一、PCA 程序

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

np.seterr(divide='ignore', invalid='ignore')

from sklearn.datasets.samples_generator import make_blobs
'''X为样本特征，Y为样本簇类别， 共1000个样本，每个样本3个特征，共4个簇'''
X, y = make_blobs(n_samples=10000, n_features=3,
                  centers=[[3, 3, 3], [0,0,0], [1,1,1], [2,2,2]],
                  cluster_std=[0.2, 0.1, 0.2, 0.2], random_state =9)
fig = plt.figure(1)
ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=30, azim=20)
plt.scatter(X[:, 0], X[:, 1], X[:, 2], marker='o')

'''先不降维，只对数据进行投影，看看投影后的三个维度的方差分布'''
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca.fit(X)
print (pca.explained_variance_ratio_)
print (pca.explained_variance_)
print ('--------------------------------')

'''从三维降到2维'''
pca = PCA(n_components=2)
pca.fit(X)
print (pca.explained_variance_ratio_)
print (pca.explained_variance_)
print ('--------------------------------')
```

```python
'''转化后的数据分布'''
fig = plt.figure(2)
X_new = pca.transform(X)
plt.scatter(X_new[:, 0], X_new[:, 1],marker='o')
plt.show()

'''现在不直接指定降维的维度，而指定降维后的主成分方差和比例。'''
pca = PCA(n_components=0.95)
pca.fit(X)
print (pca.explained_variance_ratio_)
print (pca.explained_variance_)
print (pca.n_components_)
print ('--------------------------------')

pca = PCA(n_components=0.99)
pca.fit(X)
print (pca.explained_variance_ratio_)
print (pca.explained_variance_)
print (pca.n_components_)
print ('--------------------------------')
```

二、利用 sklearn 库和 iris 数据集实现分类

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import  train_test_split
import matplotlib.pyplot as plt


# data
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, [0, 1, -1]])
    for i in range(len(data)):
        if data[i,-1] == 0:
            data[i,-1] = -1
    # print(data)
    return data[:,:2], data[:,-1]


X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)


plt.scatter(X[:50,0],X[:50,1], label='0')
plt.scatter(X[50:,0],X[50:,1], label='1')
plt.legend()

from sklearn.svm import SVC
clf = SVC()
clf.fit(X_train, y_train)

print (clf.score(X_test, y_test))
```

三、自己实现 SVM 算法

分离超平面：$w^T x + b = 0$

点到直线距离：$r = \frac{|w^T x+b|}{||w||_2}$

$||w||_2$为2-范数：$||w||_2 = \sqrt[2]{\sum_{i=1}^{m} w_i^2}$

直线为超平面，样本可表示为：

$w^T x + b \ \geq +1$

$w^T x + b \ \leq +1$

margin:

函数间隔：$label(w^T x + b) \ or \ y_i(w^T x + b)$

几何间隔：$r = \frac{label(w^T x+b)}{||w||_2}$，当数据被正确分类时，几何间隔就是点到超平面的距离

为了求几何间隔最大，SVM基本问题可以转化为求解:($\frac{r^*}{||w||}$ 为几何间隔，(r^* 为函数间隔)

$$\max \ \frac{r^*}{||w||}$$

$$(subject\ to)\ y_i(w^T x_i + b) \geq r^*,\ i = 1, 2, .., m$$

分类点几何间隔最大，同时被正确分类。但这个方程并非凸函数求解，所以要先①将方程转化为凸函数，②用拉格朗日乘子法和KKT条件求解对偶问题。

①转化为凸函数：

先令$r^* = 1$，方便计算（参照衡量，不影响评价结果）

$$\max \ \frac{1}{||w||}$$

$$s.t.\ \ y_i(w^T x_i + b) \geq 1,\ i = 1, 2, .., m$$

再将$\max \ \frac{1}{||w||}$转化成$\min \ \frac{1}{2}||w||^2$求解凸函数，1/2是为了求导之后方便计算。

$$\min \ \frac{1}{2}||w||^2$$

$$s.t.\ \ y_i(w^T x_i + b) \geq 1,\ i = 1, 2, .., m$$

②用拉格朗日乘子法和KKT条件求解最优值：

$$\min \ \frac{1}{2}||w||^2$$

$$s.t.\ \ -y_i(w^T x_i + b) + 1 \leq 0,\ i = 1, 2, .., m$$

整合成：

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i(-y_i(w^T x_i + b) + 1)$$

推导：$\min \ f(x) = \min\max \ L(w, b, \alpha) \geq \max\min \ L(w, b, \alpha)$

根据KKT条件：

$$\frac{\partial}{\partial w} L(w, b, \alpha) = w - \sum \alpha_i y_i x_i = 0,\ w = \sum \alpha_i y_i x_i$$

$$\frac{\partial}{\partial b} L(w, b, \alpha) = \sum \alpha_i y_i = 0$$

带入$L(w, b, \alpha)$

$$\begin{aligned}
\min \ L(w, b, \alpha) &= \tfrac{1}{2}||w||^2 + \textstyle\sum_{i=1}^{m} \alpha_i(-y_i(w^T x_i + b) + 1) \\
&= \tfrac{1}{2}w^T w - \textstyle\sum_{i=1}^{m} \alpha_i y_i w^T x_i - b\sum_{i=1}^{m} \alpha_i y_i + \sum_{i=1}^{m} \alpha_i \\
&= \tfrac{1}{2}w^T \textstyle\sum \alpha_i y_i x_i - \sum_{i=1}^{m} \alpha_i y_i w^T x_i + \sum_{i=1}^{m} \alpha_i \\
&= \textstyle\sum_{i=1}^{m} \alpha_i - \tfrac{1}{2}\sum_{i=1}^{m} \alpha_i y_i w^T x_i \\
&= \textstyle\sum_{i=1}^{m} \alpha_i - \tfrac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i x_j)
\end{aligned}$$

再把max问题转成min问题：

$\max \ \ \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i x_j) = \min \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i x_j) - \sum_{i=1}^{m} \alpha_i$

$s.t.\ \ \sum_{i=1}^{m} \alpha_i y_i = 0,$

$\alpha_i \geq 0, i = 1, 2, \ldots, m$

以上为SVM对偶问题的对偶形式

kernel

在低维空间计算获得高维空间的计算结果，也就是说计算结果满足高维（满足高维，才能说明高维下线性可分）。

soft margin & slack variable

引入松弛变量$\xi \geq 0$，对应数据点允许偏离的functional margin 的量。

目标函数：$\min\ \frac{1}{2}||w||^2 + C\sum\xi_i \qquad s.t.\ \ y_i(w^Tx_i + b) \geq 1 - \xi_i$

对偶问题：

$$\max\ \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j (x_i x_j) = \min\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j (x_i x_j) - \sum_{i=1}^{m}\alpha_i$$

$$s.t.\ \ C \geq \alpha_i \geq 0, i = 1, 2, \ldots, m \qquad \sum_{i=1}^{m}\alpha_i y_i = 0,$$

Sequential Minimal Optimization

首先定义特征到结果的输出函数：$u = w^T x + b.$

因为$w = \sum\alpha_i y_i x_i$

有$u = \sum y_i\alpha_i K(x_i, x) - b$

$$\max \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j < \phi(x_i)^T, \phi(x_j) >$$

$$s.t.\ \ \sum_{i=1}^{m}\alpha_i y_i = 0,$$

$$\alpha_i \geq 0, i = 1, 2, \ldots, m$$

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import  train_test_split

'''data'''
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, [0, 1, -1]])
    for i in range(len(data)):
        if data[i,-1] == 0:
            data[i,-1] = -1
    '''print(data)'''
    return data[:,:2], data[:,-1]

X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

```python
class SVM:
    def __init__(self, max_iter=100, kernel='linear'):
        self.max_iter = max_iter
        self._kernel  = kernel

    def init_args(self, features, labels):
        self.m, self.n = features.shape
        self.X = features
        self.Y = labels
        self.b = 0.0

        '''将Ei保存在一个列表里'''
        self.alpha = np.ones(self.m)
        self.E = [self._E(i) for i in range(self.m)]
        '''松弛变量'''
        self.C = 1.0

    def _KKT(self, i):
        y_g = self._g(i)*self.Y[i]
        if self.alpha[i] == 0:
            return y_g >= 1
        elif 0 < self.alpha[i] < self.C:
            return y_g == 1
        else:
            return y_g <= 1

    '''g(x)预测值，输入xi（X[i]）'''
    def _g(self, i):
        r = self.b
        for j in range(self.m):
            r += self.alpha[j]*self.Y[j]*self.kernel(self.X[i], self.X[j])
        return r
```

```python
    '''核函数'''
    def kernel(self, x1, x2):
        if self._kernel == 'linear':
            return sum([x1[k]*x2[k] for k in range(self.n)])
        elif self._kernel == 'poly':
            return (sum([x1[k]*x2[k] for k in range(self.n)]) + 1)**2

        return 0

    '''E（x）为g(x)对输入x的预测值和y的差'''
    def _E(self, i):
        return self._g(i) - self.Y[i]

    def _init_alpha(self):
        '''外层循环首先遍历所有满足0<a<C的样本点，检验是否满足KKT'''
        index_list = [i for i in range(self.m) if 0 < self.alpha[i] < self.C]
        '''否则遍历整个训练集'''
        non_satisfy_list = [i for i in range(self.m) if i not in index_list]
        index_list.extend(non_satisfy_list)

        for i in index_list:
            if self._KKT(i):
                continue

            E1 = self.E[i]
            '''如果E2是+，选择最小的；如果E2是负的，选择最大的'''
            if E1 >= 0:
                j = min(range(self.m), key=lambda x: self.E[x])
            else:
                j = max(range(self.m), key=lambda x: self.E[x])
            return i, j
```

```python
    def _compare(self, _alpha, L, H):
        if _alpha > H:
            return H
        elif _alpha < L:
            return L
        else:
            return _alpha

    def fit(self, features, labels):
        self.init_args(features, labels)

        for t in range(self.max_iter):
            '''train'''
            i1, i2 = self._init_alpha()

            '''边界'''
            if self.Y[i1] == self.Y[i2]:
                L = max(0, self.alpha[i1]+self.alpha[i2]-self.C)
                H = min(self.C, self.alpha[i1]+self.alpha[i2])
            else:
                L = max(0, self.alpha[i2]-self.alpha[i1])
                H = min(self.C, self.C+self.alpha[i2]-self.alpha[i1])

            E1 = self.E[i1]
            E2 = self.E[i2]
            '''eta=K11+K22-2K12'''
            eta = self.kernel(self.X[i1], self.X[i1]) \
            + self.kernel(self.X[i2], self.X[i2]) \
            - 2*self.kernel(self.X[i1], self.X[i2])
            if eta <= 0:
                '''print('eta <= 0')'''
                continue
```

```python
            alpha2_new_unc = self.alpha[i2] + self.Y[i2] * (E2 - E1) / eta
            alpha2_new = self._compare(alpha2_new_unc, L, H)

            alpha1_new = self.alpha[i1] + self.Y[i1] * self.Y[i2] * (self.alpha[i2] - alpha2_new)

            b1_new = -E1 - self.Y[i1] * self.kernel(self.X[i1], self.X[i1]) \
            * (alpha1_new-self.alpha[i1]) - self.Y[i2] * self.kernel(self.X[i2], self.X[i1]) \
            * (alpha2_new-self.alpha[i2])+ self.b
            b2_new = -E2 - self.Y[i1] * self.kernel(self.X[i1], self.X[i2]) \
            * (alpha1_new-self.alpha[i1]) - self.Y[i2] * self.kernel(self.X[i2], self.X[i2]) \
            * (alpha2_new-self.alpha[i2])+ self.b

            if 0 < alpha1_new < self.C:
                b_new = b1_new
            elif 0 < alpha2_new < self.C:
                b_new = b2_new
            else:
                '''选择中点'''
                b_new = (b1_new + b2_new) / 2

            '''更新参数'''
            self.alpha[i1] = alpha1_new
            self.alpha[i2] = alpha2_new
            self.b = b_new

            self.E[i1] = self._E(i1)
            self.E[i2] = self._E(i2)
        return 'train done!'
```

```python
    def predict(self, data):
        r = self.b
        for i in range(self.m):
            r += self.alpha[i] * self.Y[i] * self.kernel(data, self.X[i])

        return 1 if r > 0 else -1

    def score(self, X_test, y_test):
        right_count = 0
        for i in range(len(X_test)):
            result = self.predict(X_test[i])
            if result == y_test[i]:
                right_count += 1
        return right_count / len(X_test)

    def _weight(self):
        # linear model
        yx = self.Y.reshape(-1, 1)*self.X
        self.w = np.dot(yx.T, self.alpha)
        return self.w

svm = SVM(max_iter=200)

svm.fit(X_train, y_train)

print (svm.score(X_test, y_test))
```

四、自己实现 SVM 算法-2

添加新的库 cvxopt
打开“Anaconda Prompt”，输入 conda install cvxopt

```python
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers

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

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

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

```

```
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

        '''Gram matrix'''
        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'''
        '''
         数组的flatten和ravel方法将数组变为一个一维向量（铺平数组）。
         flatten方法总是返回一个拷贝后的副本，
         而ravel方法只有当有必要时才返回一个拷贝后的副本
         （所以该方法要快得多，尤其是在大数组上进行操作时）
       '''
        a = np.ravel(solution['x'])
        '''Support vectors have non zero lagrange multipliers'''
        '''
        这里a>1e-5就将其视为非零
        '''
        sv = a > 1e-5       # return a list with bool values
        ind = np.arange(len(a))[sv]  # sv's index
        self.a = a[sv]
        self.sv = X[sv]  # sv's data
        self.sv_y = y[sv]  # sv's labels
        print("%d support vectors out of %d points" % (len(self.a), n_samples))

        '''Intercept'''
        '''
        这里相当于对所有的支持向量求得的b取平均值
        '''
        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)):
                '''linear_kernel相当于在原空间，故计算w不用映射到feature space'''
                self.w += self.a[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None

    def project(self, X):
        # w有值，即kernel function 是 linear_kernel，直接计算即可
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        # w is None --> 不是linear_kernel,w要重新计算
        # 这里没有去计算新的w（非线性情况不用计算w）,直接用kernel matrix计算预测结果
        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
        mean1 = np.array([0, 2])
        mean2 = np.array([2, 0])
        cov = np.array([[0.8, 0.6], [0.6, 0.8]])
        X1 = np.random.multivariate_normal(mean1, cov, 100)
        y1 = np.ones(len(X1))
        X2 = np.random.multivariate_normal(mean2, cov, 100)
        y2 = np.ones(len(X2)) * -1
        return X1, y1, X2, y2

    def gen_non_lin_separable_data():
        mean1 = [-1, 2]
        mean2 = [1, -1]
        mean3 = [4, -4]
        mean4 = [-4, 4]
        cov = [[1.0,0.8], [0.8, 1.0]]
        X1 = np.random.multivariate_normal(mean1, cov, 50)
        X1 = np.vstack((X1, np.random.multivariate_normal(mean3, cov, 50)))
        y1 = np.ones(len(X1))
        X2 = np.random.multivariate_normal(mean2, cov, 50)
        X2 = np.vstack((X2, np.random.multivariate_normal(mean4, cov, 50)))
        y2 = np.ones(len(X2)) * -1
        return X1, y1, X2, y2
```

```python
def gen_lin_separable_overlap_data():
    # generate training data in the 2-d case
    mean1 = np.array([0, 2])
    mean2 = np.array([2, 0])
    cov = np.array([[1.5, 1.0], [1.0, 1.5]])
    X1 = np.random.multivariate_normal(mean1, cov, 100)
    y1 = np.ones(len(X1))
    X2 = np.random.multivariate_normal(mean2, cov, 100)
    y2 = np.ones(len(X2)) * -1
    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
```

```python
# 仅仅在Linears使用此函数作图，即w存在时
def plot_margin(X1_train, X2_train, clf):
    def f(x, w, b, c=0):
        # given x, return y such that [x,y] in on the line
        # w.x + b = c
        return (-w[0] * x - b + c) / w[1]

    pl.plot(X1_train[:,0], X1_train[:,1], "ro")
    pl.plot(X2_train[:,0], X2_train[:,1], "bo")
    pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")

    # w.x + b = 0
    a0 = -4; a1 = f(a0, clf.w, clf.b)
    b0 = 4; b1 = f(b0, clf.w, clf.b)
    pl.plot([a0,b0], [a1,b1], "k")

    # w.x + b = 1
    a0 = -4; a1 = f(a0, clf.w, clf.b, 1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, 1)
    pl.plot([a0,b0], [a1,b1], "k--")

    # w.x + b = -1
    a0 = -4; a1 = f(a0, clf.w, clf.b, -1)
    b0 = 4; b1 = f(b0, clf.w, clf.b, -1)
    pl.plot([a0,b0], [a1,b1], "k--")

    pl.axis("tight")
    pl.show()
```

```python
def plot_contour(X1_train, X2_train, clf):
    # 作training sample数据点的图
    pl.plot(X1_train[:,0], X1_train[:,1], "ro")
    pl.plot(X2_train[:,0], X2_train[:,1], "bo")
    # 做support vectors 的图
    pl.scatter(clf.sv[:,0], clf.sv[:,1], s=100, c="g")
    X1, X2 = np.meshgrid(np.linspace(-6,6,50), np.linspace(-6,6,50))
    X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
    Z = clf.project(X).reshape(X1.shape)
    # pl.contour做等值线图
    pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')
    pl.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1, origin='lower')
    pl.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1, origin='lower')

    pl.axis("tight")
    pl.show()

def test_linear():
    X1, y1, X2, y2 = gen_lin_separable_data()
    X_train, y_train = split_train(X1, y1, X2, y2)
    X_test, y_test = split_test(X1, y1, X2, y2)

    clf = SVM()
    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" % (correct, len(y_predict)))

    plot_margin(X_train[y_train==1], X_train[y_train==-1], clf)

def test_non_linear():
    X1, y1, X2, y2 = gen_non_lin_separable_data()
    X_train, y_train = split_train(X1, y1, X2, y2)
    X_test, y_test = split_test(X1, y1, X2, y2)

    clf = SVM(gaussian_kernel)
    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" % (correct, len(y_predict)))

    plot_contour(X_train[y_train==1], X_train[y_train==-1], clf)

def test_soft():
    X1, y1, X2, y2 = gen_lin_separable_overlap_data()
    X_train, y_train = split_train(X1, y1, X2, y2)
    X_test, y_test = split_test(X1, y1, X2, y2)

    clf = SVM(C=0.1)
    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" % (correct, len(y_predict)))

    plot_contour(X_train[y_train==1], X_train[y_train==-1], clf)

# test_soft()
# test_linear()
test_non_linear()
```