机器学习练习4 - 支持向量机Support Vector Machines

在这个练习中，我们将使用支持向量机（SVM）来构建垃圾邮件分类器。我们将从一些简单的二维数据集上的支持向量机开始，看看它们是如何工作的。然后，我们会对一组原始电子邮件进行预处理，并使用SVM在处理后的电子邮件上构建一个分类器，以确定它们是否是垃圾邮件。

我们要做的第一件事是查看一个简单的二维数据集，并观察线性SVM随着C值变化在数据集上的表现（类似于线性/逻辑回归中的正则化项）。现在先让我们加载数据。

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.io import loadmat
%matplotlib inline

raw_data = loadmat('ex6data1.mat')
raw_data
```

Out[ ]:
```
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Sun
Nov 13 14:28:43 2011',
 '__version__': '1.0',
 '__globals__': [],
 'X': array([[1.9643  , 4.5957  ],
        [2.2753  , 3.8589  ],
        [2.9781  , 4.5651  ],
        [2.932   , 3.5519  ],
        [3.5772  , 2.856   ],
        [4.015   , 3.1937  ],
        [3.3814  , 3.4291  ],
        [3.9113  , 4.1761  ],
        [2.7822  , 4.0431  ],
        [2.5518  , 4.6162  ],
        [3.3698  , 3.9101  ],
        [3.1048  , 3.0709  ],
        [1.9182  , 4.0534  ],
        [2.2638  , 4.3706  ],
        [2.6555  , 3.5008  ],
        [3.1855  , 4.2888  ],
        [3.6579  , 3.8692  ],
        [3.9113  , 3.4291  ],
        [3.6002  , 3.1221  ],
        [3.0357  , 3.3165  ],
        [1.5841  , 3.3575  ],
        [2.0103  , 3.2039  ],
        [1.9527  , 2.7843  ],
        [2.2753  , 2.7127  ],
        [2.3099  , 2.9584  ],
        [2.8283  , 2.6309  ],
        [3.0473  , 2.2931  ],
        [2.4827  , 2.0373  ],
        [2.5057  , 2.3853  ],
        [1.8721  , 2.0577  ],
        [2.0103  , 2.3546  ],
        [1.2269  , 2.3239  ],
        [1.8951  , 2.9174  ],
        [1.561   , 3.0709  ],
        [1.5495  , 2.6923  ],
        [1.6878  , 2.4057  ],
        [1.4919  , 2.0271  ],
        [0.962   , 2.682   ],
        [1.1693  , 2.9276  ],
        [0.8122  , 2.9992  ],
        [0.9735  , 3.3881  ],
        [1.25    , 3.1937  ],
        [1.3191  , 3.5109  ],
        [2.2292  , 2.201   ],
        [2.4482  , 2.6411  ],
        [2.7938  , 1.9656  ],
        [2.091   , 1.6177  ],
        [2.5403  , 2.8867  ],
        [0.9044  , 3.0198  ],
        [0.76615 , 2.5899  ],
        [0.086405, 4.1045  ]]),
 'y': array([[1],
        [1],
        [1],
        [1],
        [1],
```

```
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1]], dtype=uint8)}
```

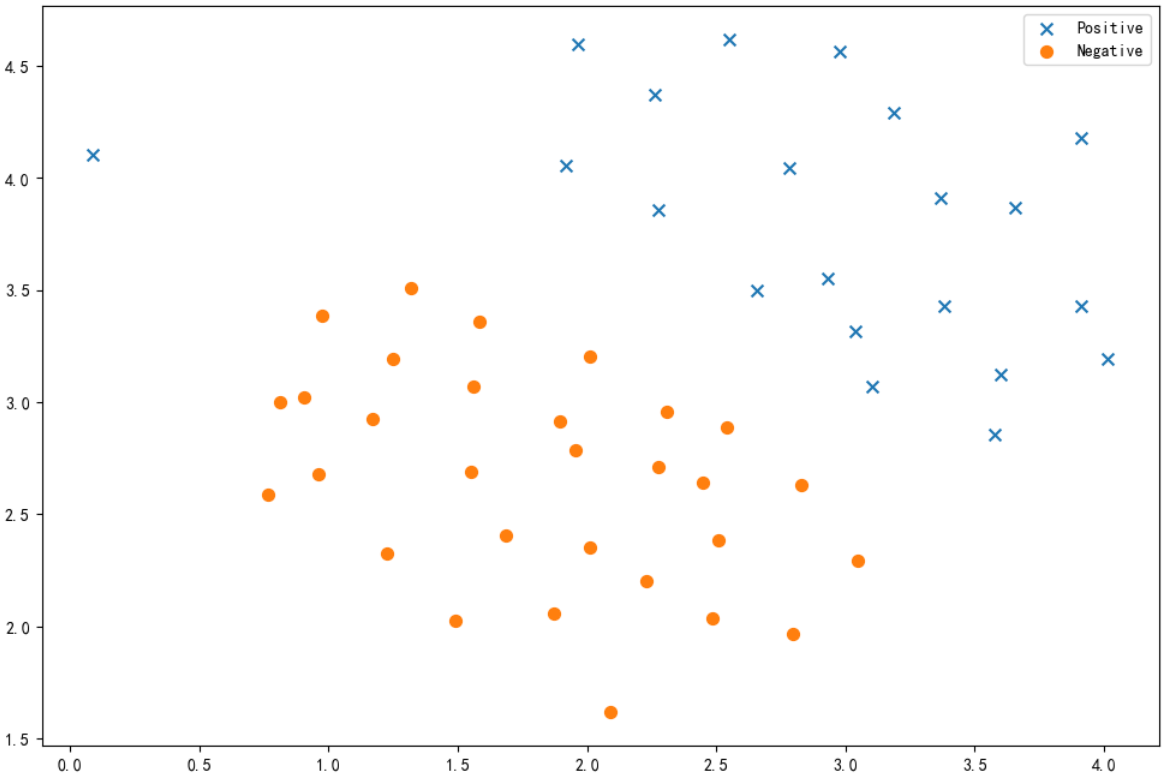
我们将把它可视化成一个散点图，其中类标签由一个符号表示（+表示正，o表示负）。

In [ ]:
```
data = pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
data['y'] = raw_data['y']

positive = data[data['y'].isin([1])]
negative = data[data['y'].isin([0])]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['X1'], positive['X2'], s=50, marker='x', label='Posit
ax.scatter(negative['X1'], negative['X2'], s=50, marker='o', label='Negat
ax.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x10be65ac0>



请注意，有一个异常的正例与其他异常例不同。这些类仍然可以线性分离，但它们非常紧密地结合在一起。我们将训练一个线性支持向量机来学习分类边界。在这个练习中，我们没有从头开始实现一个SVM的任务，我们使用scikit-learn内置的SVM。

In [ ]:
```
from sklearn import svm
svc = svm.LinearSVC(C=1, loss='hinge', max_iter=1000)
svc
```

Out[ ]:
```
LinearSVC
LinearSVC(C=1, loss='hinge')
```

在第一个实验中，我们将使用C=1，并观察它的性能。

In [ ]:
```
svc.fit(data[['X1', 'X2']], data['y'])
svc.score(data[['X1', 'X2']], data['y'])
```

```
/Users/fengzetao/miniconda3/envs/ml/lib/python3.9/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
/Users/fengzetao/miniconda3/envs/ml/lib/python3.9/site-packages/sklearn/svm/_base.py:1250: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[ ]: 0.9803921568627451

从结构可以看到，它对异常值进行了错误分类。让我们看看C值更大时会发生什么。

In [ ]:
```
svc2 = svm.LinearSVC(C=100, loss='hinge', max_iter=1000)
svc2.fit(data[['X1', 'X2']], data['y'])
svc2.score(data[['X1', 'X2']], data['y'])
```

```
/Users/fengzetao/miniconda3/envs/ml/lib/python3.9/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
/Users/fengzetao/miniconda3/envs/ml/lib/python3.9/site-packages/sklearn/svm/_base.py:1250: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

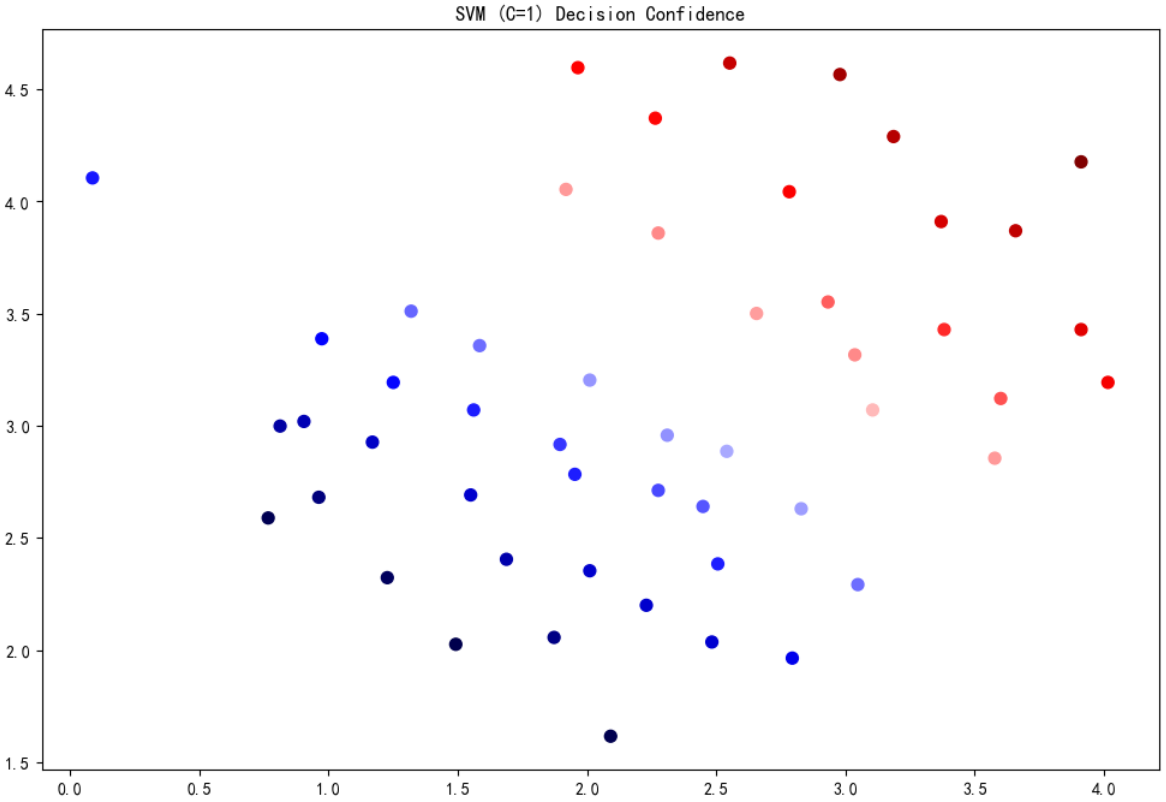
Out[ ]:
```
0.9411764705882353
```

这次我们得到了训练数据的完美分类，但是通过增加C的值，我们创建了一个与我们直觉不太一致的决策边界。我们可以通过查看每个类预测的置信度来直观地理解这一点，置信度是点与超平面距离的函数。

In [ ]:
```
data['SVM 1 Confidence'] = svc.decision_function(data[['X1', 'X2']])

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data['X1'], data['X2'], s=50, c=data['SVM 1 Confidence'], cmap
ax.set_title('SVM (C=1) Decision Confidence')
```

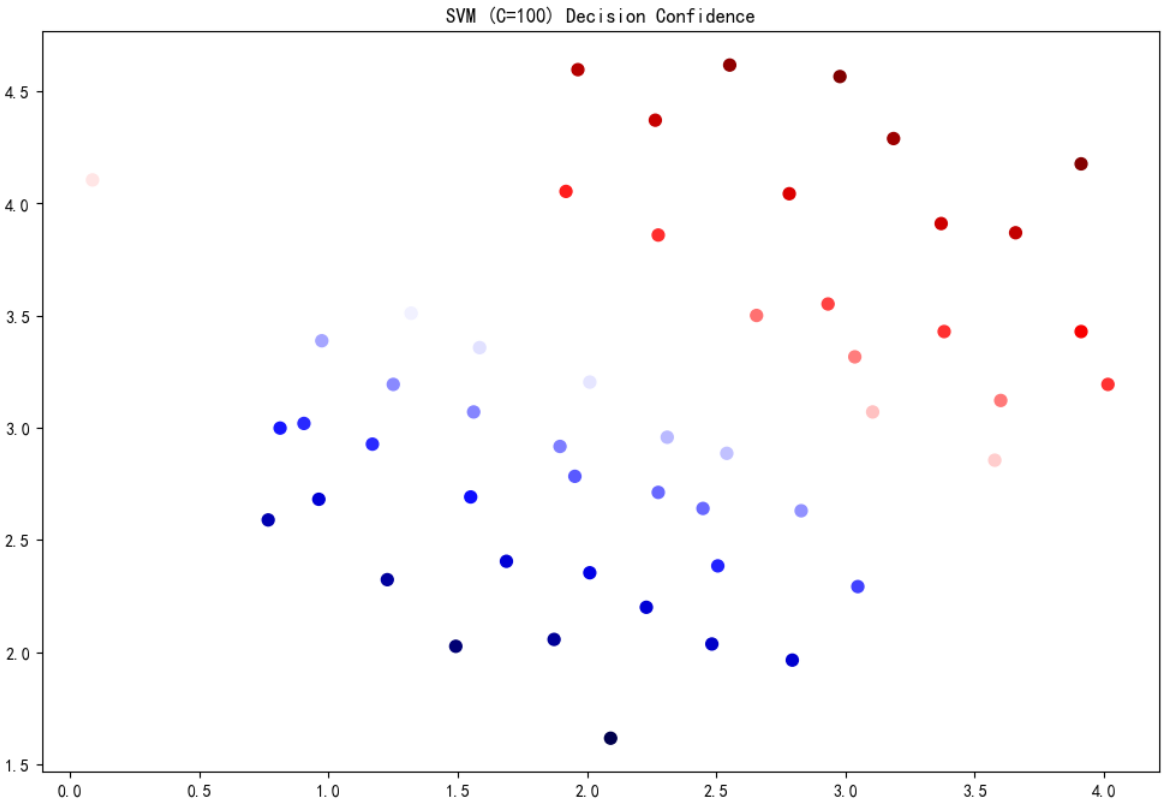
Out[ ]:
```
Text(0.5, 1.0, 'SVM (C=1) Decision Confidence')
```



In [ ]:
```
data['SVM 2 Confidence'] = svc2.decision_function(data[['X1', 'X2']])

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data['X1'], data['X2'], s=50, c=data['SVM 2 Confidence'], cmap
ax.set_title('SVM (C=100) Decision Confidence')
```

Out[ ]: Text(0.5, 1.0, 'SVM (C=100) Decision Confidence')



两个实验结果的区别不是很大。但是请仔细观察颜色的深浅变化，会发现因为C值的增大，导致分类边界与直观感觉有差别。直觉上，第一个实验结果更符合我们的直观。

现在我们要从线性SVM转向能够使用核进行非线性分类的SVM。我们的第一个任务是实现高斯核函数。虽然scikit-learn内置了高斯核，但为了透明度，我们将从头开始实现一个。

In [ ]:
```
def gaussian_kernel(x1, x2, sigma):
    return np.exp(-(np.sum((x1 - x2) ** 2) / (2 * (sigma ** 2))))
```

In [ ]:
```
x1 = np.array([1.0, 2.0, 1.0])
x2 = np.array([0.0, 4.0, -1.0])
sigma = 2

gaussian_kernel(x1, x2, sigma)
```

Out[ ]: 0.32465246735834974

接下来，我们将研究另一个数据集，这次是非线性决策边界。

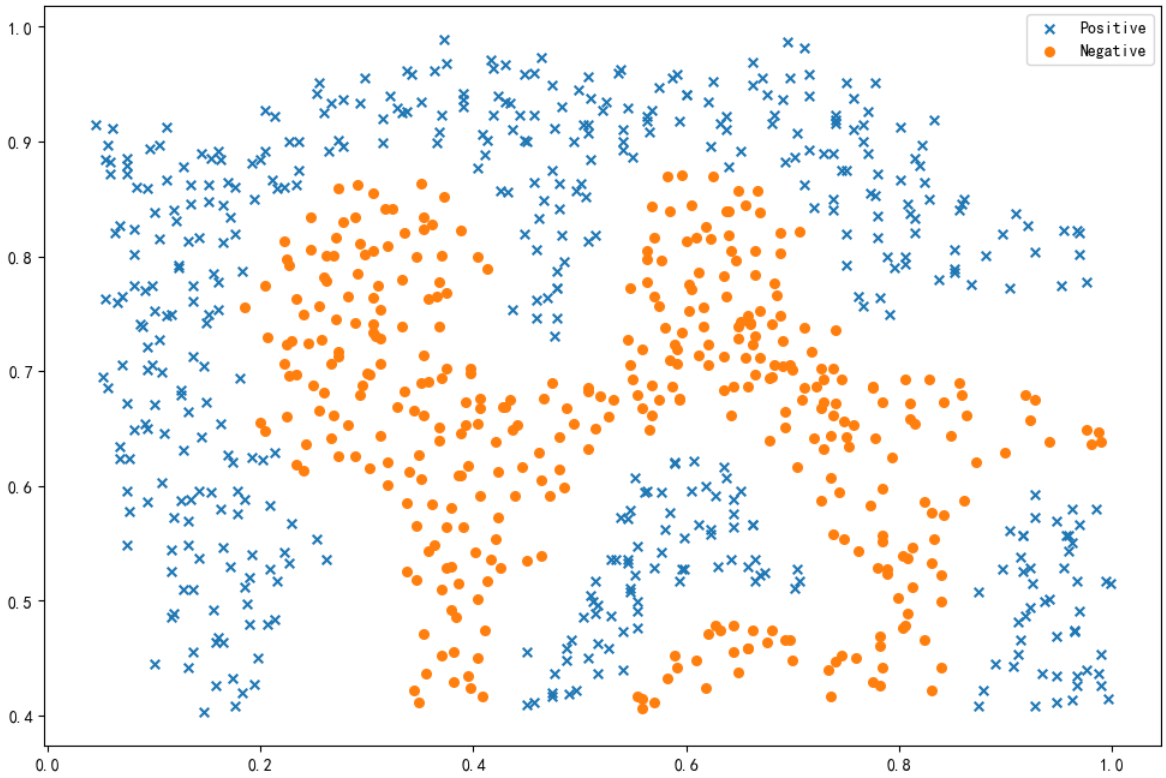
In [ ]:
```
raw_data = loadmat('ex6data2.mat')

data = pd.DataFrame(raw_data['X'], columns=['X1', 'X2'])
data['y'] = raw_data['y']

positive = data[data['y'].isin([1])]
negative = data[data['y'].isin([0])]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(positive['X1'], positive['X2'], s=30, marker='x', label='Posit
ax.scatter(negative['X1'], negative['X2'], s=30, marker='o', label='Negat
ax.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x295b3c8e0>



对于这个数据集，我们将使用内置的RBF内核构建一个支持向量机分类器，并检查它在训练数据上的准确性。为了使决策边界可视化，这次我们将根据实例具有负类标签的预测概率对点进行着色。我们将从结果中看到，它正确地识别了大部分。

In [ ]:
```
svc = svm.SVC(C=100, gamma=10, probability=True)
svc
```

Out[ ]:
```
SVC(C=100, gamma=10, probability=True)
```

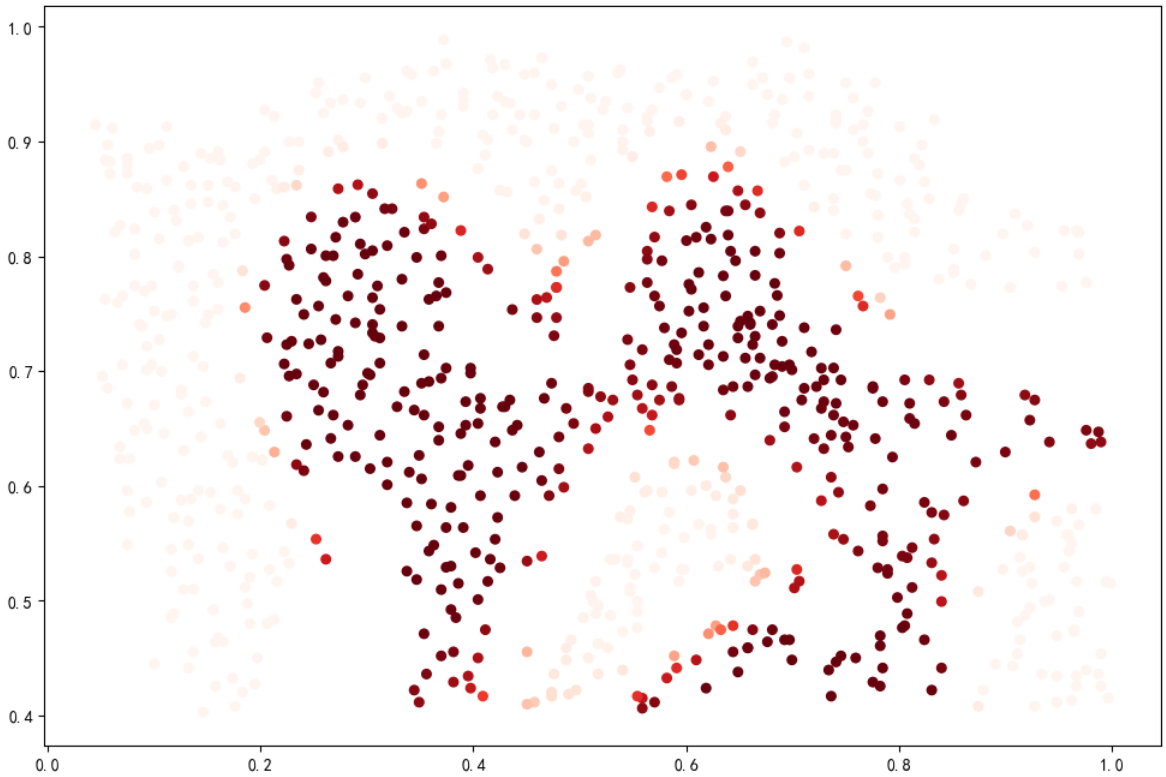
In [ ]:
```
svc.fit(data[['X1', 'X2']], data['y'])
svc.score(data[['X1', 'X2']], data['y'])
```

Out[ ]: 0.9698725376593279

In [ ]:
```
data['Probability'] = svc.predict_proba(data[['X1', 'X2']])[:,0]

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data['X1'], data['X2'], s=30, c=data['Probability'], cmap='Red
```

Out[ ]: <matplotlib.collections.PathCollection at 0x295bd01f0>



对于第三个数据集，我们给出了训练集和验证集，并负责根据验证集的性能为SVM模型找到最佳超参数。虽然我们可以使用scikit-learn的内置网格搜索来轻松完成这项工作，但本着遵循练习指导的精神，我们将从头开始实现一个简单的网格搜索。

In [ ]:
```
raw_data = loadmat('ex6data3.mat')

X = raw_data['X']
Xval = raw_data['Xval']
y = raw_data['y'].ravel()
yval = raw_data['yval'].ravel()

C_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100]
gamma_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100]

best_score = 0
best_params = {'C': None, 'gamma': None}

for C in C_values:
    for gamma in gamma_values:
        svc = svm.SVC(C=C, gamma=gamma)
        svc.fit(X, y)
        score = svc.score(Xval, yval)

        if score > best_score:
            best_score = score
            best_params['C'] = C
            best_params['gamma'] = gamma

best_score, best_params
```

Out[ ]:
```
(0.965, {'C': 0.3, 'gamma': 100})
```

现在我们将进行练习的第二部分。在这一部分中，我们的目标是使用SVM构建垃圾邮件过滤器。在练习中，有一项任务涉及一些文本预处理，文本预处理的目的是使我们的数据格

式适合SVM处理。这项任务非常简单，只需要将单词从练习中提供的词典映射到ID，其余的预处理步骤，如HTML删除、词干、规范化等，都已经完成。与其重复这些预处理步骤，本练习将直接跳到机器学习任务部分，该任务涉及从预处理的训练和测试数据集构建分类器，这些数据集由已经转换为单词向量的垃圾邮件和非垃圾邮件组成。

In [ ]:
```
spam_train = loadmat('spamTrain.mat')
spam_test = loadmat('spamTest.mat')

spam_train
```

Out[ ]:
```
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Sun
Nov 13 14:27:25 2011',
 '__version__': '1.0',
 '__globals__': [],
 'X': array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 1, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
 'y': array([[1],
        [1],
        [0],
        ...,
        [1],
        [0],
        [0]], dtype=uint8)}
```

In [ ]:
```
X = spam_train['X']
Xtest = spam_test['Xtest']
y = spam_train['y'].ravel()
ytest = spam_test['ytest'].ravel()

X.shape, y.shape, Xtest.shape, ytest.shape
```

Out[ ]:
```
((4000, 1899), (4000,), (1000, 1899), (1000,))
```

每个文档都被转换为具有1,899个维度的向量，对应于词汇表中的1,899个单词。这些值是二元的，表示文档中是否存在该单词。此时，训练和评估只是对分类器进行测试的问题。

In [ ]:
```
svc = svm.SVC()
svc.fit(X, y)
print('Training accuracy = {0}%'.format(np.round(svc.score(X, y) * 100, 2
```

```
Training accuracy = 99.32%
```

In [ ]:
```
print('Test accuracy = {0}%'.format(np.round(svc.score(Xtest, ytest) * 10
```

```
Test accuracy = 98.7%
```

这个结果是使用默认参数得到的。我们可能可以通过调整一些参数得到更高的准确率，但是95%的准确率仍然不差。

本练习到此结束！