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：

一、 线性 SVM

软边距的 SVM 的优化模型为

$$\min_{\omega,b,\xi} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$s.t. \quad y^{(i)}\left(\omega^T x^{(i)} + b\right) \geq 1 - \xi_i, \ \forall i = 1, \cdots, m$$

$$\xi_i \geq 0, \quad \forall i = 1, \cdots, m$$

转换为相应的拉格朗日对偶问题后优化目标变为

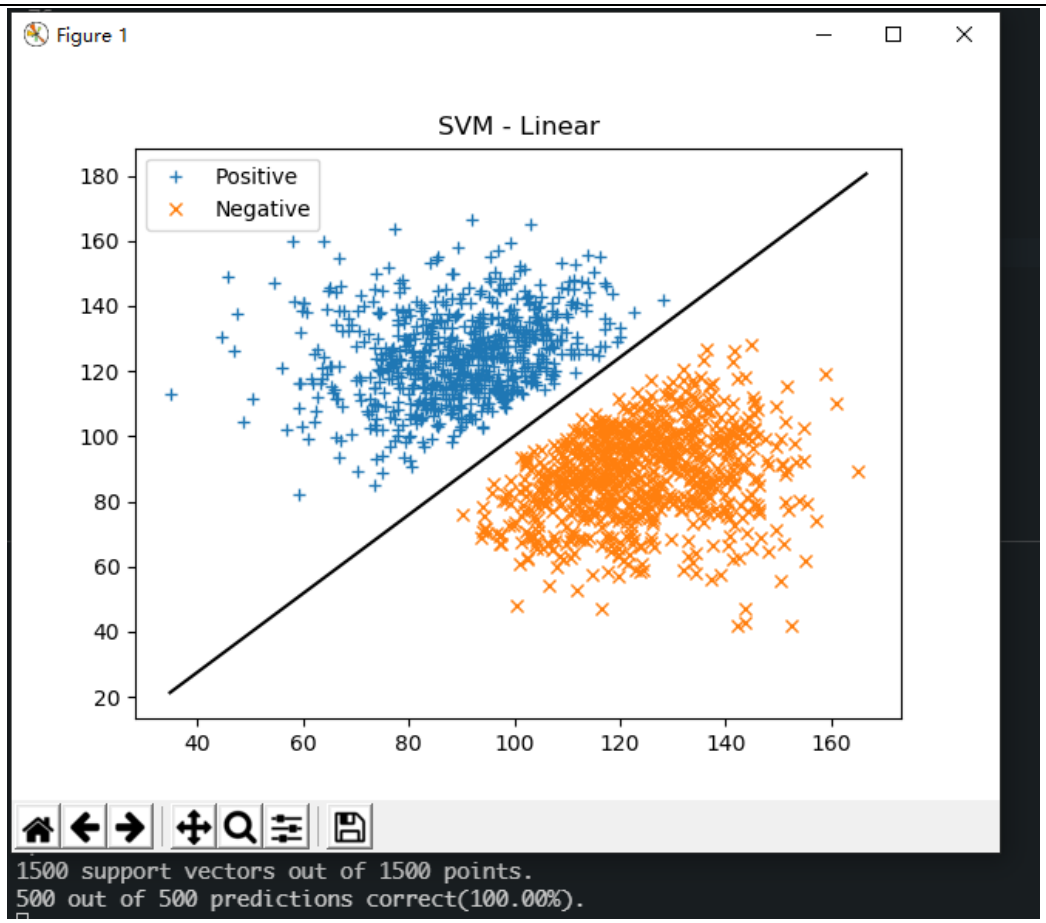
$$\max_{\alpha} \quad \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j < x^{(i)}, x^{(j)} >$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, \cdots, m$$

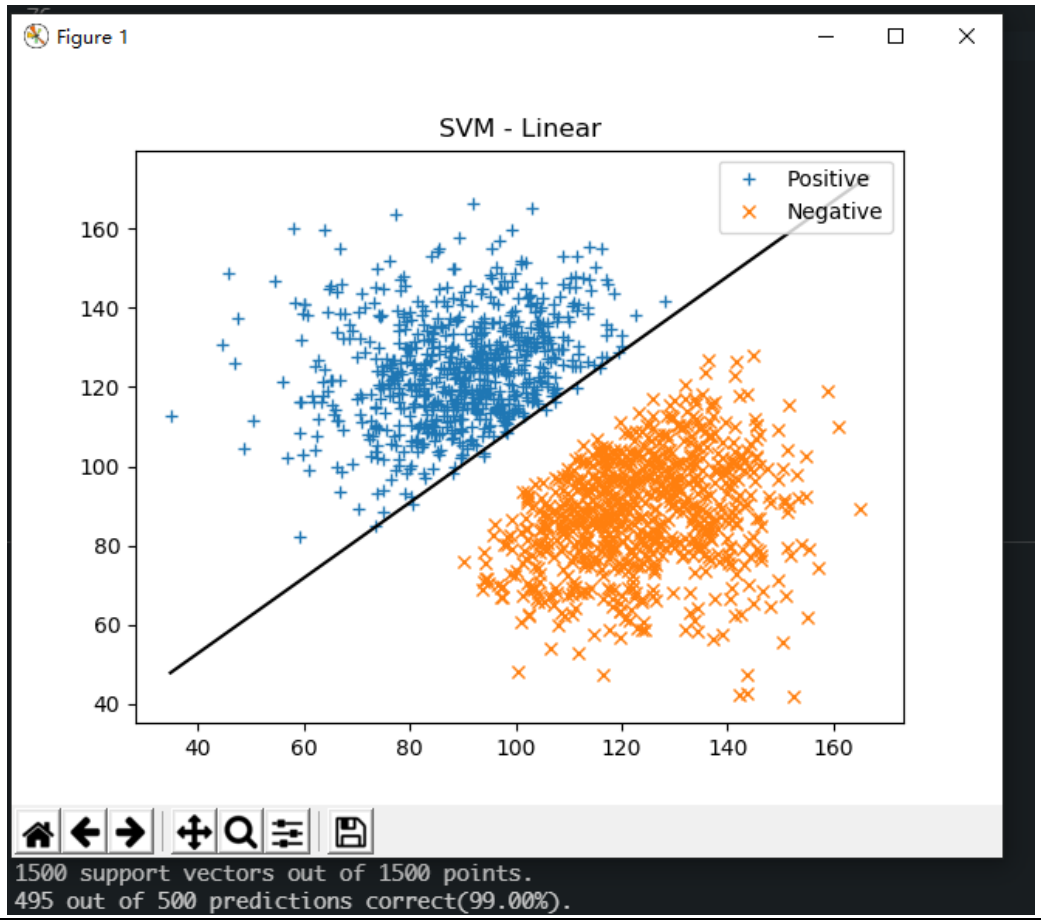
$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

这是一个二次规划 QP 问题，可以应用现成的求解器求解。

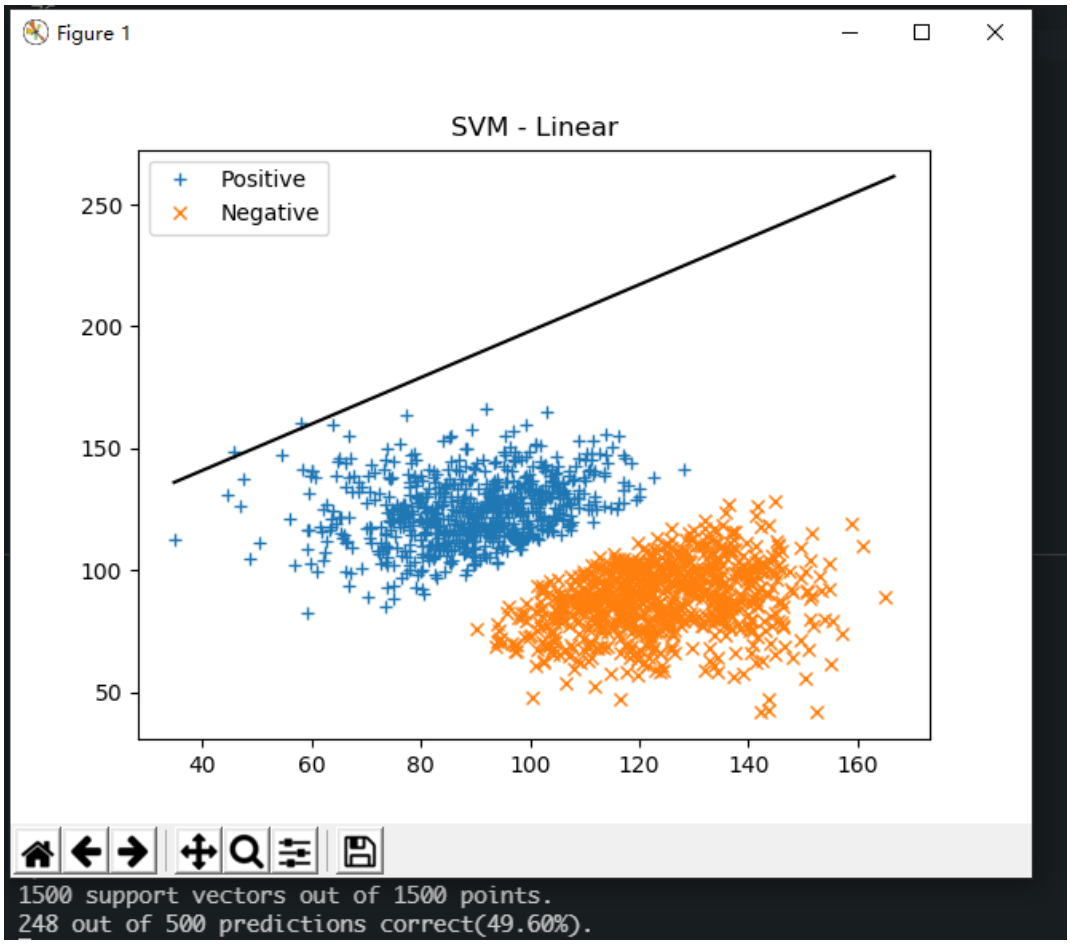
应用硬边距 C=0 时得到的结果如下图



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。

二、 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。

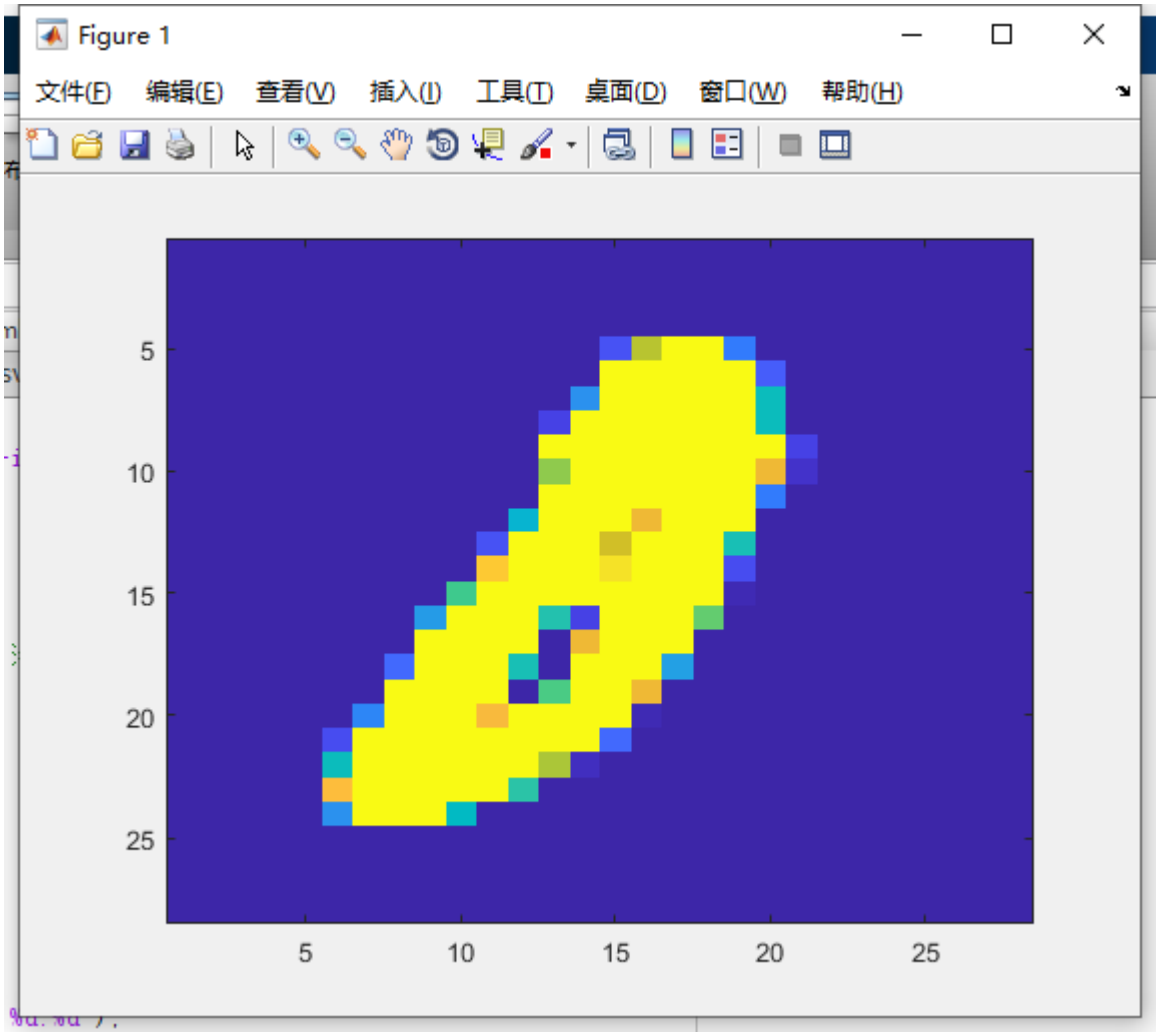
由于数据量过大，为节约时间，只挑选其中 3000 个数据进行训练。

当 C=0 的时候

```
3000 support vectors out of 3000 points.
2989 out of 3000 training examples correct(99.63%).
2108 out of 2115 predictions correct(99.67%).
```

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。

```
3000 support vectors out of 3000 points.
1944 out of 3000 training examples correct(64.80%).
1396 out of 2115 predictions correct(66.00%).
```

三、 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。

$$\max_{\alpha} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \underline{< x^{(i)}, x^{(j)} >}$$

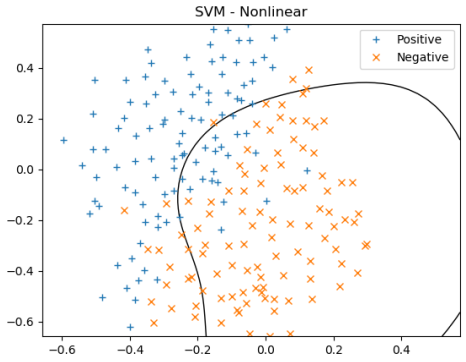
$$s.t. \quad 0 \le \alpha_i \le C, \quad \forall i = 1, \cdots, m$$

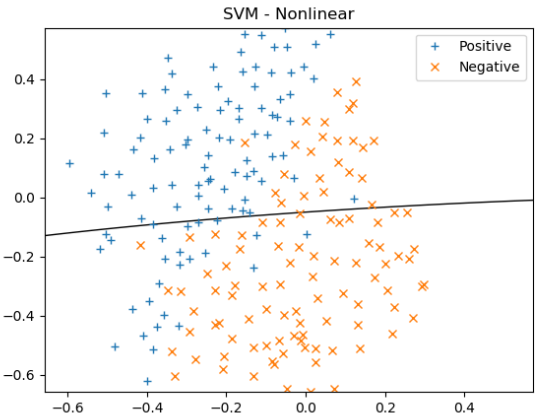
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

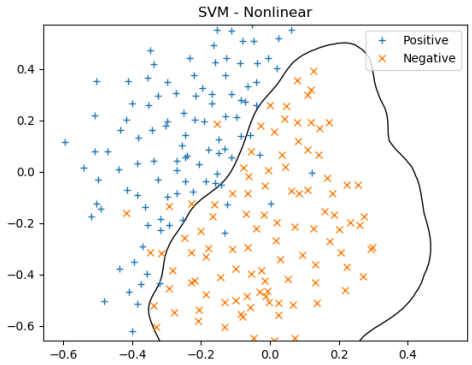
将公式红线的地方替换成 kernel 函数即可。在这个实验中，使用的是 RBFkernel。

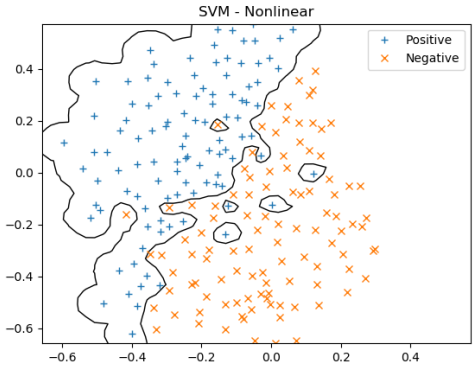
$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = \exp\left(-\gamma \|x^{(i)} - x^{(j)}\|^2\right), \quad \gamma > 0$$

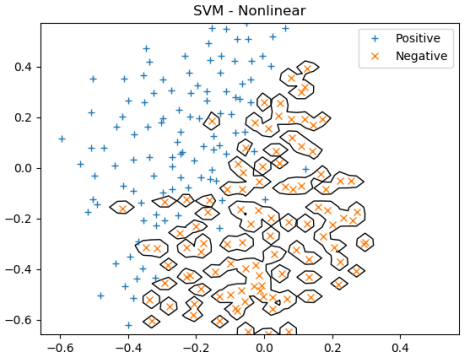
下列图片分别对应 gamma=1，10，100，1000，10000











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))
```