Lab 03
Transform Textual and Image Data into Machine Understandable Format

Lab 03 - Transform Textual and Image Data into Machine Understandable Format

1. Introduction and Objectives

1.1 Introduction

Raw data, whether it's a block of text, an audio file, or a picture, cannot be directly used by most traditional Machine Learning (ML) algorithms. These algorithms require data to be in a numerical, vector-based format. **Feature engineering** is the process of using domain knowledge to select, transform, or create features that enable an ML algorithm to work effectively.
This lab will introduce two fundamental techniques to convert common unstructured data formats—text and images—into a machine-understandable numerical format.

1.2 Learning Objectives

Upon completing this lab, students will be able to:

1. Understand why raw text and image data need transformation for ML.
2. Apply the **Bag-of-Words (BoW)** technique to convert text into a numerical feature vector.
3. Implement a simple technique to convert an image into a numerical array of pixel values.
4. Perform basic preprocessing steps like **Grayscale Conversion** for images.

2. Setup and Prerequisites

2.1 Required Libraries

The following Python libraries are required for this lab:

Library	Purpose	Installation Command
scikit-learn	For the CountVectorizer (text feature extraction).	pip install scikit-learn
Pillow (PIL)	For image loading and manipulation.	pip install Pillow
NumPy	For numerical array operations.	pip install numpy

2.2 Sample Data (No Files Needed)

To keep the lab self-contained and simple, all data will be created directly within the Python script:

- **Text Data:** A small list of short sentences (documents).
- **Image Data:** A dummy image.

3. Part A: Feature Engineering for Text Data

In this section, you will use the **Bag-of-Words (BoW)** model to convert text data into a numerical vector. BoW represents a text document as the count of word occurrences in that document, ignoring grammar and word order.

3.1 Task A.1: Vectorizing Text with CountVectorizer

Goal: Transform three sample documents into a matrix where each row is a document and each column is the count of a unique word (feature).

Python

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# 1. Define Sample Text Data (Our "Corpus")
documents = [
    "The sun is shining today.",
    "The weather is good, the sun is great.",
    "A sunny day is a wonderful day."
]
print("--- Raw Text Documents ---")
for i, doc in enumerate(documents):
    print(f"Doc {i+1}: {doc}")

# 2. Apply Feature Engineering: CountVectorizer (Bag-of-Words)
# The CountVectorizer performs tokenization and vocabulary building.
vectorizer = CountVectorizer()

# fit_transform learns the vocabulary and converts the documents into feature vectors
X_text = vectorizer.fit_transform(documents)

# 3. Analyze the Results
feature_names = vectorizer.get_feature_names_out()
text_matrix = X_text.toarray()

print("\n--- Bag-of-Words (BoW) Transformation ---")
print(f"Vocabulary (Features): {feature_names}")
print(f"Shape of Feature Matrix: {text_matrix.shape}")

# Create a DataFrame for a clean view (for student understanding)
df_text = pd.DataFrame(text_matrix, columns=feature_names,
                       index=[f"Doc {i+1}" for i in range(len(documents))])
```

```
print("\nBoW Numerical Feature Matrix (Machine Understandable Format):")
print(df_text)
```

3.2 Discussion Questions (Part A)

1. What does each column in the resulting matrix represent?
2. In this specific numerical matrix, what is the 'machine-understandable format'?
3. How is this format an oversimplification of the original text data? *(Hint: Consider lost information).*

Expected Output Structure (Self-Correction/Reference)

The final code should run and produce output similar to the following (random numbers will vary):

```
--- Raw Text Documents ---
Doc 1: The sun is shining today.
Doc 2: The weather is good, the sun is great.
Doc 3: A sunny day is a wonderful day.

--- Bag-of-Words (BoW) Transformation ---
Vocabulary (Features): ['day' 'good' 'great' 'is' 'shining' 'sun' 'sunny' 'the' 'today' 'wonderful'
'weather']
Shape of Feature Matrix: (3, 11)

BoW Numerical Feature Matrix (Machine Understandable Format):
      day  good  great  is  shining  sun  sunny  the  today  wonderful  weather
Doc 1    0     0      0   1        1    1      0    1      1          0        0
Doc 2    0     1      1   1        0    1      0    1      0          0        1
Doc 3    1     0      0   1        0    0      1    0      0          1        0
```

4. Part B: Feature Engineering for Image Data

In this section, you will convert an image into a numerical array. The raw pixel values themselves serve as the features for many basic image ML models. The feature engineering technique here involves **Grayscale Conversion** and **Pixel Value Flattening**.

4.1 Task B.1: Converting and Flattening Image Pixels

Goal: Load an image (either a file or a generated dummy), convert it to grayscale (reducing channels from 3 to 1), and then flatten the resulting pixel matrix into a single feature vector.

```
Python
import numpy as np
from PIL import Image
import os # To check if a file exists

# --- Configuration for Image Loading ---
# IMPORTANT:
```

```
# You can replace 'path/to/your/image.jpg' with the actual path to an image file on your
computer.
# Example: 'my_image.png' (if in the same directory as the script)
# or 'C:/Users/YourUser/Pictures/my_photo.jpg' (on Windows)
# or '/home/youruser/images/my_photo.png' (on Linux/macOS)
image_file_path = None # Set to None to use dummy image, or a string for your image path

# 1. Load Image or Create a Dummy Image
original_image = None
if image_file_path and os.path.exists(image_file_path):
    try:
        original_image = Image.open(image_file_path)
        print(f"--- Loaded Image from File: {image_file_path} ---")
    except Exception as e:
        print(f"Error loading image from {image_file_path}: {e}")
        print("Creating a dummy image instead.")
        # Fallback to dummy image if file loading fails
        dummy_image_data = np.random.randint(0, 256, size=(100, 100, 3), dtype=np.uint8)
        original_image = Image.fromarray(dummy_image_data, 'RGB')
        print(f"--- Created Dummy Image (100x100 RGB) ---")
else:
    print("No valid image file path provided or file not found.")
    print("Creating a dummy image instead.")
    # Create a simple 100x100 RGB Image if no valid path is given
    dummy_image_data = np.random.randint(0, 256, size=(100, 100, 3), dtype=np.uint8)
    original_image = Image.fromarray(dummy_image_data, 'RGB')
    print(f"--- Created Dummy Image (100x100 RGB) ---")

# Ensure the image is in RGB format for consistent processing if it wasn't already
original_image = original_image.convert('RGB')
original_image_array = np.array(original_image)

print(f"Original Image Size (H, W, Channels): {original_image_array.shape}")
print(f"Total Features (Pixels) in RGB: {original_image_array.shape[0] *
original_image_array.shape[1] * original_image_array.shape[2]}")

# Display the original image (optional, requires matplotlib but good for visualization)
# from matplotlib import pyplot as plt
# plt.imshow(original_image)
# plt.title("Original Image")
# plt.axis('off')
# plt.show()


# 2. Apply Simple Feature Engineering: Grayscale Conversion
# Grayscale is a simple preprocessing/feature reduction technique.
grayscale_image = original_image.convert('L') # 'L' mode is for Grayscale
grayscale_array = np.array(grayscale_image)

print(f"\n--- Grayscale Conversion (Feature Reduction) ---")
print(f"Grayscale Image Size (H, W, Channels): {grayscale_array.shape}")
print(f"Total Features (Pixels) after Grayscale: {grayscale_array.shape[0] *
grayscale_array.shape[1]}")
```

```
# Display the grayscale image (optional)
# plt.imshow(grayscale_image, cmap='gray')
# plt.title("Grayscale Image")
# plt.axis('off')
# plt.show()

# 3. Apply Simple Feature Engineering: Pixel Flattening
# Flattening converts the 2D (or 3D) matrix into a 1D feature vector
# for input into a traditional ML algorithm.
flattened_features = grayscale_array.flatten()

print(f"\n--- Pixel Flattening (Vectorization) ---")
print(f"Flattened Feature Vector Shape: {flattened_features.shape}")

print("\nFirst 10 Features (Pixel Values) in Machine Understandable Format:")
print(flattened_features[:10])

# Verify the type - it's a numerical array, ready for an ML model
print(f"\nData Type of Final Features: {flattened_features.dtype}")
```

4.2 Discussion Questions (Part B)

1. Explain how a grayscale conversion acts as a simple feature engineering technique in the context of machine learning.
2. If the original image loaded had dimensions pixels, what is the length of the final flattened feature vector after grayscale conversion?
3. Why is the step of "flattening" necessary before feeding the data to a simple ML model like a Logistic Regression or a Support Vector Machine?
4. *(New Question)* What would be the advantage of resizing all images to a consistent size (e.g., 64x64 pixels) before applying grayscale conversion and flattening, especially if you were building an ML model that processes many different images?

5. Lab Submission

Students should submit their completed Python code file along with the answers to the Discussion Questions (Part A and Part B).

Expected Output Structure (Self-Correction/Reference)

The output for Part B will now vary based on the image size if a file is loaded. If the dummy image is used, it will be 100x100 pixels.

```
No valid image file path provided or file not found.
Creating a dummy image instead.
--- Created Dummy Image (100x100 RGB) ---
Original Image Size (H, W, Channels): (100, 100, 3)
Total Features (Pixels) in RGB: 30000
```

--- Grayscale Conversion (Feature Reduction) ---
Grayscale Image Size (H, W, Channels): (100, 100)
Total Features (Pixels) after Grayscale: 10000

--- Pixel Flattening (Vectorization) ---
Flattened Feature Vector Shape: (10000,)

First 10 Features (Pixel Values) in Machine Understandable Format:
[140 220 188 123 205 156 199 110 176 211] # These values will be random

Data Type of Final Features: uint8