Practical no.7

**Aim:** Implement Batch Gradient Descent with early stopping for Softmax Regression.

# Theory

Batch gradient descent, we compute the gradient of the cost function with respect to all training examples and use the average gradient to update the parameters. In stochastic gradient descent, we randomly select a single training example, compute the gradient of the cost function with respect to that example, and use this gradient to update the parameters. Mini-batch gradient descent is a variation of stochastic gradient descent that updates the parameters using the average gradient over a small random subset of the training examples.

# Material

* sklearn
* numpy

# Program

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Load the iris dataset iris = load\_iris()

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target,test\_size=0.2, random\_state=42)

# Add bias term to the feature matrices

X\_train\_bias = np.c\_[np.ones((len(X\_train), 1)), X\_train] X\_test\_bias = np.c\_[np.ones((len(X\_test), 1)), X\_test]

# Define the softmax function

def softmax(logits): exp\_logits = np.exp(logits)

sum\_exp\_logits = np.sum(exp\_logits, axis=1, keepdims=True) return exp\_logits / sum\_exp\_logits

# Initialize the model parameters n\_inputs = X\_train\_bias.shape[1] n\_outputs = len(np.unique(y\_train))

theta = np.random.randn(n\_inputs, n\_outputs)

# Define the learning rate, number of epochs, and early stopping parameters eta = 0.1

n\_epochs = 1000

max\_epochs\_without\_improvement = 50 best\_loss = np.inf epochs\_without\_improvement = 0

# Train the model using batch gradient descent with early stopping for epoch in range(n\_epochs):

# Compute the logits and probabilities for the training set logits\_train = X\_train\_bias.dot(theta)

y\_proba\_train = softmax(logits\_train)

# Compute the loss and gradient for the training set

loss\_train = -np.mean(np.sum(np.log(y\_proba\_train) \* (y\_train.reshape(-1,1) == np.arange(n\_outputs)), axis=1))

error\_train = y\_proba\_train - (y\_train.reshape(-1, 1) == np.arange(n\_outputs)) grad = 1/len(X\_train\_bias) \* X\_train\_bias.T.dot(error\_train)

# Update the model parameters theta -= eta \* grad

# Compute the logits and probabilities for the testing set logits\_test = X\_test\_bias.dot(theta)

y\_proba\_test = softmax(logits\_test)

# Compute the loss for the testing set

loss\_test = -np.mean(np.sum(np.log(y\_proba\_test) \* (y\_test.reshape(-1, 1)== np.arange(n\_outputs)), axis=1))

# Check if the loss on the testing set has improved if loss\_test < best\_loss:

best\_loss = loss\_test epochs\_without\_improvement = 0

else:

epochs\_without\_improvement += 1

# Print the loss every 50 epochs if epoch % 50 == 0:

print("Epoch:", epoch, "Loss(train):", loss\_train, "Loss(test):",loss\_test) # Check if the early stopping criteria have been met

if epochs\_without\_improvement > max\_epochs\_without\_improvement: print("Early stopping!")

break

# Make predictions on the testing set using the trained model logits\_test = X\_test\_bias.dot(theta)

y\_proba\_test = softmax(logits\_test) y\_pred\_test = np.argmax(y\_proba\_test, axis=1)

# Compute the accuracy of the model on the testing set accuracy = accuracy\_score(y\_test, y\_pred\_test) print("Accuracy:", accuracy)

# Output:

