Project 1 感知机实验报告

一、 实验目的

实验目的有以下三个：

1. 自行生成有两个特征的线性可分数据集，实现感知机算法，对数据散点图、目标函数f和最终假设g做可视化展现。
2. 研究实例数和感知机算法收敛所需次数的关系，其中实例数分别为20, 20, 100和1000。
3. 研究学习率和感知机算法收敛所需次数的关系，其中学习率分别为1, 0.01和0.0001。

二、 实验原理

## 1. 感知机模型介绍

假设输入空间（特征空间）是，输出空间是。输入表示实例的特征向量，对应于输入空间（特征空间）的点；输出表示实例的类别。由输入空间到输出空间的如下函数：

称为感知机（Perception）。其中w和b为感知机模型参数，叫作权值（weight）或权值向量（weight vector），叫作偏置（bias），表示w和x的内积。sign是符号函数，即

感知机是一种线性分类模型，属于判别模型。感知机模型的假设空间是定义在特征空间中的所有线性分类模型或线性分类器，即函数集合。

我们可以把感知机看做是n维实例空间中的超平面决策面。对于超平面一侧的实例，感知机输出1，对于另一侧的实例输出-1.。如果存在某个超平面S能够将数据集T的正实例点和负实例点完全正确地划分到超平面的两侧，则称数据集T为线性可分样数据集。

## 2. 感知机的损失函数

假设训练数据及是线性可分的，感知机学习的目标是求得一个能够将训练集正实例点和负实例点完全正确分开的分离超平面。为了找出这样的超平面，感知机选择以误分类点到超平面S的总距离作为损失函数。为此，首先写出输入空间中任一点到超平面S的距离：

这里，||w||是w的范数。

其次，对于误分类的数据来说，有成立。因为当时，；而当时，。因此，假设超平面S的误分类点集合为M，那么所有误分类点到超平面S的总距离为，不考虑，得到感知机学习的损失函数为：

## 3. 感知机学习算法

感知机学习算法具体采用随机梯度下降法求解。首先，任意选取一个超平面 ，然后用梯度下降法不断地极小化目标函数。假设误分类点集合M是固定的，那么损失函数的梯度由

给出。

随机选取一个误分类点，对w, b进行更新：

式中是步长，在统计学习中又称为学习率。这样，通过迭代可以期待损失函数不断减小，直到为0。综上所述，得到如下算法：

|  |
| --- |
| 输入：训练数据集，其中，，；  输出：w, b; 感知机模型。  (1) 选取初值；  (2) 在训练集中选取数据；  (3) 如果（实例i被分错），  (4) 转至(2)，直至训练集中没有误分类的点。 |

注：本实验中输入的训练数据集T应当自行生成，且满足线性可分条件，并重复做实例数分别为20, 20, 100和1000，学习率分别为1, 0.01和0.0001的12种情况下的实验，对结果进行分析。

三、实验过程

## 1. 试验器材

Win10+Python3.5.4编译器。

## 2. 实验输入与输出

输入：实例数和学习率，实例数分别为20, 20, 100和1000，学习率分别为1, 0.01和0.0001。

本实验会随机生成两个大小为的二维数组作为特征，其中第一个数组是[0,50)之间的浮点数，第二个数组是[50, 100)之间的浮点数，这样保证训练数据集线性可分。再生成类别列表，其中前个元素为-1，后个元素为1。并将线性可分数据集作为感知机模型的输入。

输出：有f的散点图，和有f和感知机模型最终假设g的散点图。

## 3. 实验代码

import numpy as np

import random

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

class PerceptronDemo:

#初始化实例数和学习率

def \_\_init\_\_(self,sample\_size, l\_rate):

self.sample\_size = sample\_size

self.half\_sample = (int) (self.sample\_size/2)

self.l\_rate = l\_rate

#随机生成线性可分的数据集

def generate\_data(self):

X\_f = np.random.rand(self.half\_sample,2)\*50

X\_b = np.random.rand(self.half\_sample,2)\*50+50

X = np.concatenate((X\_f,X\_b),axis=0)

y = [-1]\*self.half\_sample+[1]\*self.half\_sample

return X, y

#画散点图并标明类别

def scatter(self,X,y):

plt.scatter(X[:self.half\_sample,0],X[:self.half\_sample,1],c='red',label='Class=-1')

plt.scatter(X[self.half\_sample:self.sample\_size,0],X[self.half\_sample:self.sample\_size,1],c='green',label='Class=1')

plt.xlabel('$X\_1$')

plt.ylabel('$X\_2$')

#在散点图上加上目标函数f

def target\_plot(self,X,y):

plt.subplot(1,2,1)

self.scatter(X,y)

plt.legend(['Class=-1','Class=1'],loc=2)

plt.title('Scatterplot with f')

xvalues = np.linspace(0,100,500)

plt.plot(xvalues,(100-xvalues))

# plt.show()

#在散点图上加上目标函数f和最终假设g

def g\_plot(self,X,y,w,b):

plt.subplot(1,2,2)

self.scatter(X,y)

plt.title('Scatterplot with f and g')

xvalues = np.linspace(0,100,500)

plt.plot(xvalues,(100-xvalues),label='f=$-100+X\_1+X\_2$')

plt.plot(xvalues,(-b/w[0,1]-w[0,0]/w[0,1]\*xvalues),label='g=$%d+%dX\_1+%dX\_2$'%((int)(b),(int)(w[0,0]),(int)(w[0,1])))

plt.legend(loc=2)

plt.show()

#训练感知机模型

def train(self,X, y):

w = np.random.rand(1,2)

b = np.random.rand()

iter\_number = 0

isWrong = True

while(isWrong):

wrong\_cnt = 0

random\_s = random.sample(range(len(X)),len(X))

for i in random\_s:

if y[i]\*(np.dot(w,X[i,])+b)<=0:

wrong\_cnt += 1

w += np.dot(y[i],X[i,])

b += y[i]

iter\_number+=1

if wrong\_cnt==0: isWrong = False

message = "Final hypothesis g is %.2f+%.2fX1+%.2fX2. Perceptron has updated the weights for %d times."%(b,w[0,0],w[0,1],iter\_number)

return w,b,message

#生成数据集、画图1、训练模型、画图2、输出说明

def main(self):

X, y = self.generate\_data()

plt.figure(figsize=(13, 6))

self.target\_plot(X, y)

w,b,m = self.train(X, y)

self.g\_plot(X,y,w,b)

print(m)

#主程序入口，遍历训练三种学习率，四种实例数下的感知机模型并作图

if \_\_name\_\_=="\_\_main\_\_":

print("We run 12 percepton demo with sample size = 20, 20, 100, 1000 and learning rate = 1, 0.01, 0.0001.")

i = 1;

for l\_rate in (1,0.01,0.0001):

for sample\_size in (20,20,100,1000):

print("\nScenario %d, we do a perceptron demo with sample size = %d and learning rate = %s."%(i,sample\_size,(str)(l\_rate)))

PerceptronDemo(sample\_size, l\_rate).main()

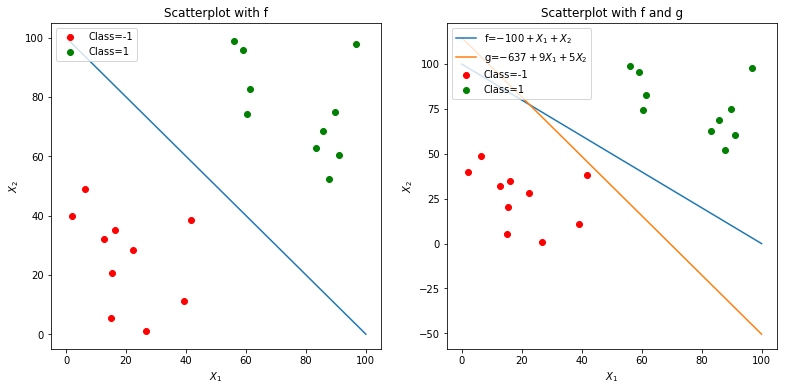
i+=1

四、实验结果

We run 12 percepton demo with sample size = 20, 20, 100, 1000 and learning rate = 1, 0.01, 0.0001.

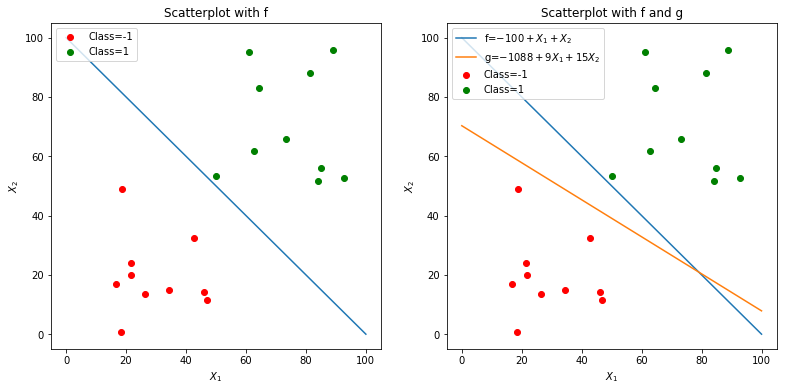
Scenario 1, we do a perceptron demo with sample size = 20 and learning rate = 1.

Final hypothesis g is -637.70+9.18X1+5.56X2. Perceptron has updated the weights for 2109 times.



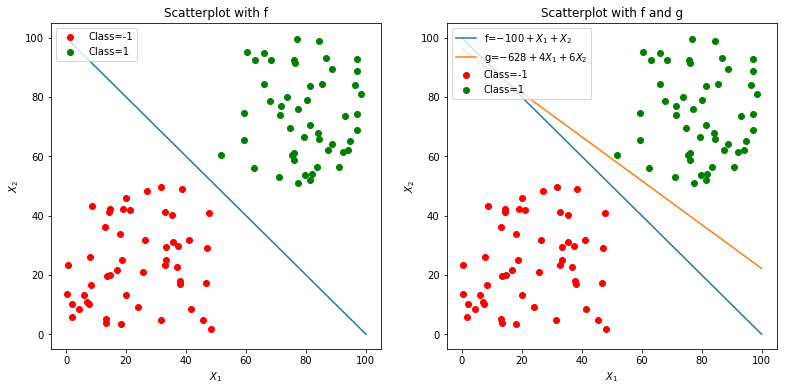
Scenario 2, we do a perceptron demo with sample size = 20 and learning rate = 1.

Final hypothesis g is -1088.96+9.67X1+15.49X2. Perceptron has updated the weights for 2592 times.



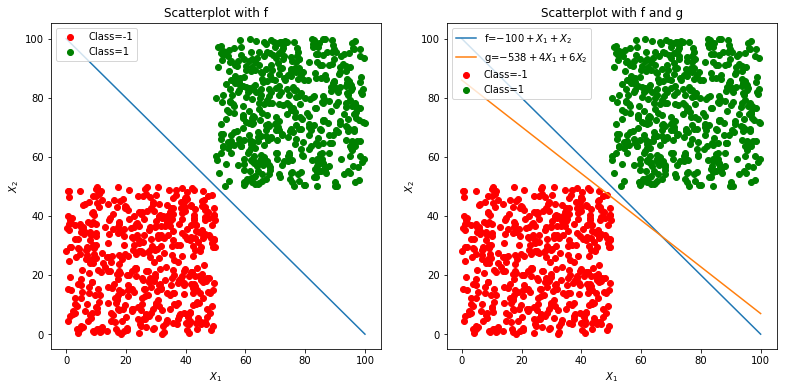
Scenario 3, we do a perceptron demo with sample size = 100 and learning rate = 1.

Final hypothesis g is -628.25+4.83X1+6.53X2. Perceptron has updated the weights for 1942 times.



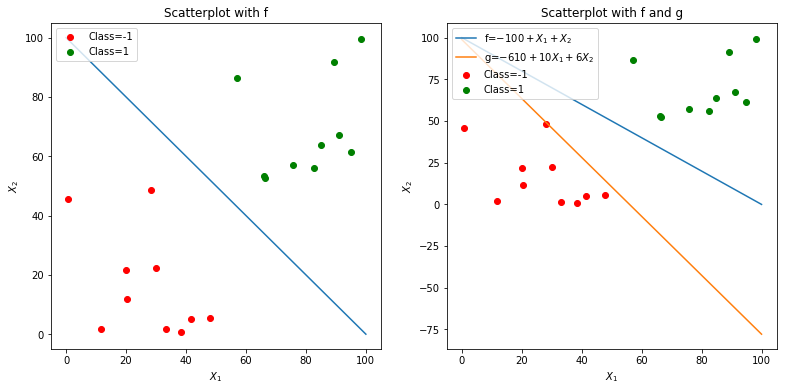
Scenario 4, we do a perceptron demo with sample size = 1000 and learning rate = 1.

Final hypothesis g is -538.67+4.95X1+6.27X2. Perceptron has updated the weights for 2628 times.



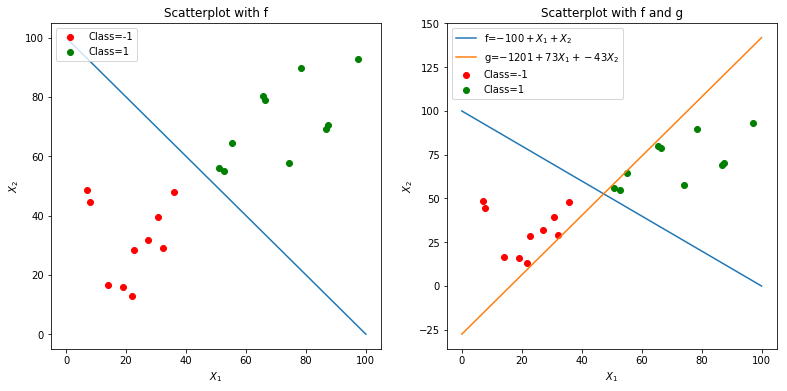
Scenario 5, we do a perceptron demo with sample size = 20 and learning rate = 0.01.

Final hypothesis g is -610.39+10.91X1+6.17X2. Perceptron has updated the weights for 1282 times.



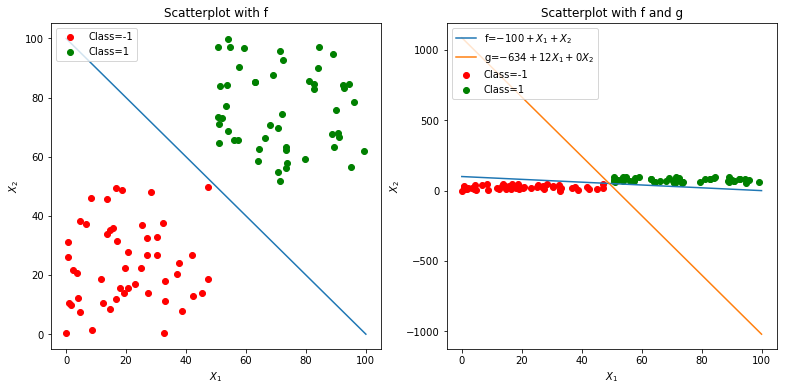
Scenario 6, we do a perceptron demo with sample size = 20 and learning rate = 0.01.

Final hypothesis g is -1201.69+74.00X1+-43.69X2. Perceptron has updated the weights for 1759 times.



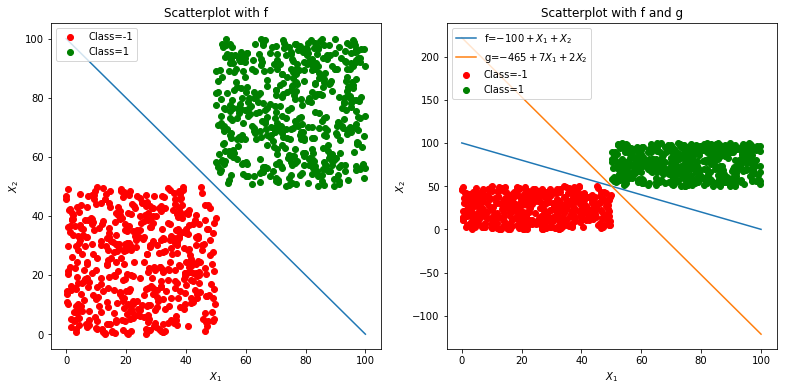
Scenario 7, we do a perceptron demo with sample size = 100 and learning rate = 0.01.

Final hypothesis g is -634.97+12.31X1+0.58X2. Perceptron has updated the weights for 1563 times.



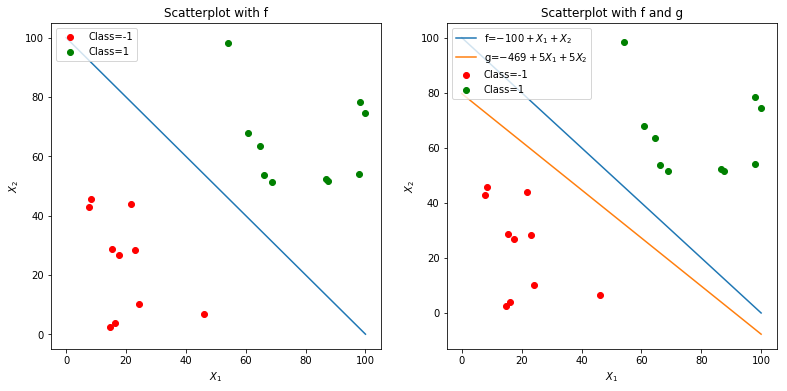
Scenario 8, we do a perceptron demo with sample size = 1000 and learning rate = 0.01.

Final hypothesis g is -465.95+7.21X1+2.10X2. Perceptron has updated the weights for 2051 times.



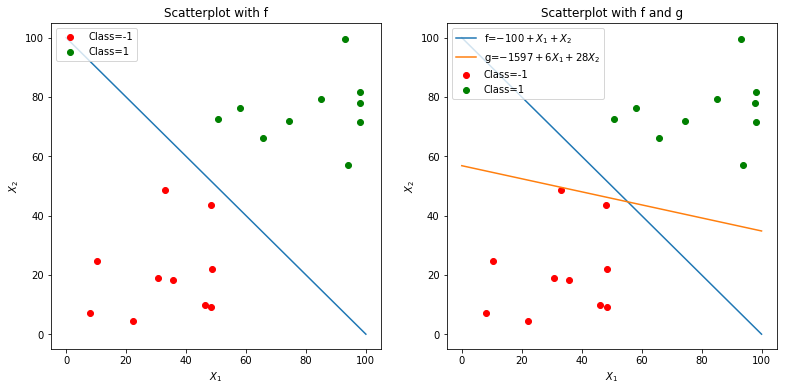
Scenario 9, we do a perceptron demo with sample size = 20 and learning rate = 0.0001.

Final hypothesis g is -469.43+5.15X1+5.89X2. Perceptron has updated the weights for 1671 times.

­­

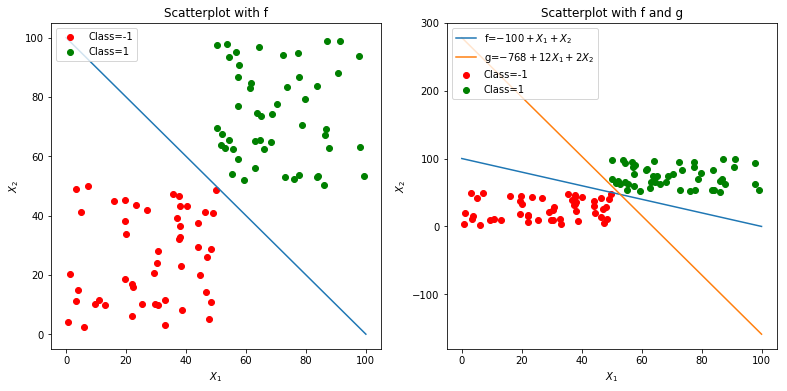
Scenario 10, we do a perceptron demo with sample size = 20 and learning rate = 0.0001.

Final hypothesis g is -1597.76+6.19X1+28.12X2. Perceptron has updated the weights for 2833 times.



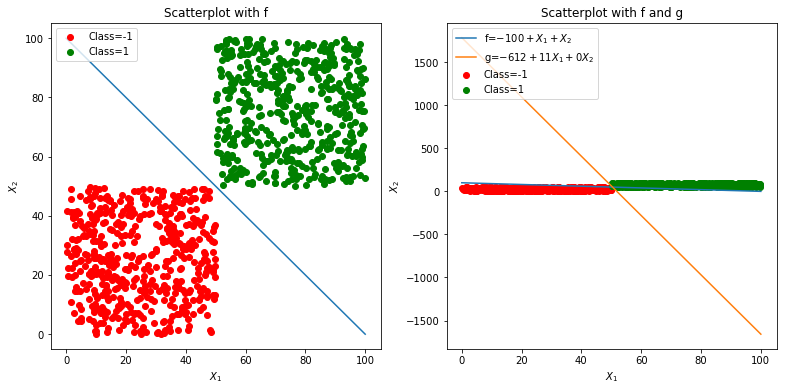
Scenario 11, we do a perceptron demo with sample size = 100 and learning rate = 0.0001.

Final hypothesis g is -768.13+12.07X1+2.76X2. Perceptron has updated the weights for 2143 times.



Scenario 12, we do a perceptron demo with sample size = 1000 and learning rate = 0.0001.

Final hypothesis g is -612.24+11.81X1+0.34X2. Perceptron has updated the weights for 3075 times.



五、结果分析

将上述12种情形下的实例数、学习率和感知机模型收敛前权重更新次数记录如下表：

表1 感知机收敛前参数更新次数

|  |  |  |  |
| --- | --- | --- | --- |
| 情形 | 实例数 | 学习率 | 感知机收敛前权重更新次数 |
| 1 | 20 | 1 | 2109 |
| 2 | 20 | 1 | 2592 |
| 3 | 100 | 1 | 1942 |
| 4 | 1000 | 1 | 2628 |
| 5 | 20 | 0.01 | 1282 |
| 6 | 20 | 0.01 | 1759 |
| 7 | 100 | 0.01 | 1563 |
| 8 | 1000 | 0.01 | 2051 |
| 9 | 20 | 0.0001 | 1671 |
| 10 | 20 | 0.0001 | 2833 |
| 11 | 100 | 0.0001 | 2143 |
| 12 | 1000 | 0.0001 | 3075 |

根据上表，下面按照三种学习率分组画出四种实例数下感知机收敛前权重更新次数。

实验发现在实例数较少（为20）时，感知机收敛所需更新次数不稳定，一次整体上低于实例数为100的情况，一次整体上明显高于实例数为100的情况，这可能与实例的随机生成有关；当实例数较多（为100和1000）时，可以看出随着实例数的增加，感知机需要更多的训练次数才能收敛。

随着学习率的增加，收敛所需更新次数先下降再上升。这说明在学习率为1时，损失函数出现了左右震荡的情形，增加了收敛所需次数；在学习率为0.001时，步长过小，模型需要过多权重更新次数才能收敛。

总结：在实例数较小时，感知机收敛前权重更新次数没有明显规律；整体上，感知机收敛前权重更新次数随着实例数的增加而增加，随着学习率的下降而先减小后增大。