

机器学习Project 1 报告

姓名	Ruiqi
Github	Ruiqi327/THU-ML-2025-Fall-Project

一、概述

本项目从零实现了K-near-neighbors (KNN)、Support vector machine (SVM)这两种机器学习算法。具体来说，我们的工作包括以下几个方面：

数据集构建

我们手动构建了一个四特征二分类的数据集，用来测试不同算法的分类效果。该数据集包括三类不同的数据分布，分别为线性、非线性和线性有噪，每一类都包含3000个样本。在此基础上，我们额外构建了一个二特征二分类数据集，用来绘制不同算法的决策边界。

KNN

- 我们实现了分别基于L1距离和L2距离的两种KNN算法，在三个自建的四特征二分类数据集上进行了测试，并分析了邻近样本参数 k 对分类效果的影响。
- 在三个二特征二分类数据集上运行了实现的两个KNN算法，绘制了决策边界。
- 我们在HIGGS数据集上测试了两种KNN算法，分析了参数 k 和训练规模对最终分类结果的影响。利用490K样本进行训练后，L1-KNN在测试集上实现了最佳64.05%的分类准确率。

SVM

- 我们实现了线性核SVM，手动编写了基于梯度下降法的线性核SVM优化器。在三个自建的四特征二分类数据集（Linear, Noisy, Nonlinear）上训练了SVM模型并分析了结果。
- 我们在三个二特征二分类数据集上采用相同的方式训练了手动实现的线性核SVM模型，并绘制了其决策边界。
- 针对HIGGS数据集，我们手动编写了基于随机梯度下降（SGD）的SVM优化算法，训练并测试了一个线性SVM模型，分析了超参数（惩罚系数）对于算法结果的影响。
- 针对HIGGS数据集，我们实现了一种基于RBF核方法的SVM模型，并采用script.minimize方法对该模型进行训练。由于这一方法计算复杂度超过 $\mathcal{O}(n^3)$ ，我们选取了小部分HIGGS数据集进行训练，报告并分析其在训练集核测试集上的表现。
- 作为对比，我们使用Sklearn自带的SVM训练工具在HIGGS数据集进行了训练。利用3,0000个样本进行训练后，该模型在测试集上实现了64.77%的分类准确率。

二、数据集准备

四特征二分类数据集

该数据集用于预测某同学是否能够通过期末考试（Pass/Fail）。包括以下四个特征：

- (1) 每天学习时长：lt: 1-8
- (2) 期中考试成绩：mid_grade:30-100

(3) 出勤率: attendance_rate:30%-100%

(4) 作业完成情况: homework 30%-100%

生成规则：一般而言，每天学习时长lr与期末考试成绩相关程度最高，其次是期中考试成绩，作业完成情况和出勤率。我们将全部特征都按列归一化，并且默认按照以下公式来判断是否能通过考试：

$$p = 0.5lt + 0.2midgrade + 0.2attendencerate + 0.1homework > 0.5. \quad (1)$$

上述公式将被用来生成Linear_dataset。为了观察算法在不同数据集分布下的表现，我们还外制作了Noisy_dataset和Nonlinear_dataset这两个数据集，其判别是否能通过期末考试的公式分别为

$$p_{noisy} = p \pm random(-0.3, 0.3) > 0.50.25 < p_{nonlinear} = p < 0.75. \quad (2)$$

我们为每一个数据集都生成了3000个样本，且各个特征均匀分布。通过期末考试的标签记为1，不通过则记为-1。在后续测试时，我们固定随机数种子为42，划分2400个样本作为训练集，300个作为验证集，300个作为测试集。

二特征二分类数据集

该数据集是上述四特征二分类数据集的简化版本，用于绘制不同算法的决策边界。具体来说，我们删去了出勤率和作业完成情况这两个特征，在将其他数据特征归一化后按照以下公式判断是否通过期末考试：

$$p = 0.6lt + 0.4midgrade > 0.5. \quad (3)$$

其他两个数据集的判别公式保持不变。同时，我们同样为每一个数据集都生成3000个样本。

HIGGS数据集

HIGGS数据集是一个源自高能物理领域的二分类数据集。它的任务是从粒子加速器实验产生的背景噪声中，区分出由希格斯玻色子（Higgs boson）产生的信号。完整数据集包含1,100万个样本，每个样本有28个特征。作业中我们选取前50万个样本进行实验，同时划分前490,000个样本作为训练集，490,001-495,000作为验证集，最后5000个样本作为测试集。

三、主要结果

1、KNN

本节我们首先实现了基于两种距离计算方式的KNN，在构建的四特征二分类数据集上进行训练并测试结果。随后，我们在二特征二分类数据集上绘制了这两种KNN的决策边界。最后，我们在HIGGS数据集上测试了它们的分类准确率，并分析训练样本和超参数对算法表现的影响。

(1) 模型实现

KNN的思想是在测试集中寻找与被测试样本距离最近的k个训练样本。实现代码如下：

```
def MyKNN(X_train, y_train, X_pred, k=5):
    num_test_samples = X_pred.shape[0]
    y_pred = np.zeros(num_test_samples)
    for i in range(num_test_samples):
        distances = np.linalg.norm(X_train - X_pred[i], axis=1) #L2 distance
        #distances = np.linalg.norm(X_train - X_pred[i], ord=1, axis=1) #L1 distance
        k_indices = np.argsort(distances)[:k]
```

```

k_nearest_labels = y_train[k_indices]
unique, counts = np.unique(k_nearest_labels, return_counts=True)
y_pred[i] = unique[np.argmax(counts)]
return y_pred

```

这里我们通过控制distance计算方式，实现了 L_1 和 L_2 两种距离。为了消除不同维度变量间差距过大的影响，在计算距离之前，我们将所有变量都min-max归一化至(0, 1)之间，代码如下：

```

X_min = X_train.min(axis=0)
X_max = X_train.max(axis=0)
X_train = (X_train - X_min) / (X_max - X_min)
X_val = (X_val - X_min) / (X_max - X_min)
X_test = (X_test - X_min) / (X_max - X_min)

```

(2) 自建数据集测试结果

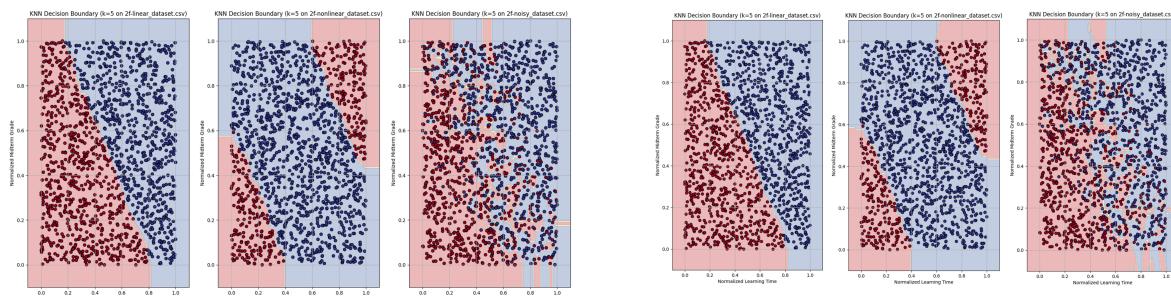
我们在自建的四分类二特征数据集上测试了实现的KNN模型，并分析了不同的参数 k 对最终分类结果的影响。由于KNN本质是一个无需训练的模型，这里我们仅报告在测试集上的结果，如下表所示。

L1-KNN / L2-KNN	Linear dataset	Nonlinear dataset	Noisy dataset
k=1	0.9767 / 0.9833	0.9500 / 0.9467	0.6800 / 0.6677
k=3	0.9833 / 0.9867	0.9433 / 0.9467	0.7200 / 0.7067
k=5	0.9767 / 0.9667	0.9533 / 0.9433	0.7167 / 0.7100
k=7	0.9677 / 0.9633	0.9433 / 0.9500	0.7200 / 0.7133
k=10	0.9700 / 0.9667	0.9500 / 0.9467	0.7067 / 0.7133
k=15	0.9677 / 0.9733	0.9333 / 0.9433	0.7333 / 0.7167
k=30	0.9633 / 0.9600	0.9333 / 0.9267	0.7400 / 0.7333

从表中可以看出，随着 k 的增大，L1-KNN和L2-KNN在linear dataset和nonlinear dataset上的表现均略有下降，但总体变化不大；但其在Noisy dataset的测试集上准确率却有较大的提升。同时，三个数据集上L1-KNN的表现都略微好于L2-KNN。上述结果表明增大 k 有利于提升KNN模型对于噪声数据的鲁棒性，但可能会损失决策边界处的决策准确率。同时，针对不同的数据集，不同的距离类型也会影响到测试结果。

(3) 决策边界绘制

我们在自制的二特征二分类数据集上测试了设计的L1-distance KNN和L2-distance KNN，并绘制出它们的决策边界。为了增加可视化的清晰度，我们仅选取300个样本作为训练集，设置 $k = 5$ 。绘制结果如下所示。（从左到右：L1-KNN / L2-KNN）



从图中可以看出，L1-KNN和L2-KNN在线性和非线性数据集上都取得了很好的分类结果，决策边界清晰地将正负样本分开。而在噪声数据集上，两种KNN都具有明显不规则的决策边界，这体现了KNN对噪声点敏感的特性。

(4) HIGGS数据集测试结果

我们在划分好的HIGGS数据集上测试了两种KNN模型的分类结果，并分析了训练集大小T size和邻居数目 k 对分类结果的影响。在测试之前，我们同样对不同维度的特征进行min-max归一化。测试结果如下：

L1-KNN / L2-KNN	T size=100,000	T size=200,000	T size=300,000	T size=490,000
k=1	0.5757 / 0.5577	0.5785 / 0.5641	0.5777 / 0.5631	0.5819 / 0.5763
k=3	0.5965 / 0.5643	0.5995 / 0.5715	0.6011 / 0.5775	0.6095 / 0.5865
k=5	0.5947 / 0.5729	0.5971 / 0.5875	0.6023 / 0.5861	0.6179 / 0.5855
k=7	0.6089 / 0.5767	0.6071 / 0.5869	0.6079 / 0.5811	0.6223 / 0.5969
k=10	0.6085 / 0.5745	0.6119 / 0.5843	0.6127 / 0.5823	0.6203 / 0.5963
k=15	0.6141 / 0.5827	0.6293 / 0.5899	0.6219 / 0.5995	0.6279 / 0.6005
k=30	0.6175 / 0.5919	0.6303 / 0.6009	0.6395 / 0.6055	0.6405 / 0.6097

基于表中数据，我们可以得到三个关键结果：

- 增大训练集样本能够有效提高KNN预测准确率。
- HIGGS是一个充满复杂噪声的数据集，在一定范围内，增大KNN算法的邻近样本数量 k 能够有效提高算法性能。但值得指出的是，这一做法并非一直有效：实验中我们发现，当 k 超过30时，继续增大 k 反而会导致性能下降。
- L1-KNN比L2-KNN更适合HIGGS数据集，其表现更好。

测试过程中，我们还统计了L2-KNN的运行时间。在单卡 RTX-5880服务器上，根据30,0000个样本预测测试集的结果花费了2397.2461s（约40分钟）。这验证了KNN的低效性（其时间复杂度为 $\mathcal{O}(mn)$ ，其中 m 是测试集样本数， n 是训练集样本数）。

2、SVM

在本节中，我们首先针对自建四特征二分类数据集实现了线性SVM和对应的梯度下降法，并在自制的二特征二分类数据集上绘制了SVM模型的决策边界。随后，我们针对HIGGS数据集实现了两个SVM算法：1) 基于随机梯度下降的线性SVM，2) 基于Script.minimize方法的RBF核SVM。

(1) 线性SVM模型实现

线性SVM模型即为优化如下函数：

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|\omega\|^2, \\ \text{s.t.} \quad & y^{(i)}(\omega^\top x^{(i)} + b) \geq 1 \quad (i = 1, 2, \dots, n) \end{aligned} \tag{4}$$

其中 ω 是系数， b 是偏置项。为了将不等式约束消除，我们利用外点法，将(4)转化为：

$$\min_{w,b} \quad \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^n \max \left(1 - y^{(i)}(\omega^\top x^{(i)} + b), 0 \right), \tag{5}$$

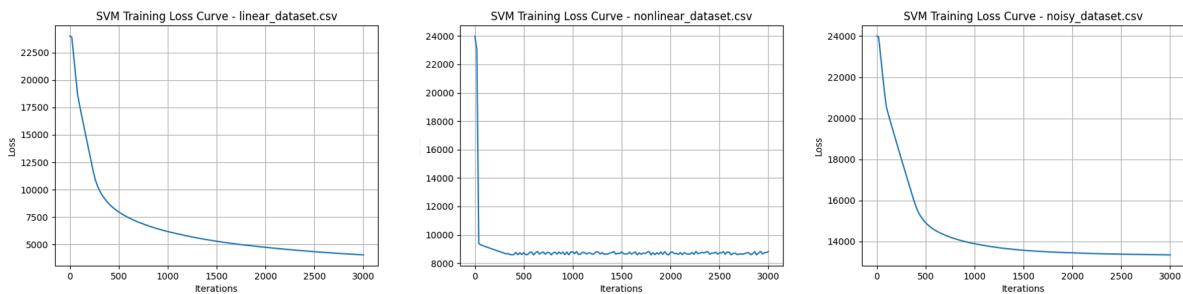
利用梯度下降法可以求解上述优化问题，具体实现如下：

```
def MySVM(X_train, y_train, iterations=1000, lr=1e-5, penalty=1):
    # initialize w, b
    num_samples, num_features = X_train.shape
    w = np.zeros(num_features)
    b = 0
    loss = np.sum(penalty * np.maximum(0, 1 - y_train * (np.dot(X_train, w) + b))) + 0.5 * np.dot(w, w)
    loss_history = [loss]
    for i in tqdm(range(iterations), desc="Training SVM"):
        gradient_w = np.zeros(num_features)
        gradient_b = 0
        for idx, x_i in enumerate(X_train):
            condition = y_train[idx] * (np.dot(w, x_i) + b) >= 1
            if not condition:
                gradient_w -= penalty * y_train[idx] * x_i
                gradient_b -= penalty * y_train[idx]
        gradient_w += w
        w -= gradient_w * lr
        b -= gradient_b * lr
    return w, b, loss_history
```

为方便监督训练，我们调用了tqdm库来实时生成进度条，并在每20次梯度更新后打印一次loss。在训练之前，我们同样对所有特征进行min-max归一化。

(2) 模型训练

我们在自建的四特征二分类数据集上测试实现的线性SVM模型。超参数包括学习率lr，惩罚系数C和总迭代次数iterations。篇幅所限，这里我们仅报告最终结果lr=2e-6，iterations=3000和C=10时的训练loss曲线和模型参数：



SVM 模型参数	w_1	w_2	w_3	w_4	b
Linear	5.3621	1.6327	1.5100	0.6158	-4.4874
Nonlinear	0.0065	0.0120	0.0099	0.0147	1.0280
Noisy	3.2573	1.1241	1.0867	0.4050	-2.9079

训练完毕的线性SVM在linear dataset, nonlinear dataset和noisy dataset的训练集上准确率分别为：96.67%、82.13%和75.08%，测试集准确率分别为96.33%、82.33%和73.33%。这表明线性SVM在线性可分问题上性能较好，但无法处理非线性或高度有噪数据集。

(3) 超参数分析

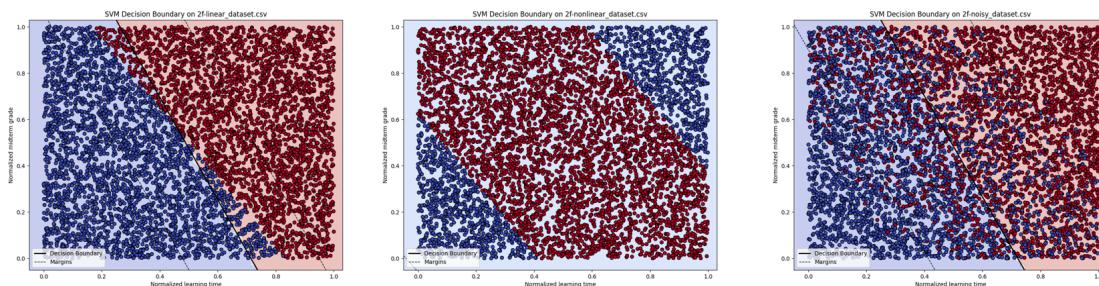
我们探索了惩罚系数C对于线性SVM模型准确率的影响。设置训练参数为lr=2e-5, iterations=1000，并保持不变，线性SVM的在测试集上的准确率如下：

线性SVM 准确率	C=0.1	C=1	C=5	C=10	C=30
Linear	0.5967	0.9233	0.9833	0.9900	0.9933
Nonlinear	0.8200	0.8200	0.8200	0.8200	0.8200
Noisy	0.5067	0.7433	0.7333	0.7333	0.7333

从表中可以看出：在Nonlinear和Noisy数据集上，增大惩罚系数C并不能有效地提高算法性能，这是因为线性SVM本身无法解决不可线性可分的分类问题。而在linear数据集上，增大惩罚系数有效地提高了算法的准确率。并且，当C较小时，算法还将出现“灾难式”崩溃，即准确率突然大幅度降低。这是因为我们采用外点法来处理SVM原始优化问题中的不等式约束，惩罚系数过小会导致约束的过于松弛化，最终导致算法崩溃。

(4) 决策边界绘制

我们在自建的二分类二特征数据集上运行了实现的线性SVM模型，并绘制了对应的决策边界。这里，线性SVM的超参数被设置为C=2，训练参数iterations=500, lr=1e-5。为方便可视化，我们仅选取600个样本作为训练集。最终SVM在linear, nonlinear和noisy 的测试集上的准确率分别为95%、74.33%和78.67%。绘制的决策边界如下：



在线性与有噪声的二分类数据集中，线性SVM的决策边界都大致均匀地将两种标签分开。而在非线性数据集中，决策边界完全偏离数据分布区域（未在图中显现），导致线性SVM预测所有样本都将“无法通过期末考试”。一方面这与数据集中样本分布不均有关，另一方面这证明了线性SVM的局限性：无法有效地对线性不可分的数据进行分类，极端情况下甚至可能出现本例中将所有样本都认为成某一类数据的现象。

(5) 线性核SVM (HIGGS数据集) —— 基于SGD优化方法

HIGGS数据集中有49,0000个训练样本，若在每步更新参数时都遍历全部数据集，会导致过高的时间复杂度。因此，这里我们手动实现了一个随机梯度下降的优化算法来训练实现的线性SVM。实现代码如下：

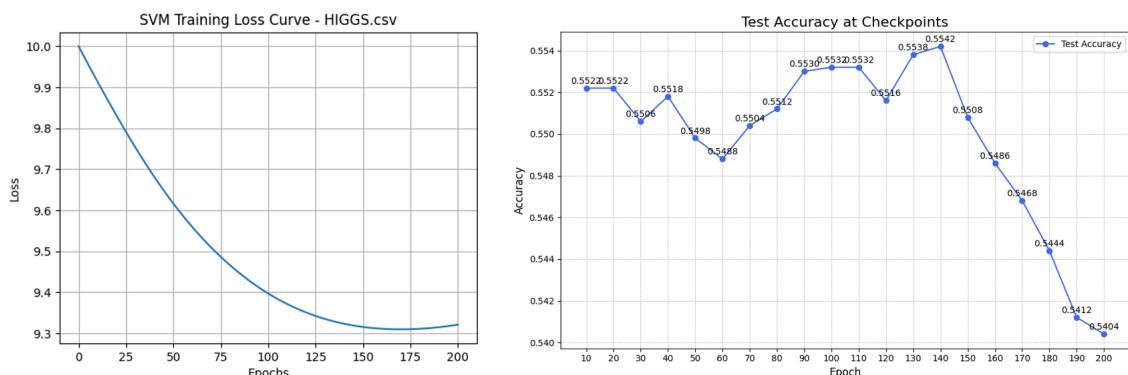
```
def MySVM(X_train, y_train, epochs=100, lr=1e-5, penalty=1, batch_size=64, X_test=None, y_test=None, results_dir=None):
    num_samples, num_features = X_train.shape
    w = np.zeros(num_features)
    b = 0
    loss = np.mean(penalty * np.maximum(0, 1 - y_train * (np.dot(X_train, w) + b))) + 0.5 * np.dot(w, w)
    loss_history = [loss]
    iter_range = tqdm(range(epochs), desc="Training", ncols=80)
    for i in iter_range:
```

```

# 在每个 epoch 开始时随机打乱数据
indices = np.arange(num_samples)
np.random.shuffle(indices)
X_train_shuffled = X_train[indices]
y_train_shuffled = y_train[indices]
for start_idx in range(0, num_samples, batch_size):
    end_idx = min(start_idx + batch_size, num_samples)
    X_batch = X_train_shuffled[start_idx:end_idx]
    y_batch = y_train_shuffled[start_idx:end_idx]
    margins = y_batch * (np.dot(X_batch, w) + b)
    misclassified_mask = margins < 1
    grad_w_reg = w
    grad_w_hinge = -penalty * np.dot((X_batch[misclassified_mask]).T, y_batch[misclassified_mask]) / len(y_batch)
    grad_b_hinge = penalty * np.sum(y_batch[misclassified_mask]) / len(y_batch)
    grad_w = grad_w_reg + grad_w_hinge
    grad_b = grad_b_hinge
    w -= lr * grad_w
    b -= lr * grad_b
return w, b

```

在完整的MySVM()函数中，我们还实现了每10个epoch就保存checkpoint的功能，以便于监控模型的训练过程。我们设置训练参数为：学习率lr=1e-6, epoch=200, 惩罚系数C=10。训练结果如下所示：



从图中可以看出，线性核SVM在HIGGS的测试集上最高准确率出现在第140个epoch，但仅达到55.42%，略高于随机猜测（50%）。这说明线性核SVM并不能解决高度线性不可分的HIGGS分类问题。

(6) RBF核SVM (HIGGS数据集) —— 基于script.minimize()优化函数

核技巧是解决SVM模型无法解决线性不可分问题的重要方法。当数据线性不可分时，需通过非线性变换 $\phi(x)$ 将数据映射到高维特征空间，使其在高维线性可分。此时，原问题中的内积 $\langle x^{(i)}, x^{(j)} \rangle$ 会变为高维空间的内积 $\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ 。由于 $\phi(x)$ 维度极高甚至无穷，导致无法计算。对偶形式则仅涉及样本之间的内积，因此可以直接引入核函数 $K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ ，无需显式计算 $\phi(x)$ ，只需通过核函数间接计算高维内积，从而实现了非线性变换。考虑考虑拉格朗日乘子 λ ，则SVM优化问题（4）可以被对偶转化成为以下问题：

$$\begin{aligned}
 & \max_{\lambda} \quad \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y^{(i)} y^{(j)} K \langle x^{(i)}, x^{(j)} \rangle, \\
 & \text{s.t.} \quad \sum_{i=1}^n \lambda_i y^{(i)} = 0, \quad \lambda_i \geq 0.
 \end{aligned} \tag{6}$$

(6) 的标准解法为SMO算法，但手动实现十分困难。因此，我们考虑使用scipy.optimize这一成熟的优化器库进行求解。这里，我们使用RBF核作为核函数：

$$K(x_1, x_2) = -\exp(\gamma \|x_1 - x_2\|^2) \tag{7}$$

其中 γ 是RBF核的超参数。值得指出的是，scipy.optimize作为一种通用的黑盒优化器，其计算复杂度过高。因此，这里我们仅仅选取少量HIGGS数据集中的样本 (<600)，观察实现的RBF核SVM的表现。经过调优，我们发现设置 $\gamma = 0.1$ 效果较好，训练结果如下：

RBF 核SVM 准确率	n=200	n=300	n=400	n=500
Train set	0.9700	0.9433	0.9500	0.9220
Validation set	0.5454	0.5578	0.5602	0.5576
Test set	0.5665	0.5673	0.5661	0.5691

从表中可以看出，在仅使用200个样本进行训练时，基于RBF核的SVM方法就已经超过了在49,0000个样本上进行训练的线性SVM。同时，在训练集上的准确率均超过了90%，这表明基于RBF核的SVM有充分的能力将HIGGS的数据集映射到高维线性可分空间。尽管由于计算资源的限制，我们无法在更多训练样本下测试RBF核SVM的表现。但从已有的数据来看，随着训练数据的增加，RBF核SVM的准确率逐步提升，证明了其在处理复杂数据上的有效性。

(7) RBF核SVM (HIGGS数据集) —— 基于Sklearn.svm自动方法

作为与我们自身实现的SVM方法的对比，这里我们使用sklearn库中的svm工具，在HIGGS数据集上训练RBF核SVM。受制于计算资源，这里我们仍然仅选取HIGGS数据集中的一部分样本来训练RBF核SVM。设置 $\gamma = 0.1$ ，训练结果如下：

RBF 核SVM 准确率	n=1000	n=5000	n=10000	n=20000	n=30000
Train set	0.9150	0.8688	0.8397	0.8173	0.8042
Validation set	0.5666	0.6072	0.6242	0.6358	0.6430
Test set	0.5801	0.6135	0.6247	0.6401	0.6477

从表中可以看出，随着训练样本的增多，RBF核SVM方法在测试集上的准确率逐渐增加，并且超过了我们自身实现的全部SVM所能达到的最好方法。这证明了基于核方法的SVM算法的强大分类能力。

四、总结

本项目从零实现了KNN和SVM两种算法。1) 我们实现了基于L1和L2距离的KNN算法，在自建数据集上测试了算法性能，分析了超参数对于算法表现的影响，绘制了决策边界，在HIGGS数据集上测试了两种算法的准确率。2) 我们实现了基于梯度下降的线性SVM、基于随机梯度下降的线性SVM和基于script.minimize()方法的RBF核SVM。在自制数据集上分析了算法的性能核超参数的影响，绘制了决策边界，在HIGGS数据集上测试了算法的准确率。此外，我们还利用sklearn机器学习库自动实现了一个RBF核SVM，测试了大样本下基于核方法的SVM的分类能力。