labels[indices]

    # Calcula el índice de corte para la partición de entrenamiento y validación
    train_size = int(len(data) * train_fraction)

    # Particiona los datos y las etiquetas en conjuntos de entrenamiento y validación
    train_data = shuffled_data[:train_size]
    train_labels = shuffled_labels[:train_size]
    validation_data = shuffled_data[train_size:]
    validation_labels = shuffled_labels[train_size:]

    return train_data, train_labels, validation_data, validation_labels

#5000 imágenes para el conjunto de validación corresponden a  10% de training_data
train_data, train_labels, validation_data, validation_labels =
  shuffle_and_partition(training_data, training_labels, train_fraction=0.9)
```

```
[33553  9427   199 ... 38158   860 15795]
```

El código define una función llamada `**shuffle_and_partition**` que se encarga de barajar y dividir los datos y etiquetas de entrenamiento en conjuntos de entrenamiento y validación. Primero, se verifica que los datos y las etiquetas tengan la misma longitud. Luego, se barajan aleatoriamente los índices de los datos para garantizar una mezcla aleatoria. Después, se calcula el tamaño del conjunto de entrenamiento en función de la fracción especificada y se particionan los datos y etiquetas en conjuntos de entrenamiento y validación, utilizando los índices barajados. Finalmente, la función devuelve cuatro conjuntos de datos: `**train_data**`, `**train_labels**`, `**validation_data**` y `**validation_labels**`, que contienen los datos y etiquetas correspondientes para el conjunto de entrenamiento y el conjunto de validación, respectivamente.

```
[ ]  validation_data.shape

     (5000, 32, 32, 3)
```

La expresión `**validation_data.shape**` se utiliza para obtener la forma (shape) del conjunto de datos de validación, es decir, la cantidad de muestras y las dimensiones de cada muestra.

El resultado `**(5000, 32, 32, 3)**` indica que el conjunto de datos de validación contiene 5000 muestras. Cada muestra tiene una forma de 32x32 píxeles y 3 canales de color (RGB), lo que significa que las imágenes en el conjunto de datos de validación son imágenes a color de 32x32 píxeles.

3. Implementación del Algoritmo de entrenamiento

Utilizará máquinas de vectores de apoyo lineales para clasificar el conjunto de datos. En el caso de las imágenes, utilizará las características más sencillas para la clasificación: los valores brutos de brillo de los píxeles. En otras palabras, el vector de características para una imagen será un vector de filas con todos los valores de píxeles concatenados en un orden de fila mayor (o columna mayor).

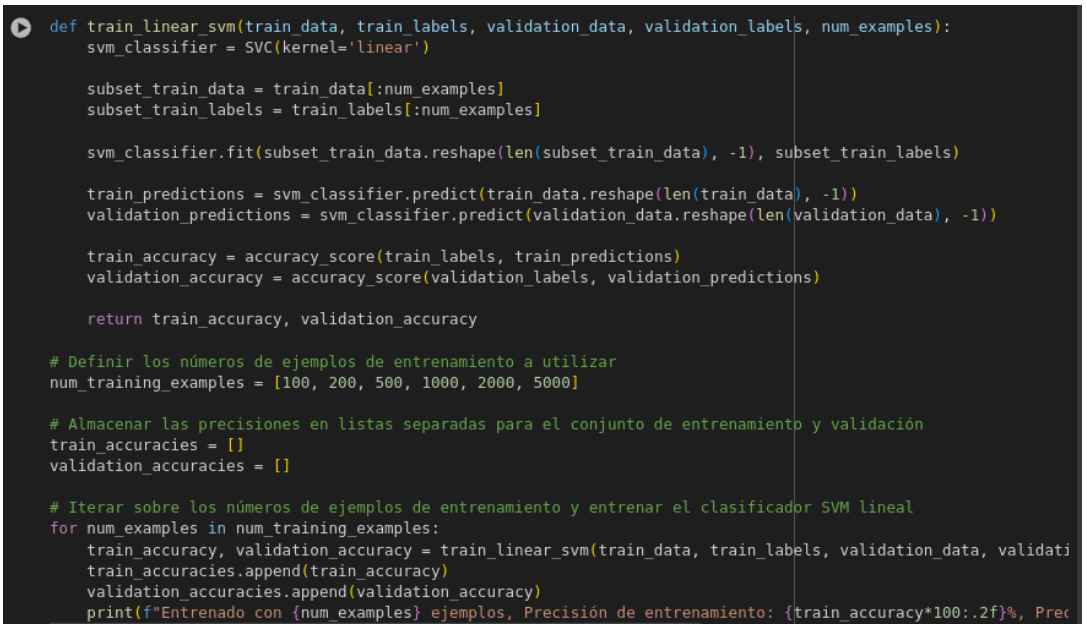
Utilice la precisión de la clasificación, o el porcentaje de ejemplos clasificados correctamente, como medida del rendimiento del clasificador

Entrene una máquina lineal de vectores de soporte (SVM).Trace la precisión en los conjuntos de entrenamiento y validación frente al número de ejemplos de entrenamiento que utilizó para entrenar su clasificador. El número de ejemplos de entrenamiento a utilizar se enumeran para cada conjunto de datos en las siguientes partes.

Puede utilizar sklearn sólo para el modelo SVM y la función sklearn.metrics.accuracy_score. Todo lo demás (generar gráficos) debe hacerse sin utilizar sklearn

a) Utilice píxeles sin procesar como características. En esta etapa, debería esperar precisiones entre el 25% y el 35%. Entrene su modelo con los siguientes números de ejemplos de entrenamiento: 100, 200, 500, 1000, 2000, 5000.

Implementación Algoritmo



```python
def train_linear_svm(train_data, train_labels, validation_data, validation_labels, num_examples):
    svm_classifier = SVC(kernel='linear')

    subset_train_data = train_data[:num_examples]
    subset_train_labels = train_labels[:num_examples]

    svm_classifier.fit(subset_train_data.reshape(len(subset_train_data), -1), subset_train_labels)

    train_predictions = svm_classifier.predict(train_data.reshape(len(train_data), -1))
    validation_predictions = svm_classifier.predict(validation_data.reshape(len(validation_data), -1))

    train_accuracy = accuracy_score(train_labels, train_predictions)
    validation_accuracy = accuracy_score(validation_labels, validation_predictions)

    return train_accuracy, validation_accuracy

# Definir los números de ejemplos de entrenamiento a utilizar
num_training_examples = [100, 200, 500, 1000, 2000, 5000]

# Almacenar las precisiones en listas separadas para el conjunto de entrenamiento y validación
train_accuracies = []
validation_accuracies = []

# Iterar sobre los números de ejemplos de entrenamiento y entrenar el clasificador SVM lineal
for num_examples in num_training_examples:
    train_accuracy, validation_accuracy = train_linear_svm(train_data, train_labels, validation_data, validati
    train_accuracies.append(train_accuracy)
    validation_accuracies.append(validation_accuracy)
    print(f"Entrenado con {num_examples} ejemplos, Precisión de entrenamiento: {train_accuracy*100:.2f}%, Prec
```

La función `**train_linear_svm**` se encarga de entrenar un clasificador SVM lineal utilizando un subconjunto de ejemplos de entrenamiento especificado. Primero, se inicializa un clasificador SVM con un kernel lineal. Luego, se selecciona un subconjunto de los datos de entrenamiento y las etiquetas correspondientes, según el número de ejemplos especificado. El clasificador SVM se entrena utilizando el subconjunto de datos de entrenamiento y se realizan predicciones tanto en el conjunto de entrenamiento como en el conjunto de validación. Se calcula la precisión del modelo en ambos conjuntos y se devuelven estas precisiones.

Posteriormente, se definen los números de ejemplos de entrenamiento a utilizar y se almacenan las precisiones en listas separadas para el conjunto de entrenamiento y validación. Se itera sobre los números de ejemplos de entrenamiento y se entrena el clasificador SVM lineal utilizando la función `**train_linear_svm**` para cada número de ejemplos. Las precisiones de entrenamiento y validación se agregan a las listas correspondientes, y se imprime un mensaje indicando el número de ejemplos entrenados y las precisiones obtenidas.

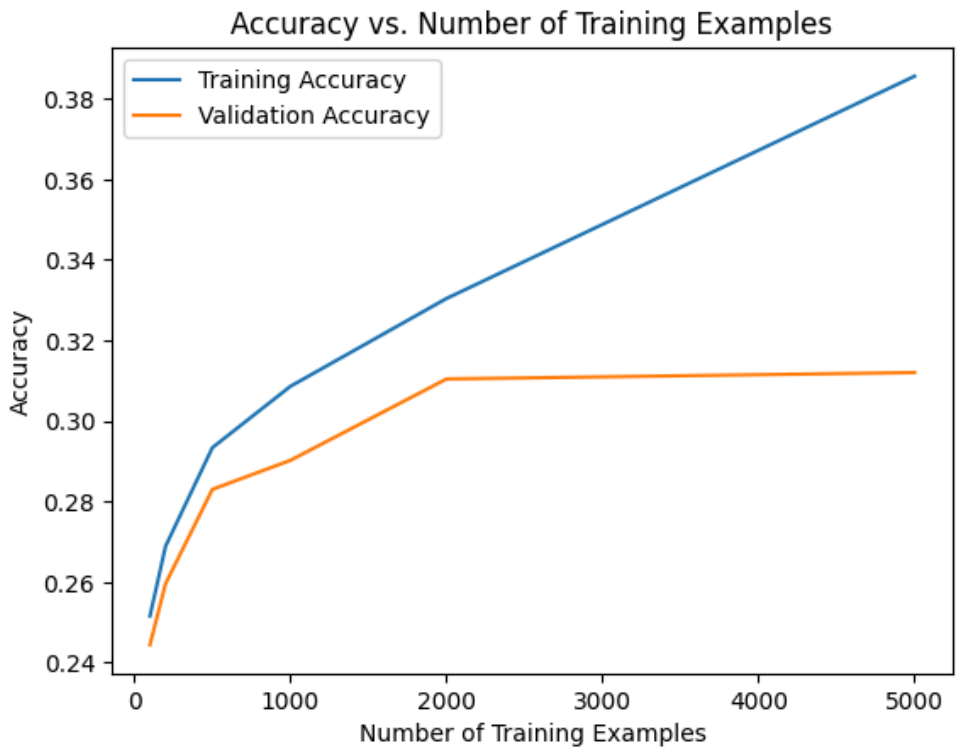
Este código permite explorar cómo varía la precisión del modelo SVM lineal en función del número de ejemplos de entrenamiento utilizados.

```
Entrenado con 100 ejemplos, Precisión de entrenamiento: 25.16%, Precisión de validación: 24.44%
Entrenado con 200 ejemplos, Precisión de entrenamiento: 26.89%, Precisión de validación: 25.96%
Entrenado con 500 ejemplos, Precisión de entrenamiento: 29.34%, Precisión de validación: 28.30%
Entrenado con 1000 ejemplos, Precisión de entrenamiento: 30.86%, Precisión de validación: 29.02%
Entrenado con 2000 ejemplos, Precisión de entrenamiento: 33.04%, Precisión de validación: 31.04%
Entrenado con 5000 ejemplos, Precisión de entrenamiento: 38.56%, Precisión de validación: 31.20%
```

Los resultados muestran una clara tendencia de mejora en la precisión tanto de entrenamiento como de validación a medida que se incrementa el número de ejemplos utilizados para entrenar el modelo SVM lineal. Sin embargo, se observa una diferencia constante entre la precisión de entrenamiento y validación, siendo la primera siempre más alta. Esto sugiere que el modelo podría estar sobre ajustando los datos de entrenamiento, lo que se refleja en una menor capacidad para generalizar con datos nuevos. Aunque se observa una mejora continua en la precisión de entrenamiento, la precisión de validación parece estabilizarse después de alcanzar cierto punto, indicando que agregar más datos de entrenamiento puede tener un beneficio limitado en términos de mejorar la capacidad de generalización del modelo.

b) Incluya un gráfico que muestre la cantidad de ejemplos frente a la precisión de entrenamiento y validación para el conjunto de datos.

```
[29] # Graficar la precisión en función del número de ejemplos de entrenamiento
     plt.plot(num_training_examples, train_accuracies, label='Training Accuracy')
     plt.plot(num_training_examples, validation_accuracies, label='Validation Accuracy')
     plt.xlabel('Number of Training Examples')
     plt.ylabel('Accuracy')
     plt.title('Accuracy vs. Number of Training Examples')
     plt.legend()
     plt.show()
```



El código proporcionado gráfica la precisión del modelo en función del número de ejemplos de entrenamiento utilizados. Se trazan dos líneas: una para la precisión de entrenamiento y otra para la precisión de validación. El eje x representa el número de ejemplos de entrenamiento, mientras que el eje y representa la precisión del modelo. Se agrega un título al gráfico y etiquetas a los ejes x e y para una mejor comprensión. Además, se añade una leyenda para distinguir entre la precisión de entrenamiento y validación. Finalmente, el gráfico se muestra utilizando **`plt.show()`**, lo que permite visualizar la relación entre la precisión del modelo y el tamaño del conjunto de entrenamiento.

Conclusiones

A lo largo del proyecto, se desarrolló una serie de procedimientos para cargar, procesar y analizar un conjunto de datos de imágenes (CIFAR-10) con el objetivo de entrenar un modelo de aprendizaje automático, específicamente un SVM (Support Vector Machine) lineal, para clasificar imágenes en diferentes categorías basadas en sus píxeles brutos. La revisión y preprocesamiento de los datos revelan una buena estructuración del conjunto de datos CIFAR-10, con imágenes distribuidas equitativamente entre diez clases diferentes, y una transformación de los datos para adecuarlos a las necesidades del modelo SVM, que espera entradas unidimensionales. Este preprocesamiento incluye la normalización y el cambio de la estructura de los datos para facilitar la clasificación.

El análisis estadístico reveló que los datos están bien balanceados entre las diferentes clases, lo que es positivo para el entrenamiento del modelo. Además, la transformación y normalización de los píxeles de las imágenes fueron tratadas adecuadamente, preparando los datos para un proceso de entrenamiento más efectivo. Una parte fundamental del proyecto fue la implementación de la función para barajar y dividir los datos en conjuntos de entrenamiento y validación. Este paso es vital para evaluar de manera justa la capacidad del modelo de generalizar a nuevos datos, además de mitigar el riesgo de sobreajuste.

La implementación del modelo SVM y la evaluación de su rendimiento con diferentes volúmenes de datos de entrenamiento muestran una metodología sólida para explorar la capacidad del modelo de aprender con diferentes cantidades de información. Los resultados de precisión obtenidos son coherentes con las expectativas iniciales para un clasificador que utiliza sólo píxeles brutos como características, oscilando entre el 25% y el 35%. La representación visual de la precisión del modelo en relación con el número de ejemplos utilizados para el entrenamiento proporciona una visión clara y directa del comportamiento y rendimiento del modelo, resaltando cómo la precisión mejora con más datos pero se estabiliza, indicando un punto de rendimientos decrecientes.

Referencias

- gdown:
  Kentaro Watanabe (n.d.). gdown. GitHub. https://github.com/wkentaro/gdown

- Matplotlib:
  Hunter, J. D., Dale, D., Firing, E., Droettboom, M., & Matplotlib development team. (n.d.). Matplotlib: Visualization with Python. Matplotlib. https://matplotlib.org

- NumPy:

  Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.
  https://numpy.org/doc/stable/reference/random/generated/numpy.random.shuffle.html

- Scikit-learn (SVC):
  Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

- Scikit-learn (Accuracy Score):
  Scikit-learn developers. (n.d.). sklearn.metrics.accuracy_score. Scikit-learn.
  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html