第三题

3.试采用核映射SVM方法对“癌症患者生存期”数据集进行分类。给出具体代码，以及训练和测试的准确率结果。

In [1]:
```
# 数据读入
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt
import numpy as np
import torch
from sklearn import preprocessing
with open('haberman.data') as f:
    raw_data = [line.strip().split(',') for line in f.readlines()]
np.random.seed(666)
raw_data = np.array(raw_data, dtype=np.int)
N_SAMPLES = raw_data.shape[0]
indices = np.arange(N_SAMPLES)
np.random.shuffle(indices)
raw_data = raw_data[indices]
raw_data
```

executed in 1.12s, finished 11:07:47 2019-12-29

Out[1]:
```
array([[47, 66,  0,  1],
       [69, 65,  0,  1],
       [36, 69,  0,  1],
       ...,
       [46, 62,  5,  2],
       [43, 63,  2,  1],
       [61, 68,  0,  1]])
```

解决方法

线性支持向量机问题的对偶表达式为： 原始优化问题转化为参数 a 的极值问题

$$A(\alpha)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j X_i X_j \tag{1}$$

$$st.\sum_{i=1}^{n}\alpha_i y_i=0\,(\alpha_i\geq 0, i=1,2,n) \tag{2}$$

$$y_i\in\{+1,-1\} \tag{3}$$

使用梯度的优化方法，直接最大化对偶优化问题 $maxA(\alpha)$

在前述方程里的核函数是点积，$x_i\cdot x_j$，其为线性核函数。该核函数是以数据点（i，j）的点积填充的方阵。

替代数据点间的点积，可以将其扩展到更复杂的函数更高维度。这看似不怎么复杂，但是如果选择函数 k，其需满足如下条件：

$$k\left(x_i,x_j\right)=\varphi\left(x_i\right)\cdot\varphi\left(x_j\right) \tag{4}$$

高斯核

$$k\left(x_i,x_j\right)=e^{-y}\left\|x_i-x_j\right\|^2 \tag{5}$$

或者参考论文 Pegasos primal estimated sub-gradient solver 的优化公式也是可以的

$$\operatorname{argmin}\frac{\lambda}{2}\sum_{i,j}\alpha_i\alpha_j K\left(x_i,x_j\right)+\frac{1}{N}\sum_i\max\left\{0,1-y_i\sum_j\alpha_j K\left(x_i,x_j\right)\right\},\alpha\geq 0 \tag{6}$$

预测函数

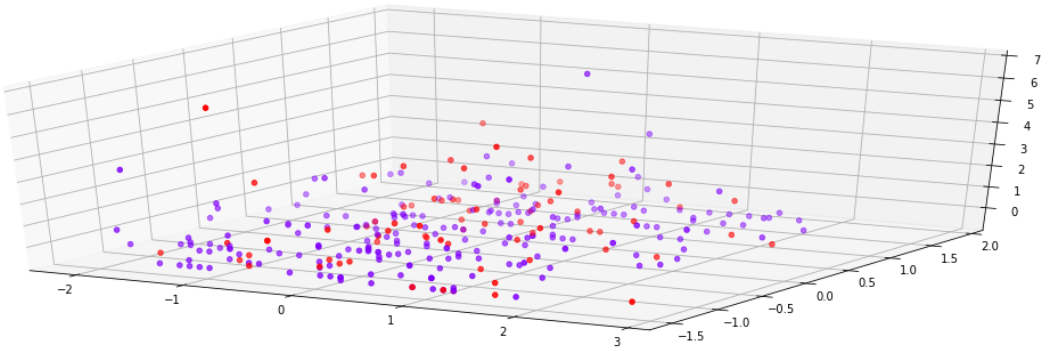
$$\text{predict}:y=\sum\alpha_i y_i K\left(x,x_i\right) \tag{7}$$

In [2]:
```
x, y = raw_data[:, :-1], raw_data[:, -1] # 数据归一化
x = preprocessing.scale(x)
print(x.mean(0), x.std(0))

```

executed in 8ms, finished 11:07:47 2019-12-29

```
[1.43313103e-16 3.84587061e-16 1.11747938e-16] [1. 1. 1.]
```

In [3]:
```
fig = plt.figure(figsize=[20,6])
# 创建3d图形的两种方式
# ax = Axes3D(fig)
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x[:,0],x[:,1],x[:,2],c=y,cmap="rainbow")
plt.show()
```
executed in 177ms, finished 11:07:47 2019-12-29



可见数据线性不可分，所以采用高斯核

In [4]:
```
y[y == 1] = -1
y[y == 2] = 1
y = y.reshape(-1, 1)
train_test_ratio = 0.6 # 划分训练集验证集合

N_TRAIN = round(N_SAMPLES*train_test_ratio)
x_train = x[:N_TRAIN]
y_train = y[:N_TRAIN]

x_test = x[-N_TRAIN:]
y_test = y[-N_TRAIN:]
```
executed in 5ms, finished 11:07:47 2019-12-29

In [5]:
```
def pairwise_mse(u: torch.Tensor, v: torch.Tensor):
    ru = (u*u).sum(-1, keepdims=True)
    rv = (v*v).sum(-1, keepdims=True)
    return ru-2*u.mm(v.t())+rv.t()

def Linear_kernal(u, v):
    return u.mm(v.t())

def Gaussian_kernal(u, v):
    # sigma = 50.0
    gamma = -10
    return torch.exp(gamma*pairwise_mse(u, v))

kernel = Gaussian_kernal

a = torch.rand(size=[N_TRAIN, 1], requires_grad=True)  # SVM paramter ai

def SVM_dul_loss(x, y):
    aij = a.mm(a.t())
    yij = y.mm(y.t())
    xij = kernel(x, x)
    assert aij.size() == yij.size() == xij.size()
    loss = a.sum()-(0.5*aij*yij*xij).sum()
    return - loss

# def SVM_dul_loss(x, y, lambda_=0.5):
#     aij = a.mm(a.t())
#     yij = y.mm(y.t())
#     xij = kernel(x, x)
#     assert aij.size() == yij.size() == xij.size()
#     loss = (lambda_ / 2 * aij * xij).sum() + \
#         (1-xij.mm(a)*y).clamp(min=0).mean()
#     return loss

def SVM_dul_predict(x_ture, y_true, x_pred):
    xij = kernel(x_ture, x_pred)
    yai = a*y_true
    y_pred = yai.t().mm(xij)
    y_pred = torch.sign(y_pred-y_pred.mean())
    return y_pred.view(-1, 1)

```
executed in 29ms, finished 11:07:47 2019-12-29

In [6]:
```
opt = torch.optim.SGD([a], lr=0.002)
x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
x_test = torch.tensor(x_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
loss_history = []
train_acc = []
val_acc = []
for epoch in range(10000):
    loss = SVM_dul_loss(x_train, y_train)
    loss.backward()
    opt.step()
    opt.zero_grad()
    with torch.no_grad():
        loss_history.append(loss.item())
        y_pred = SVM_dul_predict(x_train, y_train, x_train)
        acc = (y_train == y_pred).float().mean()
        train_acc.append(acc)
        # print(acc)
        # print(a[0])
        if epoch % 100 == 0:
            y_pred = SVM_dul_predict(x_train, y_train, x_test)
            acc = (y_test == y_pred).float().mean()
            val_acc.append(acc)
```

executed in 15.0s, finished 11:08:02 2019-12-29

In [7]:
```
with torch.no_grad():
    y_pred = SVM_dul_predict(x_train, y_train, x_test)
    acc = (y_test == y_pred).float().mean()
    print("acc", acc)
    # Plot batch accuracy
    fig = plt.figure(figsize=[20,5])
    ax = fig.add_subplot(131)
    ax.plot(train_acc, 'k-', label='Accuracy')
    ax.set_title('training Accuracy')
    ax.set_xlabel('Generation')
    ax.set_ylabel('Accuracy')
    ax.legend(loc='lower right')

    # Plot loss over time
    ax = fig.add_subplot(132)
    ax.plot(loss_history, 'k-')
    ax.set_title('Loss per Generation')
    ax.set_xlabel('Generation')
    ax.set_ylabel('Loss')

    ax = fig.add_subplot(133)
    ax.plot(val_acc, 'k-', label='Accuracy')
    ax.set_title('validation Accuracy')
    ax.set_xlabel('Generation')
    ax.set_ylabel('Accuracy')
    ax.legend(loc='lower right')
    plt.show()
```

executed in 398ms, finished 11:08:02 2019-12-29

```
acc tensor(0.7391)
```