```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
```

```python
# 1. 데이터 준비 (Iris 데이터셋)
iris = datasets.load_iris()
X = iris.data[:100, :2]  # 꽃받침 길이와 너비만 사용
y = iris.target[:100]    # 0 (Setosa) 또는 1 (Versicolor)

# 클래스 레이블을 SVM에 맞게 -1과 1로 변경
y_svm = np.where(y == 0, -1, 1)

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y_svm, test_size=0.2, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# PyTorch 텐서로 변환
X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.FloatTensor(y_train).view(-1, 1) # view를 통해 열 벡터로 변환
X_test_tensor = torch.FloatTensor(X_test)
y_test_tensor = torch.FloatTensor(y_test).view(-1, 1)
```

```python
# 2. SVM 모델 정의
# SVM은 본질적으로 선형 모델이므로, PyTorch의 nn.Linear를 사용
class SVM(nn.Module):
    def __init__(self, n_features):
        super(SVM, self).__init__()
        # 입력 특징의 수에 맞는 선형 레이어 정의
        self.linear = nn.Linear(n_features, 1)

    def forward(self, x):
        return self.linear(x)
```

```python
# 3. 모델 학습
n_samples, n_features = X_train.shape
model = SVM(n_features)

# 하이퍼파라미터 설정
learning_rate = 0.01
lambda_param = 0.01 # 규제 파라미터
n_epochs = 200

# 옵티마이저 설정
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

print("모델 학습을 시작합니다...")
for epoch in range(n_epochs):
    # 모델의 예측값 계산
    outputs = model(X_train_tensor)

    # 힌지 손실(Hinge Loss) 계산
    # max(0, 1 - y * f(x))
    hinge_loss = torch.mean(torch.clamp(1 - y_train_tensor * outputs, min=0))

    # 규제 항 (L2 Regularization)
    l2_reg = torch.tensor(0.)
    for param in model.parameters():
        l2_reg += torch.norm(param, 2)

    # 최종 손실 = 힌지 손실 + 규제 항
    loss = hinge_loss + lambda_param * l2_reg

    # 그래디언트 초기화 및 역전파, 가중치 업데이트
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch+1}/{n_epochs}], Loss: {loss.item():.4f}')
```

```python
# 4. 모델 평가
with torch.no_grad(): # 평가 시에는 그래디언트 계산이 필요 없음
    test_outputs = model(X_test_tensor)

    # 예측: 출력이 > 0 이면 1, 아니면 -1
    predicted = torch.sign(test_outputs).squeeze()

    # 정확도 계산
    correct = (predicted == y_test_tensor.squeeze()).sum().item()
    total = y_test_tensor.size(0)
    accuracy = correct / total

    print("\n모델 평가 결과")
    print(f'정확도 (Accuracy): {100 * accuracy:.2f}%')
```