BÀI THỰC HÀNH PHẦN KERNEL SUPPORT VECTOR MACHINE

Ví dụ 1: Chúng ta vẫn bắt đầu với một ví dụ trên dữ liệu nhân tạo. Trước hết ta tạo một bộ dữ liệu có 200 mẫu (N=200) phân đều vào 02 lớp – mỗi lớp 100 phần tử. Mỗi mẫu có 2 chiều (d=2). Dữ liệu sẽ được chúng ta tạo bởi make_circles trong thư viện sklearn.dataset sao cho đường biên giữa các lớp là không tuyến tính.

Tiếp theo sử dụng thư viện sklearn để minh họa phương pháp Kernel Vector Machine với các hàm nhân khác nhau và tiến hành thực hiện phân lớp trên bộ dữ liệu nói trên.

Cách 1: Tự xây dựng các module từ thư viện Numpy:

Trước hết chúng ta cần xây dựng các loại hàm nhân (kernel) khác nhau (Linear, Poly, Gaussian hay rbf):

```
import numpy as np
from numpy import linalg

from sklearn.datasets import make_circles
import matplotlib.pyplot as plt

import cvxopt
import cvxopt.solvers

def linear_kernel(x1, x2):
    return np.dot(x1, x2)

def polynomial_kernel(x, y, gamma = 1, r = 1, d=3):
    return (r + gamma*np.dot(x, y)) ** d

def gaussian_kernel(x, y, sigma=5.0):
    # gamma = 1.0/(2 * (sigma ** 2))
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))
```

Tiếp theo chúng ta xây dựng lớp SVM, trong đó có phần tạo dữ liệu (sẽ sử dụng circle của sklearn).

```
class SVM(object):

    def __init__(self, kernel=linear_kernel, C=None):
        self.kernel = kernel
        self.C = C
        if self.C is not None: self.C = float(self.C)

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Solve Quadratic Programming problem
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = self.kernel(X[i], X[j])

        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
```

```
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers a = \lambda
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
        print("%d support vectors out of %d points" % (len(self.a), n_samples))

        # Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
        self.b /= len(self.a)

        # Weight vector
        if self.kernel == linear_kernel:
            self.w = np.zeros(n_features)
            for n in range(len(self.a)):
                self.w += self.a[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None

    def project(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
                    s += a * sv_y * self.kernel(X[i], sv)
                y_predict[i] = s
            return y_predict + self.b

    def predict(self, X):
        return np.sign(self.project(X))

if __name__ == "__main__":
    import pylab as pl

    def gen_lin_separable_data():
        # generate training data in the 2-d case

        return X1, y1, X2, y2

    def gen_non_lin_separable_data():


        return X1, y1, X2, y2

    def gen_lin_separable_overlap_data():

        return X1, y1, X2, y2
```

```
    def split_train(X1, y1, X2, y2):
        X1_train = X1[:90]
        y1_train = y1[:90]
        X2_train = X2[:90]
        y2_train = y2[:90]
        X_train = np.vstack((X1_train, X2_train))
        y_train = np.hstack((y1_train, y2_train))
        return X_train, y_train

    def split_test(X1, y1, X2, y2):
        X1_test = X1[90:]
        y1_test = y1[90:]
        X2_test = X2[90:]
        y2_test = y2[90:]
        X_test = np.vstack((X1_test, X2_test))
        y_test = np.hstack((y1_test, y2_test))
        return X_test, y_test

    def plot_margin(X1_train, X2_train, clf):

    def test_linear():

    def test_non_linear():

    def test_soft():


    #test_linear()
    test_non_linear()
```

Các phương thức vẽ đồ thị, tạo dữ liệu các bạn có thể tự đọc trong tệp nguồn đính kèm.

Cách 2: Sử dụng thư viện sklearn:

Đoạn lệnh tạo dữ liệu

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.datasets import make_circles

# Generate dataset and targets
X, Y = make_circles(n_samples = 200, noise = 0.02)

# visualizing data
plt.scatter(X[:, 0], X[:, 1], c = Y, marker = '.')
plt.show()
```

Đoạn lệnh gọi thư viện và thực hiện quá trình phân loại, sau đó vẽ ra đường biên để quan sát

```
fignum = 1

# fit the model
for kernel in ('linear','sigmoid', 'poly', 'rbf'):
    clf = svm.SVC(kernel=kernel, gamma=1, coef0 = 1)
    clf.fit(X, Y)
    with PdfPages(kernel + '3.pdf') as pdf:
        # plot the line, the points, and the nearest vectors to the plane
        fig, ax = plt.subplots()
        plt.figure(fignum, figsize=(5, 5))
        plt.clf()
```

```
        plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=80,
                    facecolors='None')
        plt.plot(X[Y==0, 0], X[Y==0, 1], 'bs', markersize = 2)
        plt.plot(X[Y==1, 0], X[Y==1, 1], 'ro', markersize = 2)

        plt.axis('tight')
        x_min = -1.5
        x_max = 1.5
        y_min = -1.5
        y_max = 1.5

        XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
        Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(XX.shape)
        plt.figure(fignum, figsize=(5, 5))
        CS = plt.contourf(XX, YY, np.sign(Z), 200, cmap='jet', alpha = .2)
        plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                    levels=[-.5, 0, .5])
        plt.title(kernel, fontsize = 15)
        plt.xlim(x_min, x_max)
        plt.ylim(y_min, y_max)

        plt.xticks(())
        plt.yticks(())
        fignum = fignum + 1
        pdf.savefig()
plt.show()
```

Bài tập 1. Sử dụng thư viện để đưa ra độ chính xác Accuracy và ma trận nhầm lẫn Confusion Matrix theo mỗi phương pháp (Tham khảo code từ các bài trước).

Ví dụ 2. Trong ví dụ 2, chúng ta sử dụng dữ liệu kiểm tra chất lượng dự đoán liệu các vi mạch từ một nhà máy chế tạo có đảm bảo chất lượng hay không. Tệp dữ liệu có trong file ex2data2.txt đính kèm. Dữ liệu quan sát gồm 2 chiều, đầu ra Y có dạng 0-1.

Dưới đây là đoạn lệnh đọc dữ liệu và vẽ ra màn hình

```
import matplotlib.pyplot as plt
from sklearn import svm
from matplotlib.backends.backend_pdf import PdfPages

# visualizing data
mask = Y == 1
passed = plt.scatter(X[mask][0].values, X[mask][1].values)
failed = plt.scatter(X[~mask][0].values, X[~mask][1].values)
plt.xlabel('Microchip Test1')
plt.ylabel('Microchip Test2')
plt.legend((passed, failed), ('Passed', 'Failed'))
plt.show()
```

Tương tự Ví dụ 1, chúng ta sẽ sử dụng các nhân có trong thư viện sklearn để phân loại dữ liệu

```
fignum = 1

# fit the model
for kernel in ('linear','sigmoid', 'poly', 'rbf'):
    clf = svm.SVC(kernel=kernel, gamma=1, coef0 = 1)
    clf.fit(X, Y)
```

```
    with PdfPages(kernel + '3.pdf') as pdf:
        # plot the line, the points, and the nearest vectors to the plane
        fig, ax = plt.subplots()
        plt.figure(fignum, figsize=(5, 5))
        plt.clf()

        plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=80,
                    facecolors='None')
        plt.plot(X[mask][0].values, X[mask][1].values, 'bs', markersize = 2)
        plt.plot(X[~mask][0].values, X[~mask][1].values, 'ro', markersize = 2)

        plt.axis('tight')
        x_min = -1.5
        x_max = 1.5
        y_min = -1.5
        y_max = 1.5

        XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
        Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(XX.shape)
        plt.figure(fignum, figsize=(5, 5))
        CS = plt.contourf(XX, YY, np.sign(Z), 200, cmap='jet', alpha = .2)
        plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                    levels=[-.5, 0, .5])
        plt.title(kernel, fontsize = 15)
        plt.xlim(x_min, x_max)
        plt.ylim(y_min, y_max)

        plt.xticks(())
        plt.yticks(())
        fignum = fignum + 1
        pdf.savefig()
plt.show()
```

Bài tập tự thực hành 1. Cho tệp dữ liệu đính kèm ARgender.mat, đoạn lệnh sau đây sẽ đọc tệp dữ liệu đó và chia làm hai phần dữ liệu Training và Test Data. Chú ý sửa lại đường dẫn đến file dữ liệu theo thiết lập của các bạn:

```
import scipy.io as sio
from sklearn.svm import SVC

A = sio.loadmat('D:\\Teach_n_Train\\Machine Learning\\code\\data\\ARgender.mat')
X_train = A['Y_train'].T
X_test = A['Y_test'].T
print(X_train.shape)
N = 700
y_train = A['label_train'].reshape(N)
y_test = A['label_test'].reshape(N)
```

Hãy tham khảo các ví dụ ở trên, sau đó thực hiện train với mô hình Kernel SVM - ứng với các nhân khác nhau – trên tập dữ liệu training.

Sau đó hãy thực hiện dự đoạn (cùng mô hình) cho tập dữ liệu Test và dùng thư viện để đánh giá độ chính xác của mỗi trường hợp.

Bài tập tự thực hành 2. Cho tệp dữ liệu đính kèm dataset.csv, đoạn lệnh sau đây sẽ đọc tệp dữ liệu đó và chia làm hai phần dữ liệu Training và Test Data. Chú ý sửa lại đường dẫn đến file dữ liệu theo thiết lập của các bạn:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd

dataset = pd.read_csv('D:\\Teach_n_Train\\Machine Learning\\code\\data\\dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.model_selection import train_test_split
XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Chuyển vị ma trận dữ liệu do ta đọc bằng Pandas => mỗi record là 1 dong
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
XTrain = sc.fit_transform(XTrain)
XTest = sc.transform(XTest)
```

Hãy thực hiện các yêu cầu như với **Bài tập tự thực hành** 1.

Bài tập tự thực hành 3. Hãy thực hiện các yêu cầu như hai bài tập tự thực hành 1 và 2 với bài toán phân loại tập dữ liệu ung thư vú. Dữ liệu này có sẵn trong sklearn.dataset, chúng ta có thể dùng đoạn lệnh sau để tải dữ liệu từ thư viện này và hiện thị các trường của dữ liệu để dễ theo dõi:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
plt.style.use('ggplot')

# Breast cancer dataset for classification
data = load_breast_cancer()
print (data.feature_names)
print (data.target_names)
```

Dữ liệu cũng có thể đọc từ tệp data.csv đính kèm. Chú ý đầu ra (Y) nguyên bản của dữ liệu là 02 lớp ['malignant' 'benign'] (text) do đó ta cần kỹ thuật để chuyển chúng về dạng 0, 1 (tham khảo các bài đã có).