PhamNgocHai_21002139_Week11_HardMargin_SVM

May 10, 2024

1 Ví dụ 1

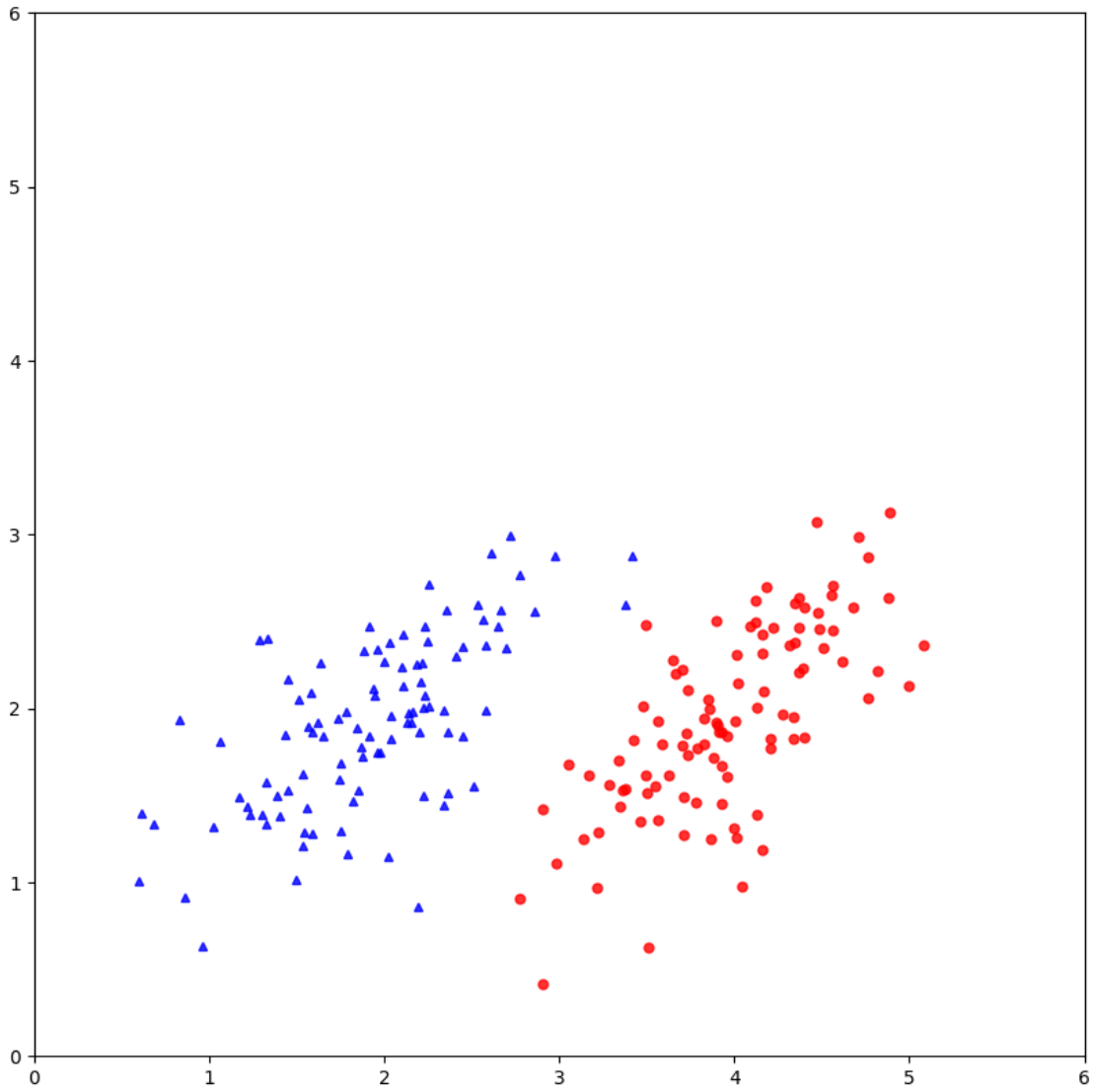
(Dữ liệu tự tạo)

1.1 Tạo data

Tạo data có 200 mẫu (N = 200) phân đều 2 class. d = 2.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(10)
means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 100
X0 = np.random.multivariate_normal(means[0], cov, N) # class 1
X1 = np.random.multivariate_normal(means[1], cov, N) # class -1
X = np.concatenate((X0.T, X1.T), axis = 1) # all data
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1) # labels
```

[ ]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10, 10))
ani = plt.cla()
#plot points
ani = plt.plot(X0.T[0, :], X0.T[1, :], 'b^', markersize = 5, alpha = .8)
ani = plt.plot(X1.T[0, :], X1.T[1, :], 'ro', markersize = 5, alpha = .8)
ani = plt.axis([0 , 6, 0, 6])
plt.show()
```



1.2 Giải bài toán tối ưu $-g(\lambda) \rightarrow min$ (sử dụng cvxopt để giải)

[ ]:
```
# !pip install cvxopt
```

```
Collecting cvxopt
  Downloading
cvxopt-1.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(1.3 kB)
Downloading
cvxopt-1.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.9
MB)
                         17.9/17.9 MB
120.2 kB/s eta 0:00:0000:0100:04
```

```
Installing collected packages: cvxopt
Successfully installed cvxopt-1.3.2
```

[ ]:
```python
from cvxopt import matrix, solvers  # build P ~ K

V = np.concatenate((X0.T, -X1.T), axis=1)
P = matrix(V.T.dot(V))  # P ~ K in slide see definition of V, K near eq (8)
q = matrix(-np.ones((2 * N, 1)))  # all-one vector
# build A, b, G, h
G = matrix(
    -np.eye(2 * N)
)  # for all lambda_n >= 0! note that we solve -g(lambda) -> min
h = matrix(np.zeros((2 * N, 1)))
A = matrix(y)  # the equality constrain is actually y^T lambda = 0
b = matrix(np.zeros((1, 1)))
solvers.options["show_progress"] = False
sol = solvers.qp(P, q, G, h, A, b)
l = np.array(sol["x"])  # lambda
print("lambda = ")
print(l.T)
```

```
lambda =
[[6.43830372e-10 4.72988123e-10 6.96301261e-10 5.79332567e-10
  5.85394978e-10 5.88997480e-10 4.89184241e-10 5.67664894e-10
  4.93959294e-10 8.11159682e-10 7.35998382e+01 5.84752419e-10
  6.33981206e-10 5.70452608e-10 5.33265242e-10 6.21992621e-10
  6.30547706e-10 6.42898737e-10 6.23309373e-10 7.19237900e-10
  6.37162151e-10 5.62750933e-10 5.43352984e-10 5.34317125e-10
  6.12473191e-10 5.54489279e-10 5.15178968e-10 6.11949009e-10
  7.11337568e-10 6.01640553e-10 6.16174828e-10 7.08582794e-10
  6.28907978e-10 9.47883549e-10 5.60470453e-10 5.73546044e-10
  5.40206464e-10 5.84839273e-10 4.52413498e-10 5.71685468e-10
  6.69868839e-10 4.47707836e-10 3.99526097e-10 5.50379371e-10
  8.79639914e-10 6.28698683e-10 6.73190935e-10 6.03403938e-10
  5.36740042e-10 1.22567795e-09 8.50987129e-10 4.93131967e-10
  6.05327442e-10 5.32588255e-10 7.14356409e-10 6.36992251e-10
  7.63436068e-10 5.89166800e-10 5.36104164e-10 9.45023473e-10
  6.67949637e-10 6.12129203e-10 6.46493582e-10 6.24525894e-10
  6.65715637e-10 7.38947965e-10 5.15745377e-10 5.13029354e-10
  6.48787617e-10 6.79888598e-10 7.55647115e-10 6.22557618e-10
  5.91892815e-10 5.35768954e-10 5.80917611e-10 5.86736483e-10
  6.43635145e-10 5.53024324e-10 5.05359026e-10 1.19232725e+00
  6.44378233e-10 5.83131769e-10 7.06922935e-11 6.61352697e-10
  6.43299553e-10 4.39175630e-10 6.26901927e-10 5.54463008e-10
  5.88873708e-10 6.48288101e-10 5.04176204e-10 5.88872421e-10
  7.06296853e-10 5.98173540e-10 5.24485213e-10 6.06670807e-10
  5.16257207e-10 5.33768519e-10 7.85256306e-10 6.88721496e-10
  1.57912121e-09 9.54002721e-10 7.48687674e-10 7.69958051e-10
```

```
 7.55083119e-10 1.91710501e-09 7.50186134e-10 9.20641853e-10
 1.81081431e-09 1.37252117e-09 1.28982692e-09 6.53070458e-10
 9.39352557e-10 8.25117923e-10 8.09559605e-10 1.11725719e-09
 9.09058978e-10 9.99946008e-10 7.55549754e-10 1.54787850e-09
 1.11345584e-09 7.86469293e-10 8.69278933e-10 6.46910456e-10
 9.39659840e-11 8.03530436e-10 1.11459401e-09 1.89196326e-09
 1.08765182e-09 7.37342360e-10 5.91226612e-10 6.72409995e-10
 9.17859643e-10 1.11415592e-09 7.04015518e-10 8.38887931e-10
 7.30608121e-10 9.58771122e-10 6.43897608e-10 7.69011272e-10
 1.01345763e-09 8.42643707e-10 3.98479616e-10 7.77126768e-10
 6.19818574e-10 8.44492610e-10 8.64815459e-10 8.32615061e-10
 8.04370289e-10 1.21977423e-09 1.09156750e-09 7.64487428e-10
 9.89715502e-10 6.87934467e-10 6.65874020e-10 4.75856878e-10
 5.91373303e-10 9.81754613e-10 8.10817319e-10 8.99737005e-10
 8.32877232e-10 1.60994572e-09 9.76367233e-10 8.06051300e-10
 6.79106997e-10 8.70814735e-10 6.81603958e-10 7.55269573e-10
 1.56451629e-09 5.57575506e-10 8.64989141e-10 1.14278242e-09
 7.25643768e-10 7.09541779e-10 6.96928465e-10 1.00623135e-09
 7.30979530e-10 1.13867339e-09 1.11044726e-10 8.11168707e-10
 8.49604933e-10 7.01515050e-10 8.91835521e-10 7.84720139e-10
 7.47921654e+01 6.40453707e-10 6.44152830e-10 9.55430231e-10
 7.24969856e-10 1.03504336e-09 1.27563749e-09 8.28173946e-10
 7.81107943e-10 9.43370659e-10 8.04133941e-10 8.08931922e-10
 9.06812557e-10 1.13753870e-09 7.80003087e-10 8.33355799e-10]]
```

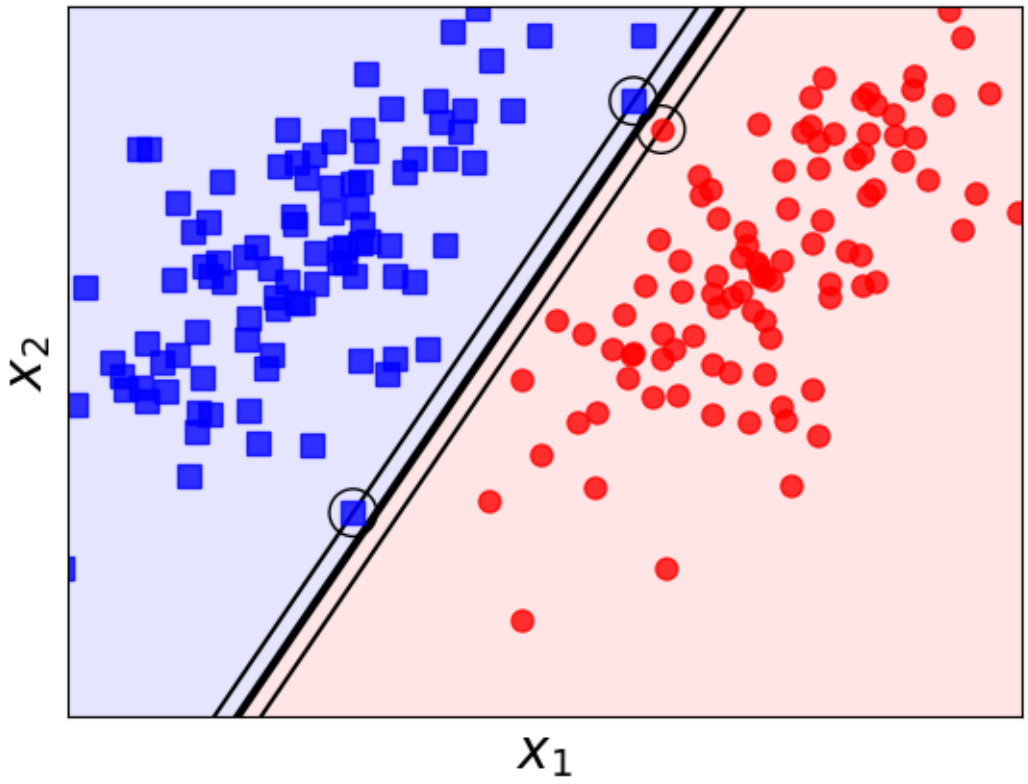
Do các giá trị λ giải bằng thư viện CVXOPT được tính theo phương pháp lặp, nên hầu hết chúng không có giá trị 0 tuyệt đối mà chỉ là rất nhỏ (cỡ 10^{-9}). Vì vậy chúng ta sẽ cần loại bỏ các giá trị λ quá nhỏ trong kết quả (thực chất đó là 0 nhưng phương pháp giải lặp có sai số tính toán).

[ ]:
```
epsilon = 1e-6 # just a small number, greater than 1e-9, to filter values of␣
 ↪lambda
S = np.where(l > epsilon)[0]
VS = V[:, S]
XS = X[:, S]
yS = y[:, S]
lS = l[S]
# calculate w and b
w = VS.dot(lS)
w0 = np.mean(yS.T - w.T.dot(XS))
print('W = ', w.T)
print('W0 = ', w0)
```

```
W =  [[-10.10601344   6.8886009 ]]
W0 =  17.241417859332664
```

[ ]:
```
from matplotlib.backends.backend_pdf import PdfPages
with PdfPages('output/svm_hardmargin_output_vd1.pdf') as pdf:
    # draw
```

```
    # plot points
    fig, ax = plt.subplots()
    x1 = np.arange(-10, 10, 0.1)
    y1 = -w[0, 0]/w[1, 0]*x1 - w0/w[1, 0]
    y2 = -w[0, 0]/w[1, 0]*x1 - (w0-1)/w[1, 0]
    y3 = -w[0, 0]/w[1, 0]*x1 - (w0+1)/w[1, 0]
    plt.plot(x1, y1, 'k', linewidth = 3)
    plt.plot(x1, y2, 'k')
    plt.plot(x1, y3, 'k')
    y4 = 10*x1
    plt.plot(x1, y1, 'k')
    plt.fill_between(x1, y1, color='red', alpha=0.1)
    plt.fill_between(x1, y1, y4, color = 'blue', alpha = 0.1)
    plt.plot(X0[:, 0], X0[:, 1], 'bs', markersize = 8, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'ro', markersize = 8, alpha = .8)
    plt.axis('equal')
    plt.ylim(0, 3)
    plt.xlim(2, 4)
    # hide tikcs
    cur_axes = plt.gca()
    cur_axes.axes.get_xaxis().set_ticks([])
    cur_axes.axes.get_yaxis().set_ticks([])
    # add circles around support vectors
    for m in S:
        circle = plt.Circle((X[0, m], X[1, m] ), 0.1, color='k', fill = False)
        ax.add_artist(circle)
    plt.xlabel('$x_1$', fontsize = 20)
    plt.ylabel('$x_2$', fontsize = 20)
    # plt.savefig('svm4.png', bbox_inches='tight', dpi = 300)
    # pdf.savefig()
    plt.show()
```




```
/tmp/ipykernel_6512/3408228427.py:2: MatplotlibDeprecationWarning: Keeping empty
pdf files is deprecated since 3.8 and support will be removed two minor releases
later.
  with PdfPages('output/svm_hardmargin_output_vd1.pdf') as pdf:
```

1.3 Thực hiện SVM HardMargin qua thư viện

[ ]:
```
from sklearn.svm import SVC
# Copy and put code for generate data here
y1 = y.reshape((2*N,))
X1 = X.T # each sample is one row
clf = SVC(kernel = 'linear', C = 1e5) # just a big number: then will be hard␣
 ↪margin | else if nearly 0 then be soft margin
# if C is small, method will be "SoftMagin SVM",
# if C is large enough, method is near to hard margin
clf.fit(X1, y1)
w = clf.coef_
w0 = clf.intercept_
print('w = ', w)
print('W0 = ', w0)
```

```
w =  [[-10.10242378   6.88613861]]
```

```
W0 =  [17.23542499]
```

1.4 So sánh bộ trọng số của cách dùng đúng công thức lý thuyết & dùng thư viện sklearn

Bộ trọng số của cách dùng công thức lý thuyết khá tương đồng với Bộ trọng số của cách dùng thư viện sklearn

2 Bài thực hành 1

(Dữ liệu tự tạo ở ví dụ 1 (train set) - Dữ liệu tự tạo mới (test set))

[ ]:
```
means = [[2, 2], [4, 2]]
cov = [[0.3, 0.2], [0.2, 0.3]]
N = 100
X0_test = np.random.multivariate_normal(means[0], cov, N)  # class 1
X1_test = np.random.multivariate_normal(means[1], cov, N)  # class -1
X_test = np.concatenate((X0_test.T, X1_test.T), axis=1)  # all data
y_test = np.concatenate((np.ones((1, N)), -1 * np.ones((1, N))), axis=1)  #␣
↪labels
```

[ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix

# Test the model
y_pred = clf.predict(X_test.T)
accuracy = accuracy_score(y_test.T, y_pred)
print("Accuracy:", accuracy)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test.T, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.995

Confusion Matrix:
[[100   0]
 [  1  99]]
```

3 Bài thực hành 2

[ ]:
```
from sklearn import datasets

cancer_data = datasets.load_breast_cancer()
# show to test record 5th
print(cancer_data.data[5])
print(cancer_data.data.shape)
```

```
# target set
print(cancer_data.target[5])
print(cancer_data.target.shape)
```

```
[1.245e+01 1.570e+01 8.257e+01 4.771e+02 1.278e-01 1.700e-01 1.578e-01
 8.089e-02 2.087e-01 7.613e-02 3.345e-01 8.902e-01 2.217e+00 2.719e+01
 7.510e-03 3.345e-02 3.672e-02 1.137e-02 2.165e-02 5.082e-03 1.547e+01
 2.375e+01 1.034e+02 7.416e+02 1.791e-01 5.249e-01 5.355e-01 1.741e-01
 3.985e-01 1.244e-01]
(569, 30)
0
(569,)
```

569 bản ghi và 30 trường dữ liệu

[ ]:
```
from sklearn.model_selection import train_test_split

cancer_data = datasets.load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer_data.data, cancer_data.target, test_size=0.3, random_state=109
)
```

[ ]:
```
from sklearn.svm import SVC

# Train the SVM model
clf = SVC(kernel='linear', C=1e5) # just a big number: then will be hard margin␣
 ↪| else if nearly 0 then be soft margin
# if C is small, method will be "SoftMagin SVM",
# if C is large enough, method is near to hard margin
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.9649122807017544

Confusion Matrix:
[[ 61   2]
 [  4 104]]
```

4 Ví dụ 2

(Dữ liệu sóng thủy âm Sonar)

Tệp dữ liệu có 60 cột ứng với 60 thuộc tính (trường) không có tiêu đề, là tham số của các mẫu sóng âm phản hồi; cột thứ 61 là đầu ra phân loại (y), với ký tự "R" nghĩa là Rock; ký tự "M" nghĩa là Mine (vật thể kim loại hình trụ).

Toàn bộ tệp có 208 bản ghi. Thông tin thêm về dữ liệu có thể tìm hiểu tại link https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks) hoặc có thể lấy trong tệp Sonar.all-data.csv đính kèm.

Tham khảo lại phần hướng dẫn cho mô hình perceptron để nắm các thao tác đọc dữ liệu, định dạng lại dữ liệu (chuyển M thành class 1 và chuyển R thành class -1).

Đọc dữ liệu và sử dụng mô hình SVM với hard margin để phân loại cho dữ liệu này

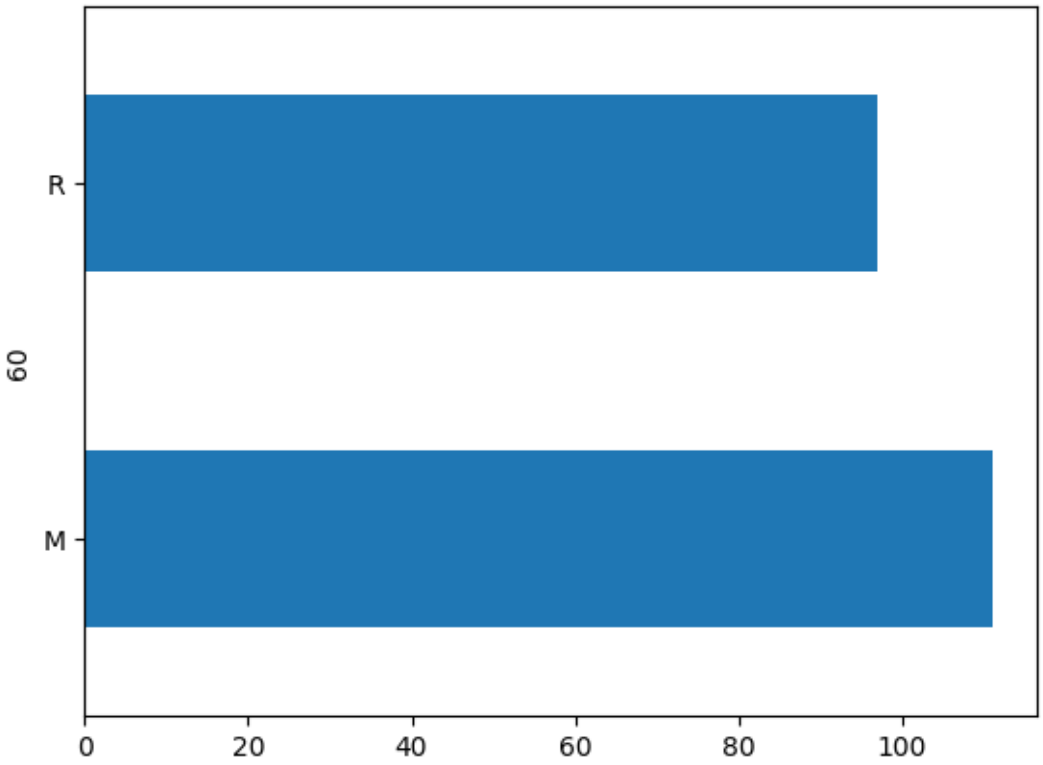
4.1 Read data

[ ]:
```
# %pip install pandas
# %pip install seaborn
```

[ ]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix

# Đọc dữ liệu từ tệp CSV
data = pd.read_csv("data/sonar.csv", header=None)
```

[ ]:
```
# Kiểm tra độ cân bằng của data
data[60].value_counts().plot(kind='barh')
```

[ ]:
```
<Axes: ylabel='60'>
```



4.2 Mã hóa data

[ ]:
```
# Chuyển đổi nhãn "M" thành 1 và nhãn "R" thành -1
data[60] = data[60].apply(lambda x: 1 if x == "M" else -1)

# Tách dữ liệu thành features (X) và nhãn (y)
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

4.3 Phân tách data

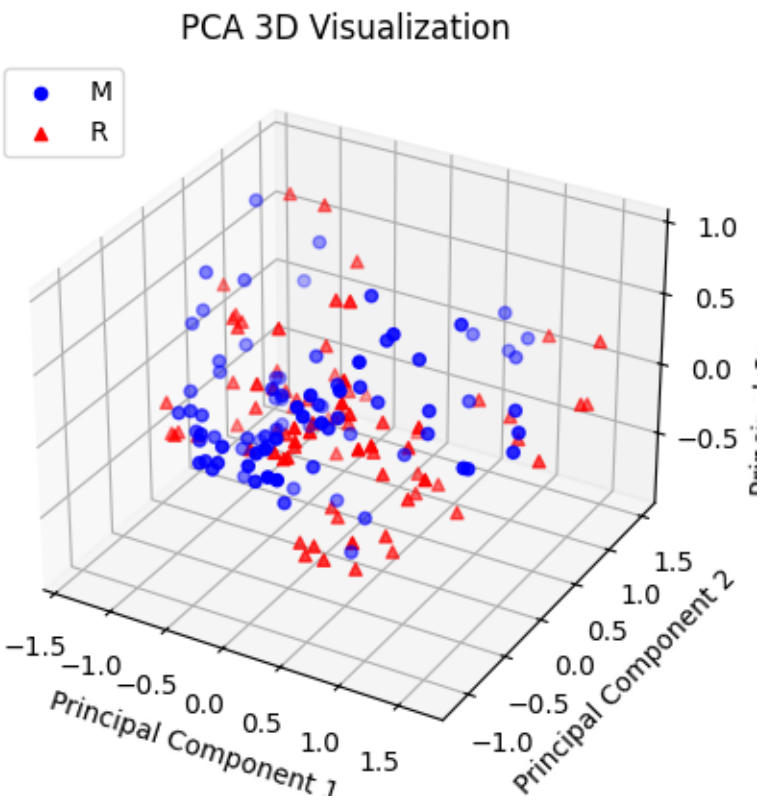
[ ]:
```
# Tách dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

4.4 Visualize data

[ ]:
```
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Giảm chiều dữ liệu xuống 3D bằng PCA
pca = PCA(n_components=3)
```

```
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

[ ]:
```
# Trực quan hóa dữ liệu
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Tách dữ liệu thành 2 lớp để trực quan hóa
X_class1 = X_train_pca[y_train == 1]
X_class2 = X_train_pca[y_train == -1]

# Trực quan hóa dữ liệu huấn luyện
ax.scatter(X_class1[:, 0], X_class1[:, 1], X_class1[:, 2], c='b', marker='o',␣
 ↪label='M')
ax.scatter(X_class2[:, 0], X_class2[:, 1], X_class2[:, 2], c='r', marker='^',␣
 ↪label='R')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
plt.legend(loc='upper left')
plt.title('PCA 3D Visualization')
plt.show()
```



4.5 Dùng SVM Hard Margin

[ ]:
```
# Huấn luyện mô hình SVM với hard margin
model = SVC(kernel='linear', C=1e10)
model.fit(X_train, y_train)
```

[ ]:
```
SVC(C=10000000000.0, kernel='linear')
```

4.6 Đánh giá mô hình

[ ]:
```
# Đánh giá mô hình trên tập kiểm tra
y_pred_train = model.predict(X_train)
accuracy_train = accuracy_score(y_train, y_pred_train)
print("Accuracy:", accuracy_train)
```

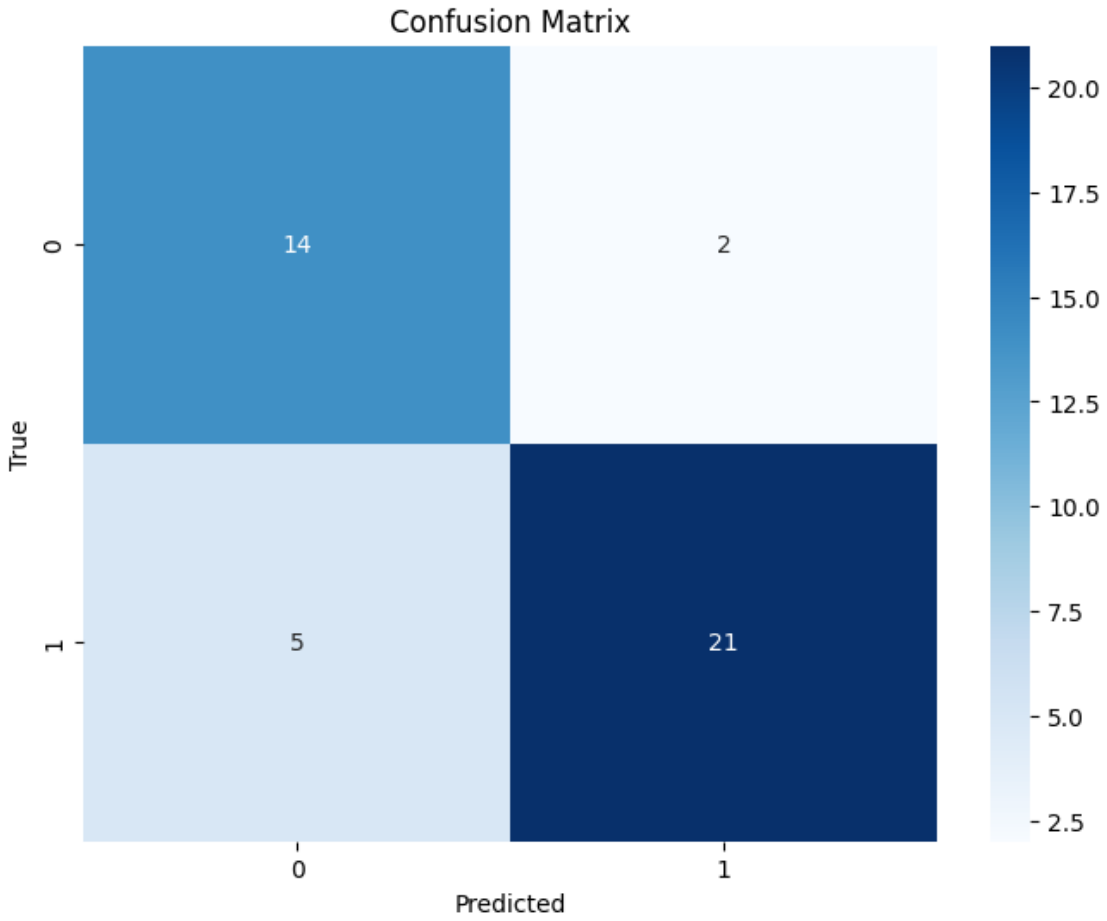
```
Accuracy: 1.0
```

[ ]:
```
# Đánh giá mô hình trên tập kiểm tra
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8333333333333334
```

[ ]:
```
# Ma trận nhầm lẫn
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Trực quan hóa ma trận nhầm lẫn
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

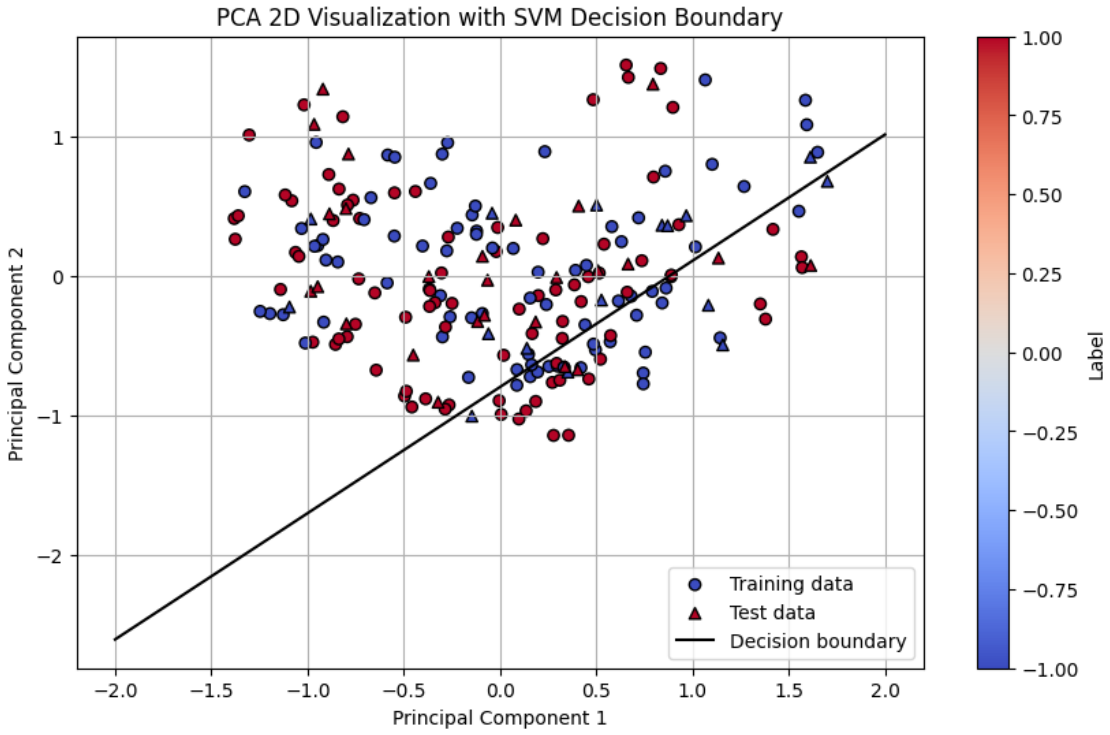
```
Confusion Matrix:
[[14  2]
 [ 5 21]]
```



4.7 Giải thích kết quả

[ ]:
```
# Giảm chiều dữ liệu xuống 2D bằng PCA
pca = PCA(n_components=2)
X_train_pca_2d = pca.fit_transform(X_train)
X_test_pca_2d = pca.transform(X_test)

# Vẽ dữ liệu và đường phân loại
plt.figure(figsize=(10, 6))

# Vẽ điểm dữ liệu
plt.scatter(X_train_pca_2d[:, 0], X_train_pca_2d[:, 1], c=y_train,␣
 ↪cmap='coolwarm', marker='o', edgecolors='k', label='Training data')
plt.scatter(X_test_pca_2d[:, 0], X_test_pca_2d[:, 1], c=y_test,␣
 ↪cmap='coolwarm', marker='^', edgecolors='k', label='Test data')

# Vẽ đường phân loại
```

```
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-2, 2)
yy = a * xx - (model.intercept_[0]) / w[1]
plt.plot(xx, yy, 'k-', label='Decision boundary')

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA 2D Visualization with SVM Decision Boundary')
plt.legend()
plt.colorbar(label='Label')
plt.grid(True)
plt.show()
```



Hình phân loại này hơi khó để trực quan và đường phân loại trực quan cho siêu phẳng 60 chiều trên có thể không đúng

Nhận xét:
Mô hình không cho độ chính xác cao = 1.0 như trên tập train, tuy nhiên độ chính xác vẫn ở mức chấp nhận được 0.83.
Lý do vì đây là HardMargin đảm bảo tập train là 1.0 (nếu data có thể tách rò ràng bằng 1 siêu phẳng tuyến tính).
Tuy nhiên đường phân loại này có thể không phân loại đúng với 1 vài trường hợp có trong tập test.
Bởi vậy có thể sử dụng theo hướng SoftMargin để xem độ chính xác phân loại trên test có thể cao

hơn 0.83 không.