CS 559-B: Machine Learning: Fundamentals and Applications

Full name: Thanapoom Phatthanaphan
CWID: 20011296

Homework 1 (Problem 6)

[Logistic Regression, MLE] In this problem, you need to use MLE to derive and build a logistic regression classifier (suppose the target/response $y \in \{0, 1\}$):

Write the codes to build and train the classifier on Iris plant dataset (https://archive.ics.uci.edu/ml/datasets/iris). The iris dataset contains 150 samples with 4 features for 3 classes. To simplify the problem, we only consider: (a) two classes, i.e., virginica and non- virginica; (b) The first 2 types of features for training, i.e., sepal length and sepal width. Based on these simplified settings, train the model using gradient descent. Please show the classification results. (Note that (1) you could split the iris dataset into train/test set. (2) You could visualize the results by showing the trained classifier overlaid on the train/test data. (3) You could tune several hyperparameters, e.g., learning rate, weight initialization method etc, to see their effects. (3) You could use sklearn or other packages to load and process the data, but you can not use the package to train the model).

In [1]:
```
# import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [2]:
```
# import the dataset
columns_name = ['Sepal length in cm', 'Sepal width in cm', 'Petal length in
data = pd.read_csv('iris.data', header=None, names=columns_name)
data
```

Out[2]:

	Sepal length in cm	Sepal width in cm	Petal length in cm	Petal width in cm	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [3]:
```
data.describe()
```

Out[3]:

	Sepal length in cm	Sepal width in cm	Petal length in cm	Petal width in cm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [4]:
```
# consider two classes: virginica and non-virginica
# define the class: virginica as 1, and non-virginica as 0
data['Class'][data['Class'] != 'Iris-virginica'] = 0
data['Class'][data['Class'] == 'Iris-virginica'] = 1
data
```

```
/var/folders/bk/66r4ld3j7hj8yg_49fhv2fr40000gn/T/ipykernel_7852/2033948320.
py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Class'][data['Class'] != 'Iris-virginica'] = 0
/var/folders/bk/66r4ld3j7hj8yg_49fhv2fr40000gn/T/ipykernel_7852/2033948320.
py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Class'][data['Class'] == 'Iris-virginica'] = 1
```

Out[4]:

	Sepal length in cm	Sepal width in cm	Petal length in cm	Petal width in cm	Class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	1
146	6.3	2.5	5.0	1.9	1
147	6.5	3.0	5.2	2.0	1
148	6.2	3.4	5.4	2.3	1
149	5.9	3.0	5.1	1.8	1

150 rows × 5 columns

In [5]:
```
# define the columns to use as training and testing data
X = data[['Sepal length in cm', 'Sepal width in cm']]
y = data['Class']

# split the dataset into training and testing data by 80% and 20%, respectiv
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

In [6]:
```
# define the sigmoid function
def sigmoid(a):

    """A function to map input values to a value between 0 and 1"""

    a = a.astype(float)
    return 1 / (1 + np.exp(-a))


# define the gradient descent function for logistic regression
def gradient_descent(X, y, learning_rate, epochs, weight, bias, n_samples, r

    """A function to minimize the error of the model"""

    for i in range(epochs):

        # get the prediction value at the current value of weight and bias i
        y_pred = sigmoid(np.dot(X, weight) + bias)
```

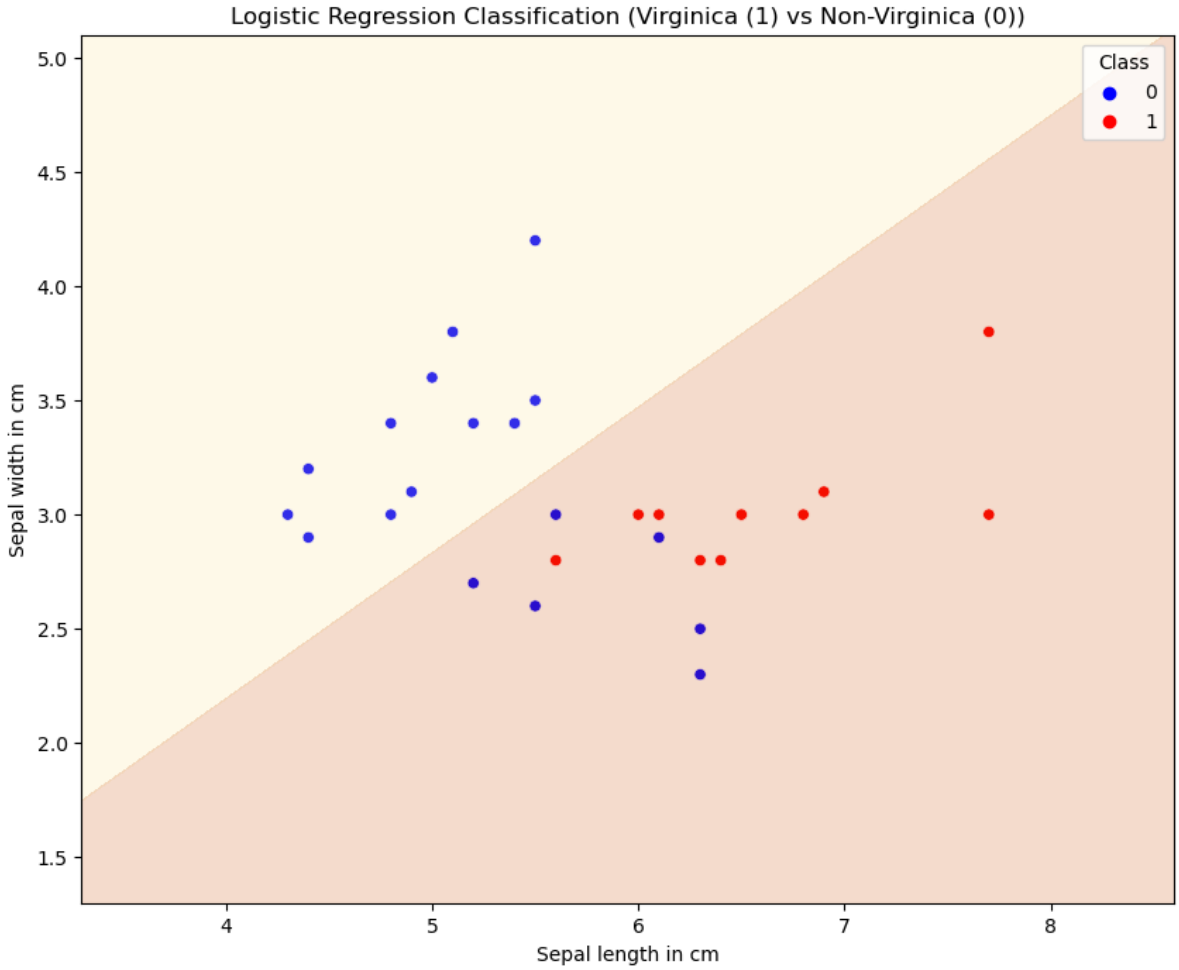
```
        # calculate the loss
        cost_dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
        cost_db = (1 / n_samples) * np.sum(y_pred - y)

        # update weight and bias in each epoch
        weight = weight - learning_rate * cost_dw
        bias = bias - learning_rate * cost_db

    return weight, bias

def predict(X, weight, bias):

    """A function to predict the class"""

    # predict the class
    y_pred = sigmoid(np.dot(X, weight) + bias)

    # put the predictions value (0 or 1) into the list
    class_pred = []

    for y in y_pred:
        if y >= 0.5:
            class_pred.append(1)
        else:
            class_pred.append(0)

    return class_pred

def evaluation(y_pred, y):

    """A function to evaluate the predictions accuracy of the model"""

    accuracy = np.sum(y_pred == y) / len(y)

    return accuracy


# train the model with different values of learning rate to find the best pa
best_acc = 0
for learning_rate in [0.01, 0.1, 1, 10]:

    # initialize parameters
    epochs = 100
    weight = np.zeros(X_train.shape[1])
    bias = 0
    n_samples, n_features = X_train.shape

    # get the final weight and bias
    weight, bias = gradient_descent(X_train, y_train, learning_rate, epochs,

    # make predictions
    y_pred = predict(X_test, weight, bias)

    # evaluate the accuracy of predictions model
    eval_acc = evaluation(y_pred, y_test)
    print('Accuracy (learning rate = {}):'.format(learning_rate), eval_acc)

    # check the best accuracy at the currect value of learning rate
    if eval_acc > best_acc:
        best_acc = eval_acc
        best_learning_rate = learning_rate

print('The predictions accuracy of the model (using the best learning rate a
```

```
# train the model with the best parameter of learning rate
# initialize parameters
epochs = 100
weight = np.zeros(X_train.shape[1])
bias = 0
n_samples, n_features = X_train.shape

# get the final weight and bias
weight, bias = gradient_descent(X_train, y_train, best_learning_rate, epochs

# make predictions
y_pred = predict(X_test, weight, bias)

# evaluate the accuracy of predictions model
eval_acc = evaluation(y_pred, y_test)
```

```
Accuracy (learning rate = 0.01): 0.6333333333333333
Accuracy (learning rate = 0.1): 0.6333333333333333
Accuracy (learning rate = 1): 0.8
Accuracy (learning rate = 10): 0.6333333333333333
The predictions accuracy of the model (using the best learning rate at 1):
0.8
```

In [17]:

```
# Generate a grid of points to plot the decision boundary
x_min, x_max = X_test['Sepal length in cm'].min() - 1, X_test['Sepal length
y_min, y_max = X_test['Sepal width in cm'].min() - 1, X_test['Sepal width ir
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, (
Z = np.array([y_pred for y_pred in (np.dot(np.c_[xx.ravel(), yy.ravel()], we
Z = sigmoid(Z)
Z = Z.reshape(xx.shape)

# plot the scatter plot to display prediction results
plt.figure(figsize=(10, 8))
custom_palette = {0: 'blue', 1: 'red'}
scatter = sns.scatterplot(x='Sepal length in cm', y='Sepal width in cm', dat
plt.title('Logistic Regression Classification (Virginica (1) vs Non-Virginic
plt.xlabel('Sepal length in cm')
plt.ylabel('Sepal width in cm')
plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], alpha=0.2, cmap='YlOrBr')
plt.show()
```


Logistic Regression Classification (Virginica (1) vs Non-Virginica (0))
Class
0
1
Sepal width in cm
Sepal length in cm
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8

In [ ]: