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 $T=\{(x_1,y_1),\ldots,(x_N,y_N)\}$ 测试数据 $x^*=(x^{*(1)},\ldots,x^{*(M)})^T$
过程： （1）　初始化参数： $$W_0=0, k=0, \alpha^{(0)}=(0,\ldots,0)$$ 计算数据的核函数矩阵$K_{ij}=K(x_i,x_j)$和$E^{(0)}$，其中 $i=1,\ldots,N, j=1,\ldots,N$ （2）　选取优化变量$\alpha_m^{(k)}, \alpha_n^{(k)}$： 选取第一个变量$\alpha_m^{(k)}$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。

选取第二个变量$\alpha_n^{(k)}$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$|E_m - E_n|$最大的$\alpha_n^{(k)}$。但是这样一来，同一个变量$\alpha_n^{(k)}$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$\alpha_n^{(k)}$。

（3） 求解两个变量的最优化问题，求得最优解$\alpha_m^{(k+1)}, \alpha_n^{(k+1)}$，更新参数$\alpha^{(k)}$为$\alpha^{(k+1)}$，并利用更新后的参数计算$W_0^{(k+1)}$和$E^{(k+1)}$，更新参数的计算公式如下：

$$\alpha_n^{(k+1)} = \max\left(\min\left(\alpha_n^{(k)} + \frac{y_n(E_m^{(k)} - E_n^{(k)})}{K_{mm} + K_{nn} - 2K_{mn}}\right), \mathrm{L}\right)$$

$$\alpha_m^{(k+1)} = \alpha_m^{(k)} + y_m y_n(\alpha_n^{(k)} - \alpha_n^{(k+1)})$$

$$W_{0(1)}^{(k+1)} = -E_m^{(k)} + W_0^{(k)} + \left(\alpha_m^{(k)} - \alpha_m^{(k+1)}\right) y_m K_{mm} + (\alpha_n^{(k)} - \alpha_n^{(k+1)}) y_n K_{nm}$$

$$W_{0(2)}^{(k+1)} = -E_n^{(k)} + W_0^{(k)} + \left(\alpha_m^{(k)} - \alpha_m^{(k+1)}\right) y_m K_{mn} + (\alpha_n^{(k)} - \alpha_n^{(k+1)}) y_n K_{nn}$$

$$W_0^{(k+1)} = \frac{W_{0(1)}^{(k+1)} + W_{0(2)}^{(k+1)}}{2}$$

$$E_j^{(k+1)} = \mathrm{g}(y_j) - y_j = \left(\sum_{i=1}^{N} \alpha_i^{(k+1)} y_i K_{ij} + W_0^{(k+1)}\right) - y_j$$

（4） 重复步骤(2)(3)直到精度ϵ范围内满足停机条件。取$\widehat{\alpha} = \alpha^{(k+1)}$

（5） 根据 $\widehat{\alpha}$ 计算 $f(x^*)$

$$\mathrm{W}_0 = W_0^{(k+1)}$$

$$f(x^*) = \sum_{i=1}^{N} \widehat{\alpha}_i y_i K(x_i, x^*) + \mathrm{W}_0$$

（6） 输出数据所属类别y^*

$$y^* = \begin{cases} 1, & f(x^*) \geq 0 \\ -1, & f(x^*) < 0 \end{cases}$$

输出：测试数据$x^* = (x^{*(1)}, \dots, x^{*(M)})^T$ 所属类别 y^*

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

```
# 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']
```

2、 建立 SVM 的模型，并初始化

```
# 模型初始化
svm_model = SVM_Model()
# SVM模型初始化
svm_model.init(C=10,kernel="rbf",max_iter=100)
```

3、 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

```
# 对数据进行标准化
X,Y = svm_model.normalize(dataset,target)
```

4、 划分训练集和测试集

```python
# 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)
```

5、 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

```python
# 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]
```

② 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

```python
# 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)
```

③ 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

```python
# 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
```

④ 根据选取的优化变量更新模型参数

```
# 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
```

⑤ 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

```
# 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
```

6、对输入数据进行预测

```
# 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
```

7、 对模型输入进行验证

```
# 验证数据
def validate(self,y_predict,y_test):
    accuracy = np.sum(y_predict==y_test)/len(y_test)
    return accuracy
```

8、 与 sklearn 的 svm 模型进行比对

```
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='rbf',C=1.0,random_state= 0)
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("----------------------------------")
```

9、 对模型进行交叉验证，评价模型指标

```
# 交叉验证
svm_model.cross_validate(X,Y,cv=7)
```

五、实验结果

① 使用 RBF 核(gamma = 1/30)

交叉验证

```
===============Cross validation Begin===============
Iter 1:
SVM Time Cost = 3.666 s
accuracy : 0.9643
Iter 2:
SVM Time Cost = 3.702 s
accuracy : 0.9464
Iter 3:
SVM Time Cost = 2.882 s
accuracy : 0.8929
Iter 4:
SVM Time Cost = 3.023 s
accuracy : 0.9286
Iter 5:
SVM Time Cost = 2.788 s
accuracy : 0.8750
Iter 6:
SVM Time Cost = 3.062 s
accuracy : 0.9464
Iter 7:
SVM Time Cost = 3.164 s
accuracy : 0.9286
Iter 8:
SVM Time Cost = 3.459 s
accuracy : 0.9286
Iter 9:
SVM Time Cost = 2.942 s
accuracy : 0.8036
Iter 10:
SVM Time Cost = 3.461 s
accuracy : 0.9821
===============Cross validation End===============
Average accuracy : 0.9196
```

模型比较

```
==============Model comparison==============
train size: 483  test size: 86
----------------------------------
My SVM Model:
Time Cost = 3.052 s
My SVM accuracy : 0.9651
----------------------------------
Time Cost = 0.005 s
Sklearn SVM accuracy : 0.9651
----------------------------------
```

② 使用多项式核(degree=3，coef0=0)

交叉验证

```
==============Cross validation Begin==============
Iter 1:
SVM Time Cost = 2.464 s
accuracy : 0.9821
Iter 2:
SVM Time Cost = 2.219 s
accuracy : 0.9821
Iter 3:
SVM Time Cost = 2.258 s
accuracy : 0.8393
Iter 4:
SVM Time Cost = 2.635 s
accuracy : 1.0000
Iter 5:
SVM Time Cost = 2.007 s
accuracy : 0.9107
Iter 6:
SVM Time Cost = 2.307 s
accuracy : 0.9286
Iter 7:
SVM Time Cost = 2.301 s
accuracy : 0.8214
Iter 8:
SVM Time Cost = 2.587 s
accuracy : 0.8571
Iter 9:
SVM Time Cost = 2.340 s
accuracy : 0.9107
Iter 10:
SVM Time Cost = 2.552 s
accuracy : 0.9286
==============Cross validation End==============
Average accuracy : 0.9161
```

模型比对

```
==============Model comparison==============
train size: 483  test size: 86
----------------------------------
My SVM Model:
Time Cost = 2.062 s
My SVM accuracy : 0.8837
----------------------------------
Time Cost = 0.004 s
Sklearn SVM accuracy : 0.8605
----------------------------------
```

附件

实验代码：SMO 算法

```python
# -*- 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_ite
r = 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_upd
ate_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)
```

```python
        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()
```

[redacted]