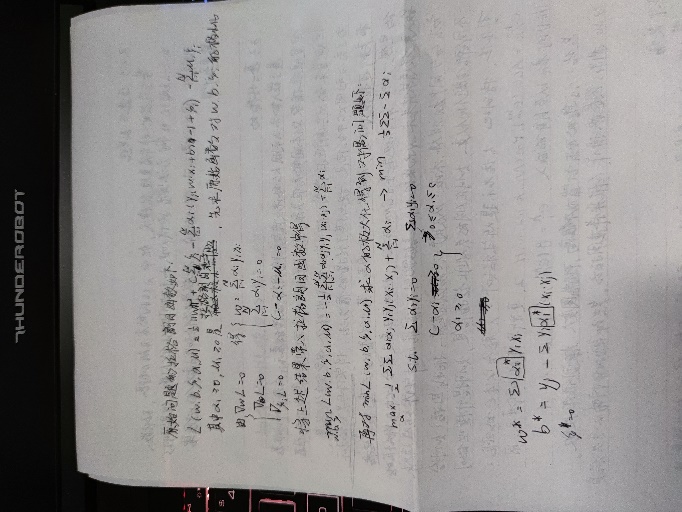
1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：原始问题的对偶问题如何得出

讨论后的理解：

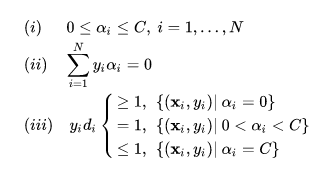


提出的问题2：SMO为什么需要优化两个变量

讨论后的理解：由限制,如果优化变量为1，则常量可以推出该变量。通常变量1违反KKT条件最严重，变量2的更新范围最大。

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：SMO停机条件。

自己的理解：



问题4：希尔伯特空间与普通线性空间区别？

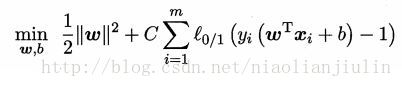
自己的理解：

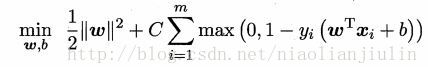
* 1. 线性空间与度量空间是两个不同的概念，没有交集。
  2. 赋范线性空间就是赋予了范数的线性空间，也是度量空间（具有线性结构的度量空间）
  3. 内积空间是赋范线性空间
  4. 希尔伯特空间就是完备的内积空间。

问题5：为什么线性支持向量机学习等价于最小化二阶范数正则化的合页函数？

自己的理解：

采用合页损失函数时，下式



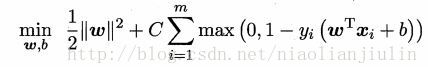
变成：  


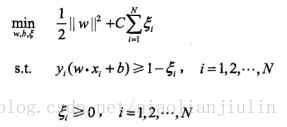
令 max(0,1−yi(wTxi+b))=ξimax(0,1−yi(wTxi+b))=ξi ，则 ξi≥0ξi≥0 。

当 1−yi(wTxi+b)>01−yi(wTxi+b)>0 ，则 yi(wTxi+b)=1−ξiyi(wTxi+b)=1−ξi ;

当 1−yi(wTxi+b)≤0,ξi=01−yi(wTxi+b)≤0,ξi=0 ， yi(wTxi+b)≥1=1−ξiyi(wTxi+b)≥1=1−ξi ;

综上有 yi(wTxi+b)≥1−ξiyi(wTxi+b)≥1−ξi ;

则合页损失函数：  


可写成如下形式：  


这就是线性SVM的原始最优化问题。

三、（必填）读书计划

1、本周完成的内容章节：李航书第七章

2、下周计划：李航书第8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）

#### 输入空间和特征空间

假设输入空间和特征空间为两个不同的空间，输入空间为欧氏空间或离散集合，特征空间为欧氏空间或希尔伯特空间。线性可分支持向量机、线性支持向量机假设这两个空间的元素一一对应，并将输入空间中的输入映射为特征空间中的特征向量。非线性支持向量机利用一个从输入空间到特征空间的非线性映射将输入映射为特征向量。所以，输入都由输入空间转换到特征空间，支持向量机的学习是在特征空间进行的。

一般地，当训练数据集线性可分时，存在无穷多个分离超平面可将两类数据正确分开。感知机利用误分类最小的策略，求得分离超平面，不过这时的解有无穷多个。线性可分支持向量机利用间隔最大化求最优分离超平面，这时，解是唯一的。

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))