《人工智能与深度学习》课程

实 验 报 告

（第 1 次实验）

学号：12345678

姓名：LiPtP

日期：2025 年 3 月 25 日

实验 1：支持向量机

一、实验目的

了解支持向量机（Supported Vector Machine, SVM）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。

二、实验内容

本实验使用 Python 语言（Python 3.10.6）完成，依赖的第三方库包括 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）。

本实验使用的数据集如表 1 所示。在 SVM 训练的实验中主要使用前三个训练集，而对于垃圾邮件分类，则需要训练集、测试集以及垃圾邮件与正常文件的样本。多维样本以`.mat` 格式给出，文本格式的样本以`.txt` 格式给出。

2.1 线性可分 SVM

对应代码：附录 A.1 的 part1 函数

数据集	意义
dataset_1.mat	线性可分 SVM 数据集
dataset_2.mat	非线性可分 SVM 数据集
dataset_3.mat	非线性可分 SVM 数据集
spamTrain.mat	垃圾邮件训练集
spamTest.mat	垃圾邮件测试集
vocab.txt	词典映射文件
emailSample1.txt	邮件样本 1
emailSample2.txt	邮件样本 2
spamSample1.txt	垃圾邮件样本 1
spamSample2.txt	垃圾邮件样本 2

表 1: 实验使用数据集

首先使用 `scipy` 库导入`.mat` 类型的线性可分数据集 1，并使用 `matplotlib` 库绘图。绘图的结果如图 1 所示。

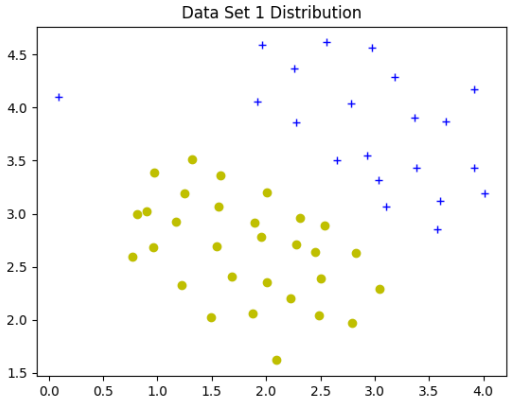


图 1: 数据集 1（线性可分）可视化

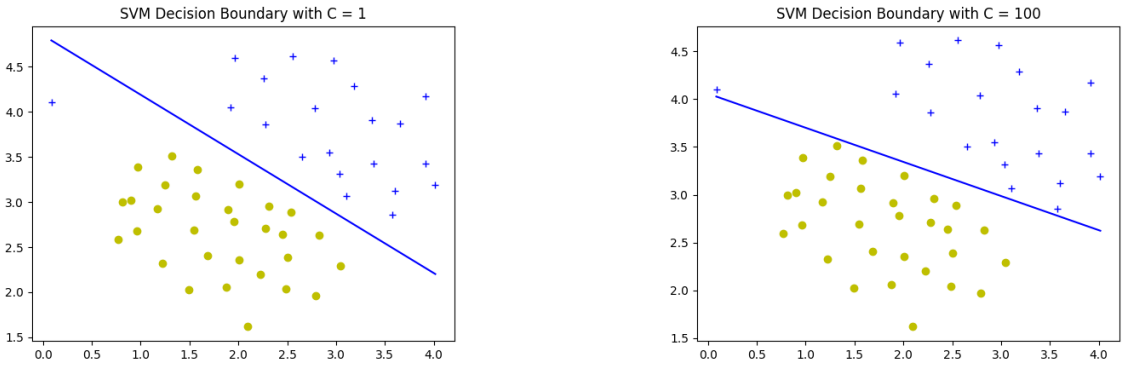
图表分析：显然可以看到两种类型的数据大体可以通过线性决策边界分开，但有少量用于测试模型性能的离群点。因此可称这种数据集为线性可分数据集。

分别调用 $C = 1$ 和 $C = 100$ 下的绘制 SVM 决策边界函数（`visualize_boundary`），得到不同 C 参数下数据集 1 的 SVM 决策边界如图 2。可以看到，$C = 100$ 时的决策边界包含了蓝色离群点，这是因为 $C = 100$ 时模型错误分类的惩罚权重更大，因此模型更倾向于包含离群点。

2.2 非线性可分 SVM

对应代码：附录 A.1 的 part2 函数

线性不可分指的是无法通过一个线性决策面来完美分隔数据集，这时候会出现分类错误。尽管线性可分的数据集分类方式简单，但这种数据集不具备普遍性。因此，需要找到一种方法处理非线性可分的数据集。



(a) $C = 1$ (b) $C = 100$

图 2: 不同 C 参数下数据集 1 的 SVM 决策边界

为了处理非线性可分的数据集，需要引入高斯核。它在 SVM 中也被称为径向基核函数（Radial Basis Function, RBF），定义如下：

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \tag{1}$$

其中，γ 是一个超参数，控制高斯核的影响范围。较大的 γ 使得模型更加关注局部信息，而较小的 γ 使得模型能够捕捉全局结构。

为了测试高斯核函数，使用它计算两个一维样本 `[1,2,1]` 和 `[0,4,-1]` 的相似度。这段代码的输出结果为：Similarity between object 1 and object 2: 0.324652。（对应代码：附录 A.1 的 part2 函数，gaussian_kernel 函数）结果验证正确。

接下来训练非线性 SVM。首先绘制出非线性可分的数据集 2 如图 3。可以看到正负样本间存在非线性的边界。

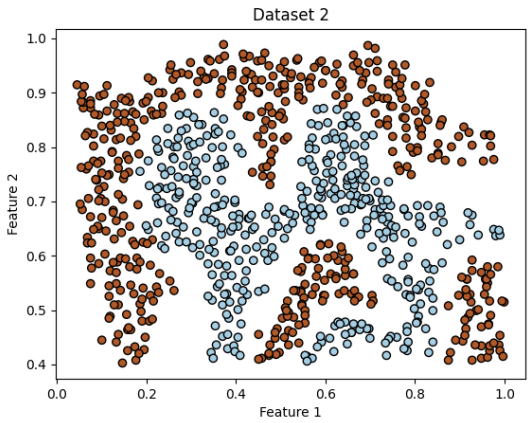


图 3: 数据集 2（线性不可分）可视化

使用 SVM 的 `fit` 方法训练 SVM，画出拟合决策边界如图 4。可以看到 SVM 基本正确地拟合了数据集 2 的非线性边界。

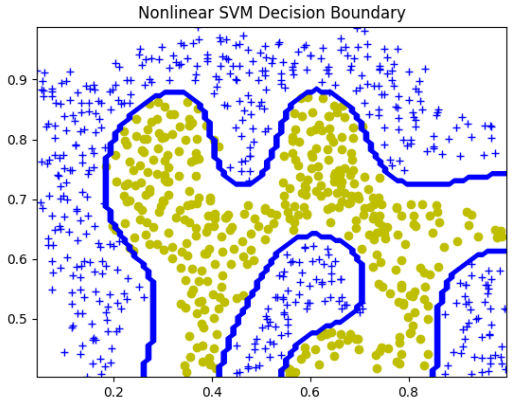


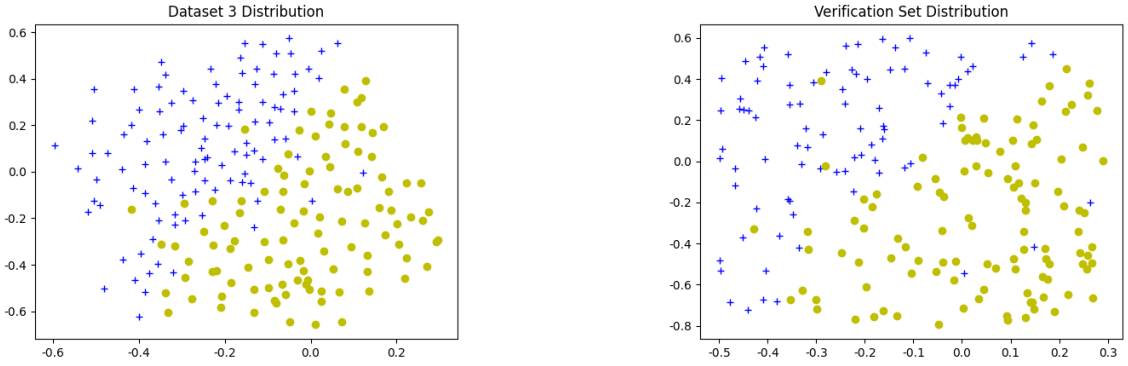
图 4: 数据集 2 的 SVM 决策边界

实验结果表明，高斯核方法能够有效处理非线性可分的数据集，并在较为复杂的数据结构中取得良好的分类效果。此外，超参数 γ 的选择对分类性能有重要影响，需要通过交叉验证或网格搜索进行优化。

2.3 最优参数搜索

对应代码：附录 A.1 的 part3 函数

在数据集 3 中，数据被划分为了训练集以及验证集。绘制训练集和验证集以观察数据（图 5 ）：



(a) 训练集 (b) 验证集

图 5: 数据集 3 的训练集和验证集

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。参数 C 和 σ 的搜索空间为 `[0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]`。确定最优参数组合后（对应代码：附录 A.1 的 params_search 函数 ）训练，得到的决策边界如图 6 所示。从图中可以看出 SVM 模型已经尽可能正负样本分开，但是还是存在少量样本未被彻底分开，如果要将其分开可能导致过拟合。

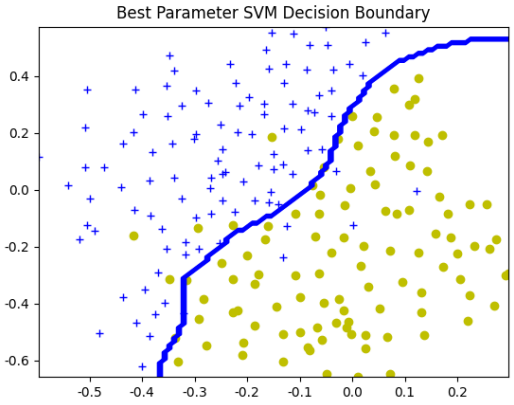


图 6: 数据集 3 的决策边界

2.4 垃圾邮件分类

根据 SVM 解决分类问题的方法，将垃圾邮件分类的步骤分为 4 步：

1. 邮件预处理：去除 HTML 标签等无关信息，将邮件中的单词映射到查找表中；
2. 邮件特征提取：将邮件映射为 n 维向量，当出现查找表中的一个单词时，对应行标 1；
3. SVM 训练：使用高质量训练集确定高维非线性边界；
4. 邮件预测：基于边界判断是否为垃圾邮件。

运行代码（对应代码：附录A.2 ），得到命令行输出如下所示，这符合我们的预期。他们分别对应四个部分的关键信息，意义在代码栏中已进行展示。

```
# 邮件预处理：单词对应下标
[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]
# 邮件预处理：待处理邮件中在查找表中的词汇
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
# 邮件特征提取：待处理邮件被映射为的n维向量
[[0.]
 [0.]
 [0.]
 ...
 [1.]
 [0.]
 [0.]]
# SVM训练：模型准确率（训练集、测试集）
99.825%
98.9%
# SVM训练：垃圾邮件中高词频单词
spam
that
urgent
wrong
datapow
linux
round
numberth
useless
unsubscrib
august
ratio
xp
toll
http

# 邮件预测：对第二封邮件进行垃圾邮件判决
[1]
邮件被分类为垃圾邮件。
```

三、提高训练

3.1 细化线性可分 SVM 的 C 参数

更改线性可分 SVM 的 C 参数，进一步观察是否会出现过拟合现象。挑选四个典型的 C 参数进行测试，画出的线性决策边界如图 7 所示。可以看到尽管 C 参数选得很大，但因为其为线性边界，过拟合现象并未发生。

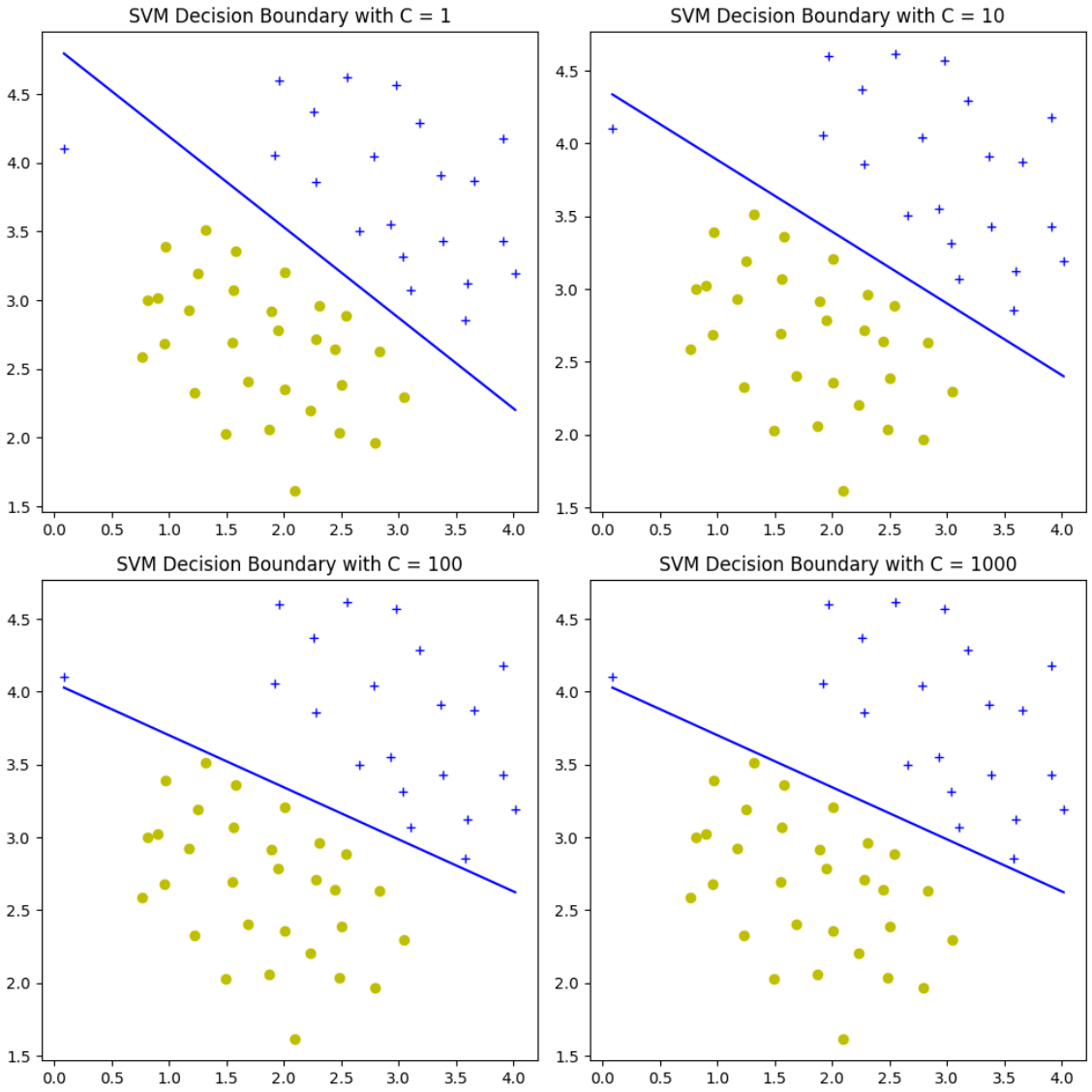
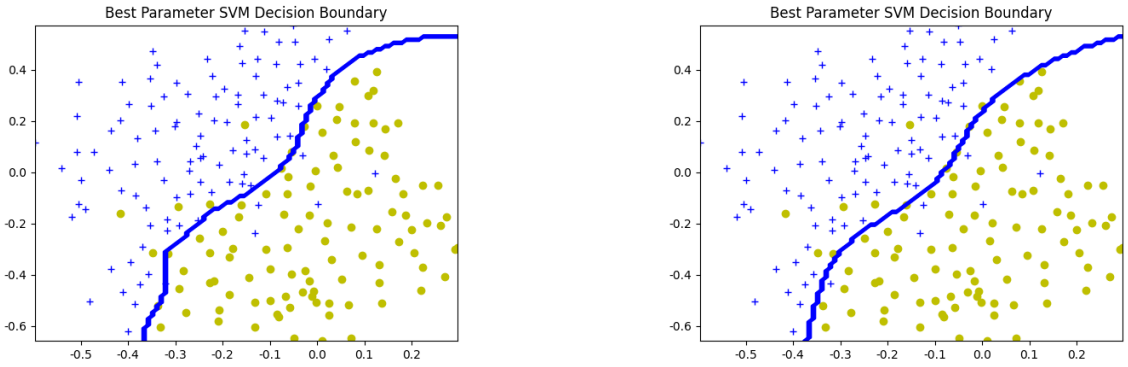


图 7: $C = 1,10,100,1000$ 时的线性可分 SVM 决策边界

另外，为了直接生成子图，对原有的 part1 部分代码进行了重写。新版的 part1 代码见附录 A.3。

3.2 调整搜索参数后的最优参数搜索

在章节 2.3 中，设定了参数 C 和 σ 的搜索空间为 `[0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]`。现在将其修改为：`[0.03, 0.02, 0.3, 0.2, 3, 2, 30, 20]`，在此参数搜索空间下，SVM 训练后得到的非线性边界有所不同。新的搜索空间下，分类结果变差了。



(a) 原搜索空间的边界　　(b) 更新搜索空间后的边界

图 8: 不同搜索空间下非线性边界的变化

打印两种搜索空间下选出的最优参数，前者为：

```
{
    'error': 0.03500000000000003,
    'C': 1,
    'sigma': 0.1,
    'gamma': 49.99999999999999
}
```

后者为：

```
{
    'error': 0.040000000000000036,
    'C': 0.3,
    'sigma': 0.2,
    'gamma': 12.499999999999998
}
```

可以看出，后者的误差变得更大了，且和前者相比，选择了更容易出偏差的参数。因此，在算力有限的情况下，优化参数选择需要精心设计搜索空间。

四、实验心得

在本次实验中，我理解了支持向量机的基本工作流程以及使用 Python 训练 SVM 模型，拟合线性、非线性边界的基本方法。并基于 SVM 完成了垃圾邮件分类识别的小实验。另外，我还独立编写了 LaTeX 模板（`SEU-AI-Report.cls`），用于本实验报告的撰写。在这个过程中也学到了不少关于 LaTeX 的进阶知识。

参考文献

[1] 邱锡鹏神经网络与深度学习. 北京: 机械工业出版社, 2020. https://nndl.github.io/.

附录 A 实验代码

该部分列举本实验中使用的所有实验代码。

A.1 SVM 训练相关代码

```
# -*- coding: utf-8 -*-

import scipy.misc, scipy.io, scipy.optimize
from sklearn import svm
from sklearn import model_selection
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['axes.unicode_minus'] = False

def plot(data):
    positives = data[data[:, 2] == 1]
    negatives = data[data[:, 2] == 0]

    plt.plot(positives[:, 0], positives[:, 1], 'b+')
    plt.plot(negatives[:, 0], negatives[:, 1], 'yo')

# 绘制SVM决策边界
def visualize_boundary(X, trained_svm):
    kernel = trained_svm.get_params()['kernel']
    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-')
    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')

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

def part1():
    mat = scipy.io.loadmat("dataset_1.mat")
    X, y = mat['X'], mat['y']

    plt.title('Data Set 1 Distribution')
    plot(np.c_[X, y])
    plt.show(block=True)

    linear_svm = svm.SVC(C=1, kernel='linear')
    linear_svm.fit(X, y.ravel())

    plt.title('SVM Decision Boundary with C = 1')
    plot(np.c_[X, y])
    visualize_boundary(X, linear_svm)
    plt.show(block=True)

    linear_svm = svm.SVC(C=100, kernel='linear')
    linear_svm.fit(X, y.ravel())

    plt.title('SVM Decision Boundary with C = 100')
    plot(np.c_[X, y])
    visualize_boundary(X, linear_svm)
    plt.show(block=True)

def part2():
    x1 = np.array([1, 2, 1])
    x2 = np.array([0, 4, -1])
    sigma = 2
    print("Similarity between object 1 and object 2: %f" %
   gaussian_kernel(x1, x2, sigma))

    mat_data = scipy.io.loadmat("dataset_2.mat")
```

```
    X = mat_data['X']
    y = mat_data['y'].ravel()

    plt.figure()
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, edgecolors
   ='k')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("Dataset 2")
    plt.show()

    sigma = 0.1
    rbf_svm = svm.SVC(C=1, kernel='rbf', gamma=1.0 / (2 * sigma ** 2)
   )
    rbf_svm.fit(X, y)

    plt.title('Nonlinear SVM Decision Boundary')
    plot(np.c_[X, y])
    visualize_boundary(X, rbf_svm)
    plt.show(block=True)

def part3():
    mat = scipy.io.loadmat("dataset_3.mat")
    X, y = mat['X'], mat['y']
    X_val, y_val = mat['Xval'], mat['yval']

    plt.title('Dataset 3 Distribution')
    plot(np.c_[X, y])
    plt.show(block=True)

    plt.title('Verification Set Distribution')
    plot(np.c_[X_val, y_val])
    plt.show(block=True)

    best = params_search(X, y, X_val, y_val)
    rbf_svm = svm.SVC(C=best['C'], kernel='rbf', gamma=best['gamma'])
    rbf_svm.fit(X, y.ravel())

    plt.title('Best Parameter SVM Decision Boundary')
    plot(np.c_[X, y])
    visualize_boundary(X, rbf_svm)
```

```
    plt.show(block=True)

def params_search(X, y, X_val, y_val):
    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]

    best = {'error': 999, 'C': 0.0, 'sigma': 0.0}

    for C in c_values:
        for sigma in sigma_values:
            rbf_svm = svm.SVC(C=C, kernel='rbf', gamma=1.0 / (2 *
   sigma ** 2))
            rbf_svm.fit(X, y.ravel())
            error = 1 - rbf_svm.score(X_val, y_val)

            if error < best['error']:
                best['error'] = error
                best['C'] = C
                best['sigma'] = sigma

    best['gamma'] = 1.0 / (2 * best['sigma'] ** 2)
    return best

def main():
    np.set_printoptions(precision=6, linewidth=200)
    part1()
    part2()
    part3()

if __name__ == '__main__':
    main()
```

程序 1: ex3.py

A.2 垃圾邮件分类相关代码

```
# -*- coding: utf-8 -*-

import csv
import re
import pickle
```

```
6
7  import numpy as np
8  import nltk, nltk.stem.porter
9  import scipy.misc, scipy.io, scipy.optimize
10 from sklearn import svm
11
12
13 def vocaburary_mapping():
14     vocab_list = {}
15     with open('vocab.txt', 'r') as file:
16         reader = csv.reader(file, delimiter='\t')
17         for row in reader:
18             vocab_list[row[1]] = int(row[0])
19
20     return vocab_list
21
22 def feature_extraction(word_indices):
23     features = np.zeros((1899, 1))
24     for index in word_indices:
25         features[index] = 1
26     return features
27
28
29 def email_preprocess(email):
30     with open(email, 'r') as f:
31         email_contents = f.read()
32     vocab_list = vocaburary_mapping()
33     word_indices = []
34     email_contents = email_contents.lower()
35     email_contents = re.sub('<[^<>]+>', ' ', email_contents)
36     email_contents = re.sub('[0-9]+', 'number', email_contents)
37     email_contents = re.sub('(http|https)://[^\s]*', 'httpaddr',
   email_contents)
38     email_contents = re.sub('[^\s]+@[^\s]+', 'emailaddr',
   email_contents)
39     email_contents = re.sub('[$]+', 'dollar', email_contents)
40     stemmer = nltk.stem.porter.PorterStemmer()
41     tokens = re.split('[ ' + re.escape("@$/#.-:&*+=[]?!(){},'\">_<;%\
   n") + ']', email_contents)
42
43     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])

    return word_indices, ' '.join(tokens)



# 预处理
def part_1():
    word_indices, processed_contents = email_preprocess('emailSample1.txt')
    print(word_indices)
    print(processed_contents)


# 特征提取
def part_2():
    word_indices, processed_contents = email_preprocess('emailSample1.txt')
    features = feature_extraction(word_indices)
    print(features)


# SVM模型训练
def part_3():
    #加载训练集
    mat = scipy.io.loadmat("spamTrain.mat")
    X, y = mat['X'], mat['y']

    # linear_svm = pickle.load(open("linear_svm.svm", "rb")) # 模型加载
    #训练SVM
    linear_svm = svm.SVC(C=0.1, kernel='linear')
    linear_svm.fit(X, y.ravel())
    # pickle.dump(linear_svm, open("linear_svm.svm", "wb")) # 模型保
```

```
    存
    # 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))

    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])

def part_4():

    word_indices, processed_contents = email_preprocess('spamSample2.
   txt')
    features = feature_extraction(word_indices)
    linear_svm = svm.SVC(C=0.1, kernel='linear')
    linear_svm = pickle.load(open("linear_svm.svm", "rb"))
    # Prediction
    prediction = linear_svm.predict(features.T)
    print(prediction)
    # Output
    if prediction == 1:
        print("邮件被分类为垃圾邮件。")
    else:
        print("邮件被分类为非垃圾邮件。")


part_1()
part_2()
part_3()
```

```
part_4()
# print(vocaburary_mapping())
```

程序 2: ex3_spam.py

A.3 绘制线性可分 SVM 子图相关代码

```
def plot(data, ax=None):
    if ax is None:
        ax = plt.gca()  # 获取当前的 Axes
    positives = data[data[:, 2] == 1]
    negatives = data[data[:, 2] == 0]

    ax.plot(positives[:, 0], positives[:, 1], 'b+', label='Positive')
    ax.plot(negatives[:, 0], negatives[:, 1], 'yo', label='Negative')

# 绘制SVM决策边界
def visualize_boundary(X, trained_svm, ax=None):
    if ax is None:
        ax = plt.gca()  # 获取当前的 Axes

    kernel = trained_svm.get_params()['kernel']
    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
        ax.plot(xp, yp, 'b-')
    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)

```

```
        ax.contour(X1, X2, vals, colors='blue')


def part1():
    mat = scipy.io.loadmat("dataset_1.mat")
    X, y = mat['X'], mat['y']

    plt.title('Data Set 1 Distribution')
    plot(np.c_[X, y])
    plt.show(block=True)

    fig, axes = plt.subplots(2, 2, figsize=(10, 10))
    C_values = [1, 10, 100, 1000]
    for ax, C in zip(axes.ravel(), C_values):
        linear_svm = svm.SVC(C=C, kernel='linear')
        linear_svm.fit(X, y.ravel())

        ax.set_title(f'SVM Decision Boundary with C = {C}')
        plot(np.c_[X, y],ax=ax)
        visualize_boundary(X, linear_svm,ax=ax)

    plt.tight_layout()
    plt.show()
```

程序 3: 新版 part1 函数以及相关改动