```
# This code is written and submitted by Balkrishna Bhatt (201673048)
# import the library like numpy,


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
import warnings

# to remove all the warning in the code
warnings.filterwarnings("ignore")
dataSet = {}

# simple Binary Perceptron Implementation with consist of pseudo code
# Answer 2


def predict(row, weights):  # Make a prediction with weights
    activation = weights[0]
    for i in range(len(row)-1):
        activation += weights[i + 1] * float(row[i])
    return 'class-1' if activation >= 0.0 else 'class-0'


# random Data Set to prove concept of binary Perceptron.
dataset = [[2.7810836, 2.550537003, 3.34353454, 5.34343535, 'class-0'],
           [1.465489372, 2.362125076, 2.34353454, -1.343435350, 'class-1'],
           [3.396561688, 4.400293529, 4.34353454, 5.34343535, 'class-0'],
           [1.38807019, 1.850220317, 3.34353454, 4.34343535, 'class-0'],
           [3.06407232, 3.005305973, -1.34353454, 5.34343535, 'class-0'],
           [7.627531214, 2.759262235, 4.34353454, 5.34343535, 'class-1'],
           [5.332441248, 2.088626775, 3.34353454, 3.34343535, 'class-1'],
           [6.922596716, 1.77106367, 3.34353454, 2.34343535, 'class-1'],
           [8.675418651, -0.242068655, -2.34353454, 5.34343535, 'class-1'],
           [7.673756466, 3.508563011, 3.34353454, 2.34343535, 'class-1']]
weights = [-0.1, 0.20653640140000007, -0.23418117710000003,
           0.343534547987879, -0.34343535087987687]
correct = 0
inCorrect = 0

for row in dataset:
    if row[-1] == 'class-1' or row[-1] == 'class-0':
        prediction = predict(row, weights)
    if (row[-1] == prediction):
        correct += 1
    else:
        inCorrect += 1
    accuracy = correct / len(dataset) * 100  # to measure Accuracy
print("Here is the answer2: ")
print("This is accuracy of simple Binary Perceptron = ", accuracy, "%")

# this method is to classify and labeling all 3 types of data


def prepare_data(data):
    classes = np.unique([line.split(",")[-1].strip() for line in data])
    datasets = {c: [] for c in classes}
    for line in data:
```

```
        values = [float(v) for v in line.split(",")[:-1]]
        label = []
        if (line.split(",")[-1].strip() == "class-1"):
            {
                datasets.pop,
            }
        else:
            label = line.split(",")[-1].strip()

        for c in classes:
            if c == label:
                datasets[c].append((values, 1))
            else:
                datasets[c].append((values, -1))
        global dataSet
        dataSet = datasets
    return datasets


# this method is to read the data from train dataset
with open("train.data") as read_data:
    train_data = read_data.readlines()
train_datasets = prepare_data(train_data)

# this method is to read the data from test dataset
with open("test.data") as read_data:
    test_data = read_data.readlines()
test_datasets = prepare_data(test_data)

# simple Binary Perceptron Implementation for to train classifiers to discriminate
between class 1 and class 2, class 2 and class 3, and class 1 and class 3.
# Answer 3


def perceptron_test(data, weights, bias):
    correct = 0
    inCorrect = 0
    for x, y in data:
        y_pred = np.dot(weights, x) + bias
        if np.sign(y_pred) == y:
            correct += 1
        else:
            inCorrect += 1

    accuracy = correct / len(data) * 100
    return accuracy


def perceptron_train(data, itr=20):
    weights = None
    bias = 0
    weights = np.full((1, 4), 0)

    for it in range(itr):
        for x, y in data:

            xx = np.array(x).reshape(len(x), 1)
            y_pred = np.dot(weights, xx) + bias
```

```
            if y * y_pred <= 0:
                xx = xx.reshape(1, len(x))
                weights = weights + (y * xx)
                bias += y

    return weights, bias


print("Here is the answer3: ")
# for class-1 & class-2 classification
class1_2_train = train_datasets["class-1"] + train_datasets["class-2"]
class1_2_test = test_datasets["class-1"] + test_datasets["class-2"]
weights_1_2, bias_1_2 = perceptron_train(class1_2_test)
acc_train_1_2 = perceptron_test(class1_2_train, weights_1_2, bias_1_2)
acc_test_1_2 = perceptron_test(class1_2_test, weights_1_2, bias_1_2)
print("The accuracy of the dataset consisting of class-1 & class-2 is below. \n",
      "For the train data = ", acc_train_1_2, "%", "\n For the test data = ",
acc_test_1_2, "%")

# for class-2 & class-3 classification
class2_3_train = train_datasets["class-2"] + train_datasets["class-3"]
class2_3_test = test_datasets["class-2"] + test_datasets["class-3"]
weights_2_3, bias_2_3 = perceptron_train(class2_3_train)
acc_train_2_3 = perceptron_test(class2_3_train, weights_2_3, bias_2_3)
acc_test_2_3 = perceptron_test(class2_3_test, weights_2_3, bias_2_3)
print("The accuracy of the dataset consisting of class-2 & class-3 is below. \n",
      "For the train data = ", acc_train_2_3, "%", "\n For the test data = ",
acc_test_2_3, "%")

# for class-1 & class-3 classification
class1_3_train = train_datasets["class-1"] + train_datasets["class-3"]
class1_3_test = test_datasets["class-1"] + test_datasets["class-3"]
weights_1_3, bias_1_3 = perceptron_train(class1_3_train)
acc_train_1_3 = perceptron_test(class1_3_train, weights_1_3, bias_1_3)
acc_test_1_3 = perceptron_test(class1_3_test, weights_1_3, bias_1_3)
print("The accuracy of the dataset consisting of class-1 & class-3 is below. \n",
      "For the train data = ", acc_train_1_3, "%", "\n For the test data = ",
acc_test_1_3, "%")

# Answer 4


class BinaryPerceptron:
    def __init__(self, learning_rate=0.1, max_iter=20):
        self.learning_rate = learning_rate
        self.max_iter = max_iter

    def train(self, X, y):
        self.weights = np.zeros(X.shape[1])
        self.bias = 0
        for i in range(self.max_iter):
            for j in range(X.shape[0]):
                y_pred = self.predict(X[j])
                if y_pred != y[j]:
                    self.weights += self.learning_rate * y[j] * X[j]
                    self.bias += self.learning_rate * y[j]

    def predict(self, x):
        linear_output = np.dot(x, self.weights) + self.bias
```

```
        y_pred = np.sign(linear_output)
        return y_pred


class MultiClassPerceptron:
    def __init__(self, learning_rate=0.1, max_iter=20):
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.binary_perceptrons = {}

    def train(self, X, y):
        classes = np.unique(y)
        for c in classes:
            y_c = np.where(y == c, 1, -1)
            perceptron = BinaryPerceptron(self.learning_rate, self.max_iter)
            perceptron.train(X, y_c)
            self.binary_perceptrons[c] = perceptron

    def predict(self, X):
        y_pred = np.zeros(X.shape[0])
        for i, x in enumerate(X):
            scores = []
            for c, perceptron in self.binary_perceptrons.items():
                score = perceptron.predict(x)
                scores.append(score)
            winner = np.argmax(scores)
            y_pred[i] = list(self.binary_perceptrons.keys())[winner]
        return y_pred


# Load the data
train_data = pd.read_csv("train.data", header=None)
test_data = pd.read_csv("test.data", header=None)

X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values

X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values

# Preprocess the data
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)


# Train the model
perceptron = MultiClassPerceptron()
perceptron.train(X_train, y_train)

# Evaluate the model
y_pred_train = perceptron.predict(y_train)
train_acc = np.mean(y_pred_train == y_train)*100


y_pred_test = perceptron.predict(y_test)
test_acc = np.mean(y_pred_test == y_test) * 100
print("Here is the answer4: ")
print("Train accuracy:", train_acc, "%")
```

```
print("Test accuracy:", test_acc, "%")


# Answer 5
newData = {}
x = []
y = []


class MulticlassPerceptronL2:
    def __init__(self, lr, num_epochs, reg_coef):
        self.lr = lr
        self.num_epochs = num_epochs
        self.reg_coef = reg_coef

    def fit(self, X, y):
        X = np.insert(X, 0, 1, axis=1)  # Add bias term to X
        num_classes = len(np.unique(y))
        num_features = X.shape[1]
        self.weights = np.zeros((num_features, num_classes))
        y_onehot = np.eye(num_classes)[y]  # One-hot encode y

        for epoch in range(self.num_epochs):
            for i in range(len(X)):
                xi = X[i]
                yi = y_onehot[i]
                scores = np.dot(xi, self.weights)
                y_hat = np.argmax(scores)

                if y_hat != np.argmax(yi):
                    self.weights[:, np.argmax(
                        yi)] += self.lr * xi - self.lr * self.reg_coef *
self.weights[:, np.argmax(yi)]
                    self.weights[:, y_hat] -= self.lr * xi + \
                        self.lr * self.reg_coef * self.weights[:, y_hat]

    def predict(self, X):
        X = np.insert(X, 0, 1, axis=1)  # Add bias term to X
        scores = np.dot(X, self.weights)
        return np.argmax(scores, axis=1).astype(int)


def prepare_data(data):
    global x, y, newData
    y = [line.split(",")[-1].strip() for line in data]
    encoder = LabelEncoder()
    y = encoder.fit_transform(y)

    x = []
    for line in data:
        values = [float(v) for v in line.split(",")[:-1]]
        x.append((values))
    x = np.array(x)
    y = np.array(y)
    newData = x
    return x, y


# this method is to read the data from test dataset
```

```
with open("test.data") as read_data:
    test_data = read_data.readlines()
test_datasets = prepare_data(test_data)
X_test, y_test = prepare_data(test_data)

with open("train.data") as read_data:
    train_data = read_data.readlines()
test_datasets = prepare_data(train_data)
X_train, y_train = prepare_data(train_data)


X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42)

print(f"Here is the answer5: ")
for reg_coef in [0.01, 0.1, 1.0, 10.0, 100.0]:
    print(f"Regularization coefficient: {reg_coef}")
    clf = MulticlassPerceptronL2(lr=0.1, num_epochs=20, reg_coef=reg_coef)
    clf.fit(X_train, y_train)
    test_acc = np.mean(clf.predict(X_test) == y_test) * 100
    train_acc = np.mean(clf.predict(X_train) == y_train) * 100
    print(f"Test accuracy: {test_acc:.2f} %")
    print(f"Train accuracy: {train_acc:.2f} %")
```