**模式识别报告L2**

自卓2201 杨欣怡 U202215067

1，分别编写PLA算法和Pocket算法。

1）PLA算法

1. class PLA(object):
2. def \_\_init\_\_(self, dimension):
3. super(PLA, self).\_\_init\_\_()
4. self.dimension = dimension
5. self.W = np.zeros((1, dimension))
6. self.b = 0
7. def update\_param(self, x, y):
8. self.W += x \* y
9. self.b += y
10. def train(self, x, y):
11. num, dim = x.shape
12. if dim != self.dimension:
13. raise
14. optimized = False
15. while not optimized:
16. for i in range(num):
17. y\_estimated = np.dot(self.W, x[i]) + self.b
18. if y\_estimated \* y[i] <= 0:
19. self.update\_param(x[i], y[i])
20. break
21. if i == num - 1:
22. optimized = True
23. print('overtraining')
24. def inference(self, x, y):
25. num, dim = x.shape
26. if dim != self.dimension:
27. raise
28. y\_estimated = np.matmul(self.W, x.transpose(1, 0)) + self.b
29. y\_estimated = np.sign(y\_estimated).transpose(1, 0)
30. accuracy = 1 - len(np.nonzero(y\_estimated - y)[0]) / len(y)
31. print('PLA分类正确率: %.2f' % accuracy)
32. return y\_estimated

2）Pocket算法

1. class Pocket(object):
2. def \_\_init\_\_(self, dimension):
3. super(Pocket, self).\_\_init\_\_()
4. self.dimension = dimension
5. self.W = np.zeros((1, dimension))
6. self.b = 0
7. def \_error\_eval(self, w, b, x, y):
8. y\_estimated = np.matmul(w, x.transpose(1, 0)) + b
9. y\_estimated = np.sign(y\_estimated).transpose(1, 0)
10. a = np.nonzero(y\_estimated - y)
11. error\_idxs = np.nonzero(y\_estimated - y)[0]
12. return len(error\_idxs), error\_idxs
13. def train(self, x, y):
14. w = np.random.randn(1, 2)
15. b = np.random.randn(1)
16. for i in range(1000):
17. error\_num, error\_idxs = self.\_error\_eval(w, b, x, y)
18. if error\_num == 0:
19. break
20. else:
21. error\_idx = np.random.choice(error\_idxs)
22. error\_x = x[error\_idx]
23. error\_y = y[error\_idx]
24. w\_new = w + error\_y \* error\_x
25. b\_new = b + error\_y
26. error\_num\_new, error\_idxs\_new = self.\_error\_eval(w\_new, b\_new, x, y)
27. if error\_num\_new <= error\_num:
28. error\_num = error\_num\_new
29. error\_idxs = error\_idxs\_new
30. w = w\_new
31. b = b\_new
32. self.W = w
33. self.b = b
34. def inference(self, x, y):
35. num, dim = x.shape
36. if dim != self.dimension:
37. raise
38. y\_estimated = np.matmul(self.W, x.transpose(1, 0)) + self.b
39. y\_estimated = np.sign(y\_estimated).transpose(1, 0)
40. accuracy = 1 - len(np.nonzero(y\_estimated - y)[0]) / len(y)
41. print('Pocket分类正确率: %.2f' % accuracy)

2，（a）产生两个都具有200个二维向量的数据集和。数据集的样本来自均值向量协方差矩阵的正态分布，属于“+1”类，数据集的样本来自均值向量、协方差矩阵的正态分布，属于“-1”类，其中****是一个2\*2的单位矩阵。产生的数据中80%用于训练，20%用于测试。

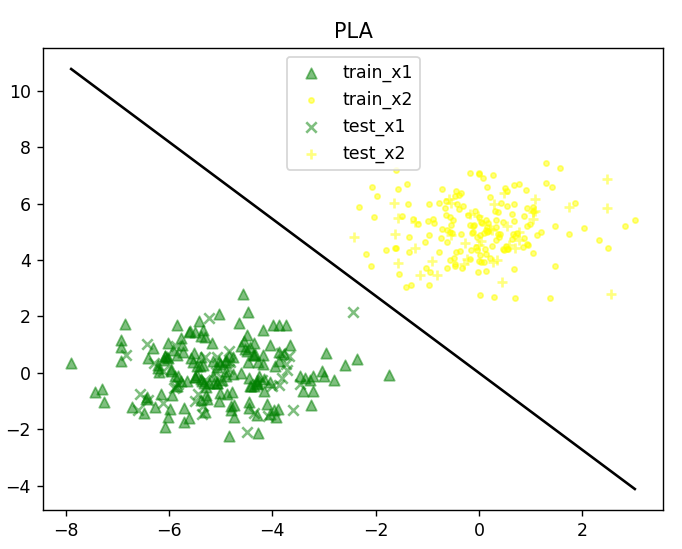
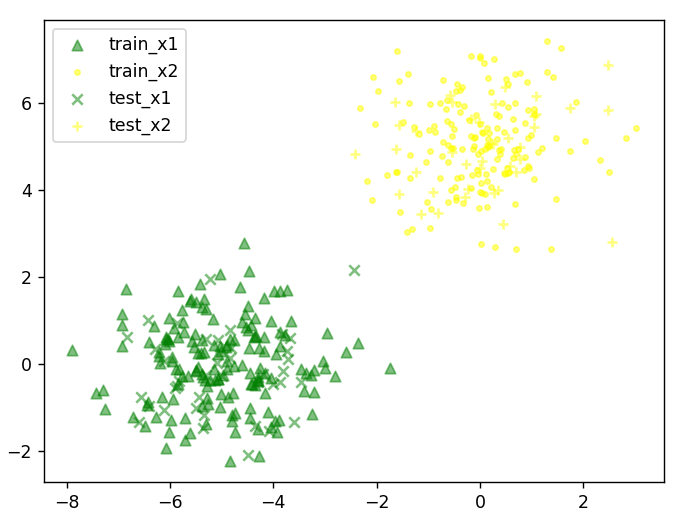
（b）在上述数据集上分别运用PLA算法和Pocket算法，利用产生的训练样本集得到分类面，算法中用到的各类超参数自定。

（c）分别在训练集和测试集上统计分类正确率。

（d）分别统计两个算法的运行时间

（e）画出数据集和分类面。

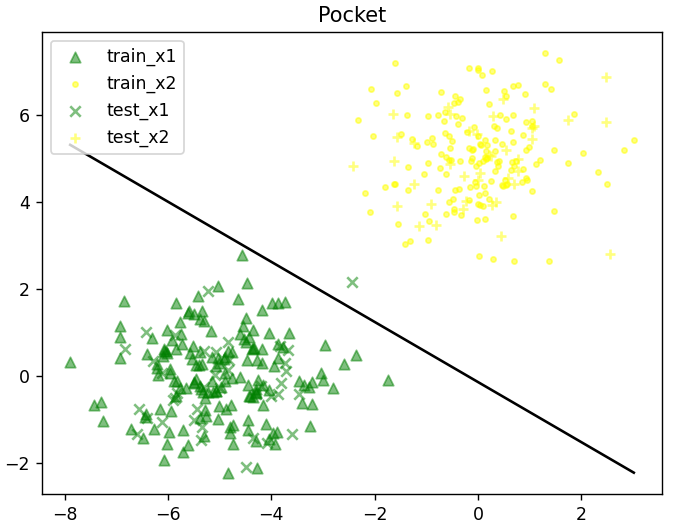
1. data2 = Dataset()
2. x\_train = data2.x\_train
3. y\_train = data2.y\_train
4. x\_test = data2.x\_test
5. y\_test = data2.y\_test
6. c1 = plt.scatter(x\_train[:160, 0], x\_train[:160, 1], alpha=0.5, marker='^', c='green')
7. c2 = plt.scatter(x\_train[160:, 0], x\_train[160:, 1], alpha=0.5, marker='.', c='yellow')
8. c3 = plt.scatter(x\_test[:40, 0], x\_test[:40, 1], alpha=0.5, marker='x', c='green')
9. c4 = plt.scatter(x\_test[40:, 0], x\_test[40:, 1], alpha=0.5, marker='+', c='yellow')
10. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1', 'train\_x2', 'test\_x1', 'test\_x2'], loc='best')
11. plt.show()



PLA算法生成分类面

分类正确率和运行时间：





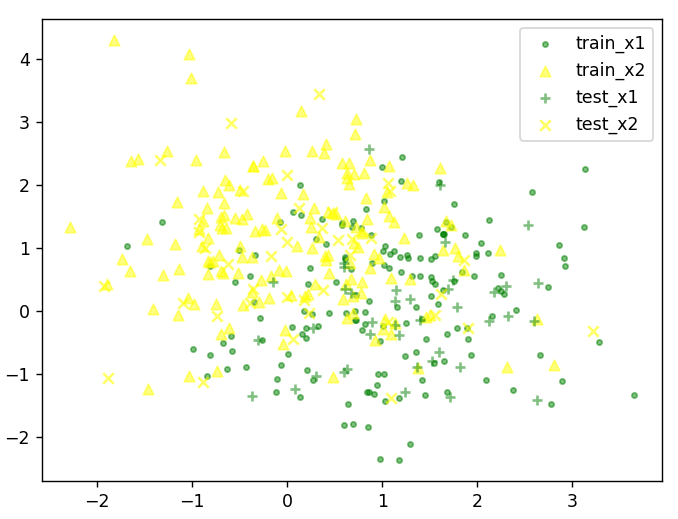
Pocket算法生成分类面

分类正确率和运行时间：



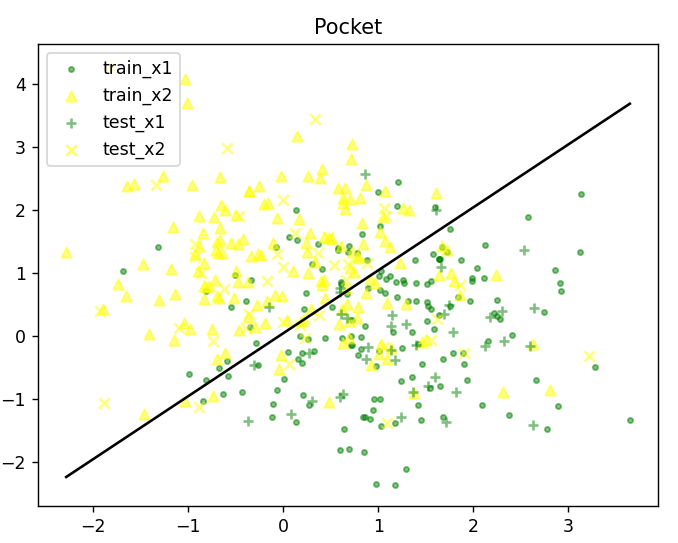
3，重复第2题的内容，但数据集和数据集的均值向量分别改为和，其他不变。

生成数据集：



由图可知生成的数据集线性不可分，PLA算法不收敛

Pocket算法：



分类正确率和运行时间：



结果讨论：

1. 样本分布的影响：

调整正负样本的均值时观察到：如果正负样本的均值点接近，PLA算法可能无法通过迭代过程找到一个合适的分类面来有效地分隔数据。在这种情况下，数据点之间的界限变得模糊，导致PLA算法难以实现其分类目标。此外，当样本均值接近时，使用Pocket算法的分类准确率也会受到影响，表现为准确度下降。

2. 样本数量的影响：

对于正负样本均值差异较小的数据分布，观察到在固定的迭代次数下，样本数量的减少会导致Pocket算法的运行时间减少。这是因为样本数量较少时，算法需要处理的数据量减少，从而加快了计算速度。同时，样本数量越少，PLA算法在有限的迭代次数内找到能够分隔数据的分类面的可能性就越大。这表明在样本数量较少的情况下，PLA算法更有可能在较少的迭代中达到分类的目的。