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

from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.utils.fixes import signature

#0. random number generator used for data generation
def randrange(n, vmin, vmax):
    '''
    Helper function to make an array of random numbers having shape (n, )
    with each number distributed Uniform(vmin, vmax).
    '''
    return (vmax - vmin)*np.random.rand(n) + vmin

rs = np.random.RandomState(1234)

#1. Training data generation
n_samples = 2500


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

data = []

for c, m, zlow, zhigh in [('r', 'o', -1000, -30), ('b', '^', -10, 1000)]:
    xs = randrange(n_samples, -100, 100)
    ys = randrange(n_samples, -100, 100)
    zs = 10*xs+ys+randrange(n_samples,zlow, zhigh)
    ax.scatter(xs, ys, zs, c=c, marker=m)
    tempx = [xs,ys,zs]
    data.append(tempx)
    if c == 'r':
        X=np.dstack((xs,ys,zs))
    else:
        X=np.append(X,np.dstack((xs,ys,zs)), axis=1)
X = np.squeeze(X)
print(np.shape(X))

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')


plt.show()

Y = np.zeros(n_samples*2)
Y[n_samples:] = 1

h = .02  # step size in the mesh

#2. SVM with different kernels initialization
C = 1.0
svc = svm.SVC(kernel='linear', C=C).fit(X, Y)
rbf_svc = svm.SVC(kernel='rbf', gamma=1.8, C=C).fit(X, Y)
poly_svc = svm.SVC(kernel='poly', degree=3, C=1.0).fit(X,Y)
lin_svc = svm.LinearSVC(C=C).fit(X, Y)

for svc_iter in (svc, lin_svc, rbf_svc, poly_svc):
    svc_iter.fit(X, Y)

    z = lambda x, y: (-svc.intercept_[0]-svc.coef_[0][0]*x-svc.coef_[0][1]*y) /
    svc.coef_[0][2]

    tmp = np.linspace(-100, 100, 100)
    x, y = np.meshgrid(tmp, tmp)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z(x, y))
    ax.plot3D(X[Y == 0, 0], X[Y == 0, 1], X[Y == 0, 2],'ob')
    ax.plot3D(X[Y == 1, 0], X[Y == 1, 1], X[Y == 1, 2],'sr')
    plt.show()
    print()
#3. Test set generation
    nt=1000
    for c, m, zlow, zhigh in [('r', 'o', -1000, -30), ('b', '^', -10, 1000)]:
        xs = randrange(nt, -100, 100)
        ys = randrange(nt, -100, 100)
        zs = 10*xs+ys+randrange(nt,zlow, zhigh)
        ax.scatter(xs, ys, zs, c=c, marker=m)
        tempx = [xs,ys,zs]
        data.append(tempx)
        if c == 'r':
            Xt=np.dstack((xs,ys,zs))
        else:
            Xt=np.append(Xt,np.dstack((xs,ys,zs)), axis=1)
    Xt = np.squeeze(Xt)
    Yt = np.zeros(nt*2)
    Yt[nt:] = 1
#4. Precision-recall score evaluation
    average_precision = average_precision_score(svc_iter.predict(Xt), Yt)

    print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
    precision, recall, _ = precision_recall_curve(svc_iter.predict(Xt), Yt)


    step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
    plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(
          average_precision))
```