

实验6. 支持向量机及文本分类

实验目的

- 1、学会使用sklearn库中的支持向量机的相关函数
- 2、使用软间隔SVM（C-SVM）体验代价函数中经验误差和松弛因子的作用
- 3、使用带有高斯核函数的SVM对非线性数据进行分类，体验高斯核函数的作用
- 4、使用交叉验证（Cross Validation）和参数网格（Grid Search）对模型参数进行调参
- 5、通过垃圾邮件检测，体验文本特征化和文本分类。

实验数据

ex6data1.mat - 二类线性可分数据集（带有一个异常点）

ex6data2.mat - 非线性二类可分数据集

ex6data3.mat - 用于调参的二分类数据集

spamTrain.mat - 用于垃圾邮件分类的训练数据集

spamTest.mat - 用于垃圾邮件分类的测试数据集

emailSample1.txt - 用于测试的正常邮件1

emailSample2.txt - 用于测试的正常邮件2

spamSample1.txt - 用于测试的垃圾邮件1

spamSample2.txt - 用于测试的垃圾邮件2

vocab.txt - 邮件中出现的单词列表

1. 使用sklearn中的支持向量机进行二分类

读取数据集

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.io import loadmat
4  from sklearn import svm
5
6  '''读取数据集'''
7
```

```

8 data1 = loadmat('ex6data1.mat')
9 data2 = loadmat('ex6data2.mat')
10 data3 = loadmat('ex6data3.mat')
11 X1 = data1['X']
12 y1 = data1['y'].flatten()
13 X2 = data2['X']
14 y2 = data2['y'].flatten()
15 X3 = data3['X']
16 y3 = data3['y'].flatten()
17 Xval = data3['Xval']
18 yval = data3['yval'].flatten()

```

1.1 调用sklearn.svm.SVC对示例代码进行二分类

通过使用sklearn（SciKit Learn）中封装的支持向量机对三个二类数据集进行二分类实验。通过实验了解支持向量机中关键因素的作用，包括最大间隔、最小训练误差、松弛因子、基于高斯核函数获得的非线性分类能力。

在实验前，先初始化好绘制正负样本plot_data()和绘制分割超平面plot_boundary()的代码。

在本实验中，在数组索引中使用逻辑表达式读取正负例的样本。此处为直接使用pos=x[y == 1]，需要读取的标签数组y从(m, 1)展平成(m,)

```

1  '''可视化数据集'''
2  def plot_data(x, y):
3      pos = x[y == 1]
4      neg = x[y == 0]
5      plt.scatter(pos[:,0], pos[:,1], c='k', marker='+')
6      plt.scatter(neg[:,0], neg[:,1], c='y', marker='o', edgecolors='k',
7                  linewidths=0.5)
8      plt.show
9
10 '''画出决策边界'''
11 def plot_boundary(clf, x1, desc=''):
12     u_span = np.max(x1[:,0]) - np.min(x1[:,0])
13     v_span = np.max(x1[:,1]) - np.min(x1[:,1])
14     u = np.linspace(np.min(x1[:,0])-u_span/10, np.max(x1[:,0])+ u_span/10,
15                     500) #为了后面可以直接调用这个函数
16     v = np.linspace(np.min(x1[:,1])-v_span/10, np.max(x1[:,1])+ v_span/10,
17                     500)
18     x1, x2 = np.meshgrid(u, v) #转为网格 (500*500)
19     y = clf.predict(np.c_[x1.flatten(), x2.flatten()]) #因为predict中是要输入一个二维的数据，因此需要展开
20     y = y.reshape(x1.shape) #重新转为网格
21     plt.contour(x1, x2, y, 1, colors='b') #画等高线
22     plt.title('The Decision Boundary'+desc)

```

scikit-learn，又写作sklearn，是一个开源的基于python语言的机器学习工具包。它通过NumPy, SciPy和Matplotlib等python数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

sklearn中SVM的算法库分为两类，一类是分类的算法库，主要包含LinearSVC，NuSVC和SVC三个类，另一类是回归算法库，包含SVR，NuSVR和LinearSVR三个类，相关模块都包裹在sklearn.svm模块中。本次实验解决分类任务我们使用到的是svm.SVC()，具体调用方法如下：

a. SVC方法参数如下：

```
1 class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3,
    gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False,
    tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,
    decision_function_shape='ovr', random_state=None)
```

涉及到本实验的参数有C，kernel和gamma三个参数。

◦ C：float，可选(默认值= 1.0)

错误术语的惩罚参数C。C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

◦ kernel：string，optional(default = 'rbf')

核函数类型，str类型，默认为'rbf'。可选参数为：

'linear'：线性核函数

'poly'：多项式核函数

'rbf'：高斯核函数

'sigmoid'：sigmoid核函数

'precomputed'：核矩阵

precomputed表示自己提前计算好核函数矩阵，这时候算法内部就不再用核函数去计算核矩阵，而是直接用你给的核矩阵，核矩阵需要为n*n的。

◦ gamma：float，optional(默认= 'auto')

核函数系数，float类型，可选参数，默认为auto。只对'rbf'，'poly'，'sigmoid'有效。如果gamma为auto，代表其值为样本特征数的倒数，即 $1/n_features$ 。

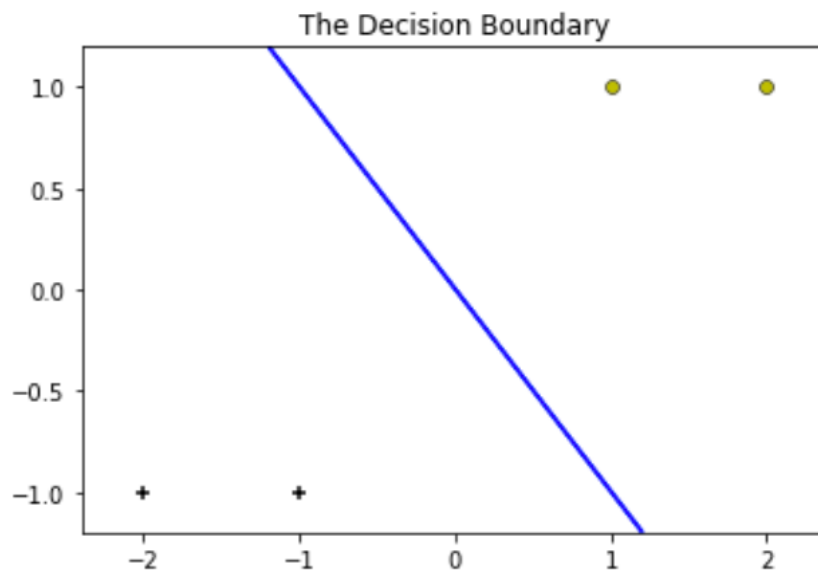
b. 模型训练fit()方法：**fit**(X, y, sample_weight=None)

c. 使用模型进行预测predict()方法：**predict**(X)

使用示例：

```
1  import numpy as np
2  from sklearn import svm
3
4  X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
5  y = np.array([1, 1, 0, 0])
6
7  # 画样本
8  plot_data(X, y)
9  clf = svm.SVC(C=1, kernel='linear')
10 clf.fit(X, y)
11 print(clf.predict([[1.5, 0.75]]))
12
13 # 绘制分类超平面
14 plot_boundary(clf, X)
15 plt.show
```

上面是构造了一个由4个样本组成的训练样本，在此基础上训练得到SVM分类器，并绘制它的分类超平面，显示如下。通过这个例子，请学会sklearn.svm.SVC()的用法，方便后面的实验。



1.2 体验软间隔SVM代价函数中经验误差和松弛因子的作用

首先对二维数据集 ex7data1.mat（正例用+表示，负例用o表示）进行可视化，然后使用sklearn.SVC对该数据集进行二分类，并显示分类判决面。

使用scipy.io.loadmat读取matlab的“.mat”数据文件，代码如下：

```
1  from scipy.io import loadmat
```

```

2 data1 = loadmat('ex7data1.mat')
3 X1 = data1['X']
4 y1 = data1['y'].flatten()

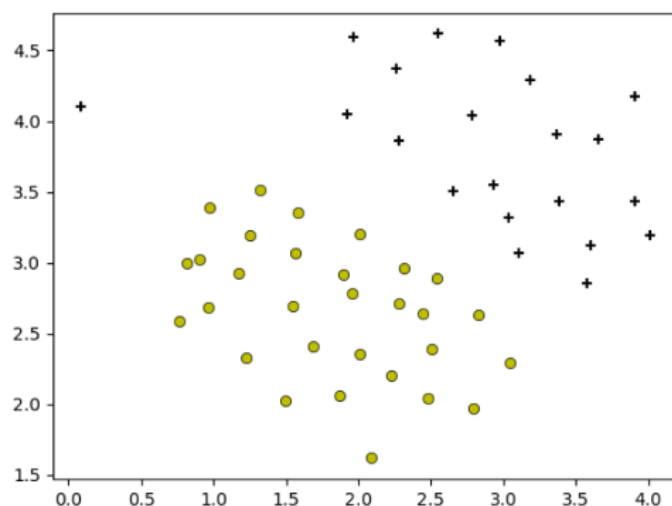
```

之后，ex7data1.mat中的数据显示如下图

```

1 # 调用plot_data()可视化ex7data1.mat
2 plot_data(X1, y1)

```



注意，在左上角有一个离群正例+，位置大致在(0.1,4.1)。面对该问题，本小节实验将通过调整参数C观察SVM代价函数中间隔最大化和引入松弛因子的训练误差这两项的均衡对该数据集的划分，并通过可视化SVM分类面进行展示。

```

1 # 1. 创建对象svm.SVC的实例linearSVMC1，设定正则化惩罚参数C=1，选择线性SVM（通过核函数选linear）
2 # 2. 使用fit()函数和X1和y1来训练分类模型
3 '''
4 linearSVMC1 = svm.SVC()
5 linearSVMC1.fit(X1, y1)
6 '''
7 #画图
8 plot_data(X1, y1)
9 plot_boundary(linearSVMC1, X1, '(C=? )')
10 plt.show
11 linearSVMC1.predict([[2, 4.5]])

```

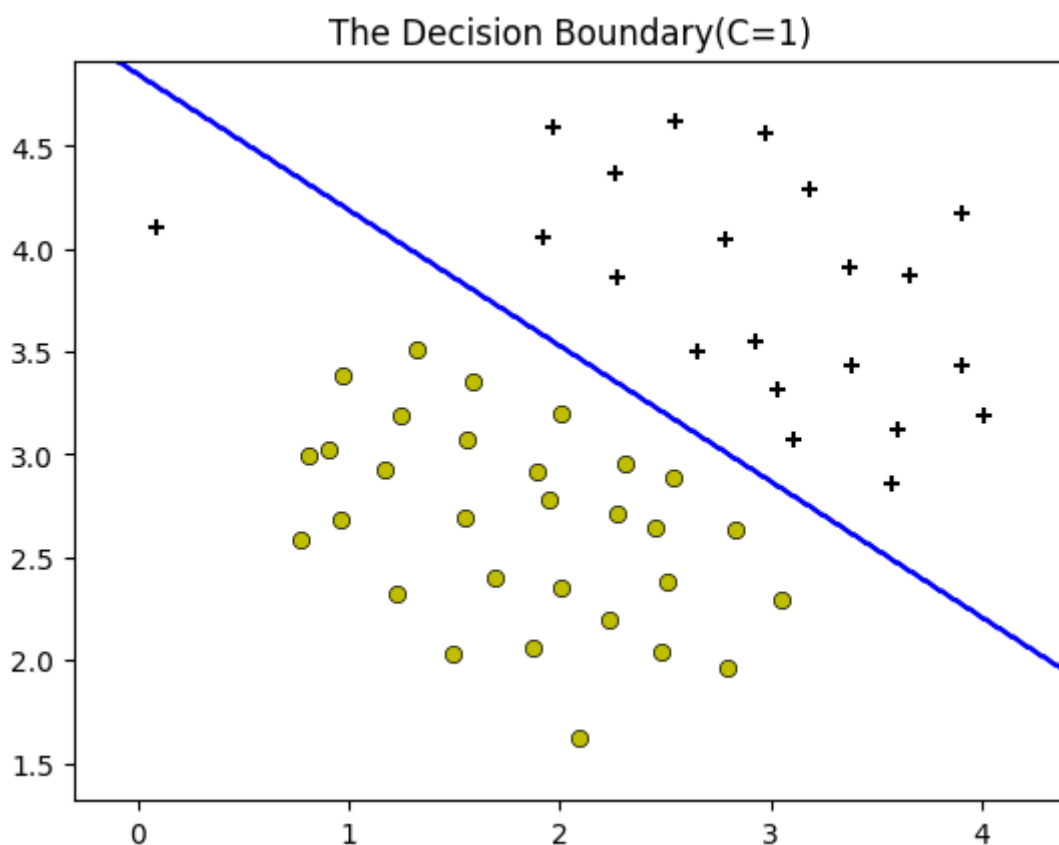
此部分需要按注释要求完成模型的训练，正确的预测结果为array([1], dtype=uint8)。

我们尝试在C-SVM（即软间隔SVM）中使用不同的C参数值。首先，回顾C-SVM的代价函数如下。一般来说，C参数是一个正值，用于控制错误分类训练误差（或经验风险）的惩罚。一个较大的C参数告诉支持向量机尽可能地正确地分类所有的样本。在这里，C的作用类似于正则化项的权重 $1/\lambda$ ，只是之前的 λ 加在正则项上，这里的C加在训练误差上。

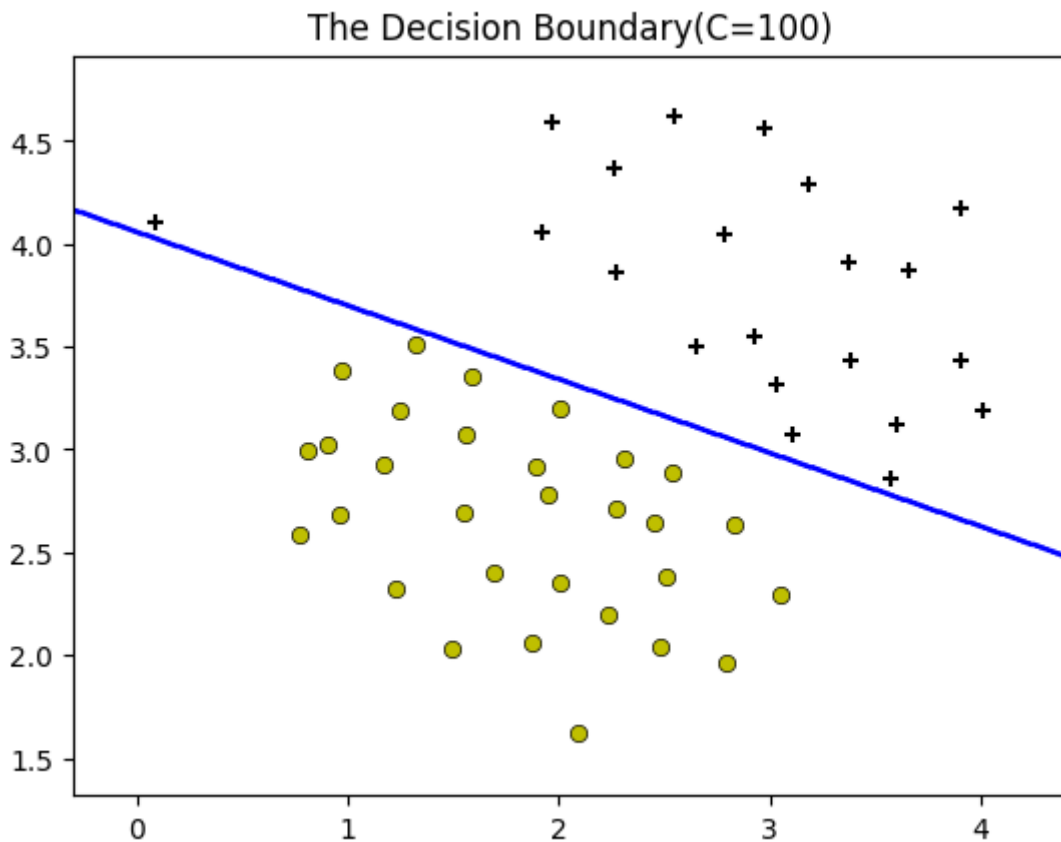
$$\begin{aligned} \min_{w,b,\xi_i} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, 2, \dots, m \end{aligned}$$

这里提供了绘制分类边界曲线的函数。

当C=1时，我们绘制的边界大致为：

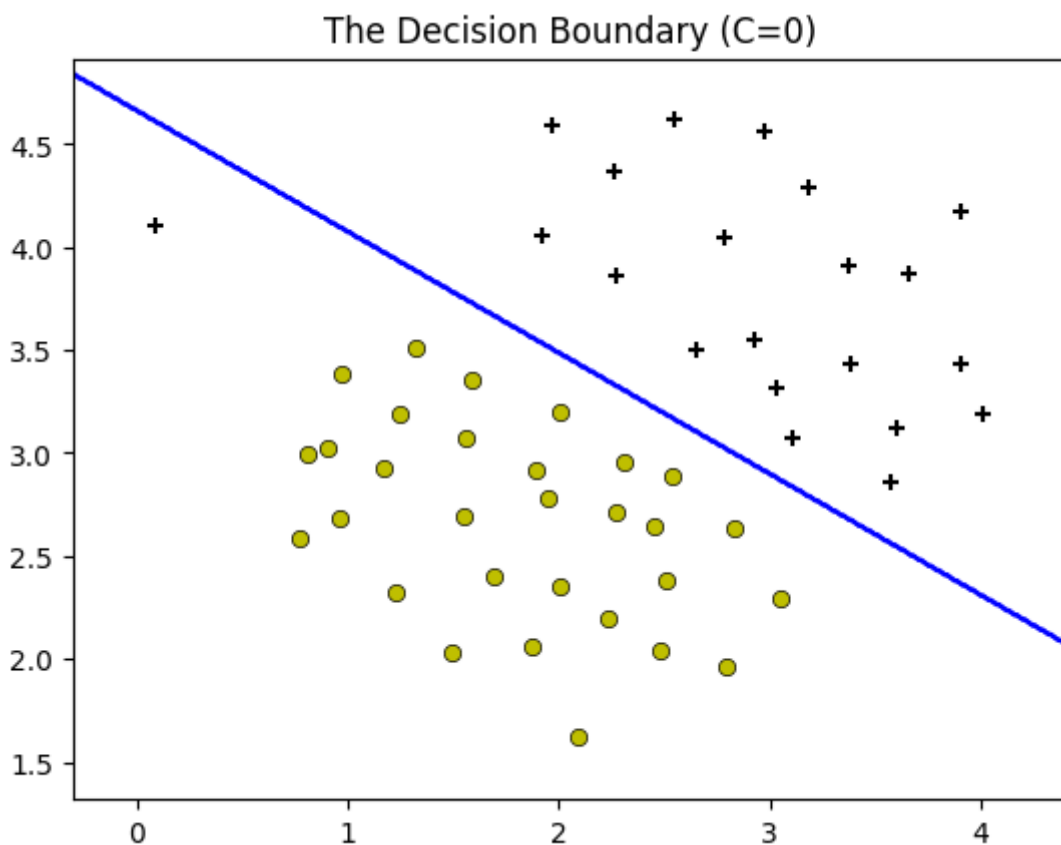


当C=100时，我们绘制的边界大致为：



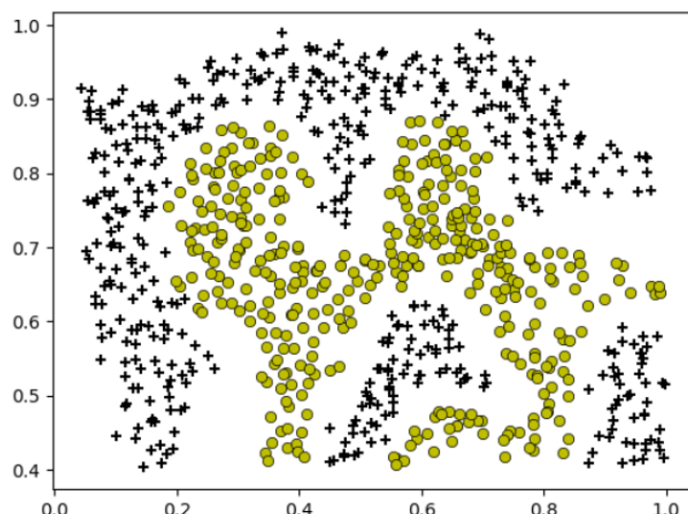
当 $C = 100$ 时，你应该会发现 SVM 现在对每个单独的示例都进行了正确的分类，但是受离群点影响，该模型的间隔很小，相应地，决策边界未在绝大多数数据中间。

当 $C=0$ 时，需要调用 `svm.LinearSVC()` 来实现。训练得到的分割超平面如下图所示：



1.3 使用SVM和高斯核函数解决非线性可分问题

在这部分练习中，你将使用SVM进行非线性分类。我们将在非线性可分的数据集ex7data2.mat上使用具有高斯核的支持向量机。数据可视化如下：



可以观察到，该数据集的正样例和反样例之间没有线性决策边界。然而，通过使用高斯核和支持向量机，你将能够学习一个非线性决策边界，它可以很好地处理数据集。

1.3.1 实现高斯核函数

为了用支持向量机找到非线性决策边界，我们需要首先实现一个高斯核。你可以把高斯核函数想象成一个相似度函数，它测量一对例子($x^{(i)}$, $x^{(j)}$)之间的“距离”。高斯核也通过带宽参数来参数化，该参数决定了随着样本距离的增加，相似度度量降低的速度(到0)。

高斯核函数的定义式为：

$$K_{gaussian}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^n (x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$

你需要完成高斯核函数的实现，当你完成后，使用所给样本a1和a2进行测试

😊 相似度结果应该近似等于0.324652。

```
1  # 定义高斯核函数
2  # 运算符 @ 是矩阵乘
3  def gaussianKernel(x1, x2, sigma):
4      '''
5
6      your code here
7  '''
```



```

1  #测试代码
2  a1 = np.array([1, 2, 1])
3  a2 = np.array([0, 4, -1])
4  sigma = 2
5  print(gaussianKernel(a1, a2, sigma)) #检查是否为0.32465246735834974

```

1.3.2 使用带有高斯核函数的SVM

在你完成高斯核函数之后，我们要用具有高斯核的向量机来对数据集分类。

在sklearn的SVM包中，高斯核函数使用径向基函数（Radial Basis Function，RBF）去表示。

除了C和kernel外，我们还要提供gamma参数。RBF核函数定义式如下：

$$e^{-\gamma \|x-x'\|^2}, \gamma > 0$$

与我们上面使用的高斯核相比，可以用下式将高斯核函数的sigma变为RBF核函数的gamma。

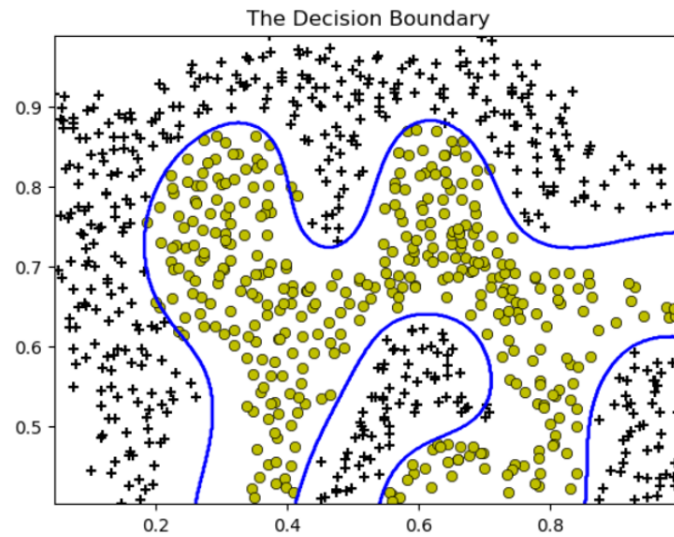
$$\gamma = \frac{1}{2\sigma^2}$$

```

1  #训练模型（这里使用内置高斯核，RBF：径向基函数，高斯核是它的一种）
2  sigma = 0.1
3  clf2 = svm.SVC(C = 1, kernel = 'rbf', gamma = 1 / (2 * sigma * sigma)) #对应
    sigma=0.1
4  clf2.fit(x2, y2)
5
6  #画图
7  plt.figure(2)
8  plot_data(X2, y2)
9  plot_boundary(clf2, X2)
10 plt.show

```

这就是我们实验中为svm.SVC()传入的第三个参数gamma。训练完成后，边界大致如图：



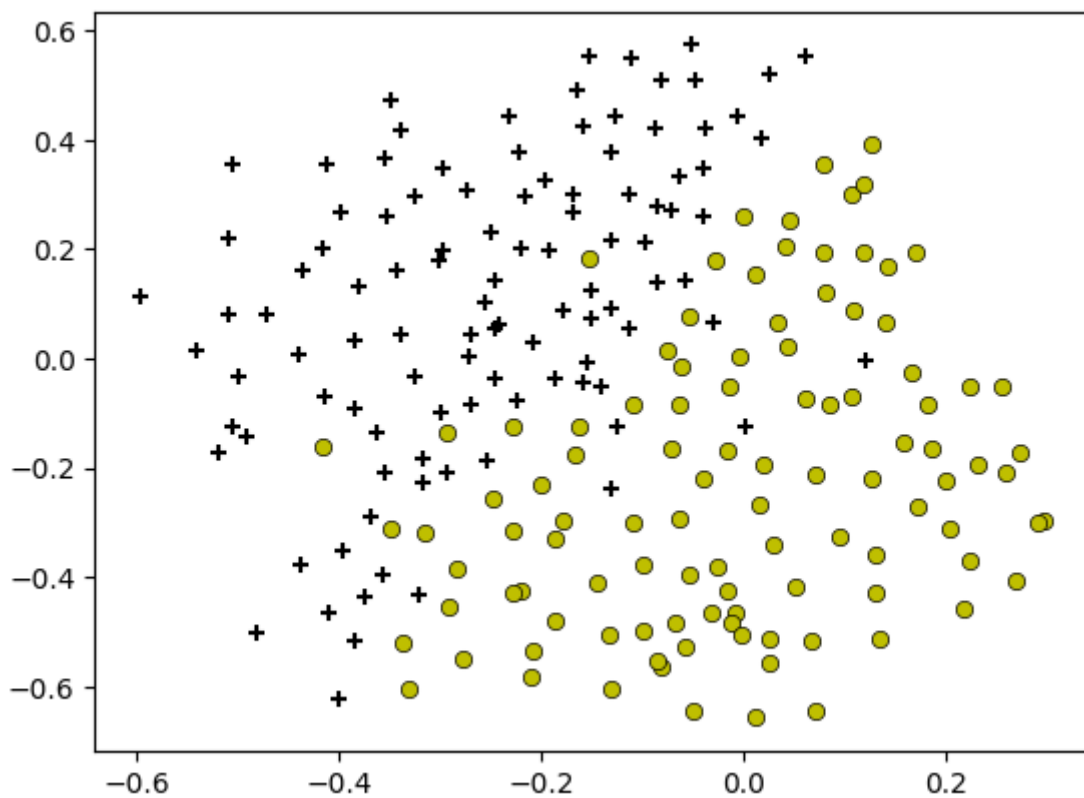
1.4 使用交叉验证和参数网格（Grid Search）进行参数调整（Parameter Tuning）

在这部分练习中，您将获得更多关于如何使用带有高斯核的SVM的实用技能。我们将加载并显示第三个数据集ex6data3.mat。你将对该数据集使用带有高斯核的支持向量机。

1.4.1 可视化训练数据集和验证数据集

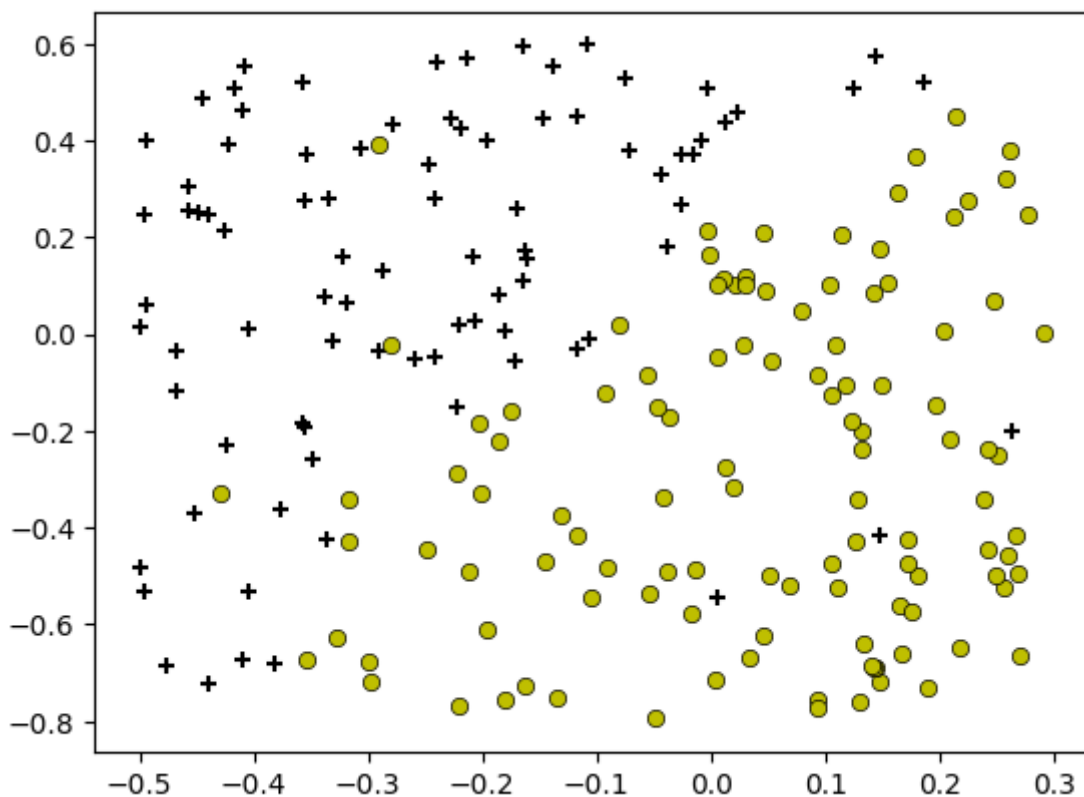
在提供的数据集中，你可以得到变量X3, y3, Xval, yval。你应该使用训练集X3, y3训练SVM分类器，使用交叉验证集Xval, yval来确定要使用的最佳C和 σ 参数。下面是X3, y3的可视化：

```
1 # 可视化训练集
2 plot_data(X3, y3)
```



下面是Xval, yval的可视化：

```
1 # 可视化验证集
2 plot_data(Xval, yval)
```



1.4.2 调参 (hyper-parameter tuning)


你应该完成下面的函数来搜索合适的参数C和 σ 。

对于C和 σ ，我们给出了一组建议尝试的数值序列try_value (0.01,0.03,0.1,0.3,1,3,10,30)。请注意，你应该尝试C和 σ 的所有可能的值对(例如，C = 0.3和 σ = 0.1)。如果你尝试上面列出的8个C和 σ 的值，你将最终训练和评估(在交叉验证集上)总共64个不同的模型。

你应该对每个模型计算其在验证集上的分类正确率，寻找正确率最高的参数组合。并绘制这组参数对应模型的分类边界曲线。最优参数组合应该为C = 1， σ = 0.1。

```
1  try_value = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])
2  #错误率
3  def error_rate(predict_y, yval):
4      m = yval.size
5      count = 0
6      for i in range(m):
7          count = count + np.abs(int(predict_y[i]) - int(yval[i])) #避免溢出错误得到
225
8      return float(count/m)
9
10 #模型选择
11 def model_selection(try_value, x3, y3, xval, yval):
12     error = 1
13     c = 1
14     sigma = 0.01
15     for i in range(len(try_value)):
16         for j in range(len(try_value)):
17             clf = svm.SVC(C=try_value[i], kernel='rbf',
gamma=np.power(try_value[j], -2)/2)
18             clf.fit(x3, y3)
19             predict_y = clf.predict(xval)
20             if error > error_rate(predict_y, yval):
21                 error = error_rate(predict_y, yval)
22                 c = try_value[i]
23                 sigma = try_value[j]
24     return c, sigma, error
25
26 c, sigma, error = model_selection(try_value, X3, y3, Xval, yval) #(1.0, 0.1,
0.035)
27 print(c, sigma, error)
```

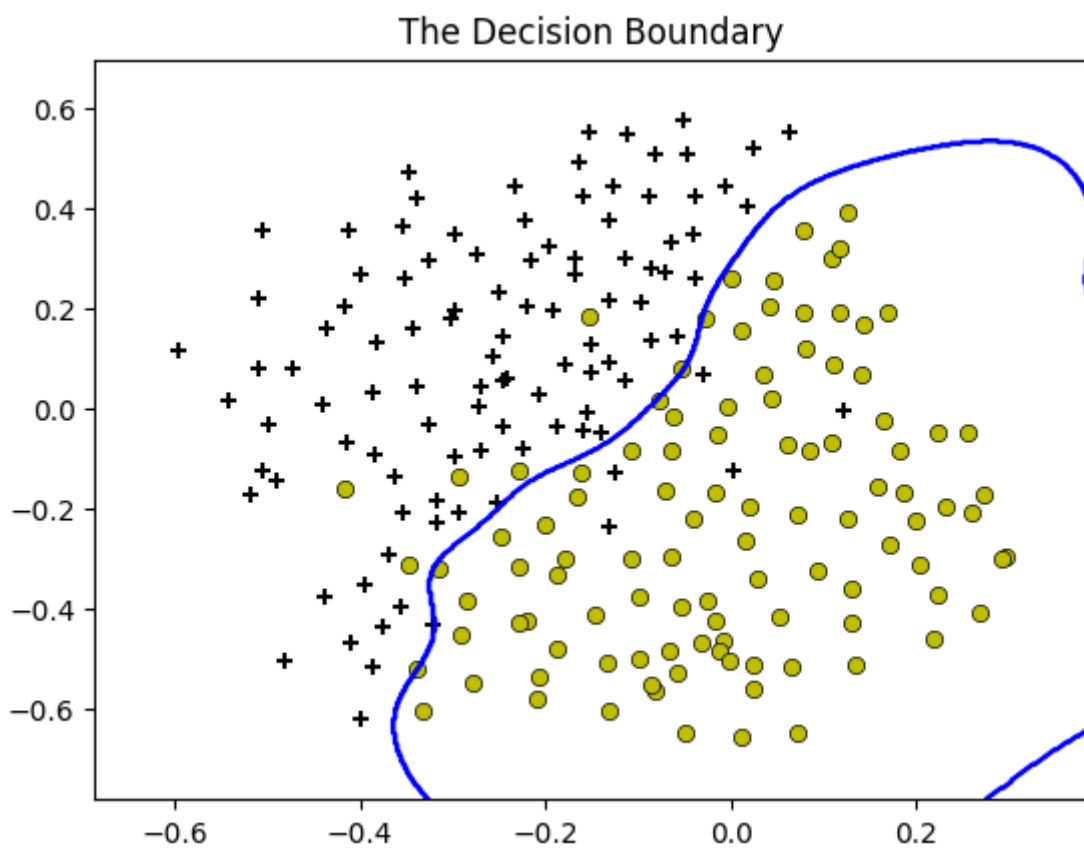
Cell输出应为：

 1.0 0.1 0.035

1.4.3 可视化最优参数下的SVM在ex7data3上的表现

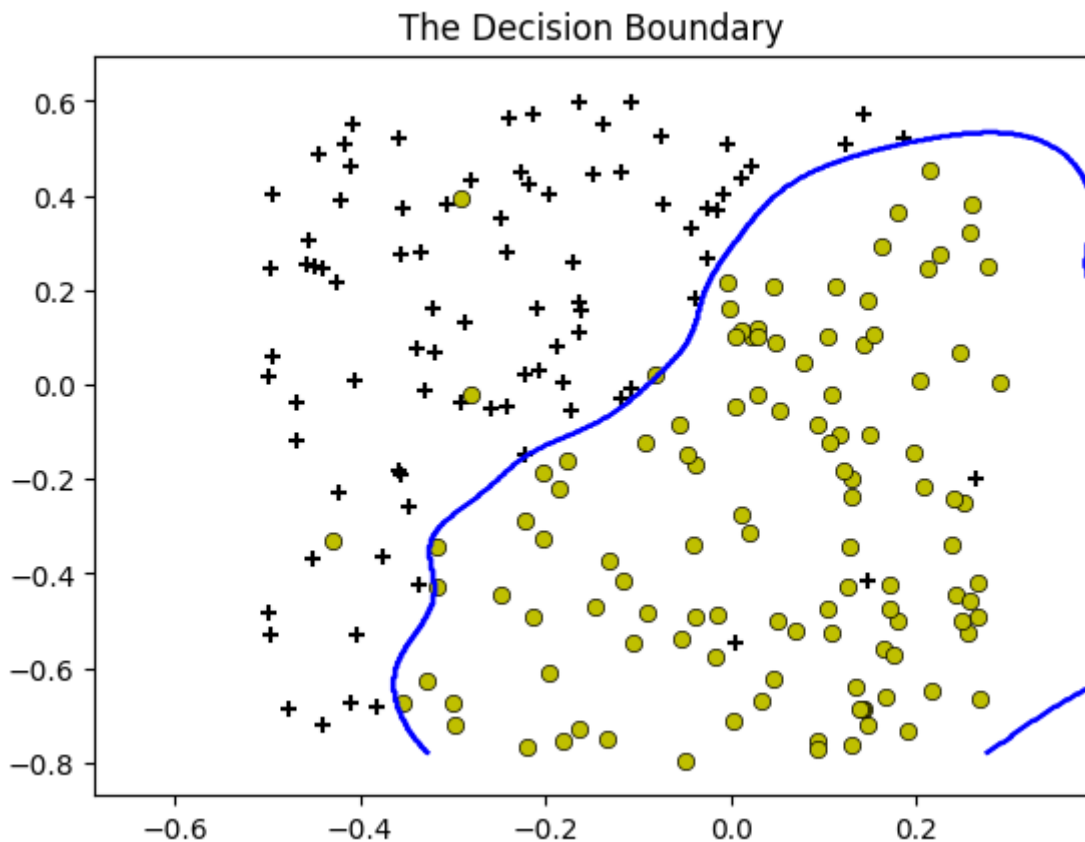
首先绘制训练数据集和训练得到的SVM

```
1 clf3 = svm.SVC(C=c, kernel='rbf', gamma=np.power(sigma, -2)/2)
2 clf3.fit(X3, y3)
3
4 #画图
5 plot_data(X3, y3)
6 plot_boundary(clf3, X3)
7 plt.show
```



然后，绘制验证数据集和SVM

```
1 plot_data(Xval, yval)
2 plot_boundary(clf3, X3)
3 plt.show
```



2. 使用SVM进行文本分类（垃圾邮件检测）

如今，许多电子邮件服务都提供垃圾邮件过滤器，能够以高精度将电子邮件分为垃圾邮件和非垃圾邮件。在本部分练习中，您将使用SVM构建自己的垃圾邮件过滤器。您将训练一个分类器来分类给定的电子邮件 x 是垃圾邮件($y = 1$)还是非垃圾邮件($y = 0$)。特别是，你需要将每个电子邮件转换为特征向量 $x \in \mathbb{R}(n)$ 。以下部分将引导你了解如何从电子邮件中构建这样的特征向量。本练习中包含的数据集是基于SpamAssassin公共语料库的一个子集。出于练习的目的，你将只使用电子邮件的正文(不包括电子邮件标题)。

2.1 对邮件数据进行预处理和向量化

在开始机器学习任务之前，通常需要查看数据集中的示例。下图显示了一个示例电子邮件，其中包含URL、电子邮件地址(在末尾)、数字和金额。

首先查看emailSample1.txt的内容：

```
1  def readEmail(emailFile):
2      f = open(emailFile, 'r', encoding='utf-8')
3      email = f.read()
4      f.close()
5      return email
6
7  emailSample1 = readEmail('./emailSample1.txt')
8  print(emailSample1)
```

emailSample1.txt的内容如下：

```
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors youre expecting. This can be
anywhere from less than 10 bucks a month to a couple of $100. You
should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if
youre running something big..

To unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

虽然许多电子邮件会包含类似类型的实体(例如，数字，其他URL或其他电子邮件地址)，但具体的实体(例如，特定的URL或特定的金额)在几乎每封电子邮件中都是不同的。因此，在处理电子邮件时经常使用的一种方法是“规范化”这些值，这样所有的url都得到相同的处理，所有的数字都得到相同的处理，等等。

例如，我们可以用唯一的字符串“httpaddr”替换电子邮件中的每个URL，以表明存在URL。这样做的效果是让垃圾邮件分类器根据是否存在任何URL而不是特定URL是否存在来做出分类决策。这通常会提高垃圾邮件分类器的性能，因为垃圾邮件发送者经常将URL随机化，因此在新的垃圾邮件中再次看到任何特定URL的几率非常小。

本实验中要完成的邮件预处理工作：

- 将整封电子邮件转换成小写
- 移除所有HTML标签（超文本标记语言）
- 将所有的URL替换为’ httpaddr’
- 将所有的地址替换为’ emailaddr’
- 将所有的数字替换为’ number’
- 将所有的美元符号(\$)替换为’ dollar’
- 将所有单词还原为词根。例如，
” discount” ,” discounts” ,” discounted” ,” discounting” 都替换为” discount”
- 移除所有非文字类型，所有的空格（tabs,newlines,spaces）调整为一个空格。

下图是预处理后的样例邮件内容。

```
anyon know how much it cost to host a web portal well it depend on how
mani visitor your expect thi can be anywher from less than number buck
a month to a coupl of dollarnumb you should checkout httpaddr or perhap
amazon ecnumb if your run someth big to unsubscrib yourself from thi
mail list send an email to emailaddr
```

我们为以上的预处理工作提供了代码实现。

```
1 def preprocess(email):
2     """做除了Word Stemming和Removal of non-words的所有处理"""
3     # 大写转小写
4     email = email.lower()
5     # 移除html标签
6     email = re.sub(r'<.*>', '', email)
7     # 移除url
8     email = re.sub(r'(http|https)://[^\s]*', 'httpaddr', email)
9     # 移除$, 解决dollar 和 number 的连接问题
10    email = re.sub(r'[\$][0-9]+', 'dollar number', email)
11    # 移除单个$
12    email = re.sub(r'\$', 'dollar number', email)
13    # 移除数字
14    email = re.sub(r'[0-9]+', 'number', email)
15    # 移除邮箱
16    email = re.sub(r'^[\s]+@[^\s]+', 'emailaddr', email)
17    return email
```

```
1 from nltk.stem import PorterStemmer
2 def preprocess2(email):
3     """预处理数据 : 提取词干, 去除非字符内容"""
4     stemmer = PorterStemmer()
5     email = preprocess(email)
6
7     # 将邮件分割为单个单词, re.split()可以设置多种分隔符
8     tokens = re.split('[ \t@$%/\#\.\-:\&*+=\[\]\?!\(\)\{\}\|\,\'\">\_\  
<\\;\\%]', email)
9
10    # 遍历每个分割出来的内容
11    tokenlist = []
12    for token in tokens:
13        # 删除任何非字母的字符
14        token = re.sub('[^a-zA-Z0-9]', '', token)
15        # 提取词根
16        stemmed = stemmer.stem(token)
17        # 去除空字符串: 里面不包含任何字符
18        if not len(token): continue
19        tokenlist.append(stemmed)
20
21    return tokenlist
```


2.2 邮件的特征向量构建

在预处理之后我们要对照单词表得到样例对应的序号列表，对应关系如图。

```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```

```
86 916 794 1077 883
370 1699 790 1822
1831 883 431 1171
794 1002 1893 1364
592 1676 238 162 89
688 945 1663 1120
1062 1699 375 1162
479 1893 1510 799
1182 1237 810 1895
1440 1547 181 1699
1758 1896 688 1676
992 961 1477 71 530
1699 531
```

左图是vocal.txt文件的示意图，这是我们实验中用到的词汇表，包含1899个单词。右图是将处理后的邮件与词汇表比对，用词汇表中的序号替代邮件中出现的单词。你需要完成以下函数来实现这个功能，其中入参email是预处理后的邮件，vocab是词汇表。返回值是一个数组，里面是邮件中所有单词的索引值。

```
1 def VocabIndex(email):
2     # 1. 对当前邮件进行预处理
3     """提取存在单词的索引"""
4     tokenlist = preprocess2(email)
5     '''
6
7     your code here
8
9     '''
10    return index
```

```
1 print(VocabIndex(emailSample1))
2 print(VocabIndex(emailSample2))
3 print(VocabIndex(spamSample1))
4 print(VocabIndex(spamSample2))
```

Cell输出:



```
[70, 85, 88, 161, 180, 237, 369, 374, 430, 476, 529, 530, 591, 687, 789, 793, 798, 809, 882,
915, 944, 960, 991, 1001, 1061, 1076, 1119, 1161, 1170, 1181, 1236, 1363, 1439, 1476,
1509, 1546, 1662, 1675, 1698, 1757, 1821, 1830, 1892, 1894, 1895]

[20, 63, 73, 74, 79, 85, 115, 123, 138, 161, 185, 203, 207, 209, 224, 229, 237, 245, 258, 379,
426, 449, 451, 463, 475, 486, 491, 530, 570, 582, 593, 625, 651, 661, 665, 676, 687, 717,
725, 734, 751, 755, 770, 784, 786, 804, 809, 824, 839, 846, 849, 875, 882, 901, 908, 939,
959, 960, 970, 979, 987, 994, 1017, 1060, 1083, 1094, 1098, 1100, 1112, 1119, 1161, 1162,
1170, 1181, 1191, 1207, 1216, 1227, 1248, 1279, 1283, 1307, 1308, 1364, 1370, 1375,
1378, 1436, 1439, 1461, 1463, 1509, 1544, 1589, 1609, 1612, 1626, 1629, 1663, 1665,
1670, 1674, 1693, 1698, 1707, 1743, 1751, 1759, 1772, 1784, 1789, 1791, 1802, 1823,
1834, 1839, 1844, 1854, 1859, 1884]

[73, 233, 386, 439, 470, 476, 607, 675, 707, 791, 809, 824, 836, 866, 868, 876, 955, 975,
996, 1047, 1063, 1069, 1092, 1112, 1116, 1119, 1181, 1190, 1229, 1230, 1241, 1264, 1286,
1345, 1376, 1489, 1630, 1664, 1665, 1675, 1698, 1808, 1826, 1843, 1851, 1892, 1894, 1895]

[9, 175, 279, 387, 459, 476, 680, 706, 738, 798, 1011, 1119, 1226, 1346, 1474, 1710, 1794,
1818]
```

在对电子邮件进行预处理之后，我们得到了每个电子邮件的单词列表。下一步是选择我们想要在分类器中使用的单词以及我们想要省略的单词。在这个练习中，我们只选择了出现频率最高的单词作为我们考虑的单词集(词汇表)。由于在训练集中很少出现的单词只出现在少数电子邮件中，它们可能会导致模型过拟合我们的训练集。

我们的词汇表是通过选择所有在垃圾邮件语料库中出现至少100次的单词来选择的，从而得到一个包含1899个单词的列表。在实践中，经常使用大约1万到5万个单词的词汇表。

给定词汇表列表，我们现在可以将预处理电子邮件中的每个单词映射到包含词汇表中单词索引的单词索引列表中。上图显示了示例电子邮件的映射。具体来说，在示例电子邮件中，“任何人”一词首先被标准化为“anyon”，然后映射到词汇表中的索引86。

现在，你将实现将每个电子邮件转换为 $R(n)$ 中的向量的特征提取。具体来说，电子邮件的特征 $x_i \in \{0,1\}$ 对应于字典中的第 i 个单词是否出现在电子邮件中。也就是说，如果电子邮件中有第 i 个单词，则 $x_i = 1$ ；如果电子邮件中没有第 i 个单词，则 $x_i = 0$ 。

下面是你要实现的特征提取函数：

```
1 def vectoringEmail(email):
2     df = pd.read_table('./vocab.txt', names=['words'])
3     '''
4
```

```
5     your code here
6
7     '''
8     return vector
```

在代码中，我们首先有一个长度为1899的空数组，对应词汇表。你将对VocalIndex()函数得到的索引序列进行遍历，其中出现的每个单词（索引），都在词汇表数组中对应单词位置置1标记。最后得到一个0-1数组，1表示单词在邮件中，0表示不在。

```
1  vector = VectoringEmail(emailSample1)
2  print(vector)
3  print('length of vector = {}\nnum of non-zero = {}'.format(len(vector),
    int(vector.sum()))))
4  vector = VectoringEmail(emailSample2)
5  print(vector)
6  print('length of vector = {}\nnum of non-zero = {}'.format(len(vector),
    int(vector.sum()))))
7  vector = VectoringEmail(spamSample1)
8  print(vector)
9  print('length of vector = {}\nnum of non-zero = {}'.format(len(vector),
    int(vector.sum()))))
10 vector = VectoringEmail(spamSample2)
11 print(vector)
12 print('length of vector = {}\nnum of non-zero = {}'.format(len(vector),
    int(vector.sum()))))
```

Cell输出:



[0. 0. 0. ... 0. 0. 0.]

length of vector = 1899

num of non-zero = 45

[0. 0. 0. ... 0. 0. 0.]

length of vector = 1899

num of non-zero = 120

[0. 0. 0. ... 0. 0. 0.]

length of vector = 1899

num of non-zero = 48

[0. 0. 0. ... 0. 0. 0.]

length of vector = 1899

num of non-zero = 18

对于样例emailSample1.txt。最终的特征向量的长度为1899，有45个非零条目。

2.3 使用支持向量机对邮件进行分类，进而进行垃圾邮件检测

最后我们要用SVM对垃圾邮件进行分类。我们将加载一个预处理的训练数据集，该数据集将用于训练SVM分类器。spamTrain.mat包含4000个垃圾邮件和非垃圾邮件的训练示例，而spamTest.mat包含1000个测试示例。这两个数据集中都保存了提取好的特征向量。我们将直接用线性核SVM进行训练，并在测试集上评估我们的SVM性能。

使用下述代码和垃圾邮件的训练数据训练用于垃圾邮件分类的线性SVM分类器。

```
1  from scipy.io import loadmat
2  # Training set
3  mat1 = loadmat('./spamTrain.mat')
4  X, y = mat1['X'], mat1['y']
5
6  clf = svm.SVC(C=0.1, kernel='linear')
7  clf.fit(X, y)
```

使用下述代码和测试数据集度量训练的SVM分类器的训练准确率和测试准确率。

```
1  # 装载测试数据集，用于性能评估
2  spamTest = loadmat('./spamTest.mat')
3  Xtest, ytest = spamTest['Xtest'], spamTest['ytest']
4  # 预测
5  perfTrain = clf.score(X, y)
6  perfTest = clf.score(Xtest, ytest)
7  print('在训练数据集上的准确率: ', perfTrain)
8  print('在测试数据集上的准确率: ', perfTest)
9
10 yTestPred = clf.predict(Xtest)
11 mse = mean_squared_error(ytest, yTestPred)
12 print(mse)
```

训练完成后，分类器的训练准确率约为99.8%，测试准确率约为98.9%。