AI and Machine Learning, Homework3

Author: Ying Yiwen
Number: 12210159

Contents

1 Code Implementation

1.1 Read in Data and Split Data

First, we need to read in the csv-format wine.data. Then randomly remove a class.

```
def read_dataset():
    # read dataset
    column_names = [
    'Class Label', 'Alcohol', 'Malic Acid', 'Ash', 'Alcalinity of Ash',
    'Magnesium', 'Total Phenols', 'Flavanoids', 'Nonflavanoid Phenols',
    'Proanthocyanins', 'Color Intensity', 'Hue', 'OD280/OD315 of Diluted Wines',
    'Proline'
    ]
    data = pd.read_csv('wine.data', header=None, names=column_names)
    # randomly remove a class
    unique_classes = data['Class Label'].unique()
    class_to_remove = np.random.choice(unique_classes)
    data_filtered = data[data['Class Label'] != class_to_remove]
    print(data_filtered)
    return data_filtered
```

As the class label may add difficulty to perception, we change the label to be 1 and -1.

```
def convert_labels(y):
    # convert labels to -1 and 1
    converted_labels = []
    y = np.array(y)
    for label in y:
    if label == y[0]:
    converted_labels.append(-1)
    else:
    converted_labels.append(1)
    return np.array(converted_labels)
```

The data format is hard to use for subsequent operations, so we need to split x and y (features and labels).

```
def split_data(data_filtered):
    # split data into features and labels
    data_filtered = data_filtered.sample(frac=1)
    # separate features and labels
    x = data_filtered.iloc[:, 1:]
    y = data_filtered.iloc[:, 0]
    y = convert_labels(y)
    return x, y
```

Then split the data into train set and test set by 0.7:0.3.

```
def split_test_and_train(x, y):
    # split data into train and test
    x_train = x.iloc[:int(len(x)*0.7), :]
    x_test = x.iloc[int(len(x)*0.7):, :]
    y_train = y[:int(len(y)*0.7)]
    y_test = y[int(len(y)*0.7):]
    return x_train, x_test, y_train, y_test
```

1.2 SGD Update Realization

SGD is stochastic gradient descent. It use only one input sample to update model weights at the same time.

```
def sgd_update(self, X, y, X_test, y_test):
    # stochastic gradient descent
    break_out = False
    epoch_no_improve = 0
    for n in range(self.n_iter):
        for i,x in enumerate(X):
            # predict the label of x
            y_pred = self._predict(x)
            # print(f"Predicted label:{y_pred}, Actual label:{y[i]}")
            # compute loss
            loss = self._loss(y[i], y_pred)
            self.loss.append(loss)
            # compute test loss
            test_loss = self._loss_batch(y_test, self._predict(X_test))
            self.test_loss.append(test_loss)
            # check if break out
            if self.tol is not None:
                if loss < self.best_loss - self.tol:
                    # update best loss
                    self.best_loss = loss
                    epoch_no_improve = 0
                elif np.abs(loss - self.best_loss) < self.tol:
                    # no improvement
                    epoch_no_improve += 1
                if epoch_no_improve >= self.patience:
                    print(f"Early stopping at epoch {n}")
                    break_out = True
                    break
                else:
                    epoch_no_improve = 0
            # compute gradient and update weights
            grad = self._gradient(x, y[i], y_pred)
            self.W = self.W - self.lr * grad
        # print(f"Epoch:{n}, Train Loss:{loss}, Test Loss:{test_loss}")
        # check if break out
        if break_out:
        break_out = False
            break
```

1.3 BGD Update Realization

BGD means batch gradient descent, so it use the whole train set as input at a time to update model weights.

```
def batch_update(self, X, y, X_test, y_test):
    # batch gradient descent
    break_out = False
    epoch_no_improve = 0
    for n in range(self.n_iter):
        # predict the label of x
        y_pred = self._predict(X)
        # compute loss
```

```
        loss = self._loss_batch(y, y_pred)
        self.loss.append(loss)
        # compute test loss
        test_loss = self._loss_batch(y_test, self._predict(X_test))
        self.test_loss.append(test_loss)
        # check if break out
        if self.tol is not None:
            if loss < self.best_loss - self.tol:
                # update best loss
                self.best_loss = loss
                epoch_no_improve = 0
            elif np.abs(loss - self.best_loss) < self.tol:
                # no improvement
                epoch_no_improve += 1
            if epoch_no_improve >= self.patience:
                print(f"Early stopping at epoch {n}")
                break_out = True
                break
            else:
                epoch_no_improve = 0
        # compute gradient and update weights
        grad = self._gradient_batch(X, y, y_pred)
        self.W = self.W - self.lr * grad
    # print(f"Epoch:{n}, Train Loss:{loss}, Test Loss:{test_loss}")
    # check if break out
    if break_out:
    break_out = False
        break
```

1.4 Performance Evaluation

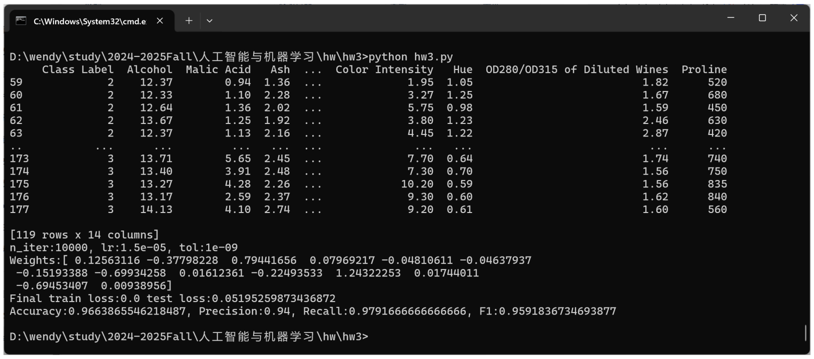
After fitting, the loss may can't totally express the performance of the fitted weights. Instead, we can use accuracy, precision, recall, F1 score to evaluate the model.

```
def evaluate(self, X, y):
    # predict the label of X
    y_pred = self._predict(self._preprocess_data(X))
    y_pred = np.where(y_pred > 0, 1, -1)
    # evaluate model accuracy, precision, recall, f1 score
    TP = sum((y_pred[i] == 1) and (y[i] == 1) for i in range(len(y)))
    FP = sum((y_pred[i] == 1) and (y[i] == -1) for i in range(len(y)))
    TN = sum((y_pred[i] == -1) and (y[i] == -1) for i in range(len(y)))
    FN = sum((y_pred[i] == -1) and (y[i] == 1) for i in range(len(y)))
    accuracy = (TP + TN) / (TP + TN + FP + FN)
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    f1 = 2 * precision * recall / (precision + recall)
    print(f"Accuracy:{accuracy}, Precision:{precision}, Recall:{recall}, F1:{f1}")
```

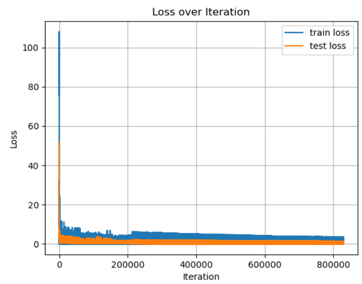
2 Model Result

2.1 SGD Update Method

Using the stochastic gradient descent, we can fit the weights to classify the label well.



(a) Running Result of SGD



(b) SGD Loss Function

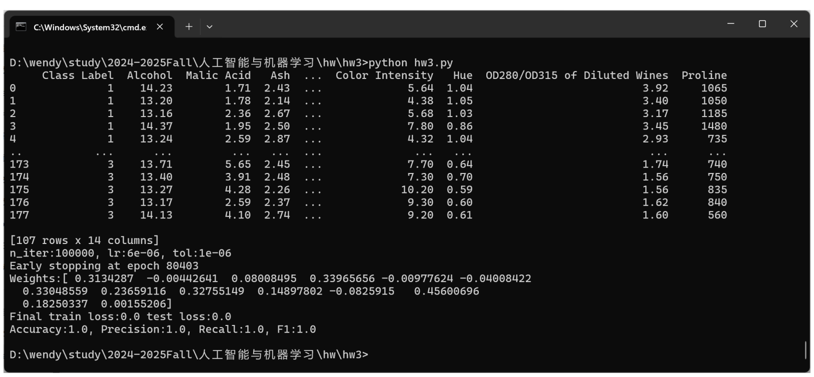
Fig. 1: The Result of SGD Fitting Method

In this test, the compiler randomly chooses label 2 and 3, so there's 119 sample data.
Superparameters: n of iteration: 10000, loss rate:1.5e-05, tolerance:1e-09, patience:100, initial W: $random(14) * 0.5$
It doesn't early stop in 10000 epoch.

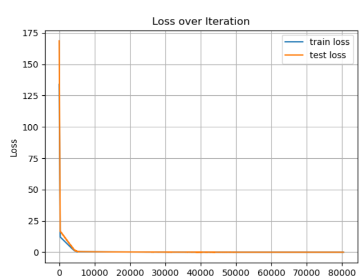
The fitted result is:
Weights:[0.12563116 -0.37798228 0.79441656 0.07969217 -0.04810611 -0.04637937 -0.15193388 -0.69934258 0.01612361 -0.22493533 1.24322253 0.01744011 -0.69453407 0.00938956]
Accuracy:0.9663865546218487, Precision:0.94, Recall:0.9791666666666666, F1:0.9591836734693877

2.2 BGD Update Method

Using the batch gradient descent, we can fit the weights to classify the label well.



(a) Running Result of BGD



(b) BGD Loss Function

Fig. 2: The Result of BGD Fitting Method

In this test, the compiler randomly chooses label 1 and 3, so there's 107 sample data.
Superparameters: n of iteration: 100000, loss rate:6e-06, tolerance:1e-06, patience:30, initial W: $random(14) * 0.5$

It early stops at epoch 80403.

The fitted result is:
Weights:[0.3134287 -0.00442641 0.08008495 0.33965656 -0.00977624 -0.04008422 0.33048559 0.23659116 0.32755149 0.14897802 -0.0825915 0.45600696 0.18250337 0.00155206]
Accuracy:1.0, Precision:1.0, Recall:1.0, F1:1.0