hw3

February 3, 2025

[31]:
```
# Implement a program to train a binary logistic regression model using␣
 ↪mini-batch SGD. Use the logistic
# regression model we derived in class, corresponding to Equation (4.90) from␣
 ↪the textbook, and where
# the feature transformation  is the identity function.

# The program should include the following hyperparameters:
# • Batch size
# • Fixed learning rate
# • Maximum number of iterations

# The program should output the following:
# • The learned weights w
# • The learned bias b
# • The final training loss

import numpy as np

EPSILON = 1e-9

# Sigmoid function
def sigmoid(z):
    """
    z: input data
    """
    return 1 / (1 + np.exp(-z))

# Predict labels
def predict(X, w, b):
    """
    X: input data
    w: weights
    b: bias
    """
    z = np.dot(X, w) + b
    return sigmoid(z)
```

```python
# Compute loss
def loss(y, a):
    """
    y: true labels
    a: predicted labels
    """
    return -np.mean(y * np.log(a + EPSILON) + (1 - y) * np.log(1 - a + EPSILON))

# Compute gradients
def compute_gradients(X, a, y, batch_size):
    """
    X: input data
    a: predicted labels
    y: true labels
    batch_size: batch size
    """
    dw = np.dot(X.T, (a - y)) / batch_size
    db = np.sum(a - y) / batch_size
    return dw, db

# Train with mini-batch SGD
def train_with_sgd(X, y, w, b, batch_size, max_iters, learning_rate):
    """
    X: input data
    y: true labels
    w: weights
    b: bias
    batch_size: batch size
    max_iters: maximum number of iterations
    learning_rate: fixed learning rate
    """
    for i in range(max_iters):
        # Randomly sample a batch of data
        indices = np.random.choice(X.shape[0], batch_size, replace=False)
        X_batch = X[indices]
        y_batch = y[indices]
        # Compute gradients based on the batch predictions
        a = predict(X_batch, w, b)
        dw, db = compute_gradients(X_batch, a, y_batch, batch_size)
        # Update weights and bias
        w -= learning_rate * dw
        b -= learning_rate * db

        if i % 10 == 0:  # Print loss every 100 iterations
            print(f"Iteration {i}, Loss: {loss(y_batch, a):.4f}")

    final_loss = loss(y, predict(X, w, b))
```

```
    return w, b, final_loss
```

[32]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score


# n this problem, you will run a logistic regression model for classification␣
 ↪on a breast cancer dataset
# (a) Download the Wisconsin Breast Cancer dataset from the UCI Machine␣
 ↪Learning Repository 1 or
# scikit-learn's built-in datasets 2.

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# (b) Split the dataset into train, validation, and test sets.
# Split the dataset into train, validation, and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
 ↪5, random_state=42)
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)


# (c) Report the size of each class in your training (+ validation) set.
# Report the size of each class in the training (+ validation) set
print('Training Set')
print('Class 0:', np.sum(y_train == 0))
print('Class 1:', np.sum(y_train == 1))
print('Validation Set')
print('Class 0:', np.sum(y_val == 0))
print('Class 1:', np.sum(y_val == 1))

# (d) Train a binary logistic regression model using your implementation from␣
 ↪problem 3. Initialize
# the model weights randomly, sampling from a standard Gaussian distribution.␣
 ↪Experiment with
```

```
# different choices of fixed learning rate and batch size.

# Initialize the weights randomly
np.random.seed(42)
w = np.random.randn(X_train.shape[1])
b = np.random.randn(1)

# Hyperparameters
batch_size = 32
max_iters = 1000
learning_rate = 0.01

# Train the model using mini-batch SGD
w, b, final_loss = train_with_sgd(X_train, y_train, w, b, batch_size,␣
 ↪max_iters, learning_rate)
print("\nFinal Training Loss:", final_loss)


# (e) Use the trained model to report the performance of the model on the test␣
 ↪set. For evaluation
# metrics, use accuracy, precision, recall, and F1-score.

# Use the learned weights to predict on the test set
z = predict(X_test, w, b)
y_pred = np.round(z)

# Evaluate the model using accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display results on the test set
print("\nModel Performance on Test Set:")
print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-score:  {f1:.4f}")


# (f) Summarize your findings.
```

```
Training Set
Class 0: 77
Class 1: 122
Validation Set
Class 0: 72
```

```
Class 1: 127
Iteration 0, Loss: 2.7426
Iteration 10, Loss: 1.2224
Iteration 20, Loss: 1.6434
Iteration 30, Loss: 2.5950
Iteration 40, Loss: 1.6533
Iteration 50, Loss: 1.6693
Iteration 60, Loss: 0.7248
Iteration 70, Loss: 1.5207
Iteration 80, Loss: 1.0582
Iteration 90, Loss: 0.4741
Iteration 100, Loss: 1.0768
Iteration 110, Loss: 1.2715
Iteration 120, Loss: 0.7635
Iteration 130, Loss: 0.2968
Iteration 140, Loss: 0.4998
Iteration 150, Loss: 0.3795
Iteration 160, Loss: 1.0238
Iteration 170, Loss: 0.3794
Iteration 180, Loss: 0.2548
Iteration 190, Loss: 0.3576
Iteration 200, Loss: 0.5790
Iteration 210, Loss: 0.2386
Iteration 220, Loss: 0.4439
Iteration 230, Loss: 0.3025
Iteration 240, Loss: 0.4022
Iteration 250, Loss: 0.2888
Iteration 260, Loss: 0.1602
Iteration 270, Loss: 0.7007
Iteration 280, Loss: 0.7724
Iteration 290, Loss: 0.5859
Iteration 300, Loss: 0.2282
Iteration 310, Loss: 0.3270
Iteration 320, Loss: 0.1556
Iteration 330, Loss: 0.1782
Iteration 340, Loss: 0.1992
Iteration 350, Loss: 0.1290
Iteration 360, Loss: 0.4175
Iteration 370, Loss: 0.3513
Iteration 380, Loss: 0.3043
Iteration 390, Loss: 0.4789
Iteration 400, Loss: 0.1586
Iteration 410, Loss: 0.1189
Iteration 420, Loss: 0.2298
Iteration 430, Loss: 0.3896
Iteration 440, Loss: 0.1835
Iteration 450, Loss: 0.1997
Iteration 460, Loss: 0.1151
```

```
Iteration 470, Loss: 0.1359
Iteration 480, Loss: 0.3971
Iteration 490, Loss: 0.1212
Iteration 500, Loss: 0.2812
Iteration 510, Loss: 0.2497
Iteration 520, Loss: 0.1093
Iteration 530, Loss: 0.0777
Iteration 540, Loss: 0.2632
Iteration 550, Loss: 0.0739
Iteration 560, Loss: 0.1132
Iteration 570, Loss: 0.3001
Iteration 580, Loss: 0.2457
Iteration 590, Loss: 0.2572
Iteration 600, Loss: 0.0577
Iteration 610, Loss: 0.1134
Iteration 620, Loss: 0.0748
Iteration 630, Loss: 0.0620
Iteration 640, Loss: 0.0862
Iteration 650, Loss: 0.2579
Iteration 660, Loss: 0.2191
Iteration 670, Loss: 0.1006
Iteration 680, Loss: 0.2086
Iteration 690, Loss: 0.1831
Iteration 700, Loss: 0.0573
Iteration 710, Loss: 0.1033
Iteration 720, Loss: 0.0935
Iteration 730, Loss: 0.2642
Iteration 740, Loss: 0.0969
Iteration 750, Loss: 0.0848
Iteration 760, Loss: 0.0893
Iteration 770, Loss: 0.1415
Iteration 780, Loss: 0.0928
Iteration 790, Loss: 0.0561
Iteration 800, Loss: 0.0395
Iteration 810, Loss: 0.0834
Iteration 820, Loss: 0.0326
Iteration 830, Loss: 0.2575
Iteration 840, Loss: 0.1905
Iteration 850, Loss: 0.0437
Iteration 860, Loss: 0.2660
Iteration 870, Loss: 0.0507
Iteration 880, Loss: 0.0618
Iteration 890, Loss: 0.0995
Iteration 900, Loss: 0.1658
Iteration 910, Loss: 0.1105
Iteration 920, Loss: 0.0614
Iteration 930, Loss: 0.1020
Iteration 940, Loss: 0.2362
```

```
Iteration 950, Loss: 0.2256
Iteration 960, Loss: 0.0876
Iteration 970, Loss: 0.0752
Iteration 980, Loss: 0.0913
Iteration 990, Loss: 0.0583

Final Training Loss: 0.10011016476489475

Model Performance on Test Set:
Accuracy:  0.9766
Precision: 0.9906
Recall:    0.9722
F1-score:  0.9813
```