山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名：黄瑞哲 | | 班级： 计科17.3 |
| 实验题目：支持向量机 | | | |
| 实验学时：10 | | 实验日期：2019.11.29 | |
| 实验目的：   1. 练习使用线性和非线性的支持向量机分类器 2. 应用SVM完成01图像的识别 | | | |
| 硬件环境：  Intel Core i5-8300H @ 2.3GHz | | | |
| 软件环境：  Windows10 Pro 1903  Python 3.7  Visual Studio Code | | | |
| 实验步骤与内容：   1. 线性SVM   软边距的SVM的优化模型为    转换为相应的拉格朗日对偶问题后优化目标变为    这是一个二次规划QP问题，可以应用现成的求解器求解。  应用硬边距C=0时得到的结果如下图    得到测试的准确率为100%。接下来应用软边距，由于C越大，表示惩罚项越高，相应的得到的分类结果更精确，C小则表示允许更多的误分类，因而尝试C=1e-8.    可以看出当C=1e-8时超平面出现了一定的偏差，准确率降至99%。接下来尝试C=1e-9.    可以看出分类器将所有的数据分成了一类，原有的数据特征因为惩罚项太小而丢失。因此在做分类之前需要预测模型是否需要使用硬边距，有时C=0的时候效果要好的很多。   1. 01图像识别   做图像识别的时候可以将图像的每个像素点的颜色值作为它的feature，在这个问题中，所有图片一共有784个像素点，因此我们可以设置每个样本有784个feature，每个feature由它的颜色值所决定，加载数据后便可以放入SVM中进行训练。  由于数据量过大，为节约时间，只挑选其中3000个数据进行训练。  当C=0的时候    可以发现训练集的准确率为99.63%，测试集的准确率为99.67%  查阅对应的图片    我们可以看出，这时候的误分类大概是因为图形的不规范。以上图为例，它在一个颜色块中挖出了几个像素点的空白，这个数字成为0，但是对于svm来说，却将其分类成了1。  后来随着C设置为1e-10，发现分类的准确率锐减。     1. 非线性SVM   在某些情况下，有些数据是线性不可分的，我们找不到一个超平面将其分成两类。因此需要将当前的feature向高维空间映射，但是映射后会不可避免地造成feature数目的增加，使得计算变得缓慢。因此引入核函数，利用核函数在低维空间完成高维空间的计算，实现在高维空间的线性分割。    将公式红线的地方替换成kernel函数即可。在这个实验中，使用的是RBFkernel。    下列图片分别对应gamma=1，10，100，1000，10000        可以看出，随着gamma的增加，svm的分类效果越来越好，但是gamma增加到1000以上的时候出现了过拟合的现象，这不是我们所期望的，而gamma取值为1或10的时候并没有正常的完成分类任务，而当gamma是100的时候，可以合理地完成分类，因此gamma设置为100时地效果最好。 | | | |
| 结论分析与体会：  由于已经有了现成的QP问题求解器，因此SVM在实现起来并不是特别复杂，但是这个实验有很多细节问题困扰了我很久。第一个问题就是支持向量的判断，按照数学分析来说当a>0的时候我们认为对应的x便是支持向量，但是在计算机存储中，需要设置一个特别小的eps来完成0的判断，这一点在第一个实验和第三个实验的时候是正常的。但是到了第二个实验中，发现有时会找不到支持向量，找到了也可能会有很低的精确度，最后发现还是eps设置的太大，考虑到python的特性，果断把eps设为0发现实现的效果很好。第二点是关于C取值的选择，在课堂中学习了解C增大会使得分类更精确，C小会导致误分类但是鲁棒性更高。在调整C的时候，发现当C取1e-10左右的极小数时才会对结果造成影响，之前的取值都是0.1这个数量级的，发现没有任何变化，还一度怀疑SVM写错了，经过大胆尝试才发现这个问题。 | | | |

附录：程序源代码

# exp5\_1.py

import numpy as np

import matplotlib.pyplot as plt

from SVM import SVM

def load\_data(file):

X = []

y = []

with open(file) as f:

for each\_line in f.readlines():

data = list(map(float, each\_line.strip().split()))

X.append(data[0:-1])

y.append(data[-1])

return np.array(X), np.array(y)

def plot\_margin(X, y, clf):

def f(w, x, b, c=0): # w'x+b=c

return (-w[0] \* x - b + c) / w[1]

X1 = X[y == 1]

X2 = X[y == -1]

# plt.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], c="r", label="Support vector")

plt.plot(X1[:, 0], X1[:, 1], "+", label="Positive")

plt.plot(X2[:, 0], X2[:, 1], "x", label="Negative")

a0 = np.min(X)

b0 = np.max(X)

plt.plot([a0, b0], [f(clf.w, a0, clf.b), f(clf.w, b0, clf.b)], "k") # w'x+b=0

# plt.plot([a0, b0], [f(clf.w, a0, clf.b, 1), f(clf.w, b0, clf.b, 1)], "k--") # w'x+b=1

# plt.plot([a0, b0], [f(clf.w, a0, clf.b, -1), f(clf.w, b0, clf.b, -1)], "k--") # w'x+b=-1

plt.title("SVM - Linear")

plt.axis("tight")

plt.legend()

plt.show()

def plot\_contour(X, y, clf):

X1 = X[y == 1]

X2 = X[y == -1]

# plt.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], c="r", label="Support vector")

plt.plot(X1[:, 0], X1[:, 1], "+", label="Positive")

plt.plot(X2[:, 0], X2[:, 1], "x", label="Negative")

a0 = np.min(X)

b0 = np.max(X)

x1, x2 = np.meshgrid(np.linspace(a0, b0, 50), np.linspace(a0, b0, 50))

x = np.array([[p1, p2] for p1, p2 in zip(np.ravel(x1), np.ravel(x2))])

z = clf.decision\_function(x).reshape(x1.shape)

plt.contour(x1, x2, z, [0.0], colors='k', linewidths=1, origin='lower')

# plt.contour(x1, x2, z + 1, [0.0], colors='grey', linewidths=1, origin='lower')

# plt.contour(x1, x2, z - 1, [0.0], colors='grey', linewidths=1, origin='lower')

plt.title("SVM - Nonlinear")

plt.axis("tight")

plt.legend()

plt.show()

def linear\_test():

X\_train, y\_train = load\_data("exp5/data/training\_1.txt")

X\_test, y\_test = load\_data("exp5/data/test\_1.txt")

clf = SVM(C=1e-9)

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

y\_predict = clf.predict(X\_test)

correct = np.sum(y\_predict == y\_test)

print("%d out of %d predictions correct(%.2f%%)." % (correct, len(y\_test), correct / len(y\_test) \* 100))

plot\_margin(X\_train, y\_train, clf)

# plot\_margin(X\_test, y\_test, clf)

def nonlinear\_test():

X\_train, y\_train = load\_data("exp5/data/training\_3.txt")

clf = SVM(kernel=SVM.RBF\_kernel, gamma=10000)

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

plot\_contour(X\_train, y\_train, clf)

if \_\_name\_\_ == "\_\_main\_\_":

# linear\_test()

nonlinear\_test()

# exp5\_2.py

import numpy as np

import random

from SVM import SVM

def load\_data(file):

X = []

y = []

with open(file) as f:

for each\_line in f.readlines():

data = each\_line.strip().split()

y.append(float(data[0]))

x = [0 for i in range(784)]

for s in data[1:-1]:

ind, color = map(int, s.split(":"))

x[ind - 1] = color \* 100 / 255

X.append(x)

return np.array(X), np.array(y)

def random\_sample(X, y, k):

ind = random.sample([i for i in range(len(X))], k)

X\_random = []

y\_random = []

for i in ind:

X\_random.append(X[i])

y\_random.append(y[i])

return np.array(X\_random), np.array(y\_random), ind

if \_\_name\_\_ == "\_\_main\_\_":

X\_train, y\_train = load\_data("exp5/data/train-01-images.svm")

X\_test, y\_test = load\_data("exp5/data/test-01-images.svm")

X\_train, y\_train, ind = random\_sample(X\_train, y\_train, 3000)

clf = SVM(C=1e-10)

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

y\_predict = clf.predict(X\_train)

correct = np.sum(y\_predict == y\_train)

print("%d out of %d training examples correct(%.2f%%)." % (correct, len(y\_train), correct/len(y\_train)\*100))

y\_predict = clf.predict(X\_test)

correct = np.sum(y\_predict == y\_test)

print("%d out of %d predictions correct(%.2f%%)." % (correct, len(y\_test), correct/len(y\_test)\*100))

# SVM.py

import numpy as np

import cvxopt

class SVM:

@staticmethod

def linear\_kernel(x1, x2):

return np.dot(x1, x2)

@staticmethod

def RBF\_kernel(x1, x2, gamma):

return np.exp(-gamma\*np.dot(x1-x2, x1-x2))

def \_\_init\_\_(self, kernel=None, C=None, \*\*kargcs):

self.kernel = SVM.linear\_kernel if kernel is None else kernel

self.C = C if C is None else float(C)

self.kargcs = kargcs

def fit(self, X, y):

m, n = X.shape

print("overall %d training datas" % m)

K = np.zeros((m, m))

for i in range(m):

for j in range(m):

K[i, j] = self.kernel(X[i], X[j], \*\*self.kargcs)

# P = cvxopt.matrix(np.outer(y, y) \* K)

print((y.T\*y).shape)

P = cvxopt.matrix(y.T\*y\*K)

q = cvxopt.matrix(np.ones(m) \* -1)

# sigma(a\*y)=0

A = cvxopt.matrix(y, (1, m))

b = cvxopt.matrix(0.0)

if self.C is None or self.C == 0: # hard-margin

# a >= 0

G = cvxopt.matrix(np.eye(m) \* -1)

h = cvxopt.matrix(np.zeros(m))

else: # soft-margin

# a >= 0 && a <= c

p1 = np.eye(m) \* -1

p2 = np.eye(m)

G = cvxopt.matrix(np.vstack((p1, p2)))

p1 = np.zeros(m)

p2 = np.ones(m) \* self.C

h = cvxopt.matrix(np.hstack((p1, p2)))

solution = cvxopt.solvers.qp(P, q, G, h, A, b)

a = np.ravel(solution['x'])

# 非0的a对应支持向量

sv = a > 0

# 支持向量对应下标

self.support\_ = np.arange(len(a))[sv]

self.a = a[sv]

self.support\_vectors\_ = X[sv]

self.support\_vectors\_y = y[sv]

print("%d support vectors out of %d points." % (len(self.a), m))

if self.kernel == SVM.linear\_kernel: # 线性可分

self.w = np.zeros(n)

for i in range(len(self.a)):

self.w += self.a[i] \* self.support\_vectors\_y[i] \* self.support\_vectors\_[i]

self.b = 0

for i in range(len(self.a)):

self.b += self.support\_vectors\_y[i] - np.dot(self.w, self.support\_vectors\_[i])

self.b /= len(self.a)

else: # 非线性

self.w = None

self.b = 0

for i in range(len(self.a)):

self.b += self.support\_vectors\_y[i]

self.b -= np.sum(self.a \* self.support\_vectors\_y \* K[self.support\_[i], sv])

self.b /= len(self.a)

def decision\_function(self, X):

if self.w is not None: # 线性可分

return np.dot(X, self.w) + self.b

y\_decision = np.zeros(len(X))

for j in range(len(X)):

for i in range(len(self.a)):

y\_decision[j] += self.a[i] \* self.support\_vectors\_y[i] \* self.kernel(X[j], self.support\_vectors\_[i], \*\*self.kargcs)

return y\_decision + self.b

def predict(self, X):

return np.sign(self.decision\_function(X))