Machine Learning Workshop

Coding Environment

With Google Colab

Get Started

You can check the tutorial also to get start: Google Colab Tutorial for Beginners

Environment Setup

1. **Open Google Colab**:
    a. Navigate to the Google Colab website (colab.research.google.com). If you're not already signed in to your Google account, you will need to sign in.
2. **Start a New Notebook**:
    a. Once you're on the Google Colab homepage, you can start a new notebook by clicking on New Notebook.
    b. This option is typically found in a prominent location, such as in the bottom right corner or in the center of the page.
3. **Rename the Notebook (Optional):**
    a. By default, Colab will name your new notebook something like "Untitled0.ipynb". To rename it, click on the filename at the top of the notebook and enter your desired name.
4. **Start Using Your Notebook:**
    a. Now that your notebook is created, you can start using it. You can write and execute Python code in code cells, and add text explanations or instructions in text cells.
5. **Saving the Notebook:**
    a. Google Colab automatically saves your notebook to your Google Drive in a folder named 'Colab Notebooks'. You can also manually save or make a copy by going to the File menu and selecting Save or Save a copy in Drive.
6. **Sharing the Notebook (Optional):**
    a. If you want to share your notebook with others, click on the Share button located at the top right of the page. You can then add the email addresses of the people you want to share it with and set their permissions (e.g., viewer, commenter, or editor).

Operating Google Colab

1. **Run a Cell:**
   a. Code Cell: To run a code cell, click on the cell and then press the play button (▶️) on the left side of the cell or use the shortcut Shift + Enter. This will execute the Python code in that cell.
   b. Text Cell: Text cells don't 'run' in the same way as code cells. Instead, they are for documentation. Double-click on a text cell to edit it, and use Shift + Enter to render the text.
2. **Add a New Cell:**
   a. Add Code Cell: Click on + Code in the top left toolbar. Alternatively, hover between cells and click on the + Code button that appears.
   b. Add Text Cell: Click on + Text in the top left toolbar. Similarly, you can hover between cells and click on the + Text button that appears.
3. **Delete a Cell:**
   a. Select the cell by clicking on it. Then, use the toolbar options to delete it, or press Ctrl/Cmd + M D.
4. **Copy, Cut, and Paste Cells:**
   a. Use the toolbar options or keyboard shortcuts (Ctrl/Cmd + C to copy, Ctrl/Cmd + X to cut, and Ctrl/Cmd + V to paste) after selecting a cell.
5. **Connect to a Runtime:**
   a. If you're not connected, click on the 'Connect' button in the top-right corner.
6. **Disconnecting from a Runtime:**
   a. To manually disconnect, click on the 'Connected' (or 'Busy', if running a cell) status in the top-right, and then click on 'Disconnect'.
7. **Change Runtime Type (Optional):**
   a. If you need a GPU or TPU for your computations, go to Runtime > Change runtime type, and select your desired hardware accelerator.
8. **Commenting and Sharing:**
   a. Add comments to any cell by clicking the comment button (speech bubble icon) in the top-right of the cell. Share the notebook using the 'Share' button.
9. **Saving and Downloading:**
   a. Your notebook is automatically saved to Google Drive. You can also download it in various formats (File > Download).

Project 1

Cricketer Classification

With python

Download Dataset

You can download the dataset from our course repository (under workshop I):

https://github.com/Kanchon-Gharami/ML_with_AI_Bangladesh

Coding

After opening a new notebook, upload the **cricket_dataset.csv** on your colab computer. Then run below.

Step 1: Import Libraries

First, you'll need to import the necessary libraries in your Colab notebook:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load the Dataset

Load your dataset into a Pandas DataFrame:

```
# Load the dataset
df = pd.read_csv('cricket_dataset.csv')

# Display the first few rows of the dataframe
print(df.head())
```

Then We Can visualize the data with scatter plot:

```
# Map labels to colors
color_dict = {'batsman': 'blue', 'bowler': 'green', 'all_rounder':
'orange'}

# Create the plot
plt.figure(figsize=(10, 5))
sns.scatterplot(x='total_runs', y='wickets', data=df, hue='label',
palette=color_dict)

# Adding titles and labels
plt.title('Dataset Visualization')
plt.xlabel('Total Runs')
plt.ylabel('Wickets')
plt.legend(title='Player Role')

# Show the plot
plt.show()
```

Step 3: Preprocess the Data

You might need to encode the categorical variables. Here encoding refer string/words to some numerical representation. Since your dataset might already be in a suitable format, this step could be minimal:

```
# Assuming df is your DataFrame
X = df[['total_runs', 'wickets']]  # Features: only total_runs and wickets
y = df['label']  # Target: label

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step 4: Train the Random Forest Classifier

Now, train the Random Forest Classifier using the training data:

```
# Initialize the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Train the classifier
clf.fit(X_train, y_train)
```

Step 5: Evaluate the Model

After training the model, evaluate its performance on the test set:

```
# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the classifier
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Step 6: Make Predictions

You can now use the trained model to make predictions on new data:

```
# Example: Predict the class for a new player based on runs and wickets
# Example: Predict the class for Lasith Malinga with 150 runs and 5 wickets
new_player_data = [[70, 4]]

# Predict the class
predicted_class = clf.predict(new_player_data)
print("Predicted Class:", predicted_class[0])
```

Step 7: Visualize the Prediction

```
# Map labels to colors
color_dict = {'batsman': 'blue', 'bowler': 'green', 'all_rounder':
'orange'}

# Create the plot for the existing dataset
plt.figure(figsize=(10, 5))
sns.scatterplot(x='total_runs', y='wickets', data=df, hue='label',
palette=color_dict)
```

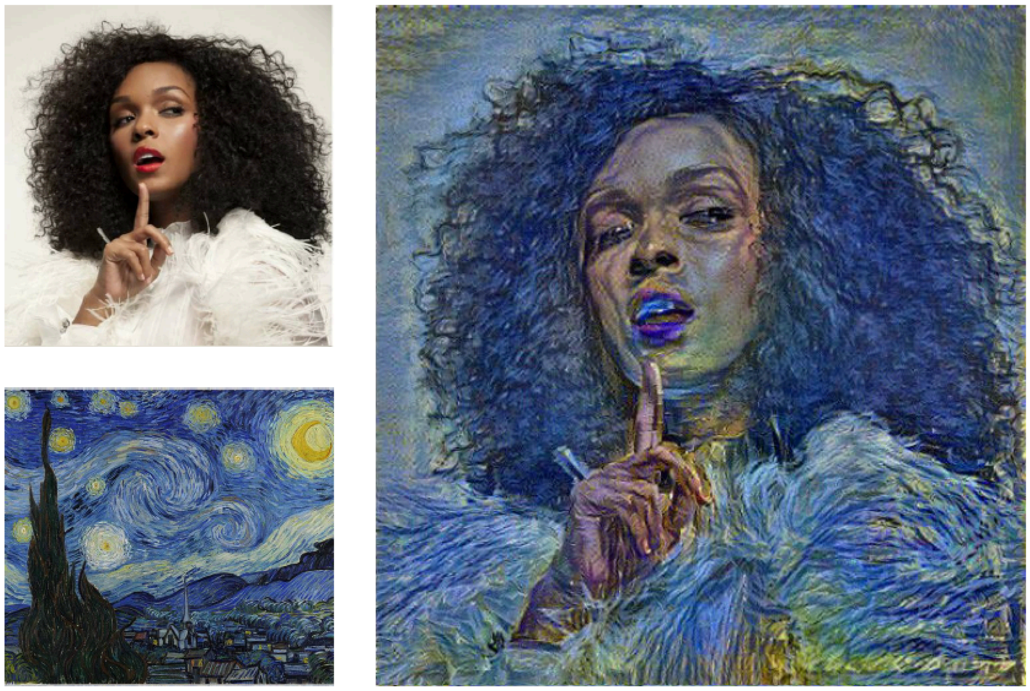
```python
# Plot the new player's data
# 'new_player_data' is a list of lists, so we index into it with [0]
plt.scatter(new_player_data[0][0], new_player_data[0][1], color='red',
label='New Player (Predicted)')

# Adding titles, labels, and legend
plt.title('Cricket Players Classification with New Player')
plt.xlabel('Total Runs')
plt.ylabel('Wickets')
plt.legend(title='Player Role')

# Show the plot
plt.show()
```

Project 2

Neural Style Transfer

With python

Neural style transfer is an optimization technique used to take two images—a content image and a style reference image (such as an artwork by a famous painter)—and blend them together so the output image looks like the content image, but “painted” in the style of the style reference image.



Coding

Step 1: Import Libraries

First, you'll need to import the necessary libraries in your Colab notebook. In this step, various Python libraries are imported to enable different functionalities. The code begins by importing TensorFlow (tf), a versatile library essential for machine learning tasks, alongside TensorFlow Hub (hub), which facilitates the use of pre-trained models. For visual representation, matplotlib.pyplot is used, enabling the creation and manipulation of plots and graphs. Numerical computations and array manipulations are handled efficiently with NumPy (np). Image processing tasks, such as loading and modifying image files, are managed through the Python Imaging Library (PIL). Finally, the os module provides utilities for file and directory operations, crucial for managing file paths and system interactions within the project. Each import plays a distinct role, collectively supporting the diverse functionalities required in machine learning and data processing tasks.

```
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import os
import requests
from io import BytesIO
```

Step 2: Load the Pre-trained Model

In Step 2, the project loads a pre-trained model from TensorFlow Hub, an online repository of machine learning models. Specifically, it retrieves an 'arbitrary image stylization' model, which is designed for applying artistic style transfer to images. This process involves blending the style of one image (like a painting) with the content of another (such as a photograph), creating visually striking results. The model is loaded using the hub.load method with a URL pointing to the desired model on TensorFlow Hub, making it ready for immediate use in style transfer tasks. This approach eliminates the need for training a model from scratch, significantly simplifying and speeding up the process.

```
# Load the model from TensorFlow Hub
hub_model =
hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-2
56/2')
```

Step 3: Prepare Content and Style Images

In Step 3, the project prepares two key images for style transfer: the content and style images. A function, load_img, is designed to load and process these images. It first reads and decodes each image, then resizes it to a maximum dimension of 512 pixels to ensure consistency and computational efficiency. The resizing maintains the image's aspect ratio and normalizes its pixel values. The function also adjusts the image format to be compatible with the model's requirements. Finally, the content and style images are loaded using this function, setting the stage for their transformation in the style transfer process.

```
def load_img_from_url(url):
    max_dim = 512

    # Fetch the image from URL
    response = requests.get(url)
    response.raise_for_status()
    img_data = response.content

    # Decode the image data
    img = tf.image.decode_image(img_data, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)

    # Resize the image
    shape = tf.cast(tf.shape(img)[:-1], tf.float32)
    long_dim = max(shape)
    scale = max_dim / long_dim
    new_shape = tf.cast(shape * scale, tf.int32)

    img = tf.image.resize(img, new_shape)
    img = img[tf.newaxis, :]
    return img
```

```
# Load your images
content_image =
load_img_from_url('https://i.pngimg.me/thumb/f/720/e6f82de227.jpg')
style_image =
load_img_from_url('https://i.pinimg.com/originals/56/ca/08/56ca0884d4c9ee80
2fe2770b30666944.jpg')
```

Step 3.1 (Optional): If you wish to load your image!

```
def load_img(path_to_img):
    max_dim = 512
    img = tf.io.read_file(path_to_img)
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)

    shape = tf.cast(tf.shape(img)[:-1], tf.float32)
    long_dim = max(shape)
    scale = max_dim / long_dim

    new_shape = tf.cast(shape * scale, tf.int32)

    img = tf.image.resize(img, new_shape)
    img = img[tf.newaxis, :]
    return img

# Load your images
content_image = load_img('/content/potrait_1.jpg')
style_image = load_img('/content/art_1.jpg')
```

Step 4: Perform Style Transfer

In Step 4, the style transfer is executed using the pre-loaded TensorFlow Hub model. The content and style images are input into the model, which then generates a new image, stylized_image, that combines the style of one image with the content of the other, demonstrating the application of neural style transfer techniques.

```
# Perform style transfer
stylized_image = hub_model(tf.constant(content_image),
tf.constant(style_image))[0]
```

Step 5: Visualize the Results

In Step 5, the project showcases the results of the style transfer. A helper function, imshow, is used to properly format and display the images. It adjusts the images, if necessary, and utilizes Matplotlib for visualization. The content, style, and stylized images are displayed side by side for comparison. This visual presentation highlights the original images and the transformative effect of the style transfer, providing a clear and immediate understanding of the process and its outcome.

```python
# Helper function to deprocess and display images
def imshow(image, title=None):
    if len(image.shape) > 3:
        image = tf.squeeze(image, axis=0)
    plt.imshow(image)
    if title:
        plt.title(title)

# Display images
plt.figure(figsize=(12, 5))
plt.subplot(1, 3, 1)
imshow(content_image, 'Content Image')

plt.subplot(1, 3, 2)
imshow(style_image, 'Style Image')

plt.subplot(1, 3, 3)
imshow(stylized_image, 'Stylized Image')
plt.show()
```