1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：116页，为什么函数间隔可以取一定值1

讨论后的理解：函数间隔虽然说是可以无穷变化，但是它与||w||之间的比值是固定的（也就是几何间隔），因此代表的是同一个平面。

提出的问题2：在高维空间更容易产生噪声，计算距离的时候如何避免

讨论后的理解：SVM的决策只基于少量的支持向量，若噪音样本出现在支持向量中，容易对决策造成影响，所以SVM对噪音敏感。尤其是维度较高的时候影响会更大。因此有必要做两件事，降维和去噪，具体的可以给核函数加不同的权重。

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. 问题3：p132，图7.6中0-1损失函数在间隔大于0的部分是什么样的。

自己的理解：0-1损失函数即为“判断正确”的话损失函数的值为0，“判断错误”的话损失函数的值为1，所以在图中左边的部分为一条恒等于1 的直线，右边的部分则是和坐标轴重合的直线。

问题4：p131，（7.58）为什么式中要有一个正则化项？

自己的理解：和本章开始的时候的线性支持向量机相同，因为函数距离在求分类超平面的时候会出现只要等比例增加w和b就能在增大函数距离的同时而不改变超平面，这里也是类似，如果仅仅使用前面的损失函数，也会出现同样的问题，所以可以在后面增加一个w的l2范数来避免这种情况。

问题5：如何找软间隔最大化时的支持向量？

自己的理解：在软间隔最大化时，因为对每个样本(xi,yi)引入了松弛变量ξi。从下图来研究软间隔最大化时支持向量的情况，第i个点到对应类别支持向量的距离为ξi||w||2。根据软间隔最大化时KKT条件中的对偶互补条件α∗i(yi(wTxi+b)−1+ξ∗i)=0有：

a) 如果α=0,那么yi(wTxi+b)−1≥0,即样本在间隔边界上或者已经被正确分类。如图中所有远离间隔边界的点。

b) 如果0<α<C,那么ξi=0,yi(wTxi+b)−1=0,即点在间隔边界上。

c) 如果α=C，说明这是一个可能比较异常的点，需要检查此时ξi

　　i)如果0≤ξi≤1,那么点被正确分类，但是却在超平面和自己类别的间隔边界之间。如图中的样本2和4.

　　ii)如果ξi=1,那么点在分离超平面上，无法被正确分类。

　　　　iii)如果ξi>1,那么点在超平面的另一侧，也就是说，这个点不能被正常分类。如图中的样本1和3.

三、（必填）读书计划

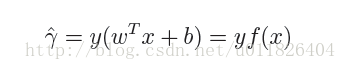
1、本周完成的内容章节：李航书7.1-7.2

2、下周计划：李航书第7章

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）

在超平面w’x+b=0确定的情况下，|w’x\*+b|能够代表点x\*距离超平面的远近，易知：当w’x\*+b>0时，表示x\*在超平面的一侧（正类，类标为1），而当w’x\*+b<0时，则表示x\*在超平面的另外一侧（负类，类别为-1），因此（w’x\*+b）y\* 的正负性恰能表示数据点x\*是否被分类正确。于是便引出了**函数间隔**的定义（functional margin）:



而超平面（w,b）关于所有样本点（Xi，Yi）的函数间隔最小值则为超平面在训练数据集T上的函数间隔：

这里写图片描述

可以看出：这样定义的函数间隔在处理SVM上会有问题，当超平面的两个参数w和b同比例改变时，函数间隔也会跟着改变，但是实际上超平面还是原来的超平面，并没有变化。例如：w1x1+w2x2+w3x3+b=0其实等价于2w1x1+2w2x2+2w3x3+2b=0，但计算的函数间隔却翻了一倍。从而引出了能真正度量点到超平面距离的概念–几何间隔（geometrical margin）。

2.代码实现

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

# data

def create\_data():

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['label'] = iris.target

df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']

data = np.array(df.iloc[:100, [0, 1, -1]])

for i in range(len(data)):

if data[i,-1] == 0:

data[i,-1] = -1

# print(data)

return data[:,:2], data[:,-1]

X, y = create\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

plt.scatter(X[:50,0],X[:50,1], label='0')

plt.scatter(X[50:,0],X[50:,1], label='1')

plt.legend()

plt.show()

class SVM:

def \_\_init\_\_(self, max\_iter=100, kernel='linear'):

self.max\_iter = max\_iter

self.\_kernel = kernel

def init\_args(self, features, labels):

self.m, self.n = features.shape

self.X = features

self.Y = labels

self.b = 0.0

# 将Ei保存在一个列表里

self.alpha = np.ones(self.m)

self.E = [self.\_E(i) for i in range(self.m)]

# 松弛变量

self.C = 1.0

def \_KKT(self, i):

y\_g = self.\_g(i) \* self.Y[i]

if self.alpha[i] == 0:

return y\_g >= 1

elif 0 < self.alpha[i] < self.C:

return y\_g == 1

else:

return y\_g <= 1

# g(x)预测值，输入xi（X[i]）

def \_g(self, i):

r = self.b

for j in range(self.m):

r += self.alpha[j] \* self.Y[j] \* self.kernel(self.X[i], self.X[j])

return r

# 核函数

def kernel(self, x1, x2):

if self.\_kernel == 'linear':

return sum([x1[k] \* x2[k] for k in range(self.n)])

elif self.\_kernel == 'poly':

return (sum([x1[k] \* x2[k] for k in range(self.n)]) + 1) \*\* 2

return 0

# E（x）为g(x)对输入x的预测值和y的差

def \_E(self, i):

return self.\_g(i) - self.Y[i]

def \_init\_alpha(self):

# 外层循环首先遍历所有满足0<a<C的样本点，检验是否满足KKT

index\_list = [i for i in range(self.m) if 0 < self.alpha[i] < self.C]

# 否则遍历整个训练集

non\_satisfy\_list = [i for i in range(self.m) if i not in index\_list]

index\_list.extend(non\_satisfy\_list)

for i in index\_list:

if self.\_KKT(i):

continue

E1 = self.E[i]

# 如果E2是+，选择最小的；如果E2是负的，选择最大的

if E1 >= 0:

j = min(range(self.m), key=lambda x: self.E[x])

else:

j = max(range(self.m), key=lambda x: self.E[x])

return i, j

def \_compare(self, \_alpha, L, H):

if \_alpha > H:

return H

elif \_alpha < L:

return L

else:

return \_alpha

def fit(self, features, labels):

self.init\_args(features, labels)

for t in range(self.max\_iter):

# train

i1, i2 = self.\_init\_alpha()

# 边界

if self.Y[i1] == self.Y[i2]:

L = max(0, self.alpha[i1] + self.alpha[i2] - self.C)

H = min(self.C, self.alpha[i1] + self.alpha[i2])

else:

L = max(0, self.alpha[i2] - self.alpha[i1])

H = min(self.C, self.C + self.alpha[i2] - self.alpha[i1])

E1 = self.E[i1]

E2 = self.E[i2]

# eta=K11+K22-2K12

eta = self.kernel(self.X[i1], self.X[i1]) + self.kernel(self.X[i2], self.X[i2]) - 2 \* self.kernel(

self.X[i1], self.X[i2])

if eta <= 0:

# print('eta <= 0')

continue

alpha2\_new\_unc = self.alpha[i2] + self.Y[i2] \* (E1 - E2) / eta # 此处有修改，根据书上应该是E1 - E2，书上130-131页

alpha2\_new = self.\_compare(alpha2\_new\_unc, L, H)

alpha1\_new = self.alpha[i1] + self.Y[i1] \* self.Y[i2] \* (self.alpha[i2] - alpha2\_new)

b1\_new = -E1 - self.Y[i1] \* self.kernel(self.X[i1], self.X[i1]) \* (alpha1\_new - self.alpha[i1]) - self.Y[

i2] \* self.kernel(self.X[i2], self.X[i1]) \* (alpha2\_new - self.alpha[i2]) + self.b

b2\_new = -E2 - self.Y[i1] \* self.kernel(self.X[i1], self.X[i2]) \* (alpha1\_new - self.alpha[i1]) - self.Y[

i2] \* self.kernel(self.X[i2], self.X[i2]) \* (alpha2\_new - self.alpha[i2]) + self.b

if 0 < alpha1\_new < self.C:

b\_new = b1\_new

elif 0 < alpha2\_new < self.C:

b\_new = b2\_new

else:

# 选择中点

b\_new = (b1\_new + b2\_new) / 2

# 更新参数

self.alpha[i1] = alpha1\_new

self.alpha[i2] = alpha2\_new

self.b = b\_new

self.E[i1] = self.\_E(i1)

self.E[i2] = self.\_E(i2)

return 'train done!'

def predict(self, data):

r = self.b

for i in range(self.m):

r += self.alpha[i] \* self.Y[i] \* self.kernel(data, self.X[i])

return 1 if r > 0 else -1

def score(self, X\_test, y\_test):

right\_count = 0

for i in range(len(X\_test)):

result = self.predict(X\_test[i])

if result == y\_test[i]:

right\_count += 1

return right\_count / len(X\_test)

def \_weight(self):

# linear model

yx = self.Y.reshape(-1, 1) \* self.X

self.w = np.dot(yx.T, self.alpha)

return self.w

svm = SVM(max\_iter=200)

svm.fit(X\_train, y\_train)

print(svm.score(X\_test, y\_test))