Lecture7-8编程作业

一.利用二次规划函数，分别编程实现原问题求解的支撑向量机算法（Primal-SVM）、对偶的支撑向量机算法（Dual-SVM）、和核函数的支撑向量机算法（Kernel-SVM）

primal_SVM:

利用二次规划问题求解最佳分类面权向量

```
def primal_SVM(data,label):                    #原数据是增广数据
    data=np.delete(data,0,1)                   #去掉增广项
    d=len(data[0])
    length=len(data)
    Q1=np.zeros((1,d+1))
    Q2=np.hstack((np.zeros((1,d)).T,np.eye(d)))
    Q=cvxopt.matrix(np.vstack((Q1,Q2)))
    p=cvxopt.matrix(np.zeros((d+1,1)))
    A=np.zeros((length,d+1))
    for i in range(length):
        a=np.insert(data[i],0,1)
        A[i]=label[i]*a
    A=cvxopt.matrix(-A)
    c=cvxopt.matrix(-np.ones((length,1)))
    u=cvxopt.solvers.qp(Q,p,A,c)               #利用二次规划问题求解问题
    w=np.ravel(u['x'])                         #得到最佳分类面
    return w
```

dual-SVM:

该对偶问题SVM可以实现升维至任意维数后利用二次规划来求解alpha,得出系统的支撑向量.

```
def dual_SVM(data,label):
    data=np.delete(data,0,1)                           #去掉数据中的增广项
    length=len(data)
    dim=2                                              #升维到dim维
    poly = PolynomialFeatures(dim)
    Z=poly.fit_transform(data)
    YZ=np.delete(Z,0,axis=1)
    Z=np.delete(Z,0,axis=1)
    for i in range(length):
        if label[i]==-1:
            YZ[i]*=-1
    Q=cvxopt.matrix(np.dot(YZ,YZ.T))
    p=cvxopt.matrix(-np.ones((length,1)))
    A=cvxopt.matrix(-np.eye(length))
    c=cvxopt.matrix(np.zeros((length,1)))
    r=cvxopt.matrix(np.array([label]))
    v=cvxopt.matrix(np.zeros(1))
    u=cvxopt.solvers.qp(Q,p,A,c,r,v)                   #利用二次规划求解alpha
    alpha=np.ravel(u['x'])                             #获得拉普拉斯乘子
    sv=alpha>1e-5
    alpha=alpha[sv]                                    #非零拉格朗日乘子对应的向量为支撑向
量
```

```
    support=Z[sv]
    support_label=label[sv]
    w=np.zeros(len(Z[0]))
    for i in range((len(alpha))):
        w+=alpha[i]*support_label[i]*support[i]     #最佳权系数w
    b=support_label[0]-np.dot(w,support[0])         #截距项
    w=np.insert(w,0,b)                              #增广后的w
    return w
```

kernel-SVM:

定义了Kernel-SVM类,初始化时可以选择核函数类型,比如高斯核函数或多项式核函数,并利用核函数来求解支撑向量

```
class Kernel_SVM:
    def __init__(self,flag,zeta,gamma,index):          #参数初始化
        self.flag=flag                                 #flag=0,则为多项式核函数;
flag=1,则为高斯核函数
        #多项式核函数参数
        self.zeta=zeta
        self.gamma=gamma
        self.index=index

    def kernel(self,x1,x2):                            #求核函数
        if self.flag==0:
            K=(self.zeta+self.gamma*np.dot(x1,x2))**self.index
        if self.flag==1:
            K=exp(-self.gamma*np.linalg.norm(x1-x2))
        return K

    def SVM(self,data,label):
        data=np.delete(data,0,1)
        length=len(data)
        K=np.zeros((length,length))
        for i in range(length):
            for j in range(length):
                K[i,j]=self.kernel(data[i],data[j])
        Q=cvxopt.matrix(np.outer(label,label)*K)
        p=cvxopt.matrix(-np.ones(length))
        A=cvxopt.matrix(-np.eye(length))
        c=cvxopt.matrix(np.zeros((length,1)))
        r=cvxopt.matrix(np.array([label]))
        v=cvxopt.matrix(np.zeros(1))
        u=cvxopt.solvers.qp(Q,p,A,c,r,v)               #利用二次规划求解alpha
        alpha=np.ravel(u['x'])                         #获得拉普拉斯乘子
        sv=alpha>1e-5
        self.alpha=alpha[sv]                           #非零拉格朗日乘子对应的向量为支
撑向量
        self.support=data[sv]
        self.support_label=label[sv]
        self.b=self.support_label[0]                   #截距项
        for i in range(len(self.alpha)):
            self.b-
=alpha[i]*label[i]*self.kernel(self.support[i],self.support[0])

    def predict(self,data,label):                      #分类器在测试集中的准确率
        data=np.delete(data,0,1)
```

```
        count=0
        for n in range(len(data)):
            all=self.b
            for i in range(len(self.alpha)):

all+=self.alpha[i]*self.support_label[i]*self.kernel(self.support[i],data[n])
            g=np.sign(all)
            if g==label[n]:
                count+=1
        precision=count/len(data)
        return precision
```

二.

1. 数据集

产生两个都具有200个二维向量的数据集 X_1 和 X_2 。数据集 X_1 的样本来自均值向量 $m_1 = [-5, 0]^T$，协方差矩阵 $s_1 = I$ 的正态分布，属于“+1”类，数据集 X_2 的样本来自均值向量 $m_2 = [0, 5]^T$,协方差矩阵 $s_2 = I$ 的正态分布，属于“-1”类，其中 I 是一个2*2的单位矩阵。其中的数据中80%用于训练，20%用于测试。

2. 训练集和测试集上，两种算法的分类正确率

Primal: $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 1.0$

Dual-SVM: $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 1.0$

Kernel-SVM(四次多项式): $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 0.9375$

Kernel-SVM(高斯核函数): $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 0.9875$

3. 对于Dual-SVM和Kernel-SVM算法，指出哪些样本是支撑向量

```
u=cvxopt.solvers.qp(Q,p,A,c,r,v)                    #利用二次规划求解alpha
alpha=np.ravel(u['x'])                              #获得拉普拉斯乘子
sv=alpha>1e-5                                       #若alpha>1e-5则认为它对应的是支撑向
量
alpha=alpha[sv]                                     #非零拉格朗日乘子对应的向量为支撑向量
support=Z[sv]                                       #找到支撑向量
support_label=label[sv]                             #找到支撑标签
```

3.1 Dual-SVM的支撑向量

找到 $\alpha > e^{-5}$ 的 α ,标记其为True（本实验对偶SVM并没有对数据维度升维）

```
Optimal solution found.
[False False False False False False False False False False  True False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False  True
 False False False False False False False False False False False False
 False False False False False False  True False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
```

True对应的样本,即这些向量为支持向量(本实验中支撑向量为下面**三个向量**,向量内 没包括w_0)

```
[[-2.5504231   1.44784009]
 [-0.62099529  2.02117227]
 [-2.24681903  4.65430002]]
```

再根据任意一个支撑向量求出 w_0 (也就是b)

最终的分界面权向量 w* 为

```
[-0.45084691 -0.87587473 -0.5408085 ]
```

3.2 Kernel-SVM的支撑向量

3.2.1 多项式核函数

参数说明: 核函数形式 $(\zeta + \gamma x_1 x_2)^\beta$,设置四次多项式核函数,各参数为 $\zeta = 1, \gamma = 1, \beta = 4$

其支撑向量为:

```
[[-2.82746019  0.36968196]
 [-2.5504231   1.44784009]
 [-0.62099529  2.02117227]]
```

本实验中多项式核函数支撑向量个数与对偶问题支撑向量个数相同都是3, 其中有两个支撑向量相同,有一个支撑向量选取不同

3.2.2 高斯核函数

参数说明: 标准高斯核函数

高斯核函数的支撑向量,较Primal-SVM 和 Dual-SVM 的支撑向量而言,增加了很多支撑向量,共129个支撑向量:

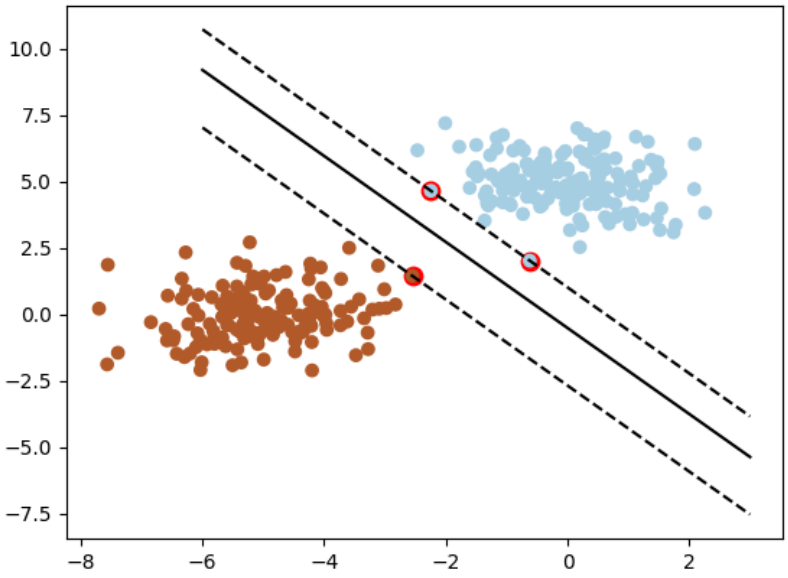
```
[[-6.08154604e+00 -7.14339340e-01]
 [-4.22643429e+00  1.90962324e+00]
 [-5.08908464e+00 -1.11915607e+00]
 [-5.84573021e+00  6.42467375e-01]
 [-3.27739484e+00 -1.31027019e+00]
 [-2.82746019e+00  3.69681959e-01]
 [-4.96151146e+00  7.83120362e-01]
 [-5.50465885e+00 -1.91946932e+00]
 [-2.55042310e+00  1.44784009e+00]
.......
 [ 4.02050907e-03  5.49808532e+00]
 [-8.95762677e-01  5.46057955e+00]]
```

4. 画出数据集和分类面、间隔面，并标注出哪些样本是支撑向量，观察是否有边界上的向量不是支撑向量的现象

图中分类面为实线，间隔面为虚线，支撑向量为图中标红的点。

从图中可以看出有三个支撑向量，这与 (3) 对偶问题根据 $\alpha > 0$ 所求支撑向量刚好符合,侧面验证了 $\alpha > 0$ 的边界样本向量即为支撑向量。

在该图中可以看出该问题所有的边界上的向量都是支撑向量。但是，并不代表所有问题的边界上的向量就都是支撑向量。边界上的点是候选支撑向量，只有在对偶问题中向量所对应的 $\alpha > 0$, 才满足为支撑向量 。



三.

1. 数据集

重复第2题的内容，但数据集 X_1 和数据集 X_2 的均值向量分别改为 $m_1 = [3, 0]^T$和 $m_2 = [0, 3]^T$ ，其他不变。

2. 训练集和测试集上，两种算法的分类正确率

Primal: $Accuracy_{(in)} = 0.975, Accuracy_{(out)} = 0.975$

Dual-SVM: $Accuracy_{(in)} = 0.975, Accuracy_{(out)} = 0.975$

Kernel-SVM(四次多项式): $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 0.975$

Kernel-SVM(高斯核函数): $Accuracy_{(in)} = 1.0, Accuracy_{(out)} = 0.9625$

3. 对于Dual-SVM和Kernel-SVM算法，指出哪些样本是支撑向量

3.1 Dual-SVM的支撑向量

找到 $\alpha > e^{-5}$ 的 α ,标记其为True（本实验对偶SVM并没有对数据维度升维)

rue对应的样本,即这些向量为支持向量(本实验中支撑向量为下面**13个向量**,向量内包括w_0)

```
[[ 1.          1.35406486  1.7866175 ]
 [ 1.          1.15963637  1.49643683]
 [ 1.          2.1354311   3.23932024]
 [ 1.          0.84131458  1.05832117]
 [ 1.          0.75464411  1.03800891]
 [ 1.          1.29725726  1.06149769]
 [ 1.          1.4137289   1.61198987]
 [ 1.          0.17328841 -0.92756252]
 [ 1.          1.58153453  1.14854839]
 [ 1.          1.92066173  1.52524288]
 [ 1.          2.258979    2.16391515]
 [ 1.          1.80310004  1.62032767]
 [ 1.          1.84901822  1.46183806]]
```

再根据任意一个支撑向量求出 w_0 (也就是b)

最终的分界面权向量 w* 为

```
[-9.33356    13.47932245 -8.62232942]
```

3.2 Kernel-SVM的支撑向量

3.2.1 多项式核函数

参数说明: 核函数形式 $(\zeta + \gamma x_1 x_2)^{\beta}$,设置四次多项式核函数,各参数为 $\zeta = 1, \gamma = 1, \beta = 4$

支撑向量为:

```
[[ 0.17328841 -0.92756252]
 [ 1.0034587  -0.03493496]
 [ 1.58153453  1.14854839]
 [ 1.80310004  1.62032767]
 [-0.13111128  0.6828912 ]
 [ 1.29725726  1.06149769]]
```

3.2.2 高斯核函数

参数说明: 标准高斯核函数

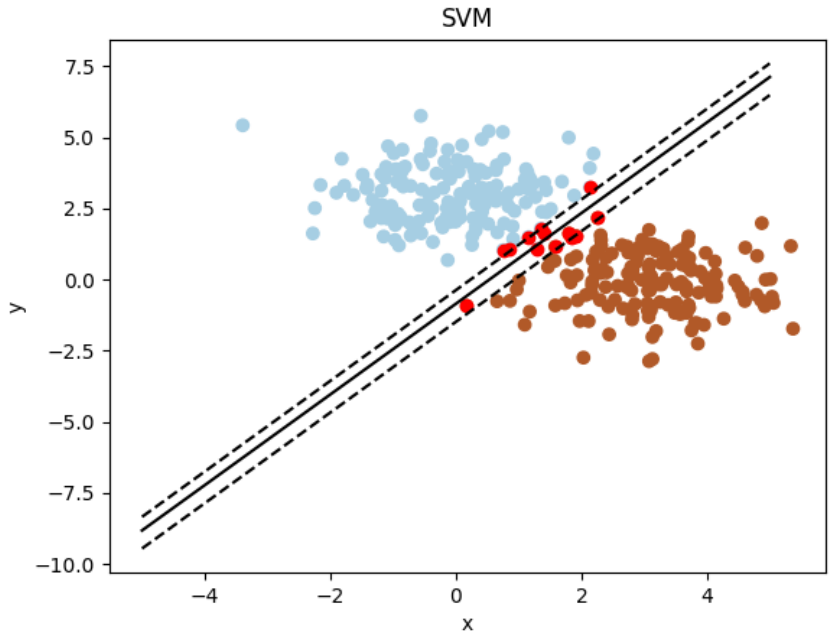
高斯核函数的支撑向量,较Primal-SVM 和 Dual-SVM 的支撑向量而言,增加了很多支撑向量,共82个支撑向量:

```
[[ 5.33582954e+00  1.17693114e+00]
 [ 4.26635105e+00 -1.37611326e+00]
 [ 3.09208276e+00 -3.80352061e-01]
 [ 3.85414275e+00 -2.25251257e+00]
 [ 3.60601138e+00 -1.19127845e+00]
 [ 1.73288412e-01 -9.27562521e-01]
 [ 4.14230845e+00 -5.95988312e-01]
 [ 3.59144047e+00 -8.38898287e-02]
 [ 3.51488100e+00 -1.73800363e-01]
  ..........
 [ 3.09036817e-01  3.64402890e+00]]
```

4. 画出数据集和分类面、间隔面，并标注出哪些样本是支撑向量，观察是否有边界上的向量不是支撑向量的现象

图中分类面为实线，间隔面为虚线，支撑向量为图中标红的点。

由于该数据部分不可分，导致最大间隔分类面内也存在支撑向量，并不仅在间隔边界上。



四.改变算法中的超参数、样本数量、样本分布等，讨论实验结果

改变样本数量对实验结果影响不大，但对于问题三来说，由于数据均值向量分别改为 $m_1 = [3, 0]^T$ 和 $m_2 = [0, 3]^T$,协方差矩阵 $s_2 = I$ 的正态分布，随着实验数据的增加，可能线性不可分的情况会加重。

样本分布对实验结果影响大，由于SVM是硬分类，当数据为线性可分情况时，边界上的向量且$\alpha > 0$则为支撑向量，而对于非线性可分情况来说，最大间隔分类面内仍然存在数据落入其中，且这些向量也可能为支撑向量。

五.钓鱼岛是中国的

训练集: 中国与日本的沿海城市的经纬度坐标向量，中国标签为+1, 日本为标签为-1.

测试集: 钓鱼岛的经纬度坐标向量

用支撑向量机设计分类器，（1）判断钓鱼岛属于哪一类；（2）增加几个非海边城市的经纬度坐标进行训练，判断这些城市是否影响分类结果，是否为支撑向量。

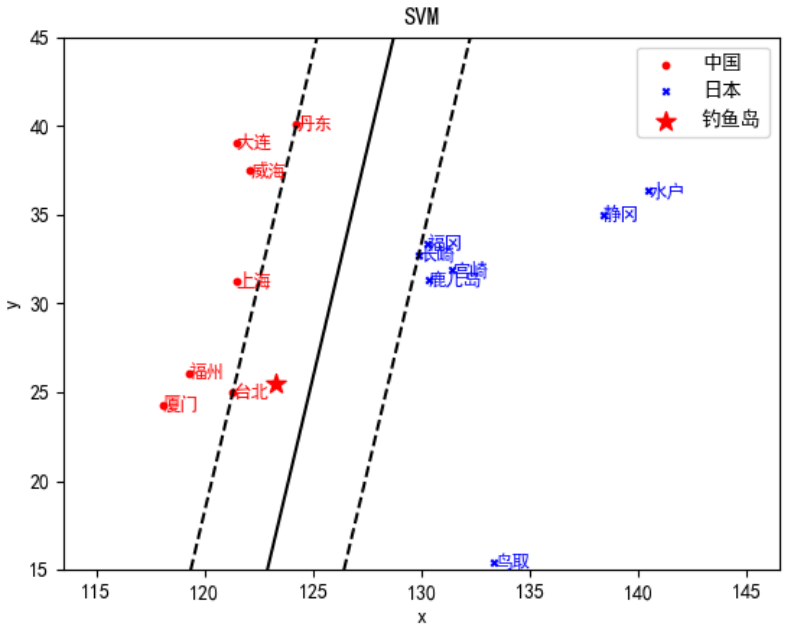
(1) 通过查阅资料求出沿海城市经纬度

```
X1 = np.array([[119.28, 26.08],     # 福州
                   [121.47, 31.23],     # 上海
                   [118.06, 24.27],     # 厦门
                   [122.10, 37.50],     # 威海
                   [121.31, 25.03],     # 台北
                   [121.46, 39.04],     # 大连
                   [124.23, 40.07]])    # 丹东

    X2 = np.array([[129.87, 32.75],     # 长崎
                   [130.24, 33.35],     # 福冈
                   [130.33, 31.36],     # 鹿儿岛
                   [131.42, 31.91],     # 宫崎
                   [133.33, 15.43],     # 鸟取
                   [138.38, 34.98],     # 静冈
                   [140.47, 36.37]])    # 水户
    X1=np.insert(X1,0,1,axis=1)                                  #增广后的数据集x1
    X2=np.insert(X2,0,-1,axis=1)                                  #增广后的数据集x2
    city = np.array(['福州', '上海', '厦门', '威海', '台北', '大连', '丹东',
                     '长崎', '福冈', '鹿儿岛', '宫崎', '鸟取', '静冈', '水户'])
    X_test = np.array([[123.28, 25.45]])    # 钓鱼岛
    y_test = np.array([1])
```

从图中可以看出**中国丹东、中国台北以及日本长崎**是边界上的点，根据对偶SVM可以计算出这三个点的$\alpha > 0$，则这三个点是支撑向量。

通过该图可以看出，分类面将钓鱼岛划分在了“+1”类，即**钓鱼岛属于中国。**



(2)增加几个非海边城市的经纬度坐标

引入南京、武汉、长沙、南昌四个非海边城市

```
X1 = np.array([[118.22,31.14],[113.41,29.58],[112.59,28.12],[115.27,28.09]])
X1=np.insert(X1,0,1,axis=1)                                    #增广后的数据集x1
city = np.array(['南京','武汉','长沙','南昌'])
```

从图中可以看出这四个**非沿海城市离分界面很远，并没有影响到钓鱼岛的分类结果。**

这些沿海城市**并不是支撑向量。**