实验三：支持向量机

一、实验简介

1.1 实验目的

了解支持向量机（Supported Vector Machine）的基本概念及原理，了解核函数的基本原理以及作用。通过实验学会 SVM 在简单分类问题中的应用并通过 SVM 构建一个垃圾邮件分类系统。

1.2 实验环境

本实验基于 Pycharm 集成开发环境使用 Python 语言（Python3）完成，依赖的第三方库主要有 Sklearn（一个强力的机器学习库）、Scipy（用于数学处理的软件包）、Numpy（一种开源的数值计算扩展库）以及 nltk（自然语言处理工具包）。
实验所用到的数据集：
dataset_1.mat, dataset_2.mat, dataset_3.mat 为实验上半部分使用的三个样例数据集。
spamTrain.mat，spamTest.mat，vocab.txt，emailSample1.txt，emailSample2.txt，spamSample1.txt，spamSample2.txt 为实验下半部分所使用的训练集、测试集、邮件样例、垃圾邮件样例。

1.3 实验原理

1.3.1 支持向量机

假定大小为 n 的训练集 $\{(x_i, y_i), \mathrm{i}=1,2,...,\mathrm{n}\}$，由二类别组成，如果 xi 属于第 1 类则标记 为正 $(y_i = 1)$，否则标记为负 $(y_i = -1)$。学习的目标是构造一个决策函数，将样本尽可能正确地 分类。而针对训练样本集分为线性和非线性两种情况。 对于线性情况，如果存在一个分类超平面 $\omega\mathrm{x}+\mathrm{b}=0$ 以最大的间隔将两类样本点划分开，使 得 $\omega x_i + \mathrm{b} \geq 1,\ y_i = 1$ 以及 $\omega x_i + \mathrm{b} \leq -1,\ y_i = -1$，则称训练集是线性可以分的。其中距离超平 面最近的样本点则称为支持向量，如图 1-1 所示：

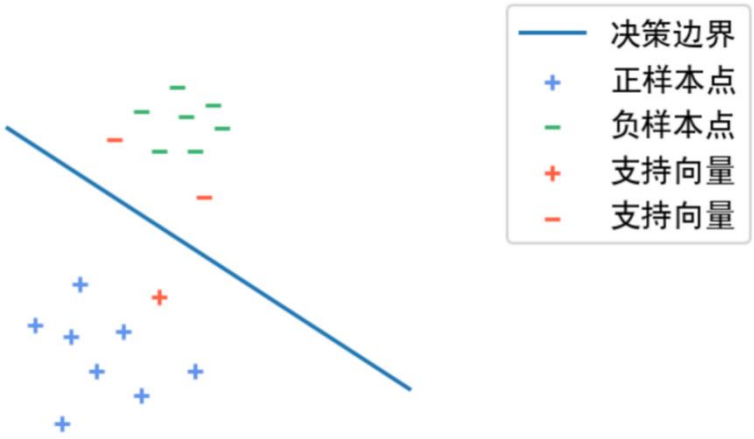

图 1-1 样例

1.3.2 核函数

对于线性不可分的样例，往往需要将样例特征映射至高维空间，但是并非所有问题都适用于此方法，维数过高同时带来的是更复杂和庞大的计算量，而核函数虽然完成了将特征从低维到高维的转换，但是它只基于低维进行计算从而避免了高维上的复杂计算量。

本实验所使用的核函数为高斯核函数（Gaussian Kernel），在支持向量机中同时也称为径向基核函数（Radial Basis Function, RBF），是非线性支持向量机最主流的核函数，公式如下所示：

$$K(x^i, x^j) = e^{(\frac{||x^i - x^j||^2}{2\sigma^2})} = e^{-\frac{\Sigma_{k=1}^{n}(x_k^i - x_k^j)^2}{2\sigma^2}}$$

公式 1-1 高斯核函数

其中 x 为具有 k 维特征的样本，x^i与x^j为样本集中的两不同样本。高斯核函数的计算结果衡量了两样本间的相似度距离，而其中 σ 则为决定了当样本之间距离愈远时相似度下降的快慢程度的一个参数。

二、支持向量机

实验的上半部分将使用 SVM 对三个数据集进行分类实验，包括线性可分和基于

高斯核函数的线性不可分的 SVM，以及如何搜索最优的模型参数。

首先需要导入程序所依赖的第三方库并对绘图参数进行适当配置：

```
# 依赖的第三方库
import scipy.misc, scipy.io, scipy.optimize
from sklearn import svm
from sklearn import model_selection
import csv
import re
import numpy as np
import matplotlib.pyplot as plt
# 使 matplotlib 绘图支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

为了使用的方便性并减少代码的冗余，我们需要实现一个图像绘制函数 plot(dataset)以及一个决策边界的绘制函数 visualize_boundary(X, model):

```
def plot(data):
    positives = data[data[:, 2] == 1]
    negatives = data[data[:, 2] == 0]
    # 正样本用+号绘制
    plt.plot(positives[:, 0], positives[:, 1], 'b+')
    # 负样本用 o 号绘制
    plt.plot(negatives[:, 0], negatives[:, 1], 'yo')


def visualize_boundary(X, trained_svm):
    kernel = trained_svm.get_params()['kernel']
    # 线性核函数
    if kernel == 'linear':
        w = trained_svm.coef_[0]
        i = trained_svm.intercept_
        xp = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
        a = -w[0] / w[1]
        b = i[0] / w[1]
```

```
        yp = a * xp - b
        plt.plot(xp, yp, 'b-')
    # 高斯核函数
    elif kernel == 'rbf':
        x1plot = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
        x2plot = np.linspace(min(X[:, 1]), max(X[:, 1]), 100)
        X1, X2 = np.meshgrid(x1plot, x2plot)
        vals = np.zeros(np.shape(X1))
        for i in range(0, np.shape(X1)[1]):
            this_X = np.c_[X1[:, i], X2[:, i]]
            vals[:, i] = trained_svm.predict(this_X)
        plt.contour(X1, X2, vals, colors='blue')
```

2.1 线性可分 SVM

在训练模型之前首先需要对数据集进行加载以及观察：

```
# 加载数据集 1
mat = scipy.io.loadmat("dataset_1.mat")
X, y = mat['X'], mat['y']
# 绘制数据集 1
plt.title('数据集 1 分布')
plot(np.c_[X, y])
plt.show(block=True)
```

绘制结果如图 2-1 所示，数据集 1 是一个简单的二维数据集并且可以清晰地看到两类数据点之间存在着明显的间隔。此外请注意在图像的最左边有一个偏离总体很远的正样本离群点，离群点对模型的拟合往往会有一定的影响作用。所以下一步请尝试给 SVM 模型的 C 参数赋予不同值以观察不同的拟合效果。其中 C 参数是调节对模型错误分类的惩罚的一个参数，越大的 C 参数则意味着模型需要更加精确地拟合样本以保证惩罚达到最小。

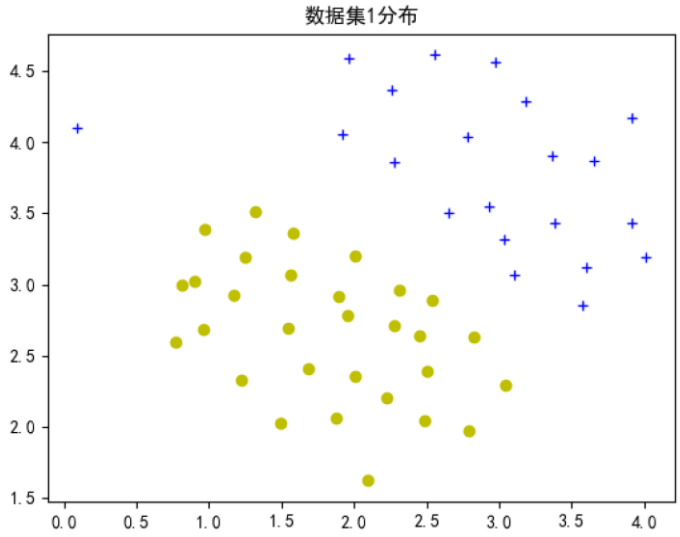

图 2-1 数据集 1 分布绘制

接下来尝试不同参数的 SVM 进行对比实验，首先是 C=1 时的 SVM 模型训练：

```
# 训练线性 SVM（C = 1）
linear_svm = svm.SVC(C=1, kernel='linear')
linear_svm.fit(X, y.ravel())
```

绘制该 SVM 拟合出的决策边界：

```
# 绘制 C=1 的 SVM 决策边界
plt.title('C=1 的 SVM 决策边界')
plot(np.c_[X, y])
visualize_boundary(X, linear_svm)
plt.show(block=True)
```

绘制结果如图 2-2 所示：

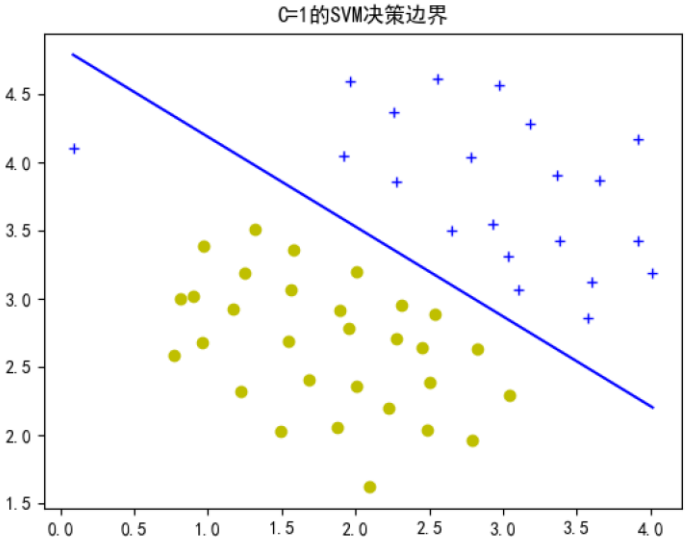

图 2-2 C=1 的 SVM 决策边界绘制

接下来是 C=100 时的 SVM 模型作为对比：

```
# 训练线性 SVM（C = 100）
linear_svm.set_params(C=100)
linear_svm.fit(X, y.ravel())
```

绘制出该 SVM 拟合出的决策边界：

```
# 绘制 C=100 的 SVM 决策边界
plt.title('C=100 的 SVM 决策边界')
plot(np.c_[X, y])
visualize_boundary(X, linear_svm)
plt.show(block=True)
```

绘制结果如图 2-3 所示：

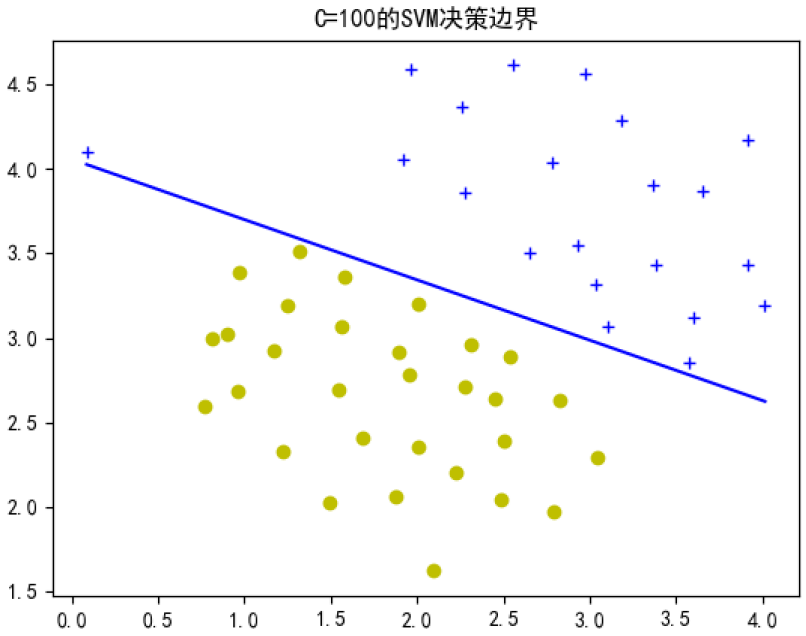

图 2-3 C=100 的 SVM 决策边界绘制

通过对比将会发现，C=1 时 SVM 训练出的决策边界大体上可以将两类样本区分，但是并没有正确地分类上面所提到的离群点，而 C=100 时训练得到的 SVM 正确地将该离群点划分到了对应类的样本中。

2.2 非线性可分 SVM

在这一步骤中将使用基于高斯核函数的 SVM 进行非线性分类。

首先基于公式 1-1 实现高斯核函数返回两样本之间的相似度距离：

```
def gaussian_kernel(x1, x2, sigma):
    return np.exp(-sum((x1 - x2) ** 2.0) / (2 * sigma ** 2.0))
```

根据高斯核函数计算给定的两个样例 x1 和 x2 的相似度：

```
# 计算高斯核函数
x1 = np.array([1, 2, 1])
x2 = np.array([0, 4, -1])
sigma = 2
print("样本 x1 和 x2 之间的相似度: %f" % gaussian_kernel(x1, x2, sigma))
```

高斯函数对样本 x1 和 x2 的计算结果如图 2-4 所示：

样本x1和x2之间的相似度：0.324652

图 2-4 高斯函数计算结果

下一步将需要加载并绘制数据集 2，用于非线性 SVM 的训练：

```
# 加载数据集 2
mat = scipy.io.loadmat("dataset_2.mat")
X, y = mat['X'], mat['y']
# 绘制数据集 2
plt.title('数据集 2 分布')
plot(np.c_[X, y])
plt.show(block=True)
```

绘制结果如图 2-5 所示：

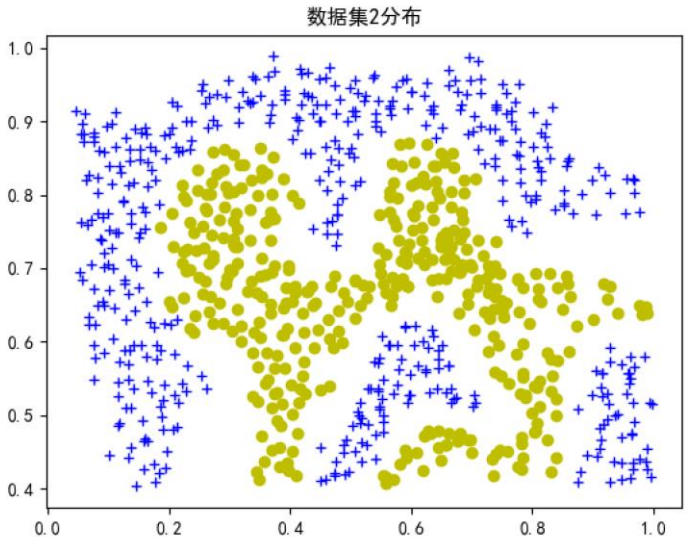

图 2-5 数据集 2 分布绘制

从图中可以观察到正样本与负样本之间并不存在线性划分边界，但是通过基于

高斯核函数的 SVM 则可以拟合出非线性的决策边界：

```
# 训练高斯核函数 SVM
sigma = 0.01
rbf_svm = svm.SVC(C=1, kernel='rbf', gamma=1.0 / sigma)  # gamma 实际上是 sigma 的倒数
rbf_svm.fit(X, y.ravel())
# 绘制非线性 SVM 的决策边界
plt.title('高斯核函数 SVM 决策边界')
plot(np.c_[X, y])
visualize_boundary(X, rbf_svm)
plt.show(block=True)
```

图 2-6 展现了基于高斯核函数的 SVM 在数据集 2 上拟合出的决策边界，可以观察到该决策边界可以将绝大部分正样本和负样本划分开。

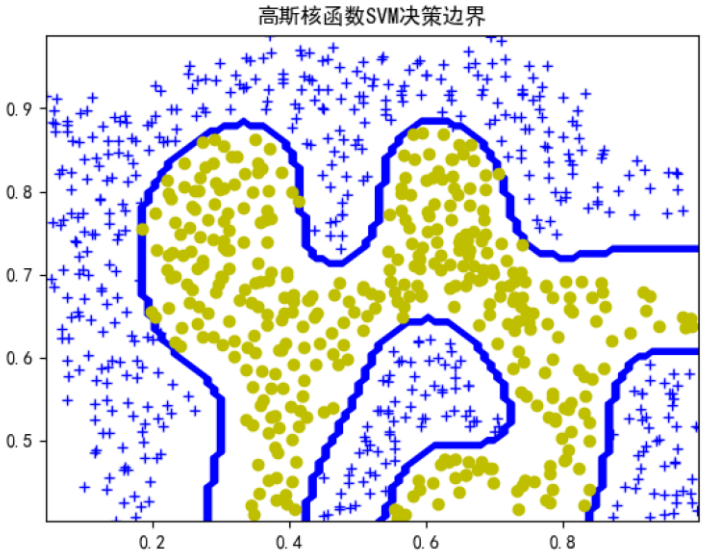

图 2-6 高斯核函数 SVM 决策边界绘制

2.3 最优参数搜索

在这一部分中将同样使用基于高斯核函数的 SVM 在数据集 3 上进行训练。在数据集 3 中，数据被划分为了训练集以及验证集。首先同样需要通过绘制训练集和验证集以观察数据：

```
# 加载数据集 3 获得训练集和验证集
mat = scipy.io.loadmat("dataset_3.mat")
X, y = mat['X'], mat['y'] #  训练集
```

```
X_val, y_val = mat['Xval'], mat['yval'] #  验证集

#  绘制数据集 3
plt.title('数据集 3 分布')
plot(np.c_[X, y])
plt.show(block=True)

#  绘制验证集
plt.title('验证集分布')
plot(np.c_[X_val, y_val])
plt.show(block=True)
```

训练集和验证集的绘制结果如图 2-7 和图 2-8 所示：

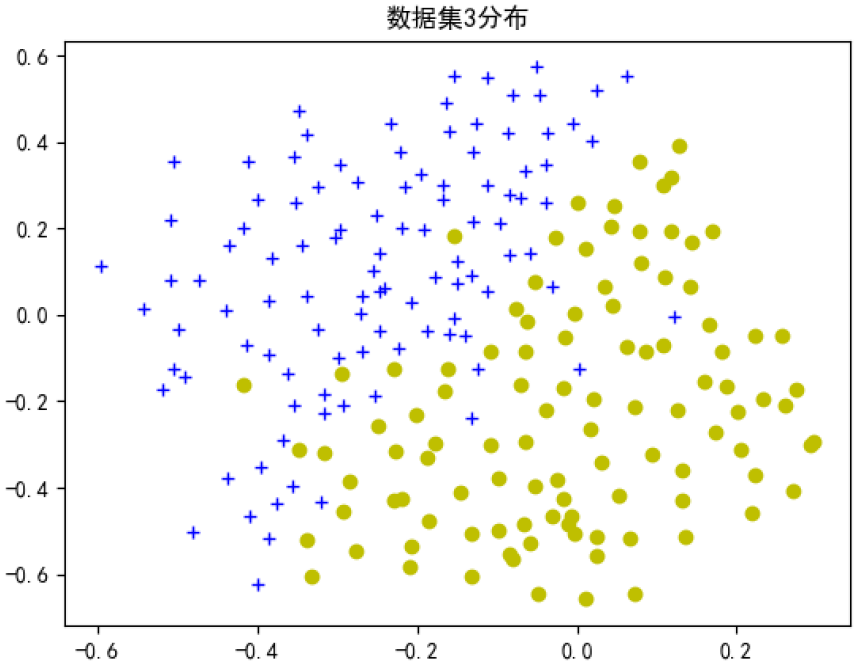

图 2-7 数据集 3-训练集

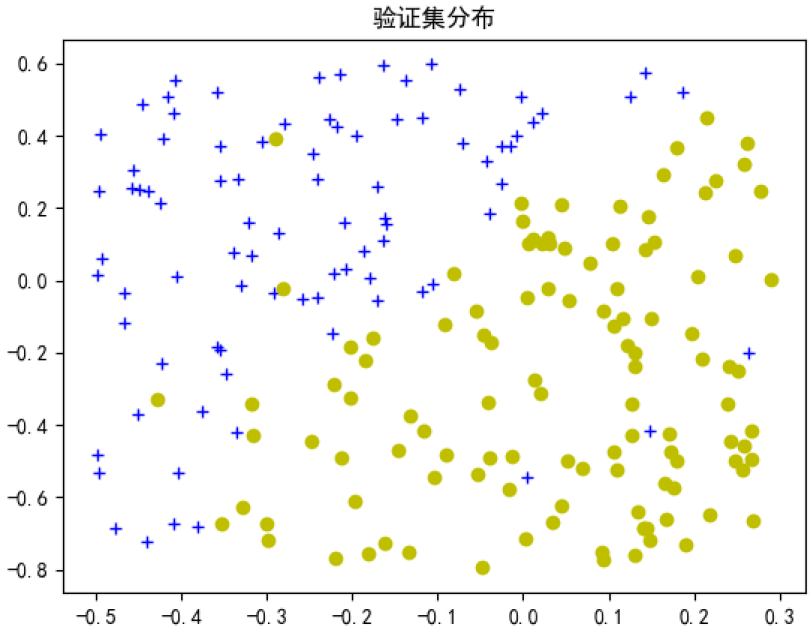

图 2-8 数据集 3-验证集

基于训练集训练的模型将在验证集上进行测试并根据验证集的反馈结果对模型参数进行微调。通过验证集对 SVM 模型的 C 参数和σ参数进行调整，对这两个参数建议的搜索范围为[0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]，也就是说在这两个参数上分别使用这 8 个值进行测试总共则需要 8*8=64 次实验。在确定了最优的 C 值与σ值之后，使用最优参数组合对模型进行训练并观察其拟合的决策边界。

首先需要实现一个用于参数搜索的函数 params_search(X, y, X_val, y_val)，注意可以通过计算分类结果的误差，例如在分类结果中，与真实值不符的结果所占比例来评估一组参数的适合程度，具有最小误差的一组参数则为最优的参数：

```
def params_search(X, y, X_val, y_val):
    np.c_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    raveled_y = y.ravel()
    m_val = np.shape(X_val)[0]

    rbf_svm = svm.SVC(kernel='rbf')
    best = {'error': 999, 'C': 0.0, 'sigma': 0.0}

    for C in np.c_values:
        for sigma in sigma_values:
```

```
            # 根据不同参数训练 SVM
            rbf_svm.set_params(C=C)
            rbf_svm.set_params(gamma=1.0 / sigma)
            rbf_svm.fit(X, raveled_y)

            # 预测并计算误差
            predictions = []
            for i in range(0, m_val):
                prediction_result = rbf_svm.predict(X_val[i].reshape(-1, 2))
                predictions.append(prediction_result[0])

            predictions = np.array(predictions).reshape(m_val, 1)
            error = (predictions != y_val.reshape(m_val, 1)).mean()

            # 记录误差最小的一组参数
            if error < best['error']:
                best['error'] = error
                best['C'] = C
                best['sigma'] = sigma
    best['gamma'] = 1.0 / best['sigma']
    return best
```

接下来则可通过参数搜索使用最优的参数组合训练模型：

```
# 训练高斯核函数 SVM 并搜索使用最优模型参数
rbf_svm = svm.SVC(kernel='rbf')
best = params_search(X, y, X_val, y_val)
rbf_svm.set_params(C=best['C'])
rbf_svm.set_params(gamma=best['gamma'])
rbf_svm.fit(X, y)
# 绘制决策边界
plt.title('参数搜索后的决策边界')
plot(np.c_[X, y])
visualize_boundary(X, rbf_svm)
plt.show(block=True)
```

得到的 SVM 决策边界如图 2-9 所示，

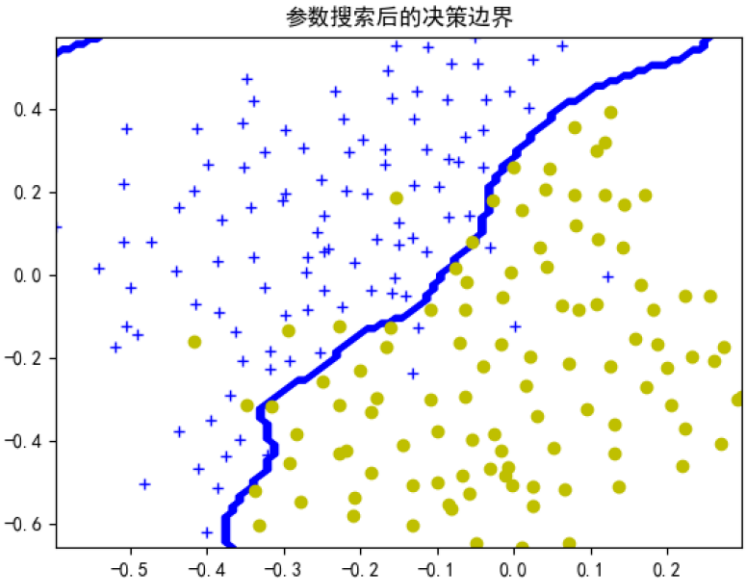

图 2-9 参数搜索 SVM

三、垃圾邮件分类

在本节实验将运用 SVM 实现一个常见的垃圾邮件分类的应用。给定的邮件 x 分为两种类型，对于垃圾邮件有 y=1，对于非垃圾邮件有 y=0。在开始训练模型前往往有一个重要的步骤，即数据的预处理与特征工程。实验中将对邮件文本内容进行预处理并将处理后的文本映射成一个特征向量，得到特征向量之后便可以开始训练分类模型。

3.1 邮件预处理

如上半部分实验所述相同，在开始训练机器学习模型之前首先对数据进行观察往往非常重要，图 3-1 是给定的两个邮件样例中的 emailSample1.txt 的文本，从文本中可以看到，邮件中除了正常词句以外还包含网页链接、邮箱地址、数字和美元等其他符号。对于垃圾邮件来说这类文本非常的常见，对此常用的方法则是对这类文本进行标准化（Normalize），这样一来对于模型来说此类特殊符号就不会受到模型的“特别关照”，而是被当作和其它普通文本一样看待。对于网页链接，我们可以将它替换成'httpaddr'，从而消除了不同网页链接之间的区别，模型将只会关心是否存在网页链接而不是关心这是百度还是知乎的链接。同理，对于邮箱地址我们也可以将其替换成'emailaddr'。

```
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors you're 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
```

图 3-1 emailSample1.txt 邮件文本

对于邮件文本使用到的预处理规则：

（1）大小写转换：将所有文本单词转换为小写形式，从而忽略大小写区别带来的影响。

（2）HTML 处理：由于很多邮件以 HTML 的格式被读取，消除 HTML 语言的标签如‘<>’从而只保留内容。

（3）网页链接：将所有网页链接转换为'httpaddr'。

（4）邮箱地址：将所有邮箱地址转换为'emailaddr'。

（5）数字：将所有数字转换为'number'。

（6）美元符号：将所有美元符号转换为'dollar'。

（7）词干提取：提取所有单词的词干。例如，'learn','learns','learned'和'learning'将被转换为'learn'。除此之外还可以直接删除单词的后缀，例如将'extract','extracts','extraction'等转换为‘extrac'。

（8）非词语删减：将词语以外的符号以及各种标点符号进行删减。

在对文本进行预处理之前将实现一个对给定的词典文件 vocab.txt 中的词汇进行标号的函数 vocaburary_mapping()：

```
def vocaburary_mapping():
    vocab_list = {}
    with open('vocab.txt', 'r') as file:
        reader = csv.reader(file, delimiter='\t')
        for row in reader:
            vocab_list[row[1]] = int(row[0])
    return vocab_list
```

函数得到的结果如图 3-2 所示，可以看到每个词汇都被映射到了一个序号，这将便于之后的文本特征提取。

```
{'aa': 1, 'ab': 2, 'abil': 3, 'abl': 4, 'about': 5, 'abov': 6, 'absolut': 7, 'abus': 8, 'ac': 9, 'accept': 10, 'access': 11, 'accord': 12,
```

图 3-2 词典映射结果

接下来将基于上述预处理规则对输入的邮件文本进行处理，并将预处理过后的邮件文本中每个词汇对应于词典中词汇的序号进行标注：

```
def email_preprocess(email):
    # 读取指定邮件文本
    with open(email, 'r') as f:
        email_contents = f.read()
    vocab_list = vocaburary_mapping()
    word_indices = []
    # 邮件文本预处理
    email_contents = email_contents.lower()
    email_contents = re.sub('<[^<>]+>', ' ', email_contents)
    email_contents = re.sub('[0-9]+', 'number', email_contents)
    email_contents = re.sub('(http|https)://[^\s]*', 'httpaddr', email_contents)
    email_contents = re.sub('[^\s]+@[^\s]+', 'emailaddr', email_contents)
    email_contents = re.sub('[$]+', 'dollar', email_contents)
    stemmer = nltk.stem.porter.PorterStemmer()
    tokens = re.split('[ ' + re.escape("@$/#.-:&*+=[]?!(){},\"'>_<;%\n") + ']', email_contents)

    for token in tokens:
        token = re.sub('[^a-zA-Z0-9]', '', token)
        token = stemmer.stem(token.strip())
        if len(token) == 0:
            continue
        if token in vocab_list:
            word_indices.append(vocab_list[token])
    # 返回邮件文本词汇与词典中词汇的对应关系，以及得到的预处理文本
    return word_indices, ' '.join(tokens)
```

对 emailSample1.txt 进行测试：

```
word_indices, processed_contents = email_preprocess('emailSample1.txt')
print(word_indices)
print(processed_contents)
```

图 3-3 则为邮件文本词汇与图 3-2 中词典词汇进行对应的结果，图 3-4 为邮件文本预处理后的结果：

[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]

图 3-3 邮件词汇映射结果(word_indices)

anyone knows how much it costs to host a web portal well it depends on how many visitors you re expecting this can be anywhere from less than number bucks a month to a couple of dollarnumber you should checkout httpaddr or perhaps amazon ecnumber if youre running something big to unsubscribe yourself from this mailing list send an email to emailaddr

图 3-4 邮件文本预处理结果(processed_contents)

3.2 邮件特征提取

在这一步中将需要实现对经过预处理后的邮件进行特征提取，即将邮件映射成一个 n 维向量 **x**，其中 n 为词典的词汇量，每一维对应词典中的一个词汇。在 **x** 中 $x_i \in \{0, 1\}$，$1 \leq i \leq n$，即对于第 i 个词汇，如果其出现在了邮件中，则有$x_i = 1$，否则$x_i = 0$。最终得到的特征向量结构如下所示：

$$\begin{bmatrix} 0 \\ \dots \\ 1 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{bmatrix}$$

基于上步对邮件词汇映射到词典所得的词汇索引 word_indices 实现特征提取：

```
def feature_extraction(word_indices):
    features = np.zeros((1899, 1))
    for index in word_indices:
        features[index] = 1
    return features
```

同样对 emailSample1.txt 进行测试：

```
features = feature_extraction(word_indices)
print(features)
```

图 3-5 则为对邮件样例 1 进行特征提取所得特征向量：

```
[[0.]
 [0.]
 [0.]
 ...
 [1.]
 [0.]
 [0.]]
```

图 3-5 emailSample1.txt 的特征向量

3.3 SVM 分类

接下来将基于一个已经预处理过数据的训练集 spamTrain.mat 进行 SVM 模型训练，该训练集包含 4000 条垃圾邮件和正常邮件的训练样本，测试集 spamTest.mat 包含 1000 条测试样本。其中每个样本已经经过了预处理已经特征提取。

```
#加载训练集
mat = scipy.io.loadmat("spamTrain.mat")
X, y = mat['X'], mat['y']
#训练 SVM
linear_svm = svm.SVC(C=0.1, kernel='linear')
linear_svm.fit(X, y.ravel())
# 预测并计算训练集正确率
predictions = linear_svm.predict(X)
predictions = predictions.reshape(np.shape(predictions)[0], 1)
print('{}%'.format((predictions == y).mean() * 100.0))
# 加载测试集
mat = scipy.io.loadmat("spamTest.mat")
X_test, y_test = mat['Xtest'], mat['ytest']
# 预测并计算测试集正确率
predictions = linear_svm.predict(X_test)
predictions = predictions.reshape(np.shape(predictions)[0], 1)
print('{}%'.format((predictions == y_test).mean() * 100.0))
```

训练结果如图 3-6 所示，训练集的准确率达到了 99.8%，测试集的准确率达到了 98.9%

```
99.825%
98.9%
```

图 3-6 SVM 训练结果

为了进一步理解垃圾邮件分类的运作原理，我们可以查看训练得到的 SVM 对各个词汇的权重系数来找出权重最高的若干词汇：

```
vocab_list = vocaburary_mapping()
reversed_vocab_list = dict((v, k) for (k, v) in vocab_list.items())
sorted_indices = np.argsort(linear_svm.coef_, axis=None)
for i in sorted_indices[0:15]:
    print(reversed_vocab_list[i])
```

结果如图 3-7 所示，当一封邮件包含如下单词时则会使得邮件有更高的概率被识别为垃圾邮件。

```
spam
that
urgent
wrong
datapow
linux
round
numberth
useless
unsubscrib
august
ratio
xp
toll
http
```

图 3-7 权重最大的前 15 个词汇

接下来尝试对给定的两封垃圾邮件样例进行预测，以 spamSample2.txt 为例：

```
word_indices, _ = email_preprocess('spamSample2.txt')
features = feature_extraction(word_indices).transpose()
print(linear_svm.predict(features))
```

输出结果如图 3-8 所示，预测结果为 y=1，即该封邮件属于垃圾邮件。

```
[1]
```

图 3-8 spamSample2.txt 预测结果