L8,9 homework

yfpeng

October 2022

1 算法实现

根据 PPT 中的公式，我们直接把它变成代码，实现如下：

```python
#以下是Primal-SVM求解
from cvxopt import solvers,matrix
def primal_svm(train_set):
    d=train_set.shape[-1]-2
    #二次规划函数的参数的定义似乎与ppt里面不太一样
    Q=np.eye(d+1,dtype=float)
    Q[0,0]=0.0
    P=np.zeros((d+1,1))
    A=-train_set[:,-1].reshape(-1,1)*train_set[:,:-1]
    C=-np.ones((train_set.shape[0],1),dtype=float)
    #转换成cvxopt需要的格式
    Q=matrix(Q)
    P=matrix(P)
    A=matrix(A)
    C=matrix(C)
    #求解
    solvers.options['show_progress'] = True
    sol = solvers.qp(Q,P,A,C)
    return sol
```

✓ 0.5s

图 1: primal svm 算法实现

```python
#以下是dual_SVM的实现
def dual_svm(train_set):
    Z=train_set[:,1:-1]
    Y=train_set[:,-1].reshape(-1,1)
    Q1=Z@np.transpose(Z)
    Q2=Y@np.transpose(Y)
    Q=Q1*Q2
    P=-np.ones((train_set.shape[0],1),dtype=float)
    A=-np.eye(train_set.shape[0],dtype=float)
    C=np.zeros((train_set.shape[0],1),dtype=float)
    R=np.transpose(Y)
    V=np.zeros((1,1),dtype=float)
    #格式转换，与cvxopt匹配
    Q=matrix(Q)
    P=matrix(P)
    A=matrix(A)
    C=matrix(C)
    R=matrix(R)
    V=matrix(V)

    sol=solvers.qp(Q, P, A, C, R, V)
    alpha=np.array(sol['x']).reshape(-1,1)
    alpha=np.absolute(alpha)
    alpha=alpha*(alpha>1e-8)
    #求w
    sv_idx=np.where(alpha>1e-8)[0]
    w=np.sum(alpha*Y*Z,axis=0).reshape(1,-1)
    #求b
    idx0=sv_idx[0]
    sv=Z[idx0,:].reshape(-1,1)
    yv=Y[idx0,0].reshape(-1,1)
    b=yv-np.dot(w,sv)
    #将b与w拼接
    w=np.concatenate((b,w),-1)
    w=np.squeeze(w)
    return w,sv_idx
```

图 2: dual svm 算法实现

以下是计算核函数的代码

```python
#以下是kernel_svm的实现
def kernel_value(x1,x2,kernel_type):
    n, dimension1 = x1.shape
    m, dimension2 = x2.shape
    if kernel_type=='poly':
        kernel_value = np.power(1+x2@np.transpose(x1),4)
    elif kernel_type=='gauss':
        x1x1=np.power(np.linalg.norm(x1,axis=-1).reshape(1,-1),2)
        x2x2=np.power(np.linalg.norm(x2,axis=-1).reshape(-1,1),2)
        x1x2=x2@np.transpose(x1)
        kernel_value=np.exp(2*x1x2-x1x1-x2x2)
    return kernel_value
```

图 3: 计算核函数

以下是 kernel svm 实现的代码

```python
#以下是核函数支撑向量机的求解
def kernel_svm(train_set,kernel_type):
    ker=kernel_value(train_set[:,1:-1],train_set[:,1:-1],kernel_type)
    yy=train_set[:,-1].reshape(-1,1)@train_set[:,-1].reshape(1,-1)
    Q=ker*yy
    P=-np.ones((train_set.shape[0],1))
    A=-np.eye(train_set.shape[0])
    R=train_set[:,-1].reshape(1,-1)
    C=np.zeros((train_set.shape[0],1))
    V=np.zeros((1,1))
    #格式转换，与cvxopt匹配
    Q=matrix(Q)
    P=matrix(P)
    A=matrix(A)
    C=matrix(C)
    R=matrix(R)
    V=matrix(V)

    #以下分别求出了支撑向量和对应的值
    sol=solvers.qp(Q, P, A, C, R, V)
    alpha=np.array(sol['x']).reshape(-1,1)
    sv_idx=np.where(alpha>1e-6)[0]
    sv_alpha=alpha[sv_idx,:].reshape(-1,1)
    sv_label=train_set[sv_idx,-1].reshape(-1,1)
    sv_points=train_set[sv_idx,1:-1]

    #求b
    xm=sv_points[0,:].reshape(1,-1)
    ym=sv_label[0,:].reshape(1,-1)
    b=ym-np.sum(sv_alpha*sv_label*kernel_value(xm,sv_points,kernel_type))

    return sv_alpha,sv_idx,sv_label,sv_points,b
```

图 4: kernel svm 算法实现

2 可视化

2.1 primal svm

由下面的结果，primal svm 可以做到分类完全正确

```
训练集:
正确率为:100.0%
测试集
正确率为:100.0%
```

图 5: 正确率评估

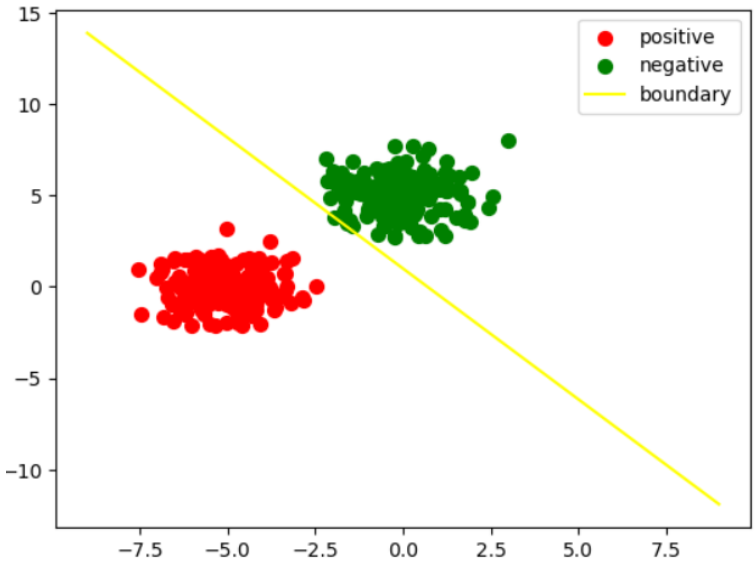

图 6: 训练集可视化

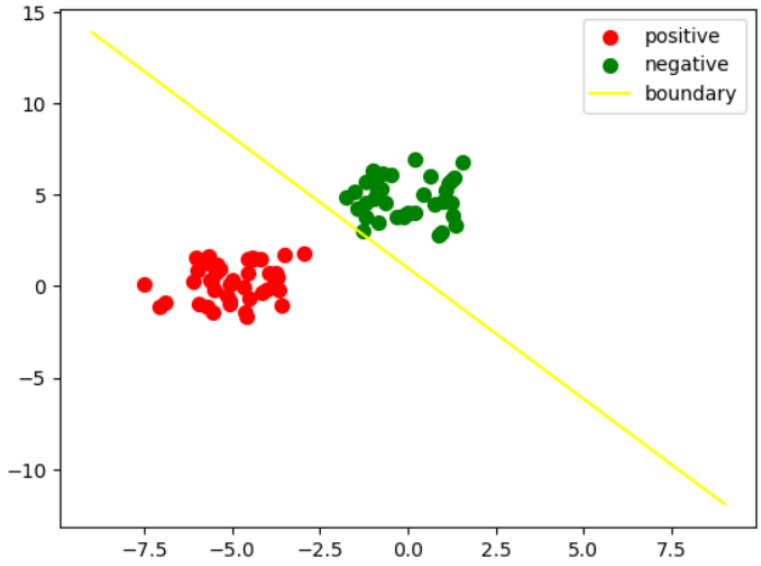

图 7: 测试集可视化

2.2 dual svm

以下展示了求出来的支撑向量对应的下标

```
     pcost       dcost       gap    pres   dres
 0: -2.4886e+01 -4.4334e+01  1e+03  4e+01  2e+00
 1: -2.6417e+01 -9.9154e+00  2e+02  7e+00  4e-01
 2: -7.2052e+00 -1.2300e+00  9e+01  2e+00  1e-01
 3: -3.9560e-01 -4.9294e-01  2e+00  3e-02  2e-03
 4: -3.3465e-01 -4.0663e-01  2e-01  4e-03  2e-04
 5: -3.9647e-01 -4.0335e-01  1e-02  1e-04  7e-06
 6: -4.0288e-01 -4.0302e-01  2e-04  2e-06  1e-07
 7: -4.0301e-01 -4.0301e-01  3e-06  2e-08  1e-09
 8: -4.0301e-01 -4.0301e-01  3e-08  2e-10  1e-11
Optimal solution found.
支撑向量对应下标为:  [192 228 292]
```

图 8: 二次规划求解过程

```
训练集:
正确率为:100.0%
测试集
正确率为:100.0%
```

图 9: 正确率评估

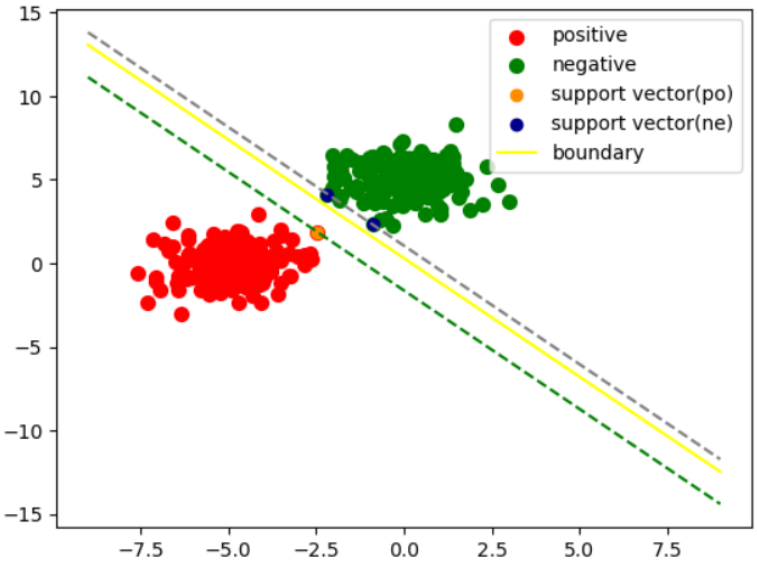

图 10: 训练集可视化

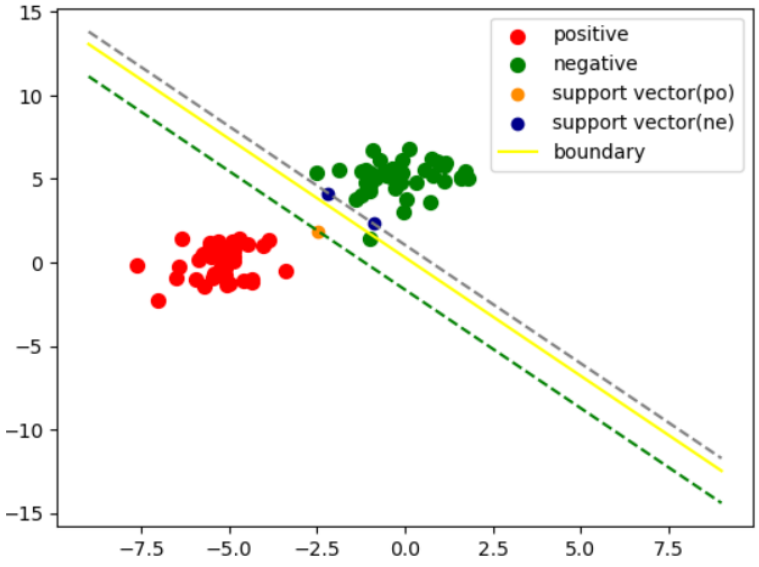

图 11: 测试集可视化

由上图，我们发现，并不是所有的在间隔面上的点都是支撑向量，只有其中的一部分才是支撑向量

2.3 kernel svm poly

以下展示的是核函数版本的支撑向量，核函数采用四次多项式

```
     pcost       dcost       gap    pres   dres
 0: -8.5643e+00 -1.5495e+01  7e+02  2e+01  2e+00
 1: -9.5565e+00 -4.3065e+00  2e+02  5e+00  4e-01
 2: -2.9333e+00 -3.0266e-01  1e+01  5e-01  4e-02
 3: -6.3636e-01 -9.5669e-02  2e+00  8e-02  7e-03
 4: -1.3151e-01 -3.6255e-02  5e-01  1e-02  1e-03
 5: -4.4907e-02 -1.0973e-02  2e-01  6e-03  5e-04
 6: -1.6931e-02 -3.9445e-03  6e-02  2e-03  1e-04
 7: -5.8069e-03 -2.7709e-03  2e-02  4e-04  4e-05
 8: -2.8146e-03 -2.2993e-03  1e-02  2e-04  2e-05
 9: -2.1535e-03 -1.8647e-03  7e-03  1e-04  1e-05
10: -1.8223e-03 -1.6026e-03  5e-03  8e-05  7e-06
11: -1.6878e-03 -1.1442e-03  3e-03  4e-05  3e-06
12: -1.3910e-03 -7.3050e-04  2e-03  2e-05  2e-06
13: -9.8002e-04 -4.9713e-04  9e-04  8e-06  7e-07
14: -7.2584e-04 -4.2829e-04  6e-04  5e-06  4e-07
15: -4.3999e-04 -3.9370e-04  1e-04  9e-07  8e-08
16: -3.9214e-04 -3.9122e-04  2e-06  1e-08  1e-09
17: -3.9117e-04 -3.9116e-04  3e-08  2e-10  2e-11
Optimal solution found.
[ 91 133 206]
```

图 12: 二次规划求解过程

```
训练集：
正确率为:100.0%
测试集：
正确率为:100.0%
```

图 13: 正确率评估

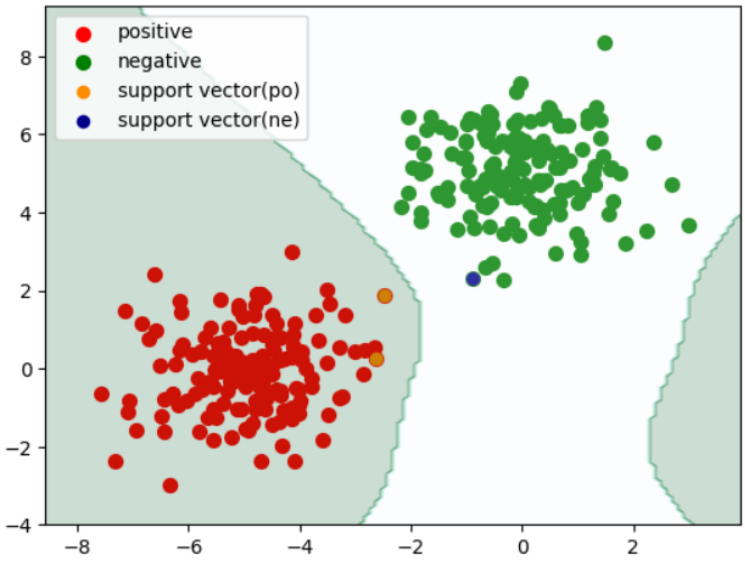

图 14: 训练集可视化

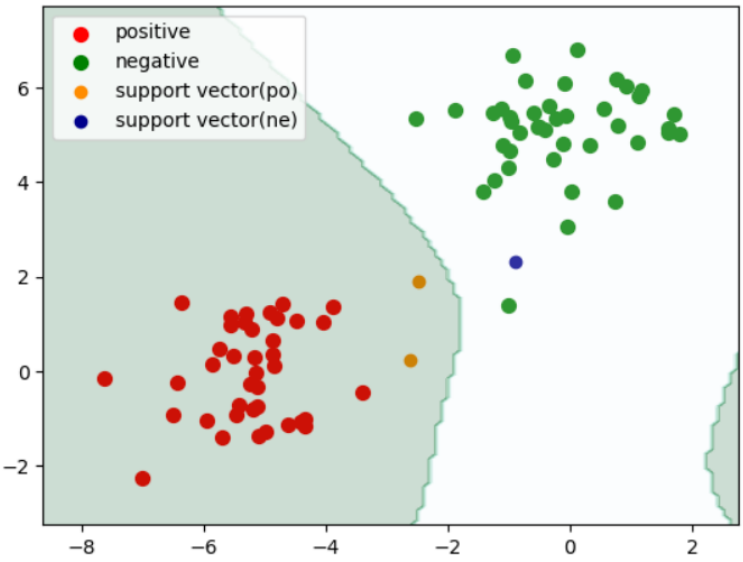

图 15: 测试集可视化

可以看出，核函数版本的 svm 有了对非线性建模的能力

2.4 kernel svm gauss

使用高斯核函数，由于它的维度是无限维，因此对非线性的建模能力更强，但是它会增加支撑向量的数量

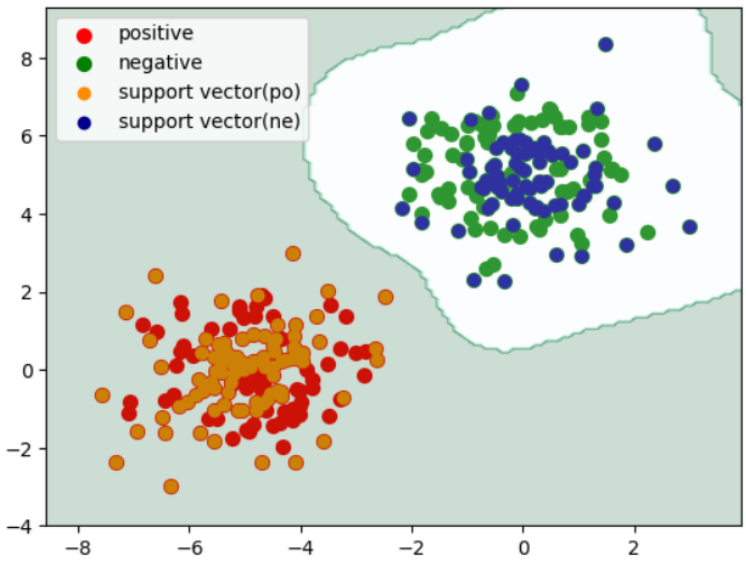

图 16: 训练集可视化

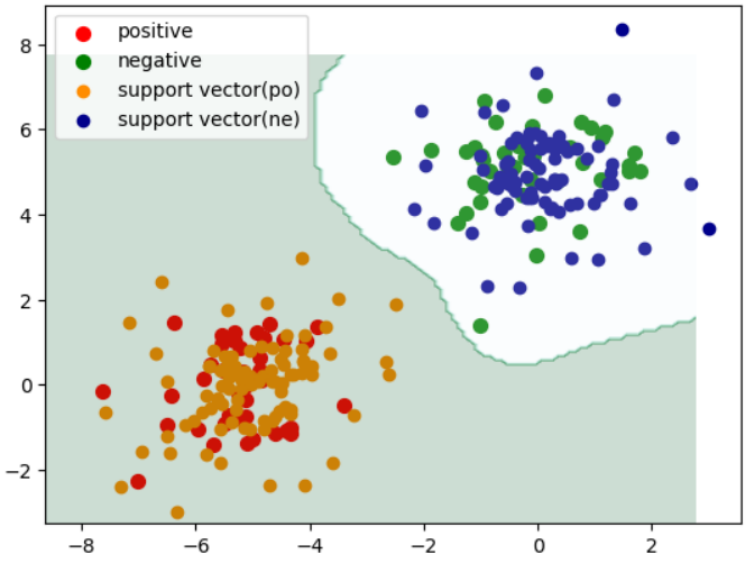
positive
negative
support vector(po)
support vector(ne)
8
6
4
2
0
−2
−8
−6
−4
−2
0
2

图 17: 测试集可视化

3 改变数据集分布

3.1 线性的分类面

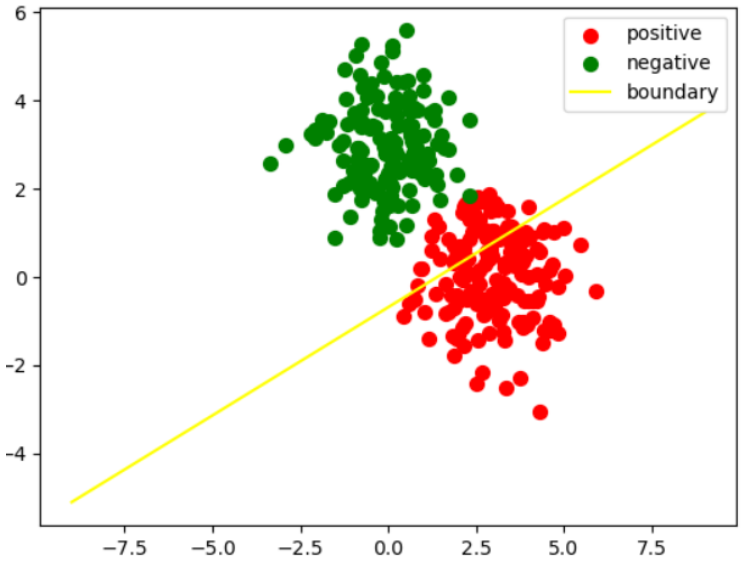

图 18: primal 训练集可视化

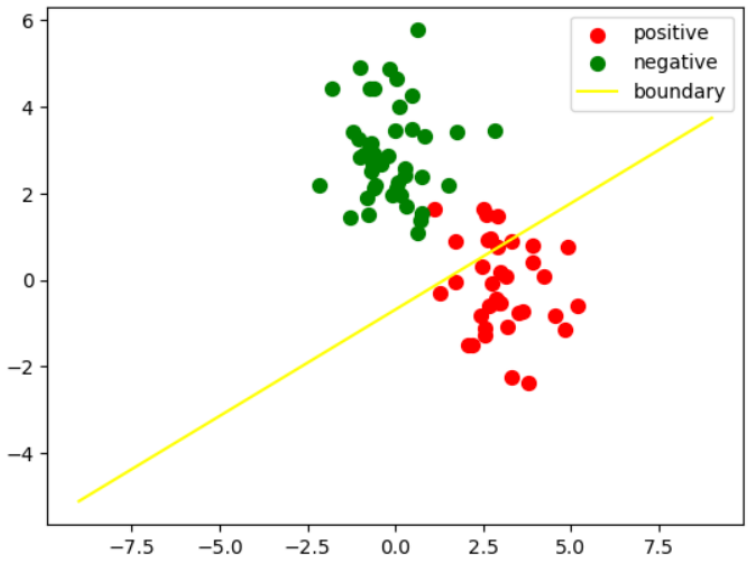

图 19: primal 测试集可视化

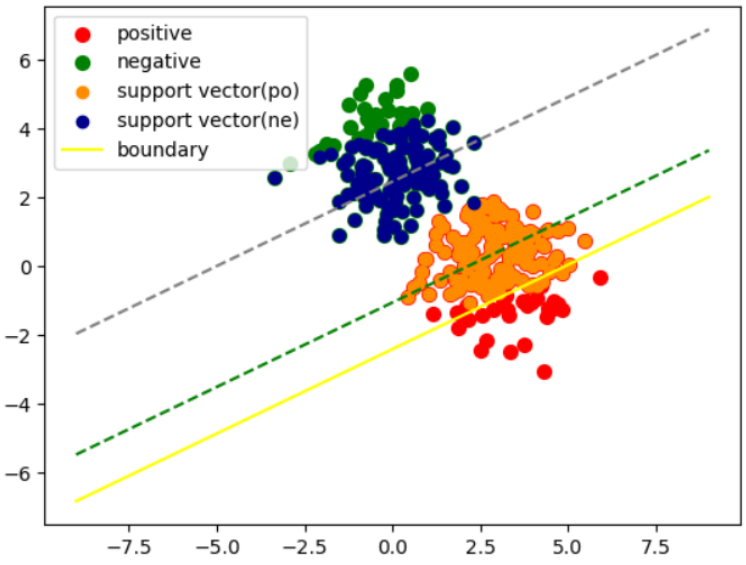

图 20: dual 训练集可视化

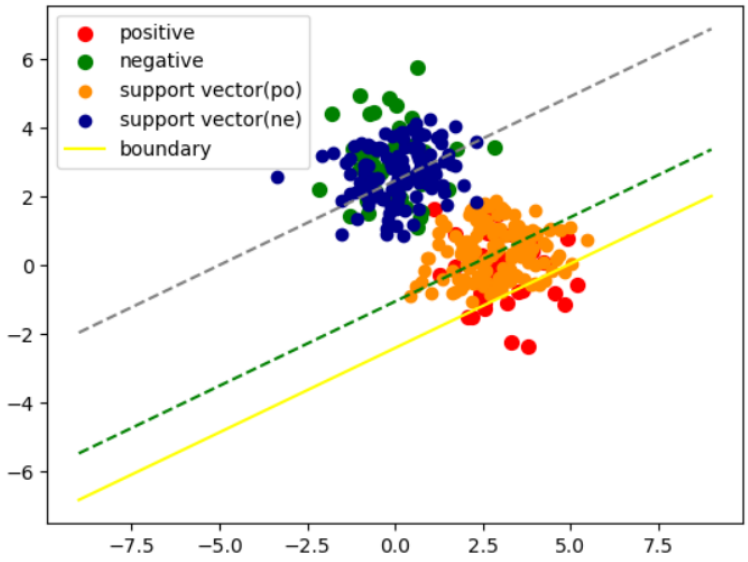

图 21: dual 测试集可视化

可见，这种线性的分类方法已经有点吃力了，表现显著下降

3.2 非线性分类面

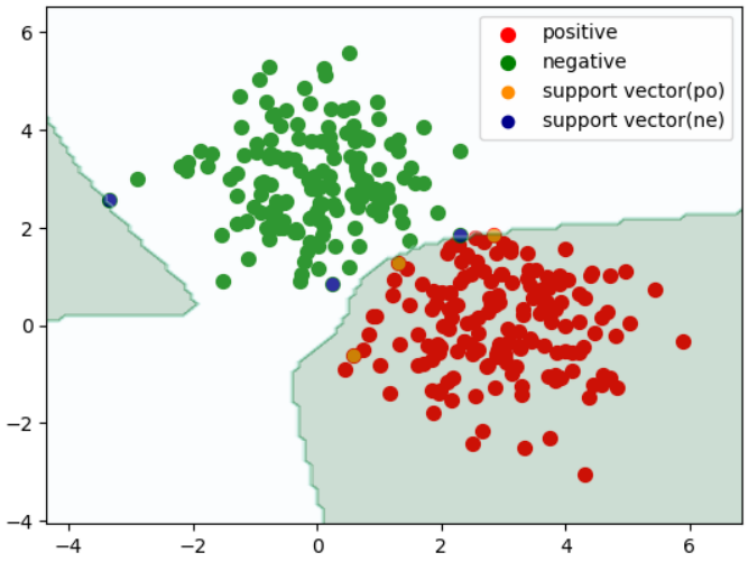

图 22: poly 训练集可视化

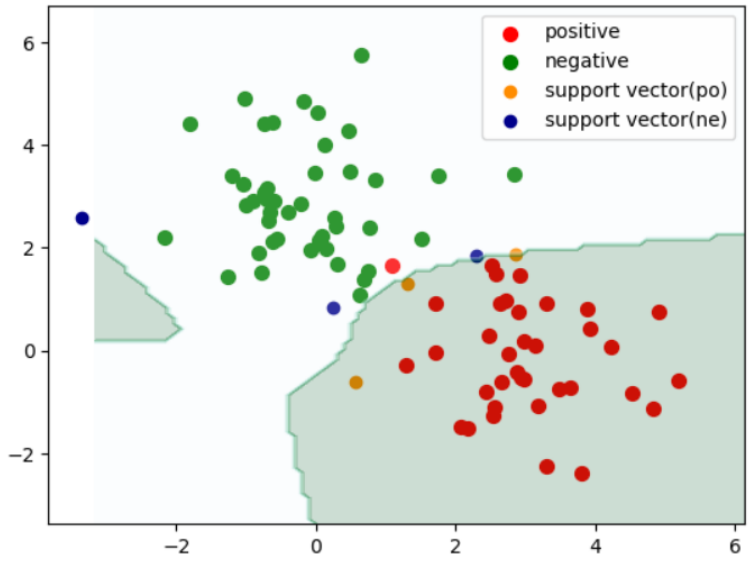

图 23: poly 测试集可视化

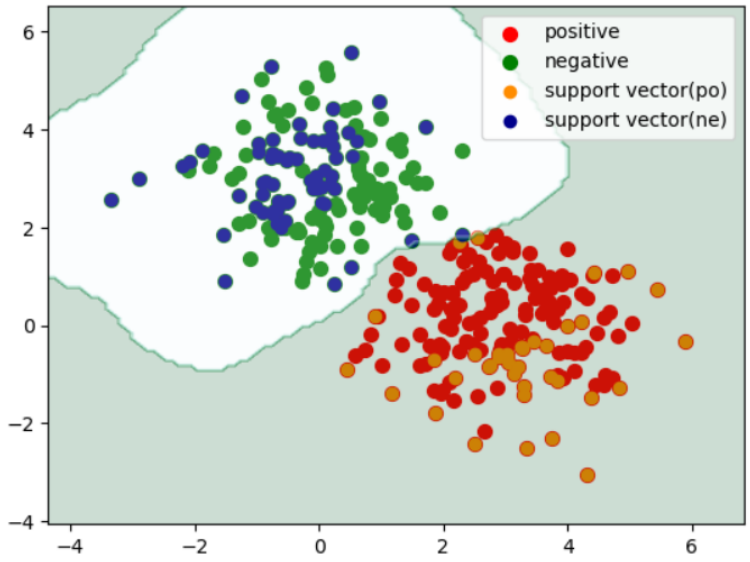

图 24: gauss 训练集可视化

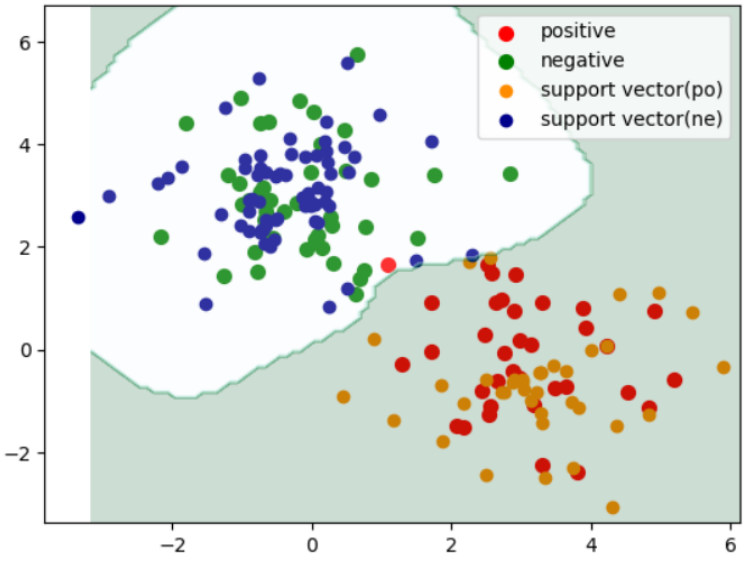

图 25: gauss 测试集可视化

这个时候，非线性变换（通过核函数实现）的优势就体现出来了。因此，若数据集非线性可分时，非线性变换是必要的

4 应用

由于海岸边界是弯曲的，因此这里采用非线性的高斯核函数的分类器：

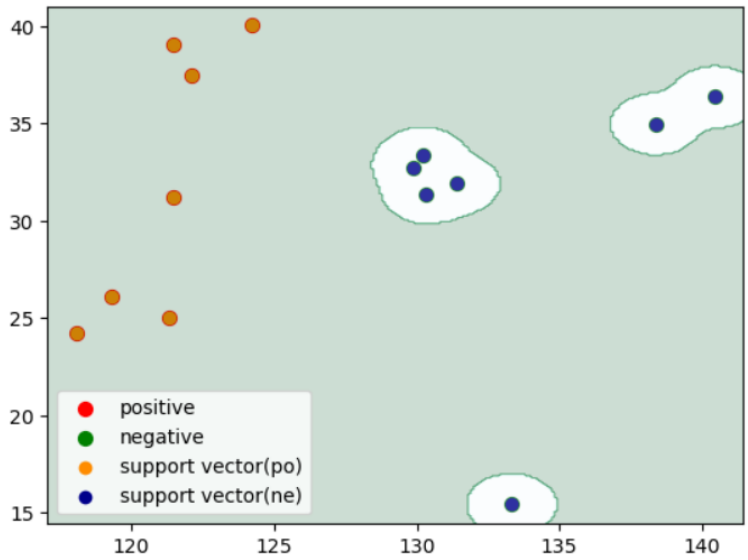

图 26: 训练集可视化

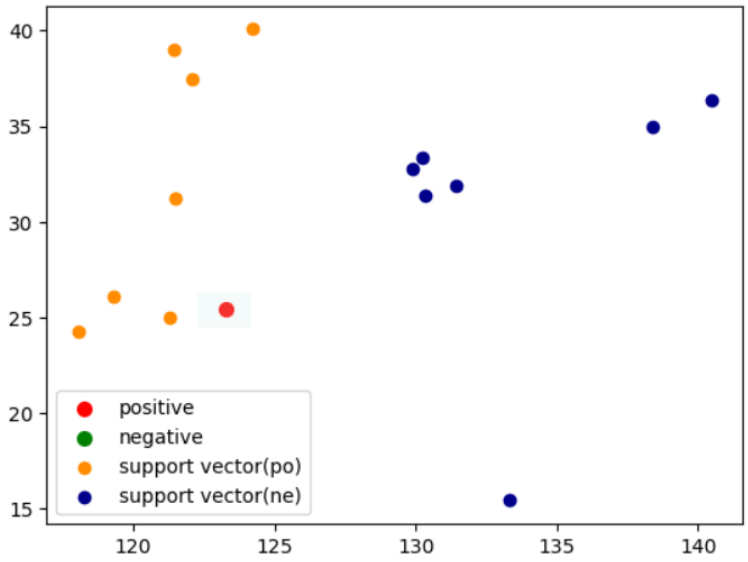

图 27: 测试集可视化

结果为，钓鱼岛属于中国

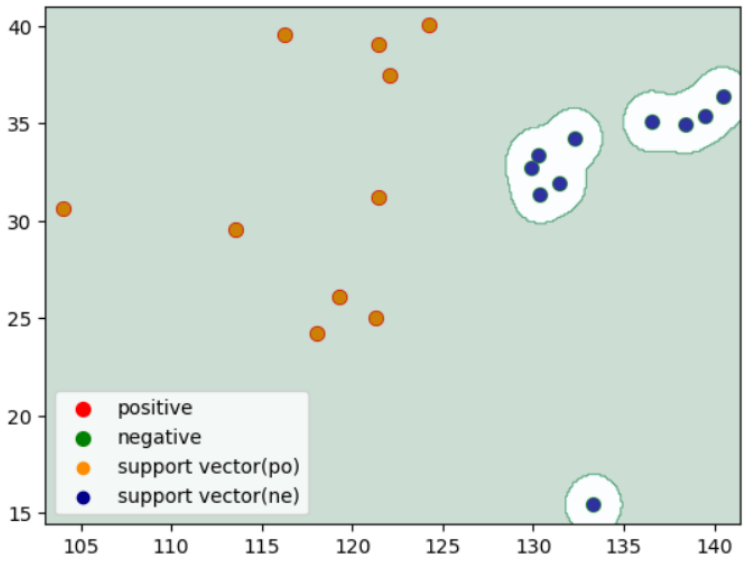

图 28: 增加样本后，训练集可视化

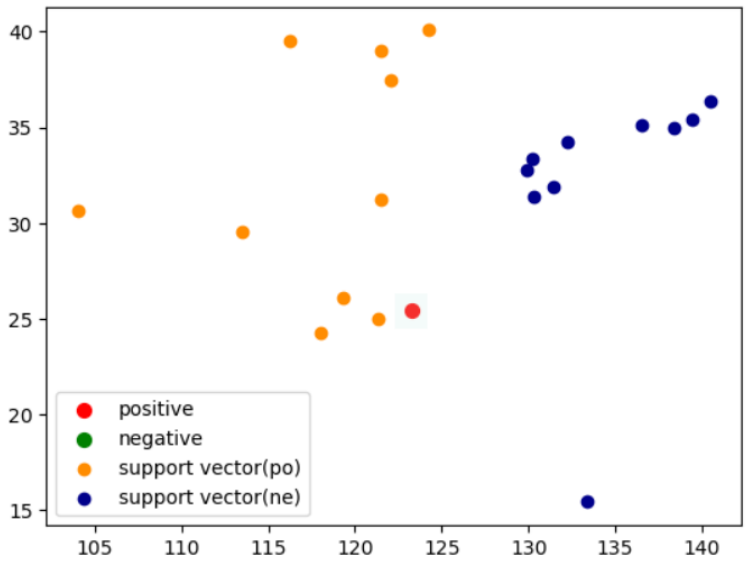

图 29: 增加样本后，测试集可视化

可见，增加内陆样本并不影响分类结果，但是对于这种非线性的核函数，似乎每个样本都变成了支撑向量