Department of Electrical Engineering

Faculty Member:	Ma'am Neelma Naz	**Date:**	October 30, 2025
Semester:	7^{th}	**Group:**	02

CS471 Machine Learning

Lab 8: Introduction to Sci-kit Learn

		PLO4	**PLO5**	**PLO5**	**PLO8**	**PLO9**
		CLO4	CLO5	CLO5	CLO6	CLO7
Student Name	**Reg. No**	**Viva / Quiz / Demo** 5 Marks	**Analysis of Data in Report** 5 Marks	**Modern Tool Usage** 5 Marks	**Ethics** 5 Marks	**Individual and Teamwork** 5 Marks
Hanzla Sajjad	403214					
Irfa Farooq	412564					

Introduction

This laboratory exercise will focus on the Scikit Learn (or SKLearn) library for machine learning implementations in python. Scikit Learn contains many useful functions for fitting models using various machine learning techniques such as linear regression, logistic regression, decision trees, support vector machines, k-means clustering, anomaly detection and more.

Objectives

The following are the main objectives of this lab:

- Extract and prepare the training and test datasets
- Implement linear regression using Scikit learn
- Implement logistic regression using Scikit learn
- Implement k-means clustering using Scikit learn

Lab Conduct

- Respect faculty and peers through speech and actions
- The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
- In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code/screenshot/plot between the #START and #END parts of these commented lines. Do NOT remove the commented lines.
- Use the tab key to provide the indentation in python.
- When you provide the code in the report, keep the font size at 12

Theory

Scikit Learn is a python library that contains a wide arsenal of functions pertaining to machine learning. It also contains its own datasets for trying out the machine learning algorithms. Scikit learns API interface can be divided into three types: estimator, predictor and transformer. The estimators are used to fit the model in accordance with some algorithm. The predictors use the fitted model to make prediction on test features. The transformers are used for the conversion of data.

A brief summary of the relevant keywords and functions in python is provided below:

print()	output text on console
input()	get input from user on console
range()	create a sequence of numbers
len()	gives the number of characters in a string
if	contains code that executes depending on a logical condition
else	connects with **if** and **elif**, executes when conditions are not met
elif	equivalent to **else if**
while	loops code as long as a condition is true
for	loops code through a sequence of items in an iterable object
break	exit loop immediately
continue	jump to the next iteration of the loop
def	used to define a function

pd.read_csv	import csv file as a dataframe
df.to_csv	export dataframe as a csv file

Lab Task 1 – Linear Regression

Download a dataset containing at least 5 feature columns and a label column containing *continuous* data. Use functions from Sci-kit learn to train a model using linear regression. You will need to split your dataset into training and validation portions. Vary the step size and regularization parameters to get at least 6 plots of the training loss and test loss. Lastly, save the weights of the best trained model, print them and use them to make at least five predictions.

Provide the codes and all of the relevant screenshots of your work. Also, provide a brief explanation of the functions you are using in your codes.

Code

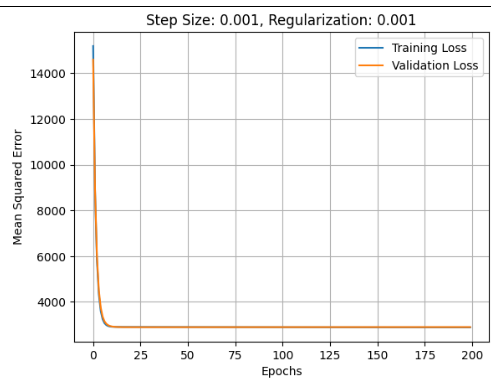
```
# Task 1
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error

# Load dataset
data = load_diabetes()
X, y = data.data, data.target

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

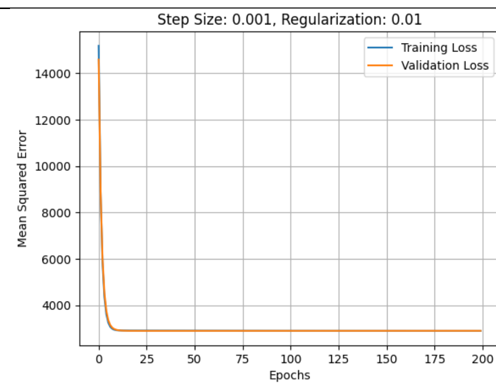
```
# Step sizes (learning rates) and regularization strengths
step_sizes = [0.001, 0.01, 0.1]
regularization_params = [0.001, 0.01]

# Track best model
best_model = None
best_val_loss = float('inf')

# Train with multiple parameter combinations
for eta in step_sizes:
    for alpha in regularization_params:
        print(f"\nTraining with step size={eta}, regularization={alpha}")

        # Model setup
        model = SGDRegressor(
            learning_rate='constant',
            eta0=eta,
            alpha=alpha,
            max_iter=1,
            warm_start=True,
            random_state=42,
            tol=None
        )

        train_losses = []
        val_losses = []
        epochs = 200

        for epoch in range(epochs):
            model.partial_fit(X_train, y_train)

            y_train_pred = model.predict(X_train)
            y_val_pred = model.predict(X_test)

            train_mse = mean_squared_error(y_train, y_train_pred)
            val_mse = mean_squared_error(y_test, y_val_pred)
```

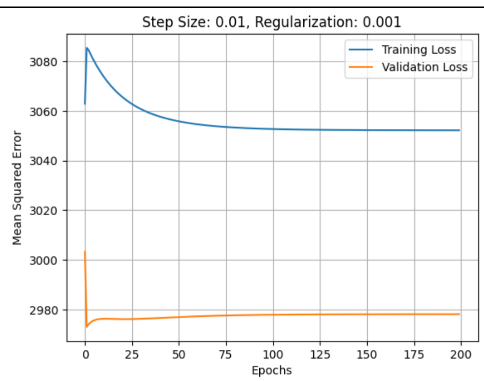
```
            train_losses.append(train_mse)
            val_losses.append(val_mse)

        # Plot training and validation loss
        plt.figure()
        plt.plot(train_losses, label="Training Loss")
        plt.plot(val_losses, label="Validation Loss")
        plt.xlabel("Epochs")
        plt.ylabel("Mean Squared Error")
        plt.title(f"Step Size: {eta}, Regularization: {alpha}")
        plt.legend()
        plt.grid(True)
        plt.show()

        # Save best model
        if val_losses[-1] < best_val_loss:
            best_val_loss = val_losses[-1]
            best_model = model

# Display best model info
print("\nBest Model Found:")
print("Step size (eta0):", best_model.eta0)
print("Regularization (alpha):", best_model.alpha)
print("Weights:", best_model.coef_)
print("Bias:", best_model.intercept_)

# Make 5 predictions with best model
X_new = X_test[:5]
y_true = y_test[:5]
y_pred = best_model.predict(X_new)

print("\nFive Predictions:")
for i in range(5):
    print(f"Input {i+1}: True value = {y_true[i]:.2f}, Predicted =
{y_pred[i]:.2f}")
```

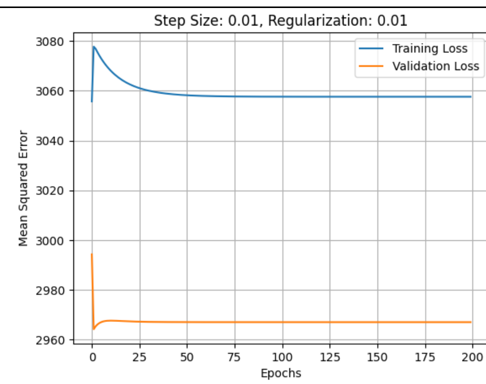
```python
# Plot true vs predicted values
plt.figure()
plt.scatter(range(len(y_true)), y_true, color='blue', label='True values')
# Added x-axis values and label
plt.scatter(range(len(y_pred)), y_pred, color='red', label='Predicted
values') # Added x-axis values and label
plt.title("True and Predicted Values (Best Model)")
plt.xlabel("Sample Index") # Added x-axis label
plt.ylabel("Diabetes Progression") # Added y-axis label
plt.legend() # Added legend
plt.grid(True)
plt.show()
```

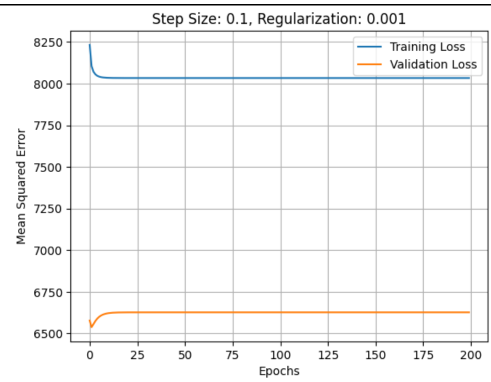
Output Console

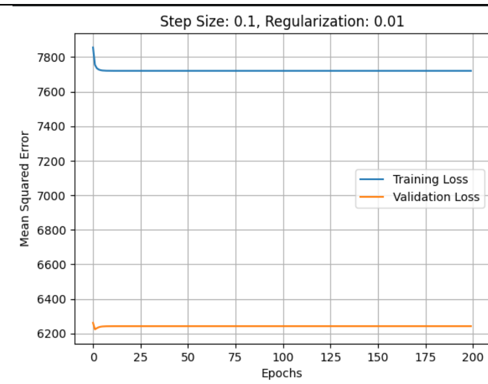




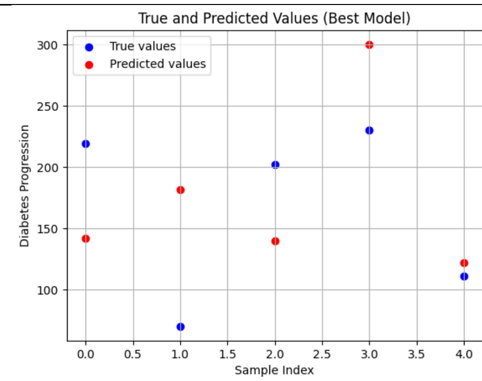









```
Best Model Found:
Step size (eta0): 0.001
Regularization (alpha): 0.01
Weights: [  2.11110165 -11.63291294  26.77724585  16.77159035 -17.16160781
   3.60967299  -4.5788064    9.82508708  25.24944374   3.33139821]
Bias: [154.1952434]

Five Predictions:
Input 1: True value = 219.00, Predicted = 142.13
Input 2: True value = 70.00, Predicted = 181.45
Input 3: True value = 202.00, Predicted = 139.67
Input 4: True value = 230.00, Predicted = 299.93
Input 5: True value = 111.00, Predicted = 122.23
```



Lab Task 2 – Logistic Regression

Download a dataset containing at least 5 feature columns and a label column containing *discrete* data. Use functions from Sci-kit learn to train a model using logistic regression. You will need to split your dataset into training and validation portions. Vary the step size and regularization parameters to get at least 6 models of the training. For each model, plot the training loss (vs. epochs), test loss (vs. epochs), precision (vs. epochs) and recall (vs. epochs). Additionally, plot the precision-recall plots for each trained model.

Lastly, save the weights of the best trained model, print them and use them to make at least five predictions. Make a scatter plot for each of your prediction. For this, you will need to show the all of the dataset examples with their labeled classes. Your prediction must be shown as a distinct point in the scatter plots.

Provide the code and all of the relevant screenshots of your work. Also, give brief explanation of the functions you are using in your codes.

Code

```
# Task 2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer  # Discrete labels (0 or 1)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import log_loss, precision_score, recall_score,
precision_recall_curve

# Load Dataset
data = load_breast_cancer()
X, y = data.data, data.target
```

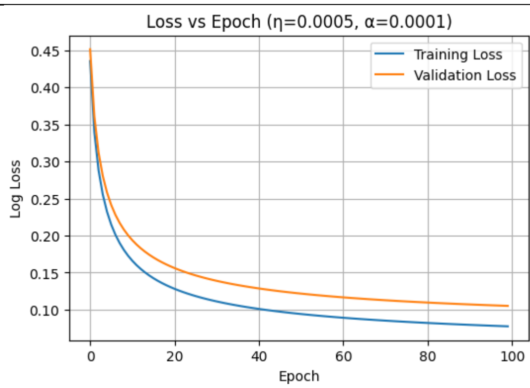
```python
# train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define Hyperparameters
step_sizes = [0.0005, 0.001, 0.005]
regularization_params = [0.0001, 0.001]

# Track best model
best_model = None
best_val_loss = float("inf")

# Train Multiple Models
for eta in step_sizes:
    for alpha in regularization_params:
        print(f"\nTraining with step size={eta}, regularization={alpha}")

        model = SGDClassifier(
            loss="log_loss",          # Logistic regression objective
            learning_rate="constant",
            eta0=eta,
            alpha=alpha,
            max_iter=1,
            warm_start=True,
            random_state=42,
            tol=None
        )

        epochs = 100
        train_losses = []
        val_losses = []
        precisions = []
```

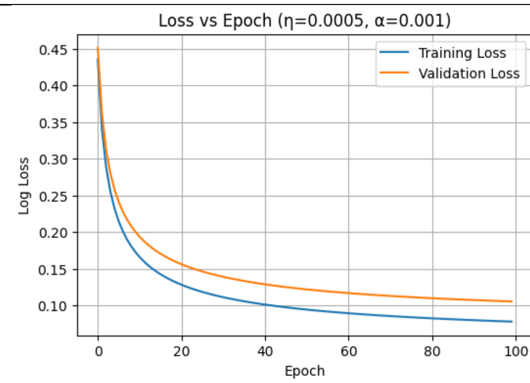
```
    recalls = []

    for epoch in range(epochs):
        model.partial_fit(X_train, y_train, classes=np.unique(y))

        # Predictions
        y_train_pred_prob = model.predict_proba(X_train)
        y_val_pred_prob = model.predict_proba(X_test)

        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_test)

        # Compute metrics
        train_loss = log_loss(y_train, y_train_pred_prob)
        val_loss = log_loss(y_test, y_val_pred_prob)

        prec = precision_score(y_test, y_val_pred)
        rec = recall_score(y_test, y_val_pred)

        train_losses.append(train_loss)
        val_losses.append(val_loss)
        precisions.append(prec)
        recalls.append(rec)

    # Plot Loss vs Epoch
    plt.figure(figsize=(6, 4))
    plt.plot(train_losses, label="Training Loss")
    plt.plot(val_losses, label="Validation Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Log Loss")
    plt.title(f"Loss vs Epoch (η={eta}, α={alpha})")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Plot Precision & Recall vs Epoch
    plt.figure(figsize=(6, 4))
    plt.plot(precisions, label="Precision")
```

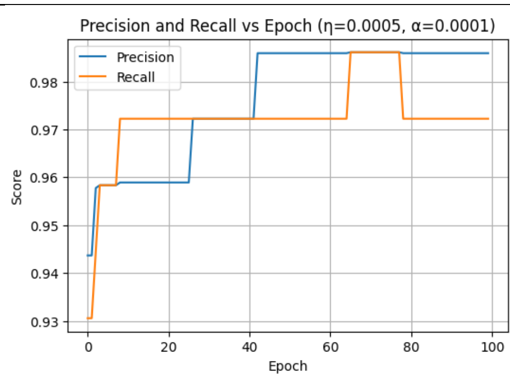
```
        plt.plot(recalls, label="Recall")
        plt.xlabel("Epoch")
        plt.ylabel("Score")
        plt.title(f"Precision and Recall vs Epoch (η={eta}, α={alpha})")
        plt.legend()
        plt.grid(True)
        plt.show()

        # Plot Precision-Recall Curve
        precision_vals, recall_vals, _ = precision_recall_curve(y_test,
y_val_pred_prob[:, 1])
        plt.figure(figsize=(6, 4))
        plt.plot(recall_vals, precision_vals)
        plt.xlabel("Recall")
        plt.ylabel("Precision")
        plt.title(f"Precision-Recall Curve (η={eta}, α={alpha})")
        plt.grid(True)
        plt.show()

        # Save Best Model
        if val_losses[-1] < best_val_loss:
            best_val_loss = val_losses[-1]
            best_model = model

# Best Model Summary
print("\nBest Model Found:")
print("Step size (η):", best_model.eta0)
print("Regularization (α):", best_model.alpha)
print("Weights:", best_model.coef_)
print("Bias:", best_model.intercept_)

# Step 7: Predictions (5 samples)
X_new = X_test[:5]
y_true = y_test[:5]
y_pred = best_model.predict(X_new)
y_prob = best_model.predict_proba(X_new)[:, 1]
```

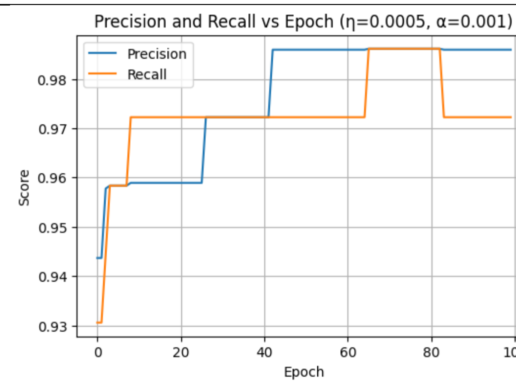
```
print("\nFive Predictions:")
for i in range(5):
    print(f"Input {i+1}: True={y_true[i]}, Predicted={y_pred[i]},
Probability={y_prob[i]:.3f}")

# Scatter Plot for Predictions
plt.figure(figsize=(7, 5))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap="coolwarm",
label="Training Data", alpha=0.6)
plt.scatter(X_new[:, 0], X_new[:, 1],
            c=y_pred, marker="X", s=150, edgecolor='black', linewidth=2,
label="Predictions")
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.title("Scatter Plot of Classes with Predictions Highlighted")
plt.legend()
plt.grid(True)
plt.show()
```

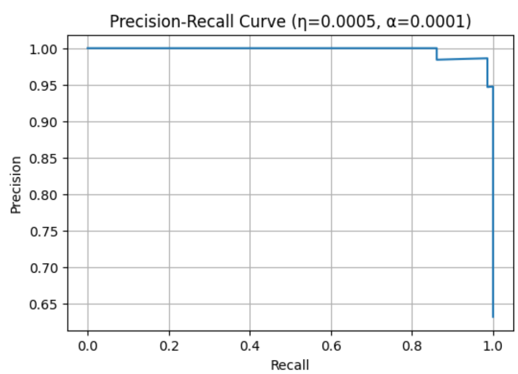
Output Console

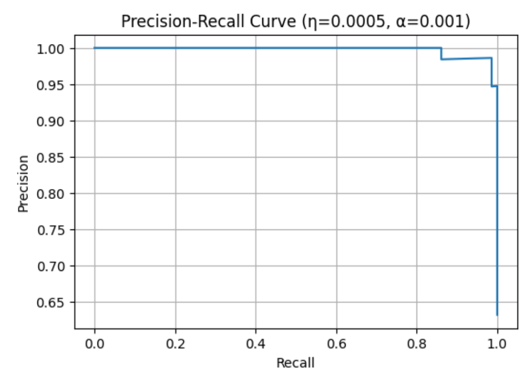


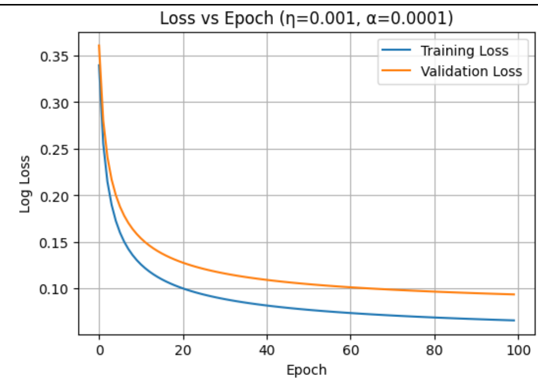


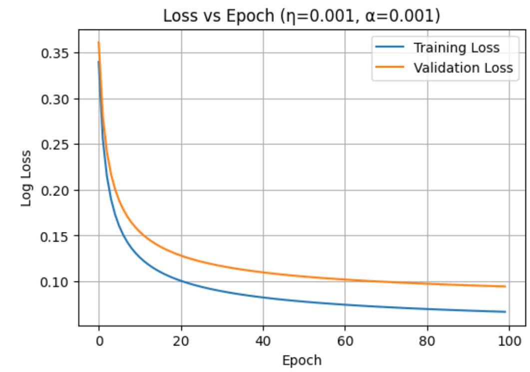


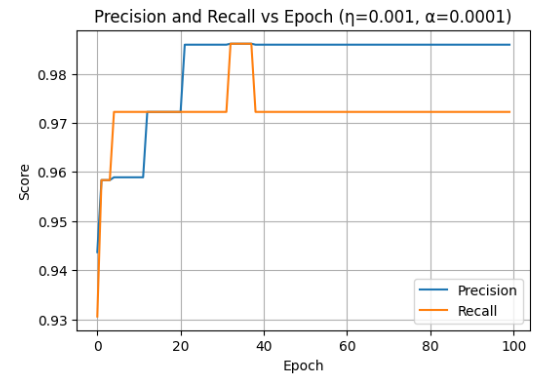


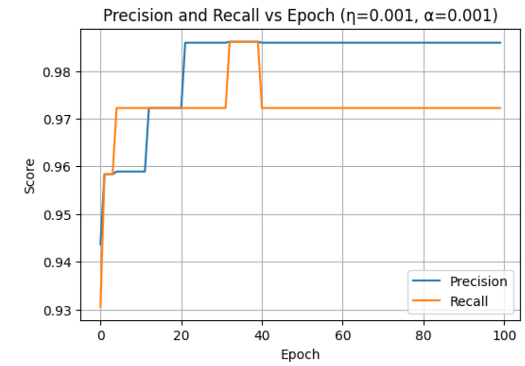


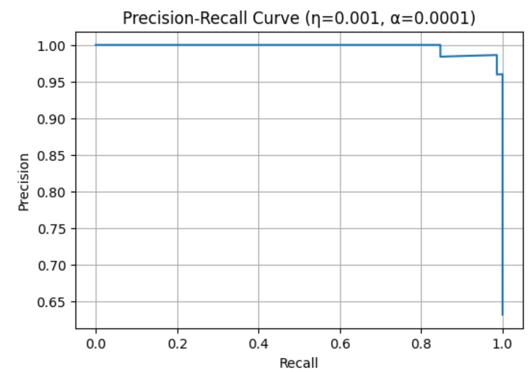


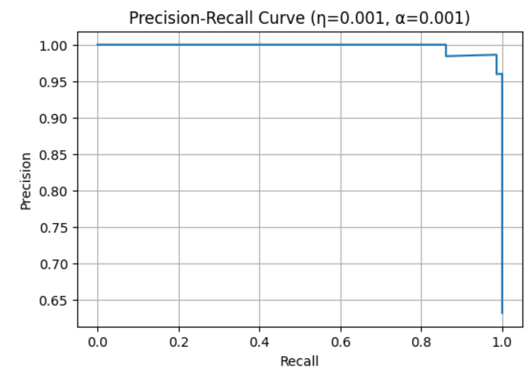

Loss vs Epoch (η=0.001, α=0.0001)
Training Loss
Validation Loss
Log Loss
Epoch

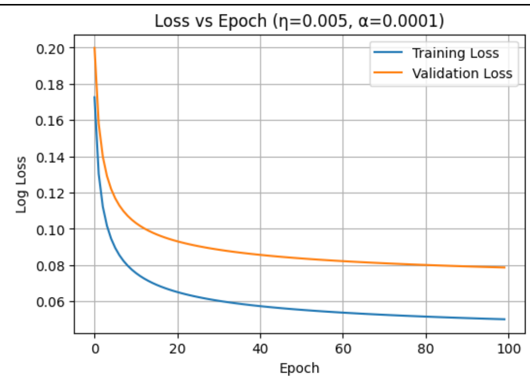

Loss vs Epoch (η=0.001, α=0.001)
Training Loss
Validation Loss
Log Loss
Epoch

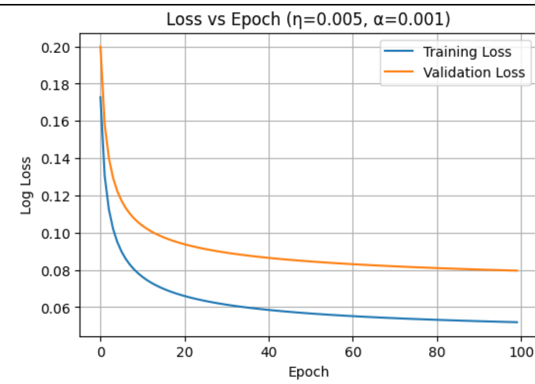

Precision and Recall vs Epoch (η=0.001, α=0.0001)
Precision
Recall
Score
Epoch

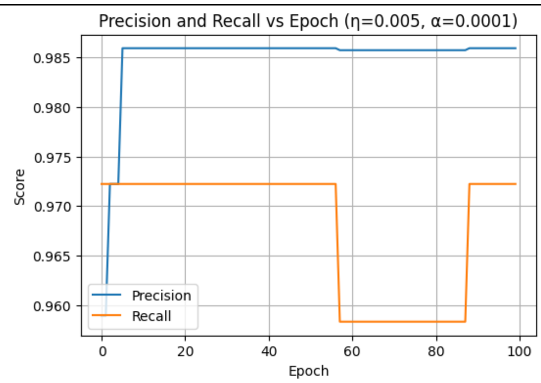

Precision and Recall vs Epoch (η=0.001, α=0.001)
Precision
Recall
Score
Epoch

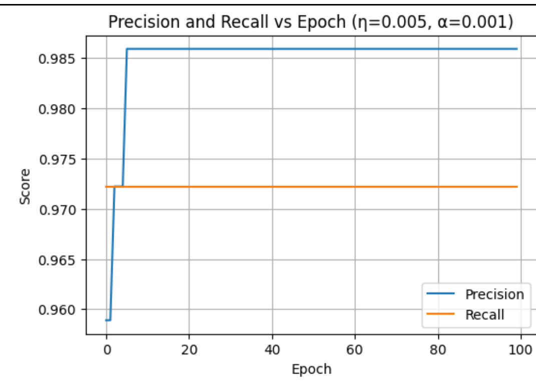

Precision-Recall Curve (η=0.001, α=0.0001)
Precision
Recall

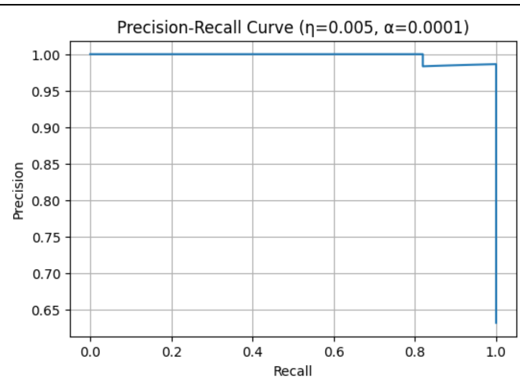

Precision-Recall Curve (η=0.001, α=0.001)
Precision
Recall

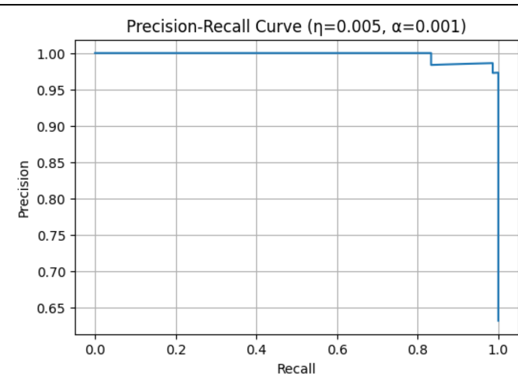

Loss vs Epoch (η=0.005, α=0.0001)
Training Loss
Validation Loss
Log Loss
Epoch


Loss vs Epoch (η=0.005, α=0.001)
Training Loss
Validation Loss
Log Loss
Epoch


Precision and Recall vs Epoch (η=0.005, α=0.0001)
Precision
Recall
Score
Epoch


Precision and Recall vs Epoch (η=0.005, α=0.001)
Precision
Recall
Score
Epoch


Precision-Recall Curve (η=0.005, α=0.0001)
Precision
Recall


Precision-Recall Curve (η=0.005, α=0.001)
Precision
Recall

```
Best Model Found:
Step size (η): 0.005
Regularization (α): 0.0001
Weights: [[-0.62466473 -0.77410025 -0.58705224 -0.66149684 -0.29046395  0.4446004
  -0.59467208 -0.73791557 -0.21106128  0.40746631 -1.08674784  0.20838655
  -0.68750509 -0.92702163 -0.19984666  0.77376031  0.11364685 -0.39478415
   0.38085302  0.60162628 -1.02411299 -1.26410265 -0.8846686  -1.00929038
  -0.92123317 -0.03359415 -0.81866621 -1.0340216  -1.00435668 -0.1680371 ]]
Bias: [0.35667871]

Five Predictions:
Input 1: True=0, Predicted=0, Probability=0.000
Input 2: True=1, Predicted=1, Probability=1.000
Input 3: True=0, Predicted=0, Probability=0.003
Input 4: True=1, Predicted=0, Probability=0.452
Input 5: True=0, Predicted=0, Probability=0.000
```