Machine learning & Deep learning Mini-Project

M.Sc Part II Computer Science

Mithun Parab 509

August 12, 2023

R.J. College of Arts, Science & Commerce
Machine learning & Deep learning
Seat number: 509

Contents

Link for GitHub or Google Colab

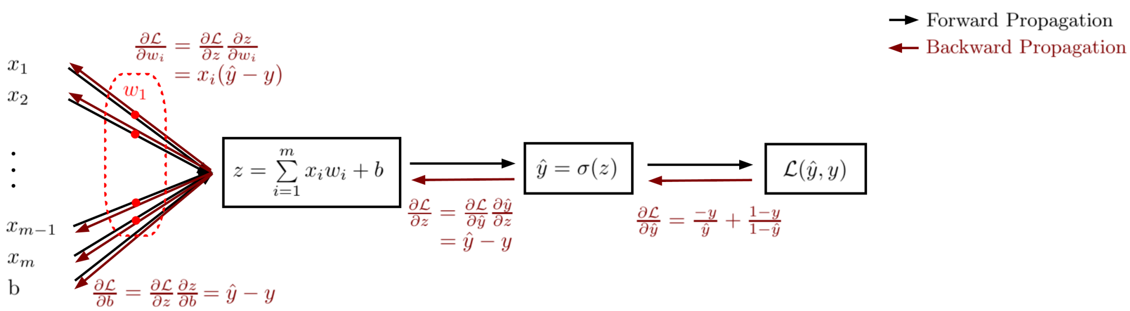
1 Mini-Project: Handwritten digit classifier using logistic regression.

In this mini project we're going to implement from scratch a one-vs-all logistic regression classifier for the MNIST digits dataset with a neural network mindset. The neural network aspect of this implementation is the use of a forward and backward propagation to claculate the value of the cost function and the partial derivatives of the cost function with respect to weights and the bias.

1.1 Introduction

We're going to employ a forward and backward propagation method to train the model as we implement the logistic regression algorithm using a Neural Netwotk mindset. The following figure shows how to predict which digit will appear in a particular image:

When it comes to the training phase, the forward propagation uses a loss function to determine the cost, and the backward propagation uses the chain rule to calculate the partial derivates of the cost function with regard to the weights and bias. This method of implementation imitates the forward and backward propagation used in neural network training. The two stages of the training process are depicted in the diagram below:



Loss and cost functions:

- Loss function:

$$\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right)$$

- Cost function:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Partial derivatives of the cost functions:

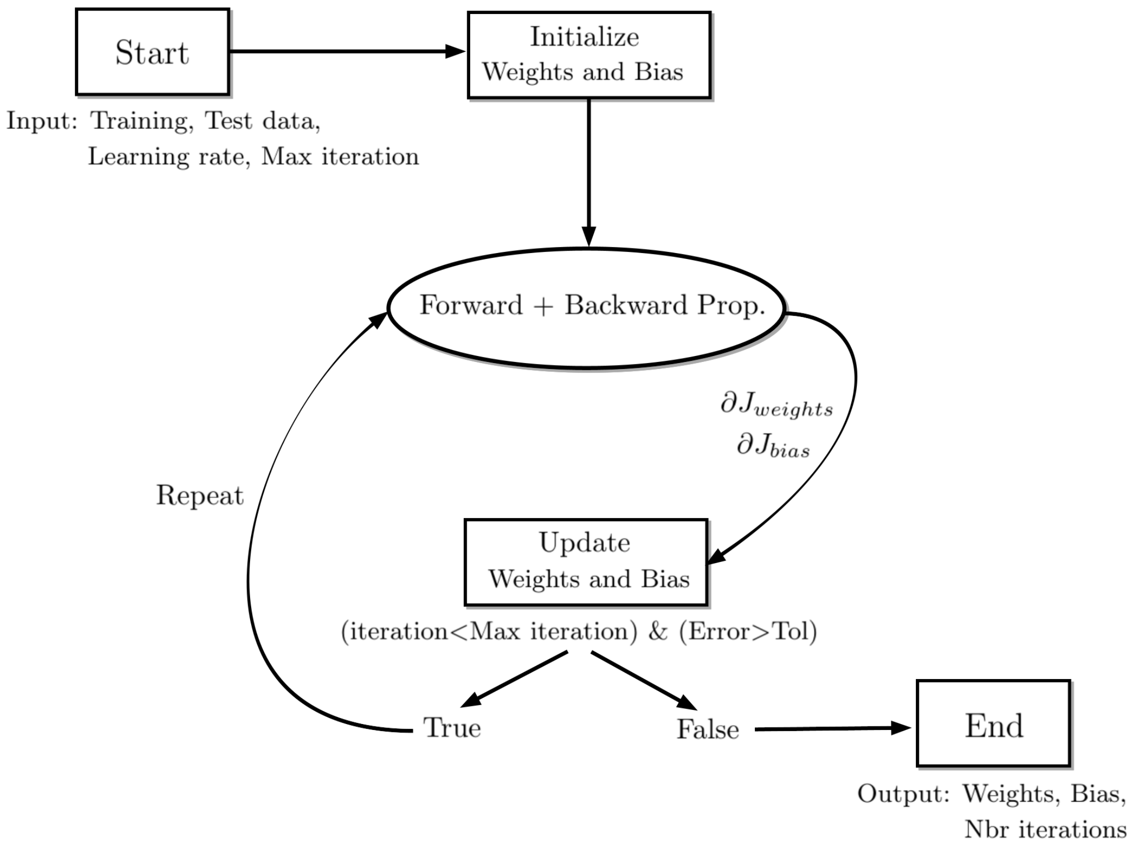
The vectorized form of the partial derivatives of the cost function J with respect to the weights and the bias are given under a vectorized form by:

$$\frac{\partial J}{\partial w} = \frac{1}{m}(X^T(\hat{Y} - Y)), \quad \frac{\partial J}{\partial b} = \frac{1}{m} Y^T \hat{Y}$$

Where,

$$X = \begin{pmatrix} \cdots Image_1 \cdots \\ \cdots Image_2 \cdots \\ . \\ . \\ . \\ \cdots Image_{m-1} \cdots \\ \cdots Image_m \cdots \end{pmatrix}_{(m\times n)} \quad , \quad Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ . \\ . \\ . \\ y^{(m-1)} \\ y^{(m)} \end{pmatrix}_{(m\times 1)} \quad , \quad \hat{Y} = \begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ . \\ . \\ . \\ \hat{y}^{(m-1)} \\ \hat{y}^{(m)} \end{pmatrix}_{(m\times 1)}$$

In order to classify 10 digits (i.e., from 0 to 9), using logistic regression. This strategy is referred to as the One-vs-all classification method and requires that we build 10 models, one for each digit. The model with the highest probability is used to categorize the provided image once each model's likelihood of a particular image has been determined. The steps of applying gradient descent to minimize the cost function are shown in the following diagram.



1.2 Download the MNIST dataset

[ ]:
```
# @title
!pip -qq install --upgrade --no-cache-dir gdown
!pip -qq  install imageio
```

```
# https://drive.google.com/file/d/1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD/view?
↪usp=sharing
!gdown 1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD
!unzip -qq mnist-original.mat.zip
```

```
Downloading...
From: https://drive.google.com/uc?id=1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD
To: /content/mnist-original.mat.zip
100% 11.4M/11.4M [00:00<00:00, 18.8MB/s]
```

1.3 Importing and preprocessing data

[ ]:
```
# @title
# Importing necessary libraries
import sys
import cv2
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from scipy.io import loadmat
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
%matplotlib inline
```

[ ]:
```
# @title
# Loading MNIST data
mnist = loadmat("mnist-original.mat")
mnist_data = mnist["data"].T
mnist_label = mnist["label"][0]
```

[ ]:
```
# @title
mnist_data.shape
```

[ ]: (70000, 784)

[ ]:
```
# @title
image_size_px = int(np.sqrt(mnist_data.shape[1]))
print("\u2705 [Info] The images size is (", image_size_px, "x", image_size_px,␣
↪")")
```

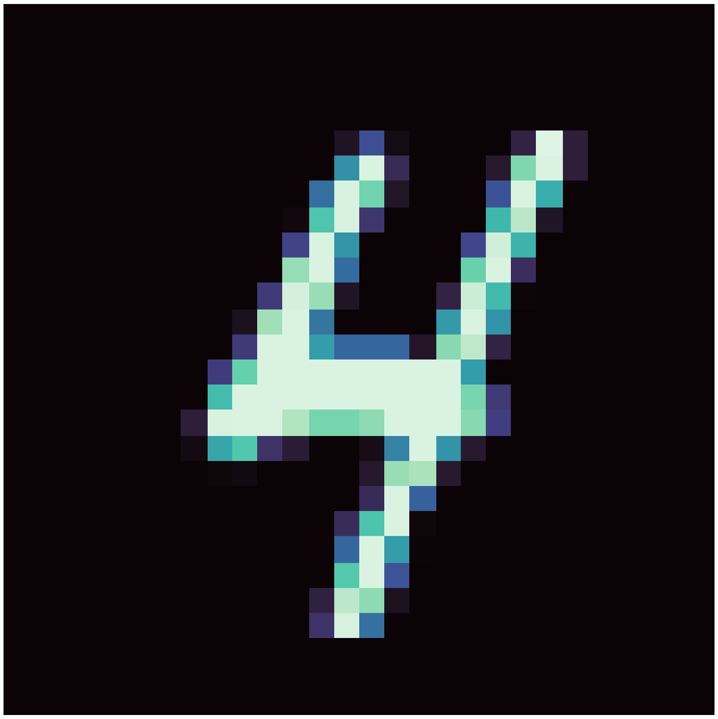
```
 [Info] The images size is ( 28 x 28 )
```

[ ]:
```
# @title
# Viewing a random MNIST image
def mnist_random_example():
```

```python
    idx = np.random.randint(70000)
    exp = mnist_data[idx].reshape(image_size_px, image_size_px)
    print("\u2705 [Info] The number in the image below is:", mnist_label[idx])
    plt.axis('off')
    plt.imshow(exp, cmap='mako')
```

[ ]:
```python
# @title
mnist_random_example()
```

```
[Info] The number in the image below is: 4.0
```



[ ]:
```python
# @title
# creating a normalization function
def normalize(data):
    mean = np.mean(data, axis=1, keepdims=True)
    std = np.std(data, axis=1, keepdims=True)
    data_normalized = (data - mean) / std
    return data_normalized
```

[ ]:
```python
# @title
# Normalizing the data
mnist_data_normalized = normalize(mnist_data)
```

[ ]:
```
# @title
# Splitting the data into Train and Test datasets
X_train, X_test, Y_train, Y_test = train_test_split(
    mnist_data_normalized, mnist_label, test_size=0.20, random_state=42
)

Y_train = Y_train.reshape(Y_train.shape[0], 1)
Y_test = Y_test.reshape(Y_test.shape[0], 1)

print("\u2705 [Info] The shape of the training set feature matrix is:", X_train.
↪shape)
print("\u2705 [Info] The shape of the training label vector is:", Y_train.shape)
print("\u2705 [Info] The shape of the test set feature matrix is:", X_test.shape)
print("\u2705 [Infp] The shape of the test label vector is:", Y_test.shape)
```

```
[Info] The shape of the training set feature matrix is: (56000, 784)
[Info] The shape of the training label vector is: (56000, 1)
[Info] The shape of the test set feature matrix is: (14000, 784)
[Infp] The shape of the test label vector is: (14000, 1)
```

[ ]:
```
# @title
# Creating new training and testing label vectors for each digit for the␣
↪one-vs-all method
Y_train_list = [(Y_train == i).astype(int) for i in range(10)]
Y_test_list = [(Y_test == i).astype(int) for i in range(10)]

# Unpack the lists to separate variables
(
    Y_train_0,
    Y_train_1,
    Y_train_2,
    Y_train_3,
    Y_train_4,
    Y_train_5,
    Y_train_6,
    Y_train_7,
    Y_train_8,
    Y_train_9,
) = Y_train_list

(
    Y_test_0,
    Y_test_1,
    Y_test_2,
    Y_test_3,
    Y_test_4,
    Y_test_5,
```

```
    Y_test_6,
    Y_test_7,
    Y_test_8,
    Y_test_9,
) = Y_test_list
```

1.4 Creating model functions

[ ]:
```
# @title
# Creating initilizer function to initialize weights and bias
def initializer(nbr_features):
    W = np.zeros((nbr_features, 1))
    B = 0
    return W, B
```

[ ]:
```
# @title
# Creating a Sigmoid function

def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s
```

[ ]:
```
# @title
# Creating the Forward and backward propagation function which calculates J, dW,␣
↪and dB

def ForwardBackProp(X, Y, W, B):
    m = X.shape[0]
    n_features = X.shape[1]

    Z = np.dot(X, W) + B
    Yhat = sigmoid(Z)

    loss = -np.sum(Y * np.log(Yhat) + (1 - Y) * np.log(1 - Yhat)) / m

    dW = np.dot(X.T, (Yhat - Y)) / m
    dB = np.sum(Yhat - Y) / m

    return loss, dW, dB
```

[ ]:
```
# @title
# Creating a prediction function which predicts the labels of the input images

def predict(X, W, B):
    Yhat_prob = sigmoid(np.dot(X, W) + B)
    Yhat = np.round(Yhat_prob).astype(int)
    return Yhat, Yhat_prob
```

```
[ ]: # @title
     # Creating the gradient descent optimizer function
     def gradient_descent(X, Y, W, B, alpha, max_iter):
         i = 0
         RMSE_threshold = 10e-6
         cost_history = []

         # Setup progress bar
         toolbar_width = 20
         print("[Info ]Training Progress: \u2705")

         while i < max_iter:
             J, dW, dB = ForwardBackProp(X, Y, W, B)
             W -= alpha * dW
             B -= alpha * dB
             cost_history.append(J)

             Yhat, _ = predict(X, W, B)
             RMSE = np.sqrt(np.mean((Yhat - Y) ** 2))

             i += 1
             if i % 50 == 0:
                 sys.stdout.write("=")
                 sys.stdout.flush()

             if RMSE <= RMSE_threshold:
                 break

         sys.stdout.write("]\n")  # End of progress bar
         return cost_history, W, B, i
```

```
[ ]: # @title
     # Creating the model function which trains a model and return its parameters.
     def LogRegModel(X_train, X_test, Y_train, Y_test, alpha, max_iter):
         # Initialize model parameters
         nbr_features = X_train.shape[1]
         W, B = initializer(nbr_features)

         # Train the model
         cost_history, W, B, i = gradient_descent(X_train, Y_train, W, B, alpha,␣
      ↪max_iter)

         # Make predictions
         Yhat_train, _ = predict(X_train, W, B)
         Yhat_test, _ = predict(X_test, W, B)

         # Calculate accuracy and confusion matrix
```

```
    train_accuracy = accuracy_score(Y_train, Yhat_train)
    test_accuracy = accuracy_score(Y_test, Yhat_test)
    conf_matrix = confusion_matrix(Y_test, Yhat_test, normalize="true")

    # Create model dictionary
    model = {
        "weights": W,
        "bias": B,
        "train_accuracy": train_accuracy,
        "test_accuracy": test_accuracy,
        "confusion_matrix": conf_matrix,
        "cost_history": cost_history,
    }

    return model
```

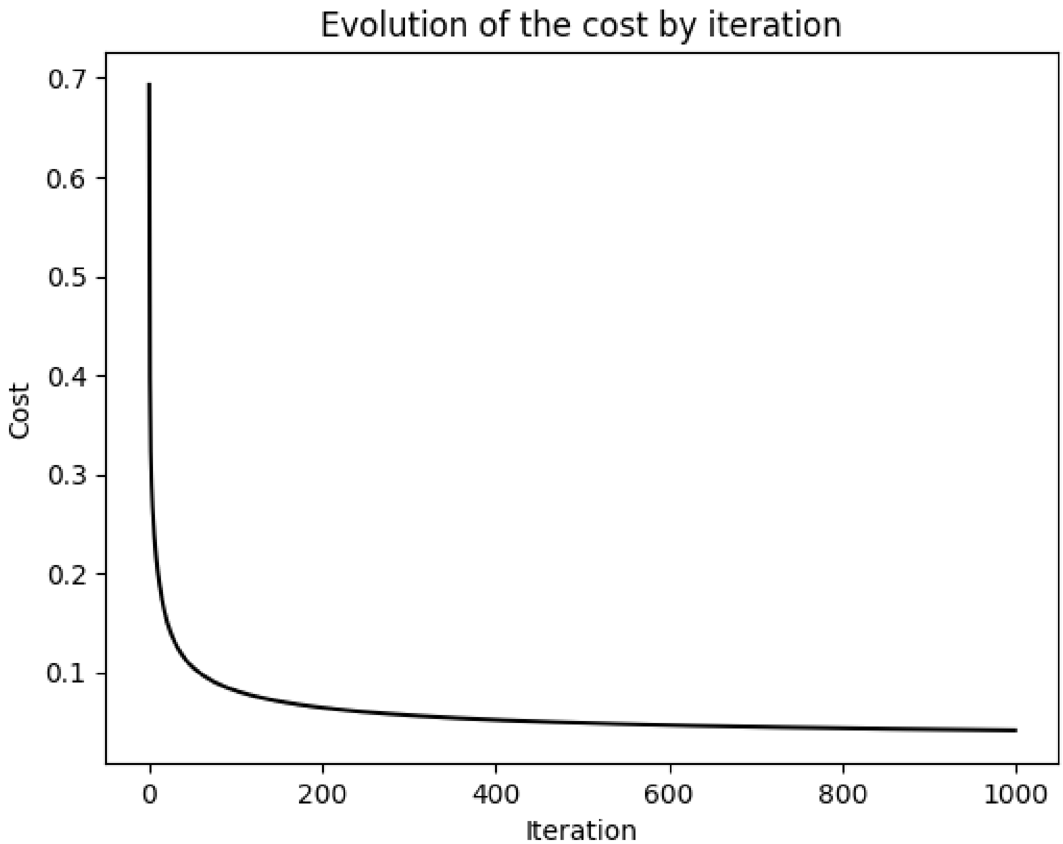
1.5 Training a test model for the digit "0"

[ ]:
```
# @title
# Testing the model function by training a classifier for the digit '0'

print("[Info] Progress bar: 1 step each 50 iteration \u2705")
model_0 = LogRegModel(X_train, X_test, Y_train_0, Y_test_0, alpha=0.01,␣
 ↪max_iter=1000)
print("[Info] Training completed! \u2705")
```

```
[Info] Progress bar: 1 step each 50 iteration
[Info ]Training Progress:
====================]
[Info] Training completed!
```

[ ]:
```
# @title
# Viewing the cost evolution over time of the trained model
cost = np.array(model_0["cost_history"]).ravel().tolist()
plt.plot(list(range(len(cost))), cost, color='k')
plt.title("Evolution of the cost by iteration")
plt.xlabel("Iteration")
plt.ylabel("Cost")
```

[ ]: `Text(0, 0.5, 'Cost')`




```
[ ]: # @title
     # Checking the accuracy of the model

     print("\u2705 [Info] The training accuracy of the model",␣
      ↪model_0["train_accuracy"])
     print("\u2705 [Info] The test accuracy of the model", model_0["test_accuracy"])
```

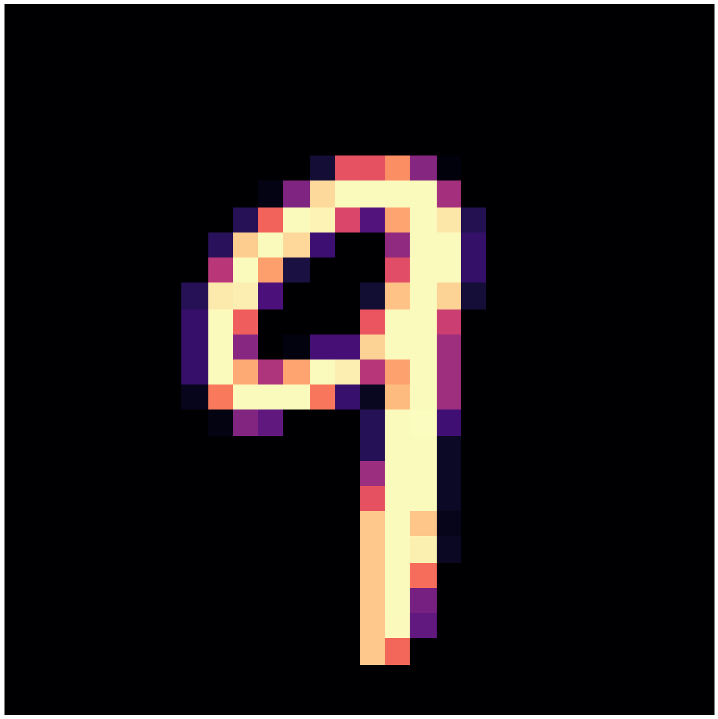
```
[Info] The training accuracy of the model 0.9882321428571429
[Info] The test accuracy of the model 0.9875
```

```
[ ]: # @title
     # Creating  afunction that shows a random image with the true and predicted label


     def check_random_pred(datum, Y, model, label):
         weights = model["weights"]
         bias = model["bias"]

         Yhat, _ = predict(datum, weights, bias)
```

```python
    if Yhat == 1:
        pred_label = label
    else:
        pred_label = "Not " + label

    if Y == 1:
        true_label = label
    else:
        true_label = "Not " + label

    print(
        "[Info] The number in the image below is:",
        true_label,
        " and predicted as:",
        pred_label,
    )

    image = datum.reshape(image_size_px, image_size_px)
    plt.axis('off')
    plt.imshow(image, cmap="magma")
```

[ ]:
```python
# @title
# Checking some random image predictions

idx = np.random.randint(X_test.shape[0])
datum = X_test[idx]
Y = Y_test_0[idx]
check_random_pred(datum, Y, model_0, "0")
```

```
[Info] The number in the image below is: Not 0  and predicted as: Not 0
```



1.6 Training a model for each digit

[ ]:
```
# @title
# Creating and training a model for each digit
models = {}
models_name_list = [
    "model_0",
    "model_1",
    "model_2",
    "model_3",
    "model_4",
    "model_5",
    "model_6",
    "model_7",
    "model_8",
    "model_9",
]

Y_train_list = [
    Y_train_0,
    Y_train_1,
    Y_train_2,
    Y_train_3,
```

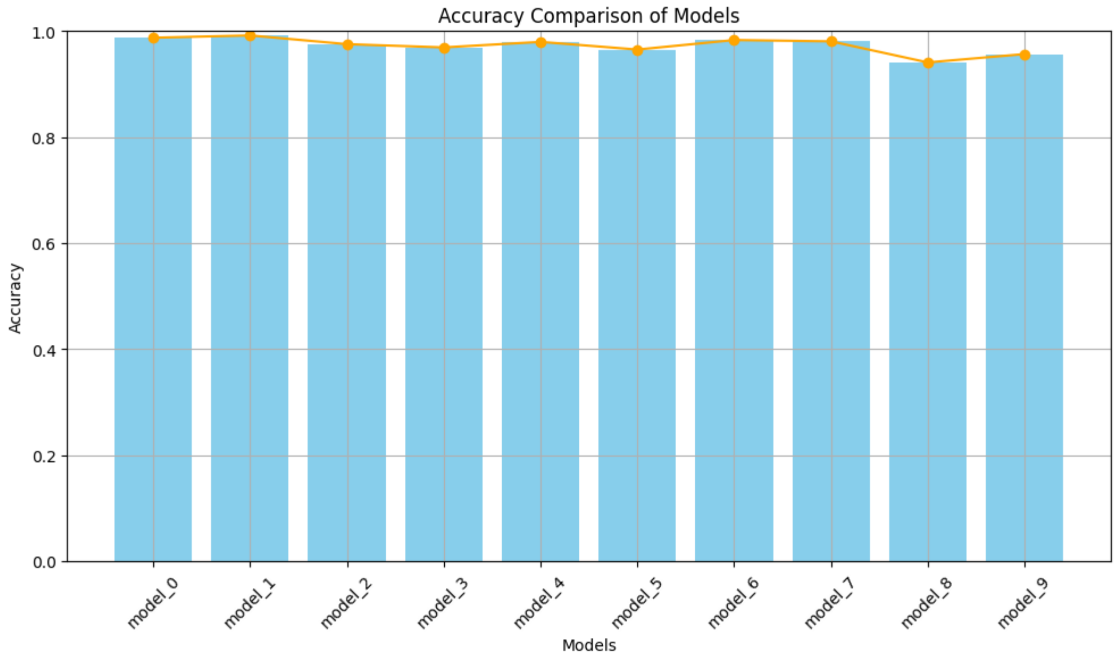
```python
    Y_train_4,
    Y_train_5,
    Y_train_6,
    Y_train_7,
    Y_train_8,
    Y_train_9,
]

Y_test_list = [
    Y_test_0,
    Y_test_1,
    Y_test_2,
    Y_test_3,
    Y_test_4,
    Y_test_5,
    Y_test_6,
    Y_test_7,
    Y_test_8,
    Y_test_9,
]

print("[Info] Training of a classifier for each digit:")
for i in range(10):
    print(f"[Info] Training model: {models_name_list[i]}, to recognize digit: 
 ↪{i} \u2705")
    print("[Info] Training progress bar: 1 step each 50 iterations \u2705")

    model = LogRegModel(
        X_train, X_test, Y_train_list[i], Y_test_list[i], alpha=0.01, 
 ↪max_iter=1000
    )
    print("[Info] Training completed! \u2705")
    print(f'[Info] Accuracy: {model["test_accuracy"]}')
    print("\u2796" * 60)

    models[models_name_list[i]] = model

print("All models trained successfully! \u2705")
```

```
[Info] Training of a classifier for each digit:
[Info] Training model: model_0, to recognize digit: 0
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9875
```

```
[Info] Training model: model_1, to recognize digit: 1
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9917857142857143

[Info] Training model: model_2, to recognize digit: 2
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9753571428571428

[Info] Training model: model_3, to recognize digit: 3
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9692142857142857

[Info] Training model: model_4, to recognize digit: 4
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9796428571428571

[Info] Training model: model_5, to recognize digit: 5
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9651428571428572

[Info] Training model: model_6, to recognize digit: 6
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9831428571428571

[Info] Training model: model_7, to recognize digit: 7
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9806428571428571
```

```
[Info] Training model: model_8, to recognize digit: 8
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9408571428571428

[Info] Training model: model_9, to recognize digit: 9
[Info] Training progress bar: 1 step each 50 iterations
[Info ]Training Progress:
====================]
[Info] Training completed!
[Info] Accuracy: 0.9566428571428571

All models trained successfully!
```

[ ]:
```
# @title
# Collect accuracy values
accuracy_values = [models[model_name]["test_accuracy"] for model_name in␣
 ↪models_name_list]

# Plotting the vertical bar plot
plt.figure(figsize=(10, 6))
plt.bar(models_name_list, accuracy_values, color='skyblue')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy of Models')
plt.ylim(0, 1)  # Set y-axis limits between 0 and 1
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()

# Plotting the line plot with markers
# plt.figure(figsize=(10, 6))
plt.plot(models_name_list, accuracy_values, marker='o', color='orange')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Models')
plt.ylim(0, 1)  # Set y-axis limits between 0 and 1
plt.xticks(rotation=45)
plt.tight_layout()
plt.grid(True)
plt.show()
```




```
[ ]: # @title
     # Calculating the average accuracy of all the models

     accuracy_list = []
     for model_name in models:
         accuracy_list.append(models[model_name]["test_accuracy"])

     ove_vs_all_accuracy = np.mean(accuracy_list)
     print("\u2705 [Info] The accuracy of the One-Vs-All model is:",␣
      ↪ove_vs_all_accuracy)
```

```
[Info] The accuracy of the One-Vs-All model is: 0.9729928571428571
```

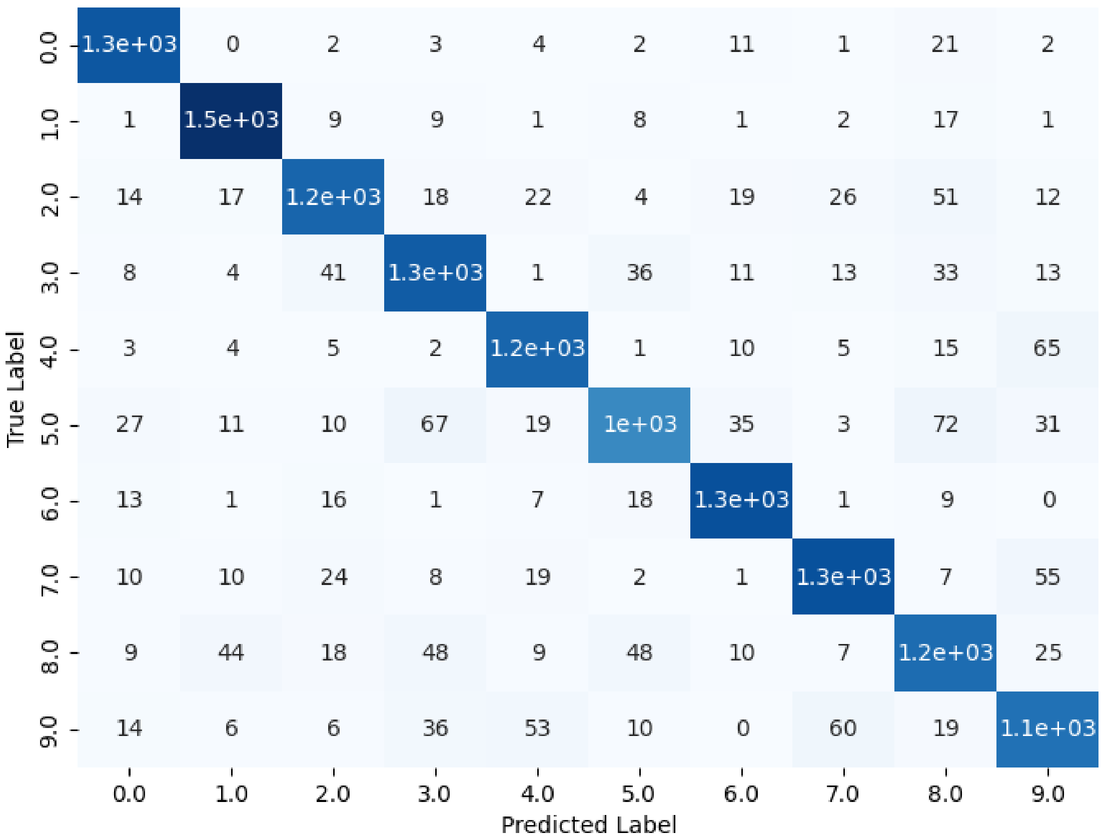
1.7 Final model for digit classification

```
[ ]: # @title
     # Creating a one-vs-all function that uses all the trained models to predict the␣
      ↪label of a random image

     def one_vs_all(data, models_dict):
         num_models = len(models_dict)
         pred_matrix = np.zeros((data.shape[0], num_models))

         for i, model_name in enumerate(models_dict):
             W = models_dict[model_name]["weights"]
             B = models_dict[model_name]["bias"]
```

```
        Yhat, Yhat_prob = predict(data, W, B)
        pred_matrix[:, i] = Yhat_prob.T

    max_prob_indices = np.argmax(pred_matrix, axis=1)
    labels = [max_prob_indices[i] for i in range(max_prob_indices.shape[0])]

    return labels
```

[ ]:
```
# @title
def plot_confusion_matrix(conf_matrix, y_true, ax, case):
    if case == 0:
        class_labels = np.unique(y_true)
        df_cm = pd.DataFrame(conf_matrix, columns=class_labels,␣
↪index=class_labels)
        df_cm.index.name = "True Label"
        df_cm.columns.name = "Predicted Label"
        sb.heatmap(
            df_cm, cmap="Blues", cbar=False, annot=True, annot_kws={"size": 10},␣
↪ax=ax
        )
    else:
        label_mapping = ["Goalkeeper", "Defender", "Midfielder", "Forward"]
        df_cm = pd.DataFrame(conf_matrix, columns=label_mapping,␣
↪index=label_mapping)
        df_cm.index.name = "True Label"
        df_cm.columns.name = "Predicted Label"
        sb.heatmap(
            df_cm, cmap="Blues", cbar=False, annot=True, annot_kws={"size": 10},␣
↪ax=ax
        )

    ax.set_yticklabels(ax.get_yticklabels(), fontsize=10)
    ax.set_xticklabels(ax.get_xticklabels(), fontsize=10)
```

[ ]:
```
# @title
pred_label = one_vs_all(X_test, models)
conf_matrix = confusion_matrix(Y_test, pred_label)

# Plot confusion matrix
fig, ax = plt.subplots(figsize=(8, 6))
plot_confusion_matrix(conf_matrix, Y_test, ax, case=0)  # Modify case as needed
plt.show()
```

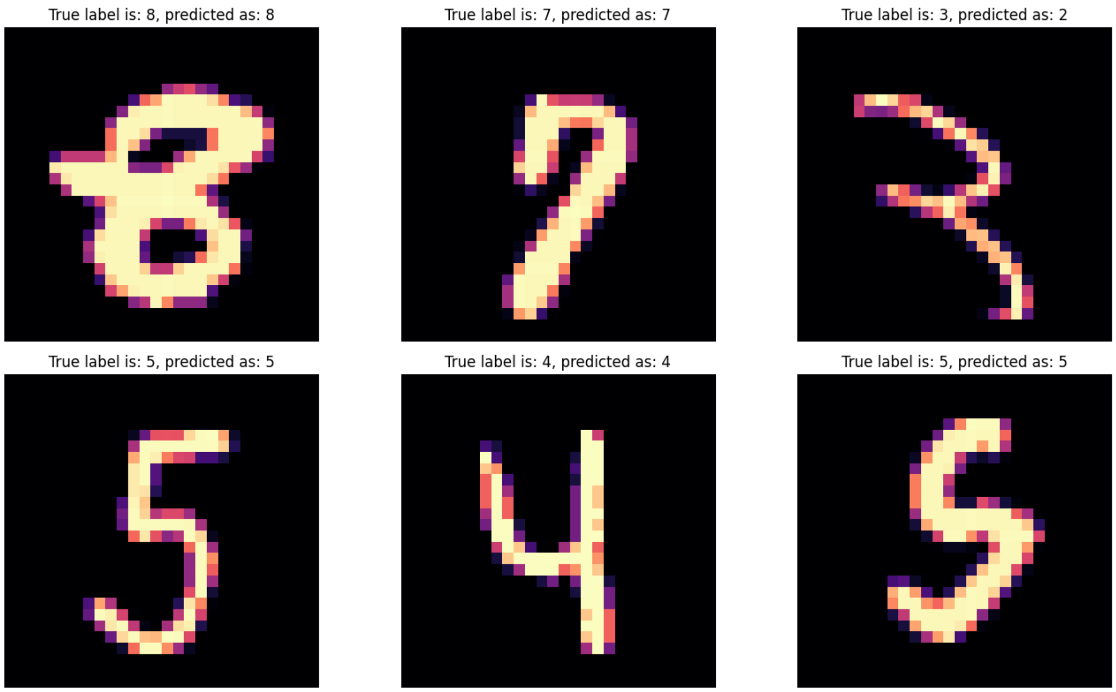


[ ]:

```
# @title
def plot_example_results(example_data, true_labels, predicted_labels,␣
↪image_size):
    examples_number = example_data.shape[0]
    plt.figure(figsize=(14, 8))

    for i in range(examples_number):
        image = example_data[i].reshape(image_size, image_size)
        plt.subplot(2, 3, i + 1)
        plt.axis('off')
        plt.imshow(image, cmap="magma")
        title = f"True label is: {true_labels[i]}, predicted as:␣
↪{predicted_labels[i]}"
        plt.title(title)

    plt.tight_layout()
    plt.show()
```

1.8 Results

[ ]:
```
# @title
examples_number = 6
index_random_sample = np.random.randint(70000, size=(1, examples_number))
example = mnist_data_normalized[index_random_sample].reshape(examples_number,␣
↪784)
true_labels = mnist_label[index_random_sample].flatten().astype(int)
predicted_labels = one_vs_all(example, models)

# Plotting example results
plot_example_results(example, true_labels, predicted_labels, image_size_px)
```



[ ]:
```
# @title
def one_vs_all_score(data, models_dict):
    num_models = len(models_dict)
    pred_matrix = np.zeros((data.shape[0], num_models))
    label_probabilities = np.zeros((data.shape[0], 10))

    for i, model_name in enumerate(models_dict):
        W = models_dict[model_name]["weights"]
        B = models_dict[model_name]["bias"]
        Yhat, Yhat_prob = predict(data, W, B)
        pred_matrix[:, i] = Yhat_prob.T
```

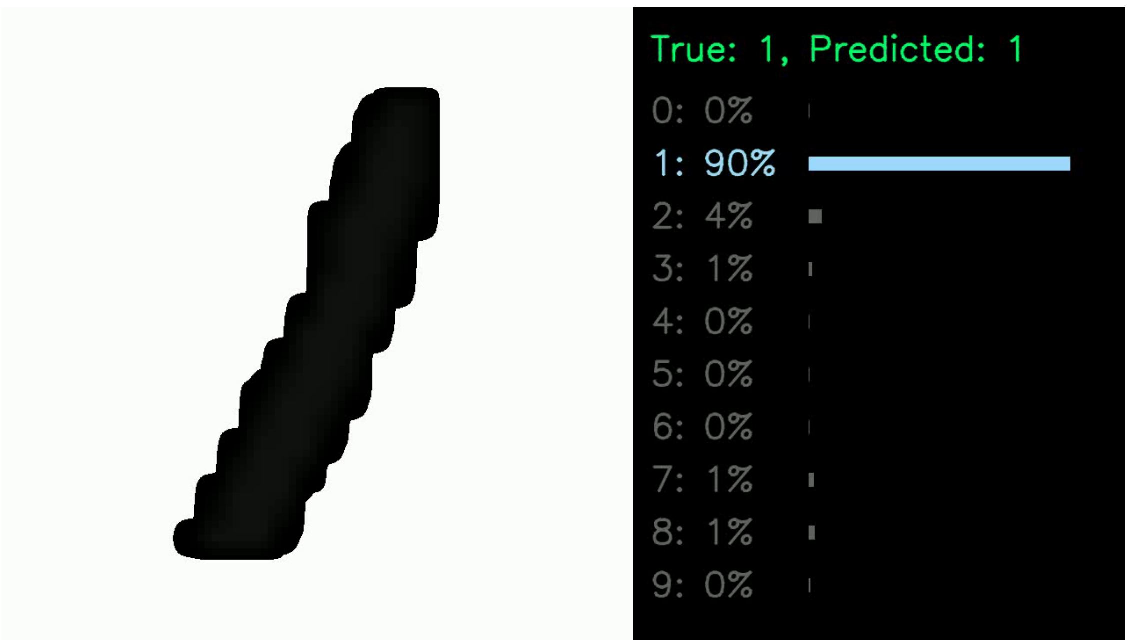
```
    max_prob_indices = np.argmax(pred_matrix, axis=1)

    for i in range(data.shape[0]):
        label_probabilities[i] = pred_matrix[i] / np.sum(pred_matrix[i])

    labels = [max_prob_indices[i] for i in range(max_prob_indices.shape[0])]

    return labels, label_probabilities
```

[ ]:
```
# @title
def render_and_save_examples(example_data, true_labels, predicted_labels,␣
↪predicted_score, image_size):
    examples_number = example_data.shape[0]
    video_output_path = 'output_video.mp4'
    codec = cv2.VideoWriter_fourcc(*'mp4v')
    vid_width_height = 1280, 720
    vw = cv2.VideoWriter(video_output_path, codec, 30, vid_width_height)

    font_face = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1.3
    thickness = 2

    for i in range(examples_number):
        image = example_data[i].reshape(image_size, image_size)
        image_disp = cv2.resize(image*5, (720, 720))

        # Check if prediction is correct or not
        is_correct = true_labels[i] == predicted_labels[i]

        title = f"True: {true_labels[i]}, Predicted: {predicted_labels[i]}"

        preds = predicted_score[i]*100  # Updated to use specific example's␣
↪probabilities

        img = np.zeros((720, 1280, 3), dtype=np.uint8)
        img[:720, :720, 0] = image_disp
        img[:720, :720, 1] = image_disp
        img[:720, :720, 2] = image_disp

        x, y = 740, 60
        txt_color = (100, 255, 0) if is_correct else (0, 0, 255)
        cv2.putText(img, text=title, org=(x, y), fontScale=font_scale,␣
↪fontFace=font_face,
                        thickness=thickness, color=txt_color, lineType=cv2.LINE_AA)

        bar_x, bar_y = 740, 130
        for j, p in enumerate(preds):
```

```
            if j < 10:
                rect_width = int(p * 3.3)
                rect_start = 180
                color = (255, 218, 158) if j == predicted_labels[i] else (100, 100,␣
↪100)
                cv2.rectangle(img, (bar_x + rect_start, bar_y - 5), (bar_x +␣
↪rect_start + rect_width, bar_y - 20),
                              color, -1)
                text = f'{j}: {int(p)}%'
                cv2.putText(img, text=text, org=(bar_x, bar_y),␣
↪fontScale=font_scale, fontFace=font_face,
                            thickness=thickness, color=color, lineType=cv2.LINE_AA)
                bar_y += 60

        vw.write(img)

    vw.release()


examples_number = 60
index_random_sample = np.random.randint(70000, size=(1, examples_number))
example = mnist_data_normalized[index_random_sample].reshape(examples_number,␣
↪784)
true_labels = mnist_label[index_random_sample].flatten().astype(int)
predicted_labels, predicted_score = one_vs_all_score(example, models)

# Render and save examples to video
render_and_save_examples(example, true_labels, predicted_labels,␣
↪predicted_score, image_size_px)
```

[ ]:
```
# @title
from IPython.display import HTML
from base64 import b64encode

video_path = 'output_video.mp4'

def show_video(video_path, video_width = 600):

  video_file = open(video_path, "r+b").read()

  video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
  return HTML(f"""<video width={video_width} controls><source␣
↪src="{video_url}"></video>""")

show_video(video_path)
```

[ ]:
```python
# @title
from IPython.display import HTML
from base64 import b64encode
import imageio

video_path = 'output_video.mp4'
gif_path = 'output_video.gif'

def convert_to_gif(video_path, gif_path):
    video = imageio.get_reader(video_path)
    frames = [frame for frame in video]
    imageio.mimsave(gif_path, frames, format='GIF', duration=0.1)
    return gif_path

gif_path = convert_to_gif(video_path, gif_path)

def show_gif(gif_path, gif_width=600):
    gif_file = open(gif_path, "rb").read()
    gif_url = f"data:image/gif;base64,{b64encode(gif_file).decode()}"
    return HTML(f"""<img src="{gif_url}" width="{gif_width}"/>""")

show_gif(gif_path)
```



1.9 Conclusion

In this mini-project, we successfully developed a handwritten digit classifier using logistic regression. The goal was to train a separate model for each digit from 0 to 9, enabling accurate recognition of individual digits. The training process involved multiple steps, and the results achieved were impressive.

Here's a summary of the achievements of this project:

- For each digit, from 0 to 9, a dedicated model was trained and tested.
- The training process was successful for all models, achieving impressive accuracy rates.
- Model accuracies varied for different digits, ranging from approximately 94% to 99%.
- The training progress was visually represented with progress bars, adding clarity and insight into the process.
- Accuracy scores were displayed after training each model, providing a clear overview of their performance.

This project demonstrated the power of logistic regression in classifying handwritten digits. The achieved accuracies showcase the effectiveness of this approach in recognizing a wide range of digits. Through this mini-project, we gained practical experience in model training, testing, and accuracy evaluation, highlighting the potential of machine learning techniques in solving real-world challenges.