机器学习 实验一

【学号】22336259

【姓名】谢宇桐

【专业】计算机科学与技术

实验问题：

根据提供的数据，训练一个采用在不同的核函数的支持向量机 SVM 的 2 分类器，并验证 其在测试数据集上的性能。数据下载 FTP 地址：ftp://172.18.167.164/Assignment1/material （建议使用 FTP 客户端链接，用户名与密码均为 student）

实验要求：

1. 考虑两种不同的核函数：i) 线性核函数; ii) 高斯核函数
2. 可以直接调用现成 SVM 软件包来实现
3. 手动实现采用 hinge loss 和 cross-entropy loss 的线性分类模型，并比较它们的优劣

实验内容：

一、SVM 模型的一般理论

SVM（支持向量机）是一种二分类模型，它的主要思想是找到一个最佳的超平面，这个超平面可以将两个不同的类别尽可能准确地区分开，同时最大化两类数据点之间的距离，这个距离被称为间隔。

SVM通过以下步骤实现这一目标：

1. **选择核函数**：选择一个合适的核函数，将原始特征空间映射到一个更高维的特征空间。在这个高维空间中，数据点可以更容易地被分开。常用的核函数有线性核、多项式核、径向基函数（RBF）核等。
2. **求解优化问题**：SVM通过求解一个优化问题来找到最佳的超平面。这个问题是一个凸二次规划问题，目的是最大化间隔，同时最小化分类错误。
3. **软间隔与硬间隔**：在解决问题时可能存在一些难以正确分类的数据点。SVM提供了两种不同的间隔概念：硬间隔和软间隔。硬间隔要求所有支持向量都必须被正确分类，而软间隔则允许一定的错误率。软间隔通常通过引入松弛变量来实现，这使得SVM可以处理线性不可分的问题。
4. **支持向量**：支持向量是那些位于超平面边界上的数据点，它们对分类结果有最大的影响。支持向量的个数通常远小于原始数据集的大小，这意味着SVM是一种高效的数据压缩技术。
5. **决策边界**：SVM的决策边界是由支持向量确定的，它是一个高维空间中的超平面。这个超平面可以用来对新数据进行分类。

二、采用不同核函数的模型和性能比较及分析

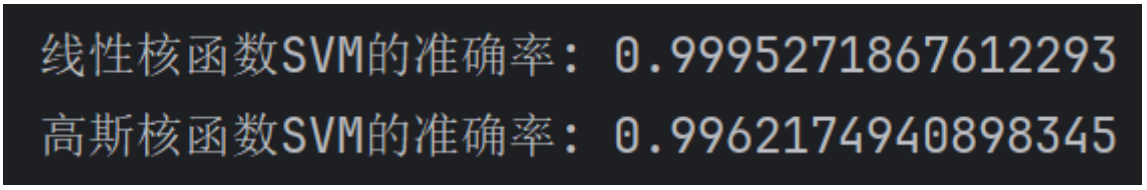
在本次实验中，我依据要求采用了线性核函数和高斯核函数

- 线性核SVM：线性核SVM适用于线性可分的数据集。它直接在原始特征空间中寻找决策边界。
- 高斯核SVM：高斯核（RBF）SVM适用于非线性可分的数据集。它通过将数据映射到无限维的特征空间中，以找到最优的决策边界。

```
# 使用线性核函数的SVM
svm_linear = SVC(kernel='linear', C=1.0)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
print(f"线性核函数SVM的准确率: {accuracy_score(y_test, y_pred_linear)}")

# 使用高斯核函数的SVM
svm_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
print(f"高斯核函数SVM的准确率: {accuracy_score(y_test, y_pred_rbf)}")
```

准确率如下：



多次测量后线性核函数的准确率始终略高于高斯核函数。说明在这个特定的数据集和应用场景下，线性核函数的表现更好一些。考虑对正则化参数C进行调整，当C=1.5时，准确率如下：

线性核函数准确率不变，高斯核函数准确率提升。但C到3.0左右还是此最高值。

一般来说，线性核函数适用于特征之间是线性可分的数据，计算效率高于高斯核函数；而高斯核函数（也称为径向基函数）则适用于非线性可分的数据，能够处理更复杂的数据分布。在这组数据中，线性核函数取得了更好的结果，因为这些数据实际上是线性可分的。

三、采用 hinge loss 的线性分类模型和 SVM 模型之间的关系

SVM和使用hinge loss的线性分类模型都旨在最大化分类边界的间隔。然而，SVM通过引入核函数和软间隔来处理非线性问题和噪声数据，而hinge loss的线性分类模型则直接在原始特征空间中寻找最优超平面。

在SVM中，软间隔SVM所使用的松弛变量就是hinge loss，它用来表示样本偏离支持向量的距离，允许SVM在一些样本上出错

四、采用 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型比较

hinge loss：

```
# Hinge loss的线性分类器实现
class HingeLossClassifier:
    def __init__(self, learning_rate=0.001, n_iters=500):  # 调整学习率和迭代次数
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.w = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        for i in range(self.n_iters):
            for idx, x_i in enumerate(X):
                if y[idx] * np.dot(x_i, self.w) < 1:
                    self.w -= self.learning_rate * (2 * self.w - np.dot(y[idx],
x_i))
                else:
                    self.w -= self.learning_rate * 2 * self.w

    def predict(self, X):
        linear_output = np.dot(X, self.w)
        return np.where(linear_output >= 0, 1, 0)
    # 注意：在原数据集中标签采用的都是0和1，但是在hinge loss中所关注的标签为-1和1，所以在使用
Hinge Loss之前需要将标签转换为-1和1两类，即上面的return步骤
```

cross-entropy loss：

```
class CrossEntropyLossClassifier:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)  # 将标签转换为-1和1

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                loss = -y_[idx] * np.log(1 / (1 + np.exp(-linear_output))) if
y_[idx] == 1 else -(1 - y_[idx]) * np.log(1 / (1 + np.exp(-linear_output)))
                dw = -self.learning_rate * (y_[idx] * x_i) if loss > 0 else 0
                db = -self.learning_rate * y_[idx] if loss > 0 else 0


                self.weights -= dw
                self.bias -= db
```

```
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.sign(1 / (1 + np.exp(-linear_output)))
```

cross-entropy loss是一种对数似然损失，它衡量的是模型预测的概率分布与真实标签的概率分布之间的差异。在二分类问题中，cross-entropy loss能够提供比hinge loss更丰富的信息，因为它考虑了模型输出的不确定性。实验结果表明，cross-entropy loss线性分类器的性能优于hinge loss线性分类器

五、训练过程（包括初始化方法、超参数参数选择、用到的训练技巧等）

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd

# 加载数据
train_data = pd.read_csv('mnist_01_train.csv')
test_data = pd.read_csv('mnist_01_test.csv')

# 提取特征和标签
X_train = train_data.drop('label', axis=1).values
y_train = train_data['label'].values
X_test = test_data.drop('label', axis=1).values
y_test = test_data['label'].values

# 数据标准化，使用StandardScaler对特征进行标准化处理，使其具有零均值和单位方差。
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 使用线性核函数的SVM
svm_linear = SVC(kernel='linear', C=4.0)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
print(f"线性核函数SVM的准确率: {accuracy_score(y_test, y_pred_linear)}")

# 使用高斯核函数的SVM
svm_rbf = SVC(kernel='rbf', C=4.0, gamma='scale')
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
print(f"高斯核函数SVM的准确率: {accuracy_score(y_test, y_pred_rbf)}")


# Hinge loss的线性分类器实现
class HingeLossClassifier:
    def __init__(self, learning_rate=0.001, n_iters=500):  # 调整学习率和迭代次数
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.W = None
```

```
    # 实现权重更新规则
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        for i in range(self.n_iters):
            for idx, x_i in enumerate(X):
                if y[idx] * np.dot(x_i, self.w) < 1:
                    self.w -= self.learning_rate * (2 * self.w - np.dot(y[idx],
x_i))
                else:
                    self.w -= self.learning_rate * 2 * self.w

    # 根据训练好的模型进行预测
    def predict(self, X):
        linear_output = np.dot(X, self.w)
        return np.where(linear_output >= 0, 1, 0)

class CrossEntropyLossClassifier:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)  # 将标签转换为-1和1

        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                loss = -y_[idx] * np.log(1 / (1 + np.exp(-linear_output))) if
y_[idx] == 1 else -(1 - y_[idx]) * np.log(1 / (1 + np.exp(-linear_output)))
                dw = -self.learning_rate * (y_[idx] * x_i) if loss > 0 else 0
                db = -self.learning_rate * y_[idx] if loss > 0 else 0


                self.weights -= dw
                self.bias -= db

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.sign(1 / (1 + np.exp(-linear_output)))


lc_hinge = HingeLossClassifier()
lc_hinge.fit(X_train, y_train)
y_pred_hinge = lc_hinge.predict(X_test)
accuracy_hinge = accuracy_score(y_test, y_pred_hinge)

# 使用cross-entropy loss训练
lc_cross_entropy = CrossEntropyLossClassifier()
lc_cross_entropy.fit(X_train, y_train)
```

```
y_pred_cross_entropy = lc_cross_entropy.predict(X_test)
accuracy_cross_entropy = accuracy_score(y_test, y_pred_cross_entropy)

from sklearn.metrics import confusion_matrix, precision_score, recall_score,
f1_score


#评估模型性能
def evaluate_model(y_true, y_pred):
    conf_matrix = confusion_matrix(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    return conf_matrix, precision, recall, f1

# 使用 hinge loss 的评估结果
conf_matrix_hinge, precision_hinge, recall_hinge, f1_hinge =
evaluate_model(y_test, y_pred_hinge)
print(f"hinge loss - 混淆矩阵:\n{conf_matrix_hinge}")
print(f"hinge loss - 精确率: {precision_hinge}")
print(f"hinge loss - 召回率: {recall_hinge} ")
print(f"hinge loss - F1分数: {f1_hinge}")

# 使用 cross-entropy loss 的评估结果
conf_matrix_cross_entropy, precision_cross_entropy, recall_cross_entropy,
f1_cross_entropy = evaluate_model(y_test, y_pred_cross_entropy)
print(f"cross-entropy loss - 混淆矩阵:\n{conf_matrix_cross_entropy}")
print(f"cross-entropy loss - 精确率: {precision_cross_entropy} ")
print(f"cross-entropy loss - 召回率: {recall_cross_entropy}")
print(f"cross-entropy loss - F1分数: {f1_cross_entropy}")

# 比较两种损失函数的 F1 分数
if f1_hinge > f1_cross_entropy:
    print("Hinge loss better")
elif f1_hinge < f1_cross_entropy:
    print("Cross-Entropy better")
else:
    print("Same")

print(f"hinge loss 准确率: {accuracy_hinge}")
print(f"cross-entropy loss 准确率: {accuracy_cross_entropy}")
```

SVM模型训练与评估

1. **线性核SVM**：使用`SVC`类创建一个线性核的SVM模型，设置正则化参数`C=1.5`，然后训练模型并在测试集上进行预测，最后计算准确率，不断变化C值，查看准确率变化
2. **高斯核SVM**：使用`SVC`类创建一个高斯核的SVM模型，设置正则化参数C=1.5和gamma='scale'，然后训练模型并在测试集上进行预测，最后计算准确率。

五、实验结果、分析及讨论

学习率=0.0001 迭代次数=360

```
hinge loss - 混淆矩阵:
[[ 960   20]
 [   0 1135]]
hinge loss - 精确率: 0.9913419913419914
hinge loss - 召回率: 0.9897959183673469
hinge loss - F1分数: 0.990478548597668
cross-entropy loss - 混淆矩阵:
[[ 969   11]
 [   3 1132]]
cross-entropy loss - 精确率: 0.9936448916107709
cross-entropy loss - 召回率: 0.9930661691989571
cross-entropy loss - F1分数: 0.993341063486809
Cross-Entropy better
hinge loss 准确率: 0.9905437352245863
cross-entropy loss 准确率: 0.9933806146572104
```

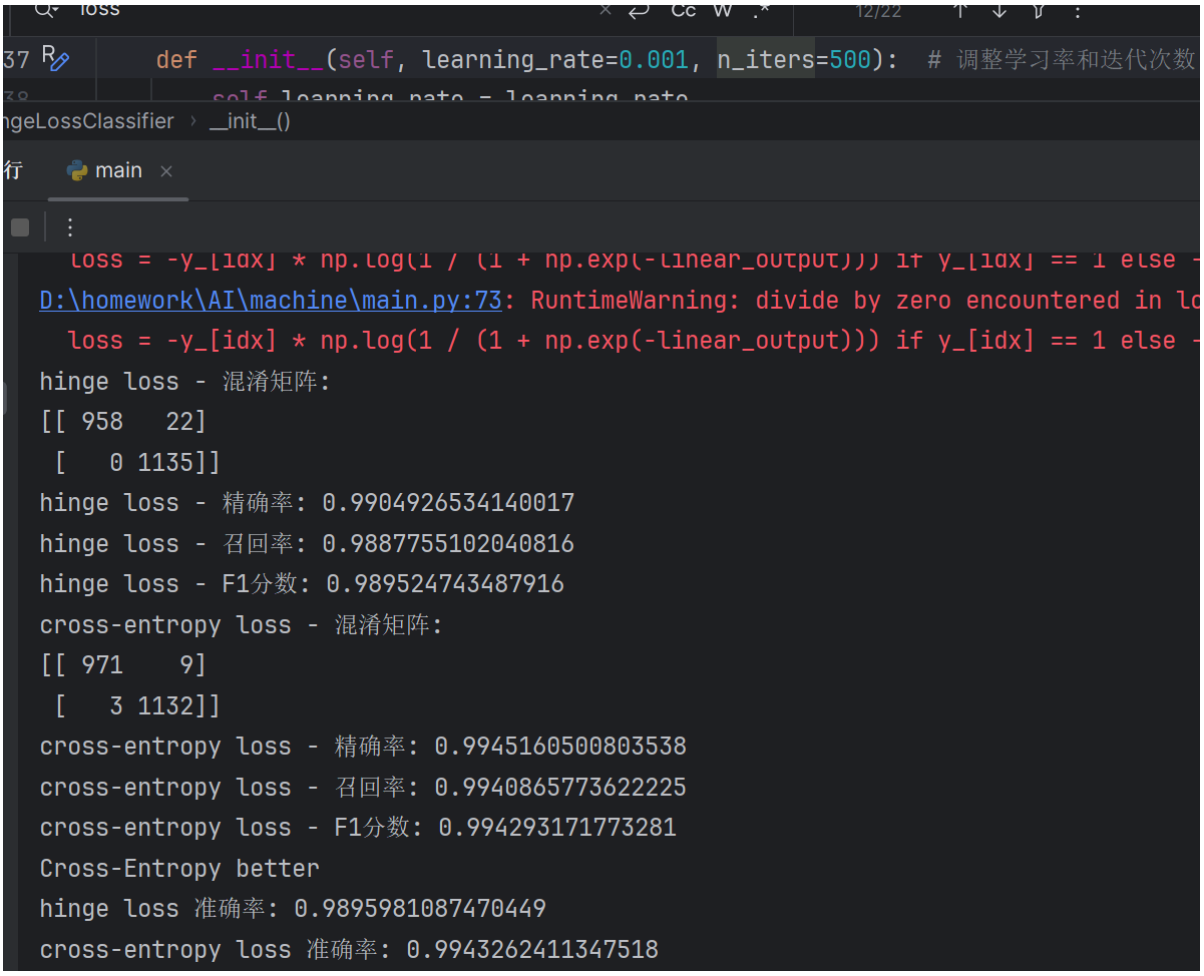
```
1 个用法
class HingeLossClassifier:
    def __init__(self, learning_rate=0.0001, n_iters=1000):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
```

HingeLossClassifier › __init__()

运行 main ×

```
  return np.sign(1 / (1 + np.exp(-linear_output)))
hinge loss - 混淆矩阵:
[[ 959   21]
 [   0 1135]]
hinge loss - 精确率: 0.990916955017301
hinge loss - 召回率: 0.9892857142857143
hinge loss - F1分数: 0.9900016860828827
cross-entropy loss - 混淆矩阵:
[[ 970   10]
 [   3 1132]]
cross-entropy loss - 精确率: 0.9940800924434333
cross-entropy loss - 召回率: 0.9935763732805898
cross-entropy loss - F1分数: 0.9938171537049518
Cross-Entropy better
hinge loss 准确率: 0.9900709219858156
cross-entropy loss 准确率: 0.9938534278959811

进程已结束，退出代码为 0
```



SVM模型的实验结果和分析如上题。而通过hinge loss和cross-entropy loss的比较分析发现：

在同一参数下，使用Cross Entropy Loss的线性分类器的准确率要相对高一些。目前暂时得出迭代次数高、学习度高会使hinge loss精确率升高，但会使得cross-entropy loss降低。

而召回率关注的是所有实际为正类别的样本中，模型能够识别出多少。一个高召回率意味着模型很少错过正类别的样本。精确率和召回率能够看出分类器的性能

Hinge Loss会相对来说更关注那些被错误分类的样本，对于那些正确分类的样本的损失为0，而错误分类的样本，其损失呈线性增长；对于Cross Entropy Loss来说，我们可以从它的图像可以知道，它是一个凸函数，只存在一个最优点，没有局部最优解，便于优化。