实验课程：机器学习与数据挖掘

实验名称：向量机SVM

专业名称：计算机科学与技术

学生姓名：吴臻

学生学号：21307371

实验目的

> 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

实验要求

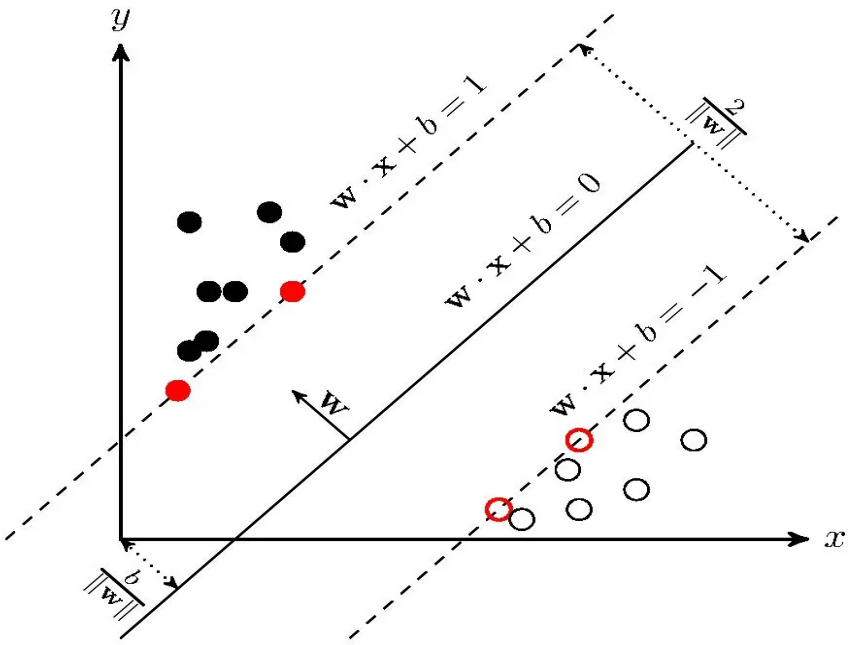
1. 考虑两种不同的核函数：i) 线性核函数; ii) 高斯核函数
2. 可以直接调用现成 SVM 软件包来实现
3. 手动实现采用 hinge loss 和 cross-entropy loss 的线性分类模型，并比较它们的优劣

实验内容

1. **SVM 模型的一般理论**

- **SVM简介**
  支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题。SVM的的学习算法就是求解凸二次规划的最优化算法。

- **SVM算法原理**
  SVM学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。如下图所示，$\boldsymbol{w} \cdot x + b = 0$ 即为分离超平面，对于线性可分的数据集来说，这样的超平面有无穷多个（即感知机），但是几何间隔最大的分离超平面却是唯一的。



- **SVM类别**
  SVM可以分为**线性SVM和非线性SVM**，非线性SVM引入了**核函数**，核函数用于将数据从输入空间映射到一个更高维度的特征空间，这个映射的目的是使数据在新的特征空间中更容易分割，从而将非线性SVM转变为线性SVM问题。
  在训练样本的特征空间中，根据其是否线性可分，引申出了两种侧重点不同的SVM模型，一种是**硬间隔SVM**（Hard Margin），另一种是**软间隔SVM**（Soft Margin），前者假定了样本在其特征空间中是线性可分的（也就是能找到一个超平面，可以将不同类别的样本划分开），后者则假定了样本在其特征空间中线性不可分，其引入了**松弛变量**（Slack Variable）$\xi i \geq 0$（表示样本偏离支持向量的距离）,允许SVM在一些样本上出错，但会得到相应的惩罚ξi,这里提到的Margin是指与分类的超平面距离最近的样本点（支持向量）之间的距离

约束最优化问题

$$\min_{w,b} \ \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$s.t. \ \ y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)\geq 1,\ i=1,2,\ldots,N$$

使用拉格朗日乘子法得到其对偶问题

$$L\left(\boldsymbol{w},b,\boldsymbol{\alpha}\right)=\frac{1}{2}\|\boldsymbol{w}\|^2-\sum_{i=1}^{N}\alpha_i\left(y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)-1\right)$$

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)-\sum_{i=1}^{N}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$\alpha_i \geq 0,\ i=1,2,\ldots,N$$

运用KKT条件

$$\boldsymbol{w}^* = \sum_{i=1}^{N}\alpha_i^* y_i \boldsymbol{x}_i$$

$$b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i\left(\boldsymbol{x_i}\cdot\boldsymbol{x_j}\right)$$

预测

$$f\left(\boldsymbol{x}\right) = sign\left(\boldsymbol{w}^*\cdot\boldsymbol{x}+b^*\right)$$

2. **采用不同核函数的模型和性能比较及分析**

- 线性核函数

```
# 训练SVM分类器（线性核函数）
linear_svm = svm.SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
y_pred_linear = linear_svm.predict(X_test)

# 计算线性SVM分类器的准确度
y_train_pred_linear = linear_svm.predict(X_train)
linear_svm_train_accuracy = accuracy_score(y_train, y_train_pred_linear)
print("Linear SVM 训练集准确度: {:.5f}%".format(linear_svm_train_accuracy * 100))
linear_accuracy = accuracy_score(y_test, y_pred_linear)
print("Linear SVM 测试集准确度: {:.5f}%".format(linear_accuracy * 100))
```

结果：（c=1）

```
Linear SVM 训练集准确度: 100.00000%
Linear SVM 测试集准确度: 99.90544%
```

c=0.0000001

```
Linear SVM 训练集准确度: 99.92104%
Linear SVM 测试集准确度: 99.95272%
```

> 分析：线性核函数（c=1）在训练集上的准确度达到了100%，但是测试集只有99.90%，考虑到有可能出现了过拟合的现象，减小惩罚系数C（参数C确定了模型对于分类错误的容忍度。具体来说，较小的C值会导致更多的容忍度，模型更倾向于容忍一些分类错误，从而可能产生一个更简单的决策边界。较大的C值会降低容忍度，模型更倾向于严格要求正确分类，可能产生更复杂的决策边界），当c=0.0000001时，测试集的准确度有所提高

- 高斯核函数

```
# 训练SVM分类器（高斯核函数）
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
y_pred_rbf = rbf_svm.predict(X_test)

# 计算高斯SVM分类器的准确度
y_train_pred_rbf = rbf_svm.predict(X_train)
gaussian_svm_train_accuracy = accuracy_score(y_train_pred_rbf, y_train)
print("RBF SVM 训练集准确度: {:.5f}%".format(gaussian_svm_train_accuracy * 100))
rbf_accuracy = accuracy_score(y_test, y_pred_rbf)
print("RBF SVM 测试集准确度: {:.5f}%".format(rbf_accuracy * 100))
```

结果:

```
RBF SVM 训练集准确度: 99.99210%
RBF SVM 测试集准确度: 99.95272%
```

> 分析：高斯核函数的测试效果较好，在不用调参的情况下，就达到和调参后线性核函数一样的结果（也可能是刚好默认参数适用于这组数据）。高斯核函数将数据映射到高维度特征空间后，执行非线性分类，在处理非线性数据和复杂模式时表现出色。
> 线性核函数和高斯核函数的表现大致相近，可以推测数据的特征空间可以用一个超平面大致分类

3. **采用 hinge loss 的线性分类模型和 SVM 模型之间的关系**

hinge loss

```
learning_rate = 0.0001
C = 1
num_epochs = 100

def linear_svm(features, labels, learning_rate, C, num_epochs):
    # 初始化权重向量W和偏置项b
    W = np.zeros(features.shape[1])
    b = 0
    # 训练模型
    for epoch in range(num_epochs):
        for i, x in enumerate(features):
            # 计算预测值
            y_pred = np.dot(W, x) + b

            # 计算Hinge Loss
            loss = max(0, 1 - labels[i] * y_pred)

            # 计算梯度
            if loss > 0:
                dW = -C * labels[i] * x
                db = -C * labels[i]
            else:
                dW = 0
                db = 0

            # 更新权重
            W -= learning_rate * dW
            b -= learning_rate * db
    return W,b
```

结果:

```
Linear (Hinge Loss) 训练集准确度: 100.00000%
Linear (Hinge Loss) 测试集准确度: 99.85816%
```

> 分析：hinge loss: $L(z) = max(0, 1 - y_i\hat{y}_i)$, 松弛变量$\xi i$的定义和hinge损失函数一样,SVM的损失函数为

$$\min_{\omega,\gamma}\left[C\sum_{i=1}^{n}\max\left\{0,1-(\omega^T x_i+\gamma)y_i\right\}+\frac{1}{2}\|\omega\|_2^2\right]$$ 。

，可以看到，SVM的损失函数可以看作Hinge损失 + L2正则化。
Hinge loss关注每一个点的正确与否，而SVM更关注支持向量（同时会衡量一些点的惩罚）

4. **采用 hinge loss 线性分类模型和 cross-entropy loss 线性分类模型比较**

cross-entropy

```
learning_rate = 0.01
num_epochs = 800

def cross_entropy_loss(y, y_pred):
    epsilon = 1e-15  # 用于避免对数函数中的除零错误
    loss = - (y * np.log(y_pred + epsilon) + (1 - y) * np.log(1 - y_pred + epsilon))
    return np.mean(loss)

def train(X, y, learning_rate, num_epochs):
    num_features = X.shape[1]
    w = np.zeros(num_features)
    b = 0
    for epoch in range(num_epochs):
        y_pred = predict(X, w, b)
        loss = cross_entropy_loss(y, y_pred)

        # 计算梯度
        dw = np.dot(X.T, (y_pred - y)) / len(y)
        db = np.mean(y_pred - y)

        # 更新参数
        w -= learning_rate * dw
        b -= learning_rate * db

        if epoch % 100 == 0:
            print(f'Epoch {epoch}, Loss: {loss}')
    return w, b
```

结果:

```
Epoch 100, Loss: 0.054564258194398 45
Epoch 200, Loss: 0.033119258221896 64
Epoch 300, Loss: 0.025523601394471872
Epoch 400, Loss: 0.02437160512378341
Epoch 500, Loss: 0.02032858052500716
Epoch 600, Loss: 0.01654854032513943
Epoch 700, Loss: 0.009000527922997343
Linear (cross-entropy) 训练集准确度: 99.99210%
Linear (cross-entropy) 测试集准确度: 99.95272%
```

> 分析：交叉熵损失函数定义为$L(W) = -\frac{1}{N}\sum_{l=1}^{N}[y^{(l)}ln(\sigma(x^{(l)}W)) + (1-y^{(l)})ln(1-\sigma(x^{(l)}W))]$
> 采用交叉熵损失函数的效果比折页损失的好，交叉熵是用来评估当前训练得到的概率分布与真实分布的差异情况。它刻画的是实际输出（概率）与期望输出（概率）的距离，也就是交叉熵的值越小，两个概率分布就越接近。hinge loss关注类别难以区分的部分样本；cross entropy关注全局所有样本，尽可能提高各样本所对应正确类别的概率。

5. **训练过程**

- 数据预处理
  （1）输入标准化（**节省时间，提高准确率**）

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

> fit_transform 会计算并应用标准化参数，以便将数据标准化为均值为0、方差为1的分布,它适应并转换数据，同时计算适应过程中所需的参数。
> transform 用于测试数据或其他未见过的数据，它只应用之前在训练数据上计算的参数，以保持数据的一致性。

未标准化

```
Linear (Hinge Loss) 训练集准确度: 100.00000%
Linear (Hinge Loss) 测试集准确度: 99.85816%
It takes 7.263657808303833 s
```

```
Linear (cross-entropy) 训练集准确度: 99.99210%
Linear (cross-entropy) 测试集准确度: 99.95272%
It takes 48.343196153640 75 s
```

标准化

```
Linear (Hinge Loss) 训练集准确度: 99.99210%
Linear (Hinge Loss) 测试集准确度: 99.95272%
It takes 7.256137371063232 s
```

```
Linear (cross-entropy) 训练集准确度: 99.86577%
Linear (cross-entropy) 测试集准确度: 99.95272%
It takes 4.733819961547852 s
```

> 分析：可以看到进行数据标准化之后，线性（hinge loss）的准确率有所提高，线性（交叉熵）的运行时间有明显地减少

（2）采用 hinge loss 线性分类模型时，要将标签0转为-1

```
y_train[y_train == 0] = -1
y_test[y_test == 0] = -1
```

- 超参数参数选择

SVM线性核	c = 0.001	c = 0.01	c = 0.1	c = 1	c = 10
训练集正确率	100.00000%	100.00000%	100.00000%	100.00000%	100.00000%
测试集正确率	99.90544%	99.90544%	99.90544%	99.90544%	99.90544%

SVM高斯核	c = 0.001	c = 0.01	c = 0.1	c = 1	c = 10
训练集正确率	99.58942%	99.72365%	99.88156%	99.99210%	100.00000%
测试集正确率	99.57447%	99.71631%	99.90544%	99.95272%	99.95272%

> 分析：随着惩罚系数C的增大，训练集正确率不断提高，但是测试集正确率会出现波动，当C过大时，会出现过拟合导致正确率下降
> 还有一个参数（学习率），因为训练过程很快便收敛了，所以学习率可以设置小一些（0.01，0.001）

实验总结

> 在本次实验中，我调用现成 SVM 软件包（sklearn），分别实现了线性核函数和高斯核函数的SVM，还手动实现了采用 hinge loss 和 cross-entropy loss 的线性分类模型。
> （1）训练SVM时，线性核函数的运行速度快于高斯核函数，高斯核函数的计算复杂度大，计算量较大。线性核函数适用于线性可分的数据，高斯核函数适用于非线性数据，其中类别之间没有清晰的线性边界，线性核函数和高斯核函数在本次实验的表现大致相近，可以推测数据的特征空间可以用一个超平面大致分类。
> （2）训练hinge loss 和 cross-entropy loss 的线性分类模型时，两者的表现差不多，但是cross-entropy loss的运行时间明显久于hinge loss，在进行数据标准化之后，运行速度得到极大改善。
> （3）虽然本次实验是使用sklearn实现SVM，编程的难度不大，但是在查阅SVM 模型的一般理论时花费了大量的时间，SVM公式的推导对我来说难度是很大的，特别是引入核函数之后，不过这次实验让我对SVM的理解又更进了一步，之前只是大致知道SVM的工作原理和粗略的实现方式，这次实验让我对所用公式的理解更加深入，同时对SVM以及其他线性分类器的区别，以及不同损失函数的选择，超参数的调整都有了更深刻的理解。

实验资料

1. SVM：https://blog.csdn.net/qq_24178985/article/details/115636659?spm=1001.2014.3001.5501
2. https://blog.csdn.net/m0_52387305/article/details/127342415
3. hinge loss：https://zh.wikipedia.org/wiki/Hinge_loss