L1 homework

yfpeng

September 2022

1 算法

以下分别列举作业中想对重要的算法的实现, 详情见于我的 notebook

1.1 PLA 算法

```
#weight为一个3x1的行向量，每个point也是一个行向量
#以下是pla算法的实现
random_init=True
if random_init==True:
    weight=np.random.normal(0,1,3)
else:
    weight=np.array([2.,2.,2.])
def pla(weight,train_data,max_itera):
    for i in range(max_itera):
        error=False
        for j in range(train_data.shape[0]):
            if np.dot(weight,train_data[j][0:-1])*train_data[j][-1]<0:
                error=True
                print('分类面更新,更新前为:\n',weight)
                weight+=train_data[j][-1]*train_data[j][0:-1]
                print('更新后为\n',weight)
        if error==False:
            print('线性可分，找到分类面')
            return weight
        elif i==max_itera-1:
            error_rate=0
            for k in range(train_data.shape[0]):
                if np.dot(weight,train_data[k][0:-1])*train_data[j][-1]<0:
                    error_rate+=1
            print('未能找到分类面，正确率为：',1-error_rate/train_data.shape[0])
            return weight
```

图 1: PLA 算法实现

1.2 Pocket 算法

```
weight=np.random.normal(0,1,3)
#取出分类错误的数据
def get_error_set(weight,set):
    error_set=[]
    for j in range(set.shape[0]):
        if np.dot(weight,set[j][0:-1])*set[j][-1]<0:
            error_set.append(set[j])
    return np.array(error_set)
def pocket(train_data,weight,iteration):
    weight_hat=np.random.normal(0,1,3)
    for i in range(train_data.shape[0]):
        error_set=get_error_set(weight,train_data)
        if error_set.shape[0]==0:
            print('线性可分')
            return weight
        print('候选w参数更新，原为:\n',weight)
        weight+=error_set[0][0:-1]*error_set[0][-1]
        print('现为:\n',weight)
        if error_set.shape[0]<get_error_set(weight_hat,train_data).shape[0]:
            print('候选w取代最优w')
            weight_hat=weight
            print('在错误集合里面随机选择一个样本用于更新')
            rand_example=error_set[int(np.random.choice(range(error_set.shape[0]),1))]
            print('候选w参数更新，原为:\n',weight)
            weight+=rand_example[0:-1]*rand_example[-1]
            print('现为:\n',weight)
    return weight_hat
new_weight=pocket(train_set,weight,200)
```

图 2: Pocket 算法实现

2 算法运行和可视化

使用 np.random 模块产生随机数，然后用 random 模块打乱。对分类面的初始化，我同时考虑了 0 初始化和随机初始化（在正态分布中抽样）。

2.1 PLA 算法结果

以下是分类面的更新情况：

```
分类面更新,更新前为:
 [2. 2. 2.]
更新后为
 [ 3.         -3.39121705  2.10471403]
分类面更新,更新前为:
 [ 3.         -3.39121705  2.10471403]
更新后为
 [ 2.         -4.03918484 -4.84238461]
线性可分，找到分类面
```

图 3: pla 结果

以下分别是在训练集和测试集上的结果:

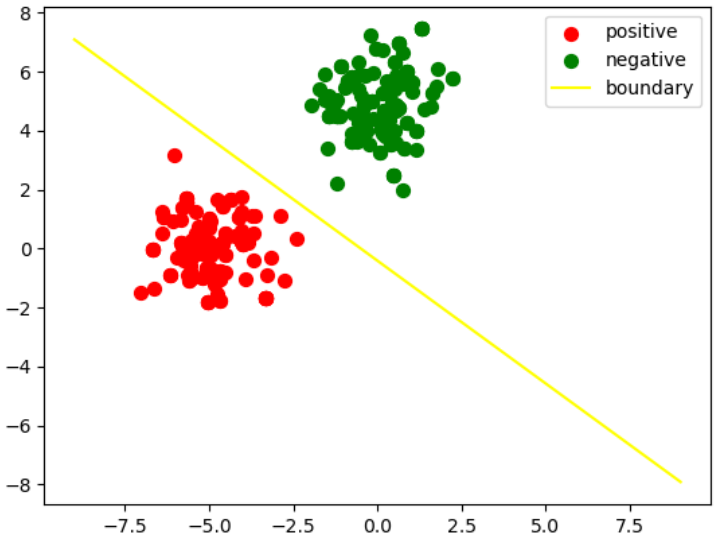

图 4: PLA 在训练集上的结果

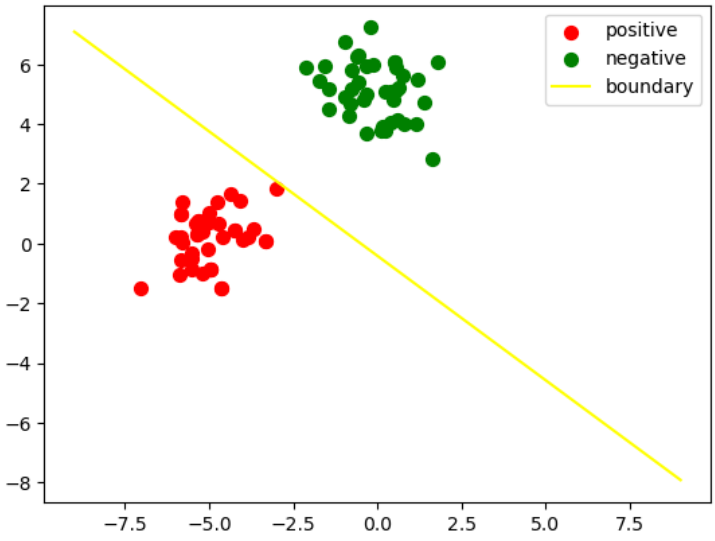

图 5: PLA 在测试集上的结果

在训练集和测试集的准确率均为 100% ，运行时间为 0.3s

2.2 Pocket 算法结果

以下是分类面的更新情况：

```
候选w参数更新，原为:
 [ 1.26056885 -0.1003935  -0.40034882]
现为:
 [ 0.26056885  1.0945933  -2.59814602]
候选w取代最优w
在错误集合里面随机选择一个样本用于更新
候选w参数更新，原为:
 [ 0.26056885  1.0945933  -2.59814602]
现为:
 [-0.73943115  2.28958011 -4.79594322]
候选w参数更新，原为:
 [-0.73943115  2.28958011 -4.79594322]
现为:
 [ 0.26056885 -3.10163695 -4.69122919]
线性可分
```

图 6: pocket 结果

以下分别是在训练集和测试集上的结果：

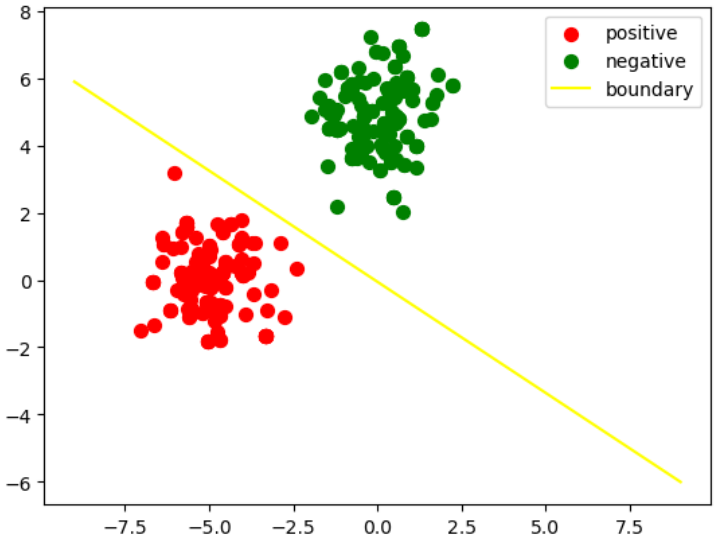

图 7: Pocket 在训练集上的结果

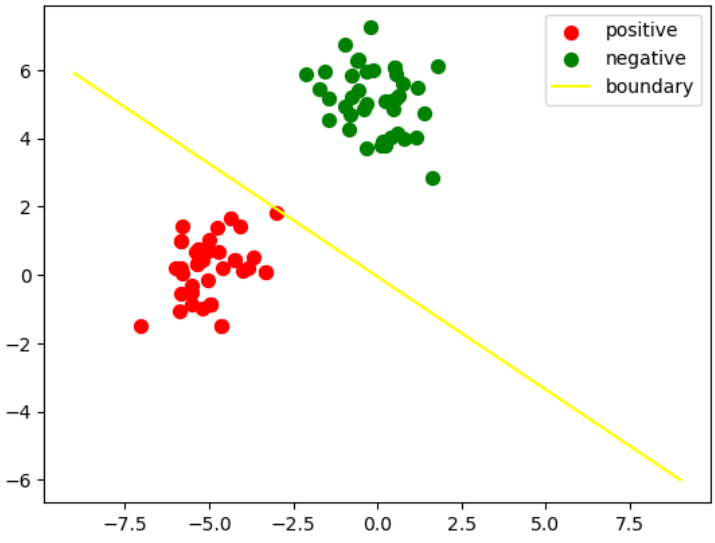

图 8: Pocket 在测试集上的结果

在训练集和测试集的准确率均为 100% ，运行时间为 0.8s

3 改变均值之后

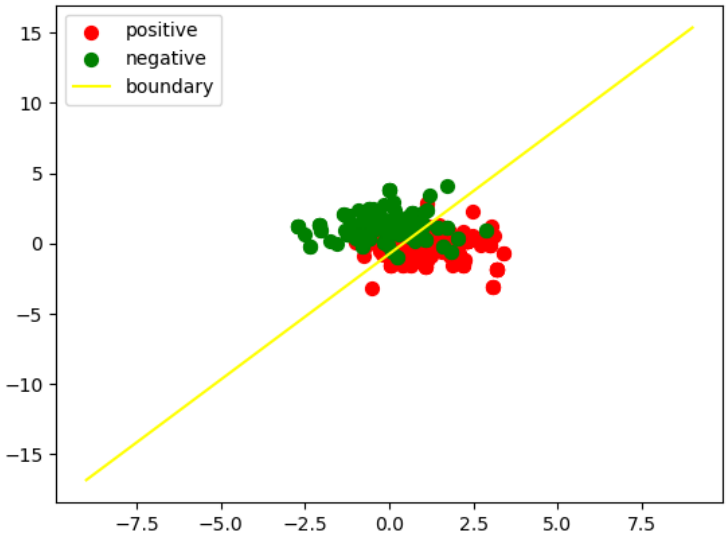

图 9: pla 算法在均值改变之后的训练集结果

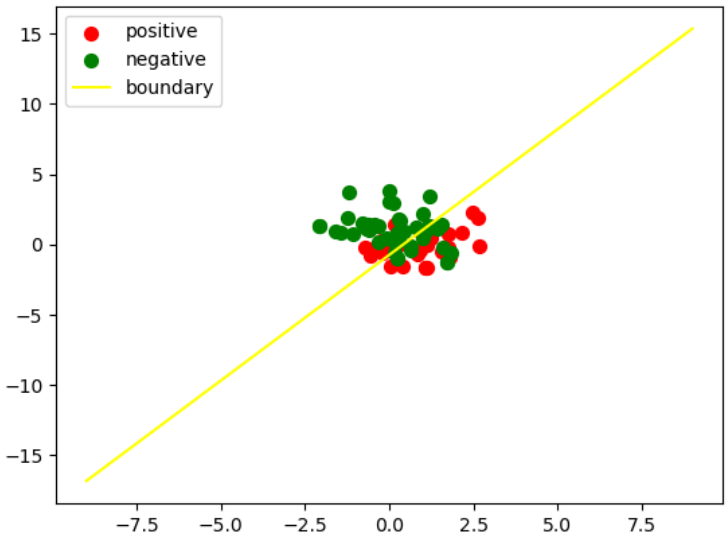

图 10: pla 算法在均值改变之后的测试集结果

这时，pla 算法在训练集和测试集准确率分别为 82.5% 和 70.0%，运行时间为 10.7s

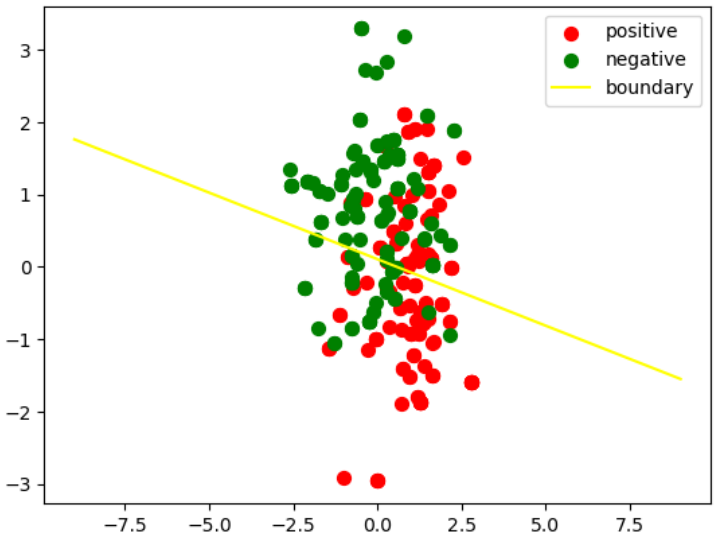

图 11: Pocket 算法在均值改变之后的训练集结果

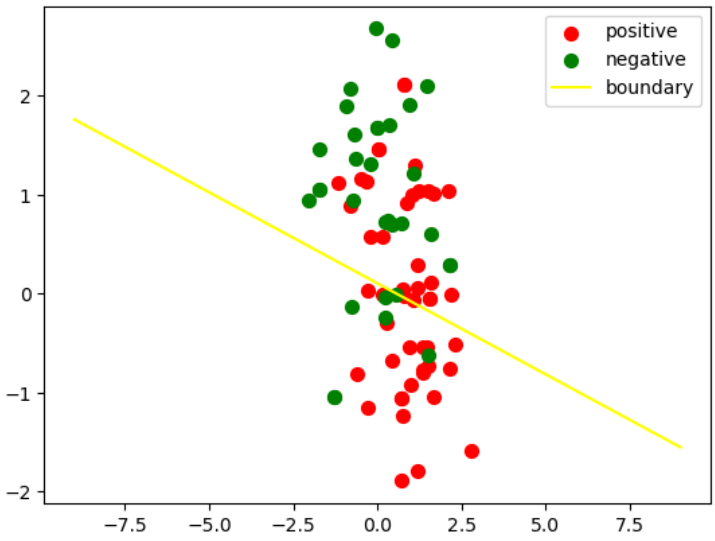

图 12: Pocket 算法在均值改变之后的测试集结果

这时，pla 算法在训练集和测试集准确率分别为 72.8% 和 72.5%，运行时间为 1.4s。由此可见，当数据本身分不开的时候，两个算法的能力都是很有限的。其中，pocket 相对于 pla 算法，效率和表现都更好。

4 改变训练轮数

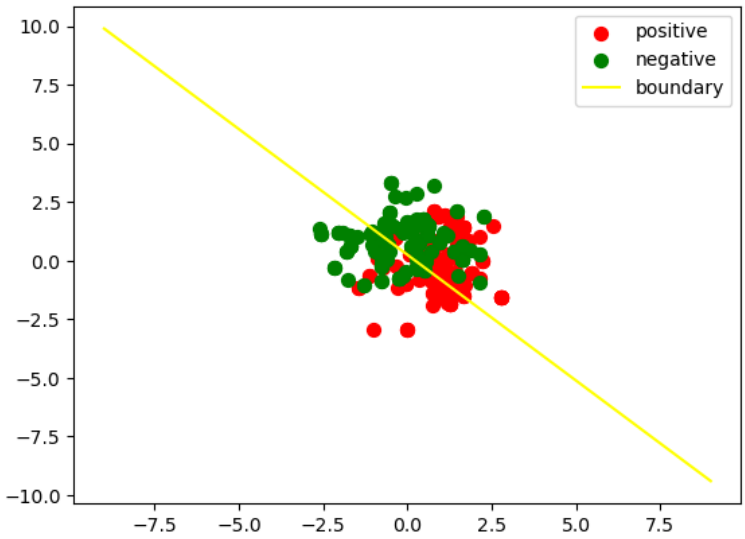

图 13: Pocket 算法在轮数改变之后的训练集结果

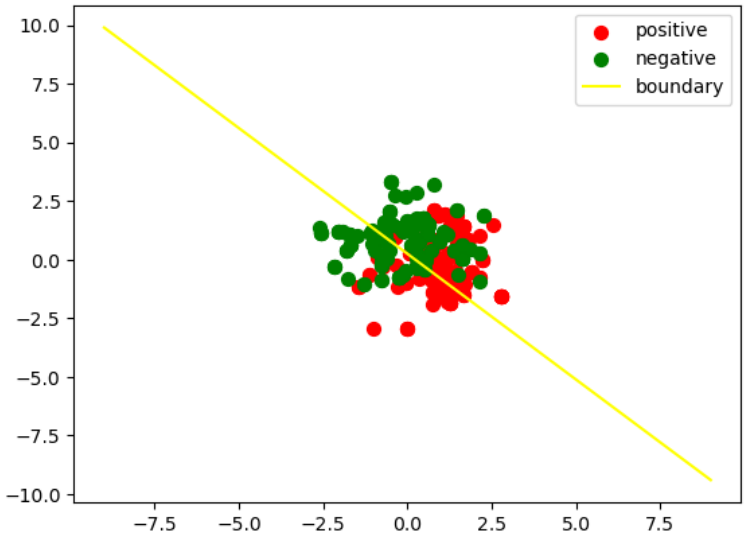

图 14: Pocket 算法在轮数改变之后的测试集结果

可见，并不是训练论述越多越好，但这也不是过拟合。

5 改变样本数量

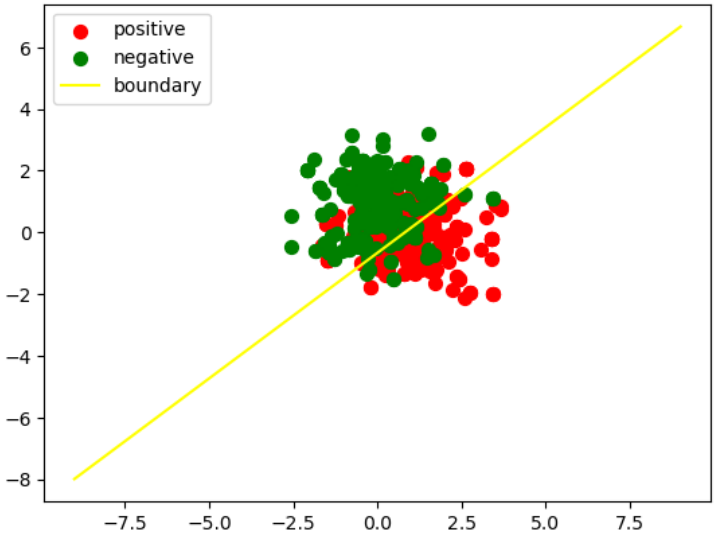

图 15: Pocket 算法在样本量改变之后的训练集结果

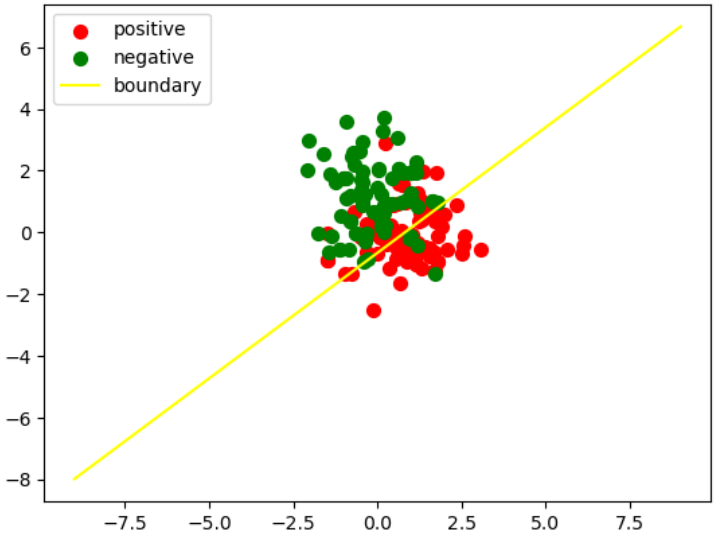

图 16: Pocket 算法在样本量改变之后的测试集结果

样本数量的提升并没有带来表现得过多提升，我认为这是样本本身的多样性就很有限造成的。

6 改变样本分布

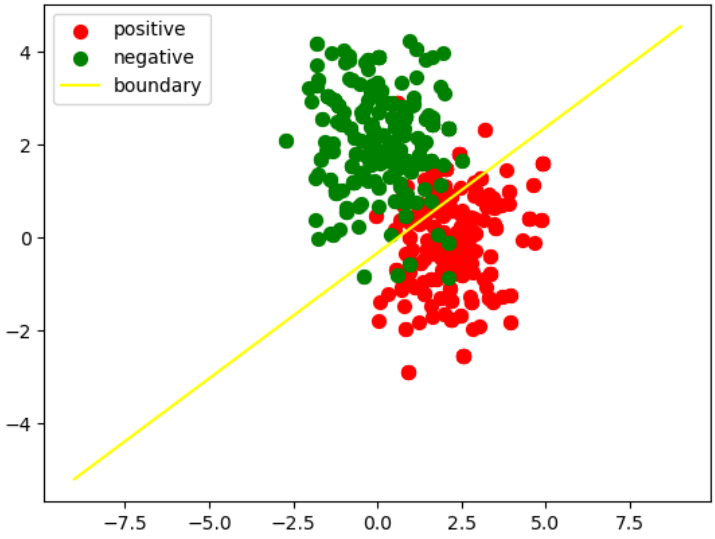

图 17: Pocket 算法在分布改变之后的训练集结果

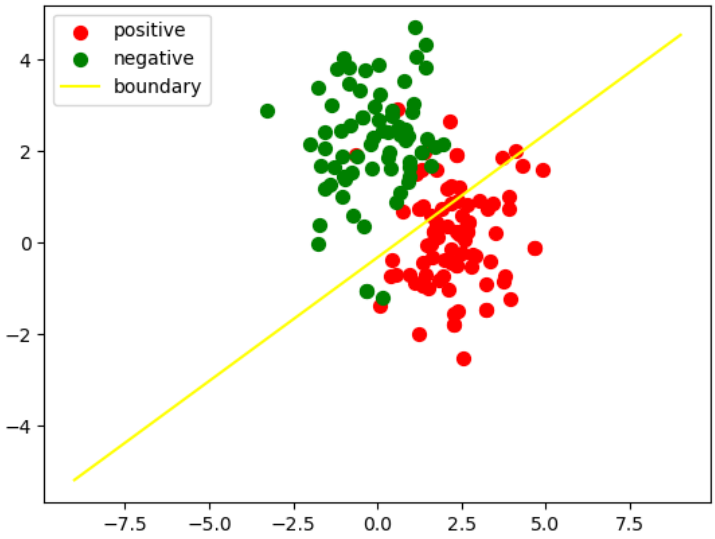

图 18: Pocket 算法在分布改变之后的测试集结果

这个分布相对于上一个分布，两个类别要分的更开，因此，表现的提升是可以理解的。