create two groups using the make\_blobs() function from Scikit-learn

use the Python library cvxpy to do convex optimization;

Generate two data difference variability:

X\_noSlack, labels\_noSlack = make\_blobs(n\_samples=100, n\_features=2, centers=2, cluster\_std=1.25, random\_state=1234)  
X\_Slack, labels\_Slack = make\_blobs(n\_samples=100, n\_features=2, centers=2, cluster\_std=2, random\_state=1234)

Plot the results:

def plot\_decision\_boundary(X, y, theta0, theta):  
 # X --> Inputs  
 # theta --> parameters  
  
 # The Line is y=mx+c  
 # So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2  
 # Solving we find m and c  
 x1 = np.array([min(X[:, 0]), max(X[:, 0])])  
 m = -theta[0] / theta[1]  
 c = -theta0 / theta[1]  
 x2 = m \* x1 + c  
  
 fig = plt.figure(figsize=(10, 8))  
 plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "r^")  
 plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], "bs")  
 plt.xlabel("Feature 1")  
 plt.ylabel("Feature 2")  
 plt.title('SVM')  
 plt.plot(x1, x2, 'y-')

1. No slack variable case:

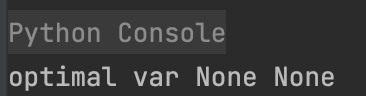
def svm\_noSlack(X, labels):  
 # split the data in the two classes. Name them class\_1 and class\_2.  
 ## Assign label 0 to class\_1  
 class\_1 = X[labels == 0,]  
 ## Assign label 1 to class\_2  
 class\_2 = X[labels == 1,]  
 # Define the variables  
 beta = cp.Variable(2)  
 beta0 = cp.Variable()  
  
 # Define the constraints  
 constraint1 = []  
 constraint2 = []  
 for i in range(class\_1.shape[0]):  
 constraint1 += [np.matmul(class\_1[i,], beta) + beta0 <= -1]  
 for j in range(class\_2.shape[0]):  
 constraint2 += [np.matmul(class\_2[j,], beta) + beta0 >= 1]  
 # Sum the constraints  
 constraints = constraint1 + constraint2  
 # Define the objective. Hint: use cp.norm  
 obj = cp.Minimize((cp.norm(beta)\*\*2)/2)  
 # Add objective and constraint in the problem  
 prob = cp.Problem(obj, constraints)  
 # Solve the problem  
 prob.solve()  
 print("optimal var", beta.value, beta0.value)  
  
 return beta.value, beta0.value

results = svm\_noSlack(X\_noSlack, labels\_noSlack)  
beta = results[0]  
beta0 = results[1]  
plot\_decision\_boundary(X\_noSlack, labels\_noSlack, beta0, beta)

Chart, scatter chart

Description automatically generated

In this case, the two sets are linear separatable, so there is a clear solution.

However, when the two sets are not linear separatable, there will be no solution returned by the svm\_noSlack function: 

1. slack variable case:

def svm\_Slack(X, labels, C):  
 # split the data in the two classes. Name them class\_1 and class\_2.  
 ## Assign label 0 to class\_1  
 class\_1 = X[labels == 0,]  
 ## Assign label 1 to class\_2  
 class\_2 = X[labels == 1,]  
 # Define the variables  
 beta = cp.Variable(2)  
 beta0 = cp.Variable()  
 xi\_1 = cp.Variable(class\_1.shape[0])  
 xi\_2 = cp.Variable(class\_2.shape[0])  
 # Define the constraints  
 constraint1 = []  
 constraint2 = []  
 for i in range(class\_1.shape[0]):  
 constraint1 += [np.matmul(class\_1[i,], beta) + beta0 - xi\_1[i] <= -1]  
 constraint1 += [xi\_1[i] >= 0]  
 for j in range(class\_2.shape[0]):  
 constraint2 += [np.matmul(class\_2[j,], beta) + beta0 + xi\_2[j] >= 1]  
 constraint2 += [xi\_2[j] >= 0]  
  
 # Sum the constraints  
 constraints = constraint1 + constraint2  
 # Define the objective. Hint: use cp.norm  
 obj = cp.Minimize(C\*(sum(xi\_1) + sum(xi\_2)) + (cp.norm(beta)\*\*2)/2)  
 # Add objective and constraint in the problem  
 prob = cp.Problem(obj, constraints)  
 # Solve the problem  
 prob.solve()  
 print("optimal var", beta.value, beta0.value)  
  
 return beta.value, beta0.value

results = svm\_Slack(X\_Slack, labels\_Slack, C=1)  
beta = results[0]  
beta0 = results[1]  
plot\_decision\_boundary(X\_Slack, labels\_Slack, beta0, beta)

Chart, scatter chart

Description automatically generated

1. support vector regression

Text, letter

Description automatically generated

X = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))  
y = np.array([5, 20, 14, 32, 22, 38])

def svm\_Regression(X, y, C=1, epsilon=0.1):  
 # Define the variables  
 beta = cp.Variable()  
 beta0 = cp.Variable()  
  
 # Define the constraints  
 xi = cp.Variable(X.shape[0])  
 xi\_hat = cp.Variable(X.shape[0])  
  
 constraints = []  
 for i in range(X.shape[0]):  
 constraints += [xi[i] >= 0]  
 constraints += [xi\_hat[i] >= 0]  
 constraints += [(X[i] \* beta + beta0) - (y[i] + epsilon) - xi[i] <= 0]  
 constraints += [(y[i] - epsilon) - (X[i] \* beta + beta0) - xi\_hat[i] <= 0]  
  
 # Define the objective  
 obj = cp.Minimize(C\*(sum(xi)+sum(xi\_hat)) + (cp.norm(beta) \*\* 2)/2)  
 # Add objective and constraint in the problem  
 prob = cp.Problem(obj, constraints)  
 # Solve the problem  
 prob.solve()  
 print("optimal var", beta.value, beta0.value)  
  
 return beta.value, beta0.value

def plot\_decision\_boundary\_regression(X, y, theta0, theta):  
 x1 = np.array([min(X), max(X)])  
 x2 = beta \* x1 + beta0  
  
 fig = plt.figure(figsize=(10, 8))  
 plt.plot(X, y, "r^")  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.title('SV Regression')  
 plt.plot(x1, x2, 'y-')

results = svm\_Regression(X, y, C=1, epsilon=0.1)  
beta = results[0]  
beta0 = results[1]  
plot\_decision\_boundary\_regression(X,y,beta0,beta)

Chart, scatter chart

Description automatically generated