**实验3 序贯最小优化算法（SMO）**

#### 一、实验目的

理解SMO算法的工作原理，编程实现该算法并用于非线性分类问题。

#### 二、实验内容

基于SMO算法，自定义一个实现核SVM分类器的类，其中核函数分别取RBF核和多项式核。利用所设计的SMO类，在乳腺癌数据训练一个SVM分类器，测试其分类精度，并与SKlearn中的SVM进行比较。

#### 三、实验环境

硬件：CPU i5-8300H，内存8G，硬盘 SAMSUNG 512G固态

软件：win10家庭版、python3.7、Visual Studio Code

数据：乳腺癌数据集

#### 四、实验原理

**1. 序贯最小优化算法（Sequential Minimal Optimization，SMO）基本思想**

SMO算法用于求解凸二次规划问题。其主要思想是选取两个变量(lagranger 乘子)作为待更新的变量，固定其他变量，针对这两个变量变量构造一个二次规划问题，用解析方法求解该子问题，如此反复直到所有的解都满足该最优化问题的KKT条件。

**2. 算法描述**

表1 算法描述

|  |
| --- |
| **输入：**训练数据集  测试数据 |
| **过程：**   1. 初始化参数：   计算数据的核函数矩阵和，其中,   1. 选取优化变量：   ①选取第一个变量的过程为外层循环。交替地在”整个样本集上”和”非界样本子集上多次遍历”选取。  ②选取第二个变量的过程为内层循环。选择这一变量的原则是使优化过程中步长最大，加速收敛。即选取最大的。但是这样一来，同一个变量很可能被多次选择，因此，在部分迭代过程中也通过随机选取来获得第二个变量。   1. 求解两个变量的最优化问题，求得最优解，更新参数   为，并利用更新后的参数计算和，更新参数的计算公式如下：   1. 重复步骤(2)(3)直到精度ϵ范围内满足停机条件。取 2. 根据 计算 3. 输出数据所属类别 |
| **输出：**测试数据 所属类别 |

**3. 类设计**

表2 SVM\_Model类的方法

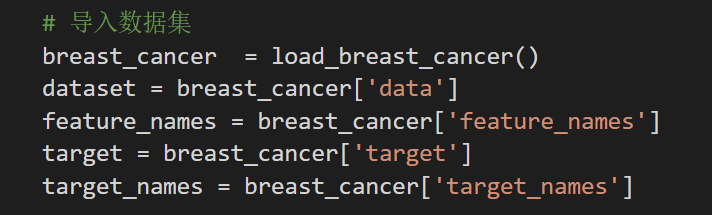
|  |  |
| --- | --- |
| **方法** | **描述** |
| init | 初始化模型参数 |
| svm\_fit | 训练svm模型 |
| predict | 对输入数据进行预测 |
| update\_am\_an | 更新smo算法的两个优化变量 |
| update\_b | 更新参数b |
| SMO | smo算法集成 |
| smo\_iter | smo算法单次迭代 |
| smo\_update | smo更新参数 |
| find\_an | 选择SMO算法的第二个优化变量 |
| KKT\_check | 检查KKT条件 |
| g\_x | 计算g(x) |
| set\_kernel | 设置核 |
| K\_xy | 计算核函数 |
| validate | 验证预测的精度 |
| cross\_validate | 对模型进行交叉验证 |

#### 五、实验步骤

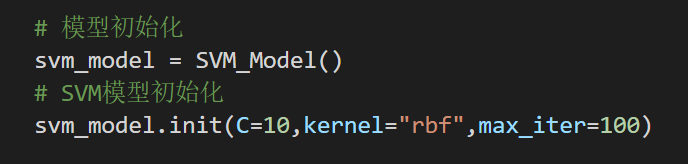
**实验数据简介：**

实验所用到的乳腺癌数据集总共569个样本数据，每个样本的特征维度为30维。

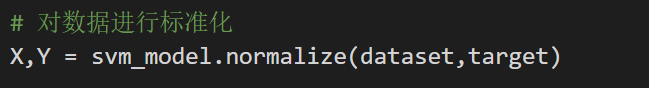
1. **载入乳腺癌数据**



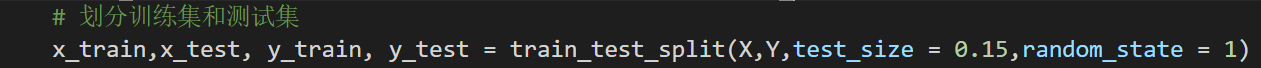
1. **建立SVM的模型，并初始化**



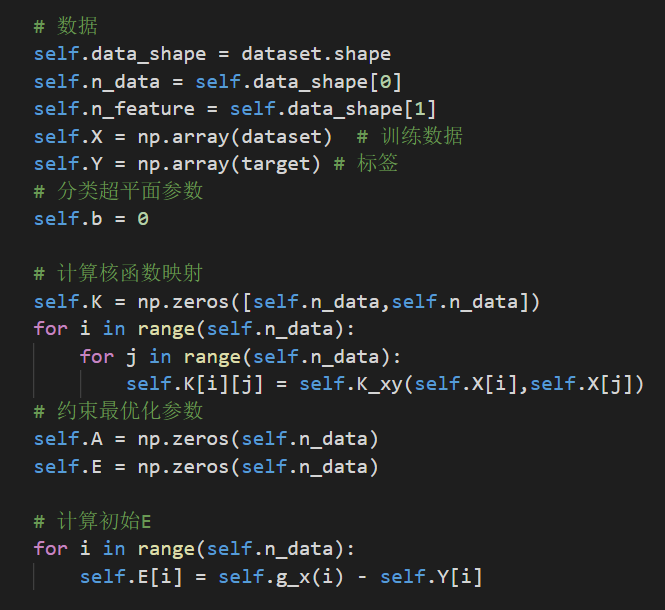
1. **对乳腺癌数据集进行标准化，以便能够正确的进行训练和分类**



1. **划分训练集和测试集**



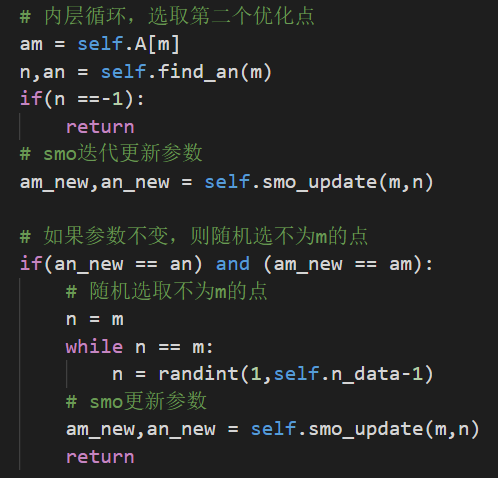
1. **训练模型——迭代**
   1. 将训练数据载入模型，同时初始化模型参数，计算核函数映射矩阵等



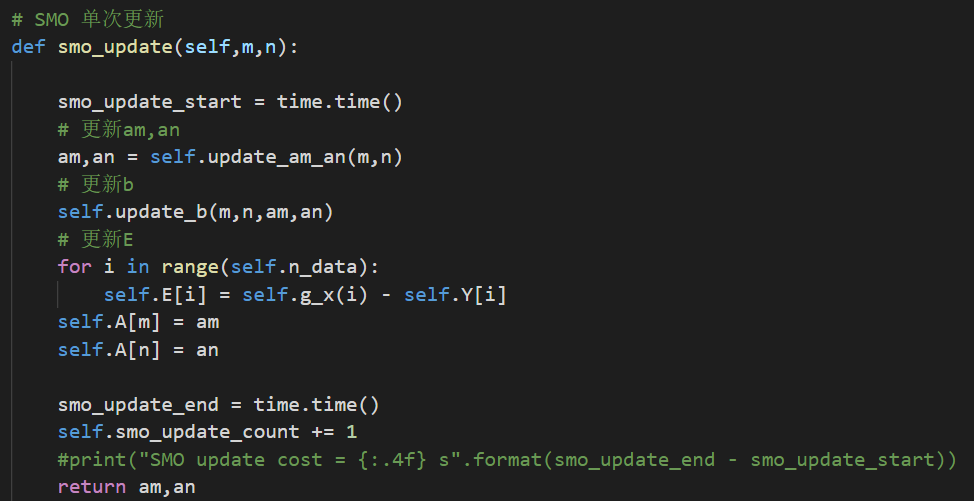
* 1. SMO算法外层循环，交替遍历所有样本和非界样本，选取第一个优化变量



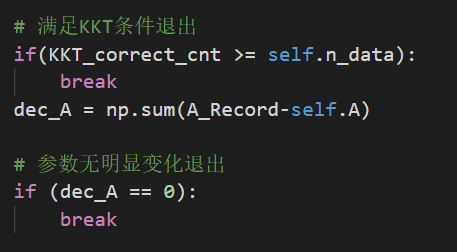
* 1. 内层循环，根据优化原则选取第二个优化变量，如果优化后参数无明显变化，则第二个优化变量采用随机选取



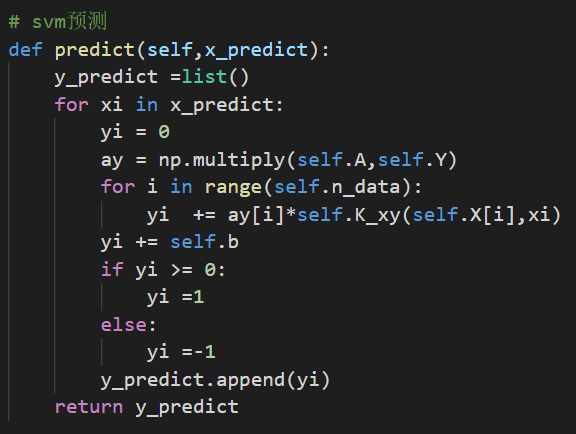
* 1. 根据选取的优化变量更新模型参数



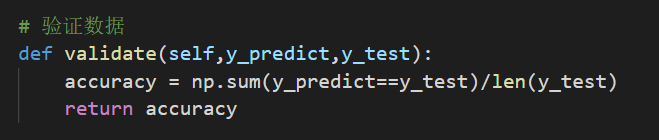
* 1. 当满足KKT条件，模型参数迭代过程中无明显变化或超出最大迭代次数时退出



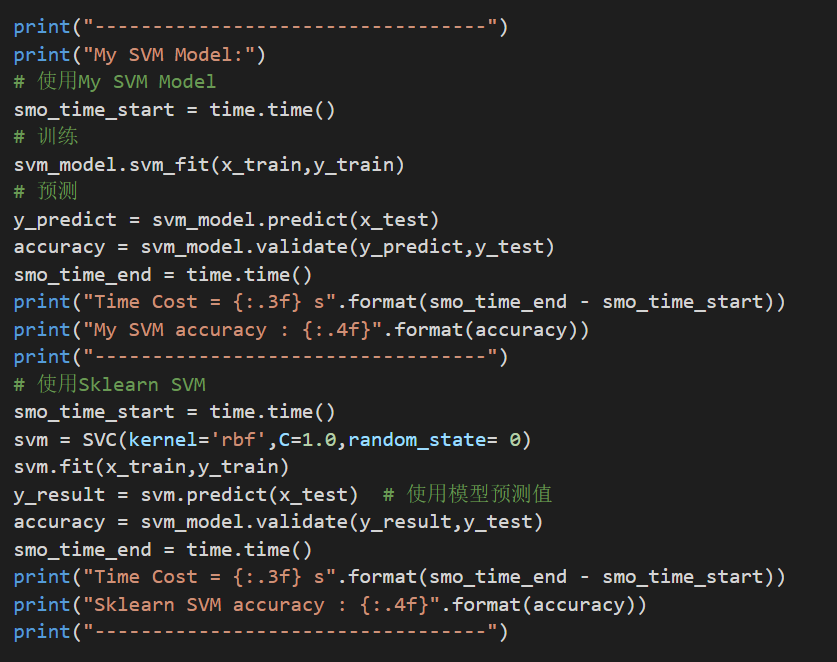
1. **对输入数据进行预测**



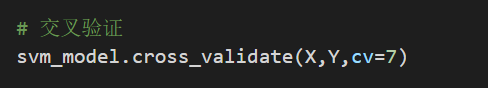
1. **对模型输入进行验证**



1. **与sklearn的svm模型进行比对**



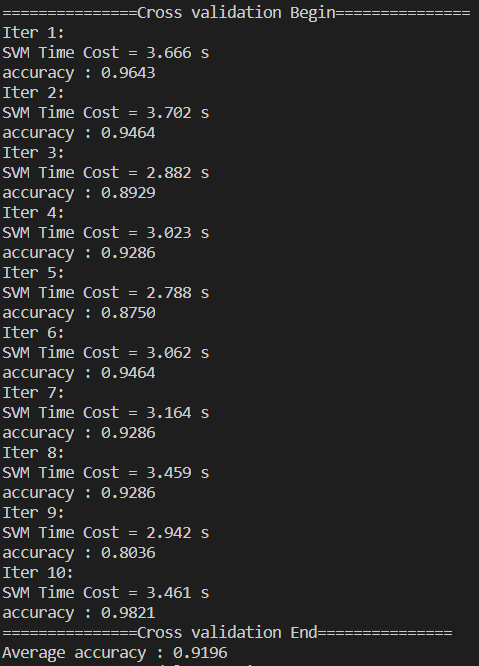
1. **对模型进行交叉验证，评价模型指标**



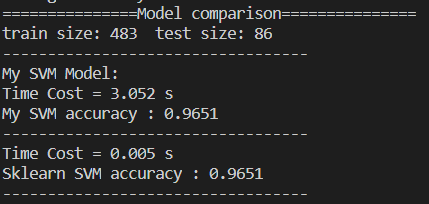
#### 五、实验结果

1. **使用RBF核(gamma = 1/30)**

交叉验证

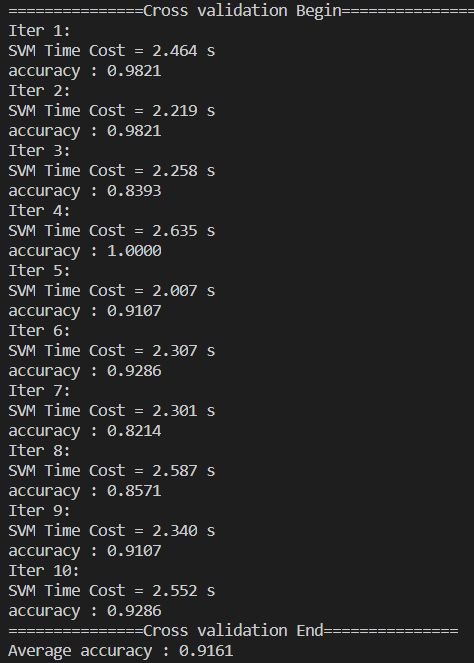


模型比较

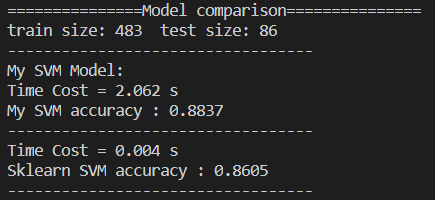


1. **使用多项式核(degree=3，coef0=0)**

交叉验证



模型比对



#### 附件

#### 实验代码：SMO算法

# -\*- coding:utf-8 -\*-

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_breast\_cancer

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

import numpy as np

import matplotlib.pyplot as plt

import math

from random import randint

import time

# SVM模型,使用SMO算法进行优化求解

class SVM\_Model:

    def SVM\_Model():

        self.init()

    # 初始化

    def init(self,C = np.inf,e\_p = 1e-6,KKT\_e=0.1,kernel = "linear",max\_iter = 100,gamma = "auto",coef0=0,degree=3):

        # 软间隔容忍因子

        self.C = C

        # 精度范围

        self.e\_p = e\_p

        # KKT条件容忍度

        self.KKT\_e = KKT\_e

        # 设置映射核函数

        self.kernel = dict()

        self.max\_iter = max\_iter

        # 设置核函数

        self.set\_kernel(kernel,gamma=gamma,coef0=coef0,degree=degree)

    # svm数据标准化

    def normalize(self,dataset,target):

        Y = np.array(target)

        for i in range(len(Y)):

            if Y[i] != 1:

                Y[i] = -1

        X = np.array(dataset)

        X -= np.mean(X,0)

        X /= np.std(X,0)

        return X,Y

    # svm训练

    def svm\_fit(self,dataset,target):

        # 数据

        self.data\_shape = dataset.shape

        self.n\_data = self.data\_shape[0]

        self.n\_feature = self.data\_shape[1]

        self.X = np.array(dataset)  # 训练数据

        self.Y = np.array(target) # 标签

        # 分类超平面参数

        self.b = 0

        # 计算核函数映射

        self.K = np.zeros([self.n\_data,self.n\_data])

        for i in range(self.n\_data):

            for j in range(self.n\_data):

                self.K[i][j] = self.K\_xy(self.X[i],self.X[j])

        # 约束最优化参数

        self.A = np.zeros(self.n\_data)

        self.E = np.zeros(self.n\_data)

        # 计算初始E

        for i in range(self.n\_data):

            self.E[i] = self.g\_x(i) - self.Y[i]

        # SMO算法选取两个a更新

        self.SMO(self.max\_iter)

    # svm预测

    def predict(self,x\_predict):

        y\_predict =list()

        for xi in x\_predict:

            yi = 0

            ay = np.multiply(self.A,self.Y)

            for i in range(self.n\_data):

                yi  += ay[i]\*self.K\_xy(self.X[i],xi)

            yi += self.b

            if yi >= 0:

                yi =1

            else:

                yi =-1

            y\_predict.append(yi)

        return y\_predict

    # SMO更新参数a1,a2

    def update\_am\_an(self,m,n):

        am = self.A[m]

        an = self.A[n]

        E1 = self.E[m]

        E2 = self.E[n]

        K11 = self.K[m,m]

        K22 = self.K[n,n]

        K12 = self.K[m,n]

        u = K11+K22 - 2\*K12

        u = max(self.e\_p,u)

        an += self.Y[n]\*(E1-E2)/u

        # 计算上下边界

        L = 0

        H = self.C

        if self.Y[m] == self.Y[n]:

            L = max(0, self.A[m] + self.A[n] - self.C)  # max(0,a2+a1-C)

            H = min(self.C, self.A[m] + self.A[n])   # min(C,a2+a1)

        else:

            L = max(0, self.A[n] - self.A[m]) # max(0,a2-a1)

            H = min(self.C, self.A[n] - self.A[m] + self.C)  # min(C,a2-a1+C)

        # 控制an范围

        if(H<L):

            print(" =============== H({}) < L({}) ===============".format(H,L))

            print("m = {} n = {}  a1 = {} a2 = {}  equal\_flag = {}".format(m,n,am,an,self.Y[m] == self.Y[n]))

            an = 0

        else:

            an = max(min(an,H),L)

        # 计算am

        am += self.Y[m]\*self.Y[n]\*(self.A[n]-an)

        # 控制am范围

        am = max(0,am)

        return am,an

    # SMO更新b

    def update\_b(self,m,n,a1,a2):

        E1 = self.E[m]

        E2 = self.E[n]

        K11 = self.K[m,m]

        K12 = self.K[m,n]

        K21 = self.K[n,m]

        K22 = self.K[n,n]

        y1 = self.Y[m]

        y2 = self.Y[n]

        # 计算b1

        b1 = -E1 - y1\*K11\*(a1 - self.A[m]) - y2\*K21\*(a2 - self.A[n]) + self.b

        # 计算b2

        b2 = -E2- y1\*K12\*(a1 - self.A[m]) - y2\*K22\*(a2 - self.A[n]) + self.b

        self.b = (b1+b2)/2

    # SMO算法求解约束最优化问题

    def SMO(self,max\_iter):

        # SMO算法迭代

        iter\_count = 0

        Out\_Loop\_flag = 0 #外层循环的flag

        # print('||===============SMO START===============||')

        while iter\_count<max\_iter:

            self.smo\_update\_count = 0

            # print('\n-----Iter {}-----'.format(iter\_count))

            iter\_time\_start = time.time()

            if(iter\_count % 5) == 0:

                Out\_Loop\_flag = 0 # 遍历所有样本

            else:

                Out\_Loop\_flag = 1 # 遍历非界样本

            iter\_count += 1

            # 首先遍历点m,n

            # 外层循环，找到最违反KKT条件的点m

            m = 0

            am = 0

            exceed\_error = 0

            error\_yg = 0

            # 保存外层循环的A

            A\_Record = self.A.copy()

            # 外层循环，选取所有违反KKT条件的参数，将其作为第一个优化参数

            for i in range(self.n\_data):

                if Out\_Loop\_flag:

                    ai = self.A[i]

                    # 遍历非界样本

                    if not self.KKT\_check(i,ai):

                        if (np.abs(ai)<self.e\_p):

                            self.smo\_iter(i)

                else:

                    # 遍历所有样本

                    ai = self.A[i]

                    if not self.KKT\_check(i,ai):

                        if (np.abs(ai)<self.e\_p):

                            self.smo\_iter(i)

            # 计算是否满足KKT条件

            KKT\_correct\_cnt = 0

            for i in range(self.n\_data):

                # 遍历所有样本

                ai = self.A[i]

                if self.KKT\_check(i,ai):

                    KKT\_correct\_cnt +=1

            # 显示KKT条件

            # print("KKT Check = ( {} / {} )".format(KKT\_correct\_cnt,self.n\_data))

            # print("Sum(aiyi) = {}".format(np.dot(self.A,self.Y)))

            iter\_time\_end = time.time()

            # print("Iter Time Cost = {:.3f} s".format(iter\_time\_end - iter\_time\_start))

            # print("Update Count = {:d} ".format(self.smo\_update\_count))

            # 满足KKT条件退出

            if(KKT\_correct\_cnt >= self.n\_data):

                break

            dec\_A = np.sum(A\_Record-self.A)

            # 参数无明显变化退出

            if (dec\_A == 0):

                break

            # print("Dec(A)= {}".format(dec\_A))

        # print("||===============SMO   END===============||")

    # SMO 单次迭代

    def smo\_iter(self,m):

        # 内层循环，选取第二个优化点

        am = self.A[m]

        n,an = self.find\_an(m)

        if(n ==-1):

            return

        # smo迭代更新参数

        am\_new,an\_new = self.smo\_update(m,n)

        # 如果参数不变，则随机选不为m的点

        if(an\_new == an) and (am\_new == am):

            # 随机选取不为m的点

            n = m

            while n == m:

                n = randint(1,self.n\_data-1)

            # smo更新参数

            am\_new,an\_new = self.smo\_update(m,n)

            return

    # SMO 单次更新

    def smo\_update(self,m,n):

        smo\_update\_start = time.time()

        # 更新am,an

        am,an = self.update\_am\_an(m,n)

        # 更新b

        self.update\_b(m,n,am,an)

        # 更新E

        for i in range(self.n\_data):

            self.E[i] = self.g\_x(i) - self.Y[i]

        self.A[m] = am

        self.A[n] = an

        smo\_update\_end = time.time()

        self.smo\_update\_count += 1

        #print("SMO update cost = {:.4f} s".format(smo\_update\_end - smo\_update\_start))

        return am,an

    # SMO 第二参数选取

    def find\_an(self,m):

        E1 = self.E[m]

        E2 = 0

        e\_f = 1

        n = -1

        an = 0

        if E1 < 0:

            # 取最大的E2

            e\_f = 1

        else:

            # 取最小的E2

            e\_f = -1

        for i in range(self.n\_data):

            Ei = self.E[i]

            if (i!=m) and (Ei\*e\_f >= E2\*e\_f):

                E2 =  Ei

                n = i

                an = self.A[n]

        return n,an

    # 核对是否满足KKT条件

    def KKT\_check(self,i,ai):

        KKT\_flag = False

        g = self.g\_x(i)

        yg = self.Y[i]\*g

        if  (np.abs(ai)<self.e\_p):

            KKT\_flag = (yg >= 1-self.KKT\_e)

        elif (ai>=self.e\_p) and (ai <=self.C-self.e\_p):

            KKT\_flag = (np.abs(yg-self.C) <self.e\_p)

        elif np.abs(ai-self.C)<self.e\_p:

            KKT\_flag = (yg <= 1 + self.KKT\_e)

        return KKT\_flag

    # 计算g(x)

    def g\_x(self,i):

        # g = w\*fi(x) + b = sumj(aj\*yj\*Kji) + b

        g = np.dot(np.multiply(self.A,self.Y),self.K[:,i]) + self.b

        return g

    # degree 当为poly核时多项式的最高次数

    # 设置映射核函数

    def set\_kernel(self,kernel = 'None',gamma = "auto",coef0=0,degree=3):

        if kernel == 'rbf':

            self.kernel['name'] = 'rbf'

            self.kernel['gamma'] = gamma

        elif kernel == 'linear':

            self.kernel['name'] = 'linear'

        elif kernel == 'poly':

            self.kernel['name'] = 'poly'

            self.kernel['degree'] = degree

            self.kernel['coef0'] = coef0

        else:

            self.kernel['name'] = 'None'

    # 计算核函数映射后的K(x,y)

    def K\_xy(self,x,y):

        K\_xy = 0

        if self.kernel['name'] == 'rbf':

            gamma = 0

            if self.kernel['gamma'] == 'auto':

                gamma = 1/self.n\_feature

            else:

                gamma = self.kernel['gamma']

            K\_xy = np.exp(-gamma\*np.linalg.norm(x-y)\*\*2)

        elif self.kernel['name'] == 'linear':

            K\_xy = np.dot(x,y)

        elif self.kernel['name'] == 'poly':

            degree = self.kernel['degree']

            coef0 = self.kernel['coef0']

            K\_xy = (np.dot(x,y) + coef0)\*\*degree

        else:

            K\_xy = np.dot(x,y)

        return K\_xy

    # 验证数据

    def validate(self,y\_predict,y\_test):

        accuracy = np.sum(y\_predict==y\_test)/len(y\_test)

        return accuracy

    # 交叉验证数据

    def cross\_validate(self,X,Y,cv = 10):

        n\_data = X.shape[0]

        n\_feature = X.shape[1]

        # 合并数据和标签

        DataSet = np.array(np.c\_[X,Y.T])

        # 打乱数据集

        np.random.shuffle(DataSet)

        # print(DataSet.shape)

        folds = list()

        # 分割数据集

        for k in range(cv):

            fold\_size = np.int(n\_data/cv)

            l\_index = max(k\*fold\_size,0)

            h\_index = min((k+1)\*fold\_size,n\_data)

            fold\_k =DataSet[l\_index:h\_index]

            folds.append(fold\_k)

        # 精度

        accuracy = 0

        print("===============Cross validation Begin===============")

        # 交叉验证

        for i in range(cv):

            # 划分数据集和测试集

            x\_train = np.array([])

            y\_train = np.array([])

            x\_test = np.array([])

            y\_test = np.array([])

            for k in range(cv):

                X\_k = folds[k][:,0:n\_feature]

                Y\_k = folds[k][:,n\_feature]

                if k != i:

                    if np.any(x\_train):

                        x\_train = np.r\_[x\_train,X\_k]

                        y\_train = np.r\_[y\_train,Y\_k]

                    else:

                        x\_train = X\_k

                        y\_train = Y\_k

                else:

                    x\_test = X\_k

                    y\_test = Y\_k

            # 计算用时

            smo\_time\_start = time.time()

            # 训练

            self.svm\_fit(x\_train,y\_train)

            # 预测

            y\_predict = self.predict(x\_test)

            # 测试

            accuracy\_k = self.validate(y\_predict,y\_test)

            accuracy += accuracy\_k

            smo\_time\_end = time.time()

            print("Iter {}:".format(i+1))

            print("SVM Time Cost = {:.3f} s".format(smo\_time\_end - smo\_time\_start))

            print("accuracy : {:.4f}".format(accuracy\_k))

        accuracy /= cv

        print("===============Cross validation End===============")

        print("Average accuracy : {:.4f}".format(accuracy))

def main():

    # 导入数据集

    breast\_cancer  = load\_breast\_cancer()

    dataset = breast\_cancer['data']

    feature\_names = breast\_cancer['feature\_names']

    target = breast\_cancer['target']

    target\_names = breast\_cancer['target\_names']

    # 模型初始化

    svm\_model = SVM\_Model()

    # SVM模型初始化

    svm\_model.init(C=10,kernel="poly",max\_iter=100)

    # 对数据进行标准化

    X,Y = svm\_model.normalize(dataset,target)

    # 交叉验证

    svm\_model.cross\_validate(X,Y,cv=10)

    print("===============Model comparison===============")

    # 划分训练集和测试集

    x\_train,x\_test, y\_train, y\_test = train\_test\_split(X,Y,test\_size = 0.15,random\_state = 1)

    print("train size: {:d}  test size: {:d}".format(len(x\_train),len(y\_test)))

    print("----------------------------------")

    print("My SVM Model:")

    # 使用My SVM Model

    smo\_time\_start = time.time()

    # 训练

    svm\_model.svm\_fit(x\_train,y\_train)

    # 预测

    y\_predict = svm\_model.predict(x\_test)

    accuracy = svm\_model.validate(y\_predict,y\_test)

    smo\_time\_end = time.time()

    print("Time Cost = {:.3f} s".format(smo\_time\_end - smo\_time\_start))

    print("My SVM accuracy : {:.4f}".format(accuracy))

    print("----------------------------------")

    # 使用Sklearn SVM

    smo\_time\_start = time.time()

    svm = SVC(kernel='poly',C=1.0,random\_state= 0,gamma='auto')

    svm.fit(x\_train,y\_train)

    y\_result = svm.predict(x\_test)  # 使用模型预测值

    accuracy = svm\_model.validate(y\_result,y\_test)

    smo\_time\_end = time.time()

    print("Time Cost = {:.3f} s".format(smo\_time\_end - smo\_time\_start))

    print("Sklearn SVM accuracy : {:.4f}".format(accuracy))

    print("----------------------------------")

if \_\_name\_\_ == '\_\_main\_\_':

    main()