example

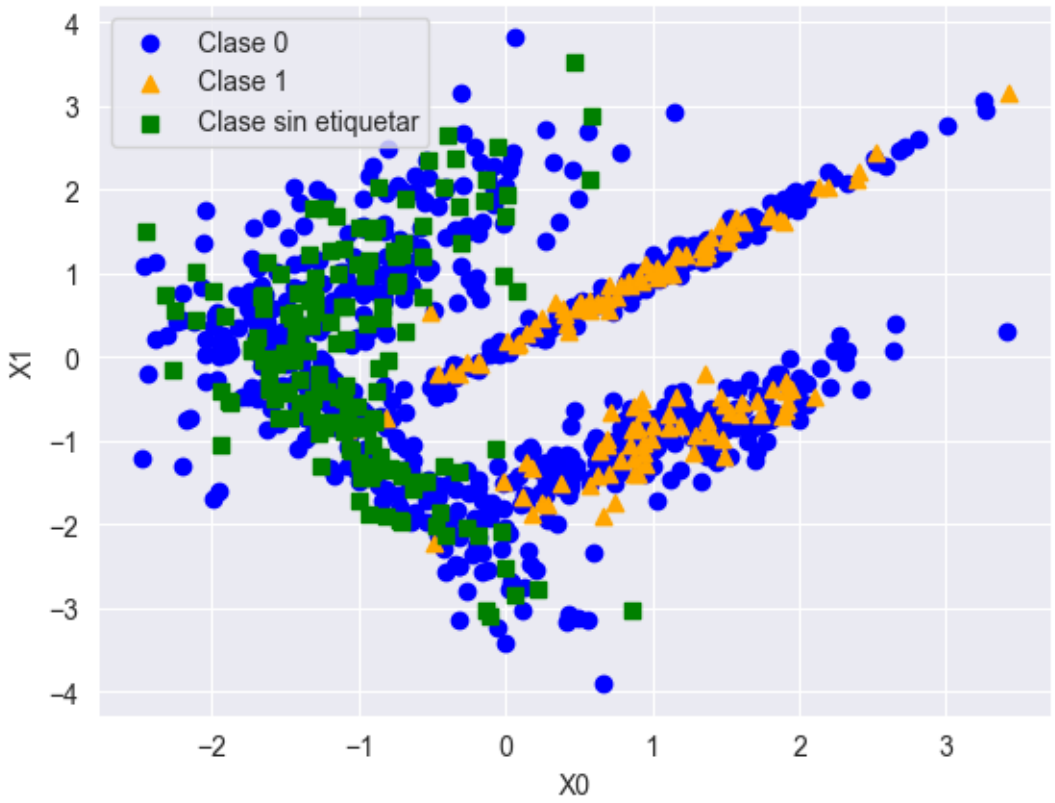
February 4, 2024

[125]:
```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
import numpy as np
```

[129]:
```
nb_samples = 1000
nb_unlabeled = 700
```

[134]:
```
X, Y = make_classification(n_samples=nb_samples, n_features=2, n_redundant=0,␣
↪random_state=1000000)
Y[Y==0] = -1
Y[nb_samples - nb_unlabeled:nb_samples] = 0
```

[135]:
```
plt.scatter(X[Y == 0, 0], X[Y == 0, 1], c='blue', marker='s', label='Clase 0')
plt.scatter(X[Y == 1, 0], X[Y == 1, 1], c='red', marker='^', label='Clase 1')
plt.scatter(X[Y == -1, 0], X[Y == -1, 1], c='green', marker='o', label='Clase␣
↪sin etiquetar')
plt.xlabel('X0')
plt.ylabel('X1')
plt.legend()
plt.show()
```



[136]:
```
w = np.random.uniform(-0.1, 0.1, size=X.shape[1])
eta = np.random.uniform(0.0, 0.1, size=nb_samples - nb_unlabeled)
xi = np.random.uniform(0.0, 0.1, size=nb_unlabeled)
zi = np.random.uniform(0.0, 0.1, size=nb_unlabeled)
b = np.random.uniform(-0.1, 0.1, size=1)
C = 1.0
theta0 = np.hstack((w, eta, xi, zi, b))
```

[137]:
```
vmin = np.vectorize(lambda x1, x2: x1 if x1 <= x2 else x2)
```

[138]:
```
def svm_target(theta, Xd, Yd):
    wt = theta[0:2].reshape((Xd.shape[1], 1))

    s_eta = np.sum(theta[2:2 + nb_samples - nb_unlabeled])
    s_min_xi_zi = np.sum(vmin(theta[2 + nb_samples - nb_unlabeled:2 +␣
 ↪nb_samples],theta[2 + nb_samples:2+ nb_samples + nb_unlabeled]))
    return C * (s_eta + s_min_xi_zi) + 0.5 * np.dot(wt.T, wt)
```

[139]:
```
def labeled_constraint(theta, Xd, Yd, idx):
    wt = theta[0:2].reshape((Xd.shape[1], 1))
```

```
    c = Yd[idx] * (np.dot(Xd[idx], wt) + theta[-1])+theta[2:2 + nb_samples -␣
↪nb_unlabeled][idx] - 1.0
    return (c >= 0)[0]
```

[140]:
```
def unlabeled_constraint_1(theta, Xd, idx):
    wt = theta[0:2].reshape((Xd.shape[1], 1))
    c = np.dot(Xd[idx], wt) - theta[-1] +theta[2 + nb_samples - nb_unlabeled:2␣
↪+ nb_samples][idx - nb_samples + nb_unlabeled] - 1.0
    return (c >= 0)[0]
def unlabeled_constraint_2(theta, Xd, idx):
    wt = theta[0:2].reshape((Xd.shape[1], 1))
    c = -(np.dot(Xd[idx], wt) - theta[-1]) +theta[2 + nb_samples:2 + nb_samples␣
↪+ nb_unlabeled ][idx - nb_samples + nb_unlabeled] - 1.0
    return (c >= 0)[0]
```

[141]:
```
def eta_constraint(theta, idx):
    return theta[2:2 + nb_samples - nb_unlabeled][idx] >= 0
def xi_constraint(theta, idx):
    return theta[2 + nb_samples - nb_unlabeled:2 + nb_samples][idx - nb_samples␣
↪+ nb_unlabeled] >= 0
def zi_constraint(theta, idx):
    return theta[2 + nb_samples:2 + nb_samples+nb_unlabeled ][idx - nb_samples␣
↪+ nb_unlabeled] >= 0
```

[142]:
```
svm_constraints = []
for i in range(nb_samples - nb_unlabeled):
    svm_constraints.append({
            'type': 'ineq',
            'fun': labeled_constraint,'args': (X, Y, i)
        })
    svm_constraints.append({
            'type': 'ineq',
            'fun': eta_constraint,
            'args': (i,)
        })
for i in range(nb_samples - nb_unlabeled, nb_samples):
    svm_constraints.append({
            'type': 'ineq','fun': unlabeled_constraint_1,'args': (X, i)
        })
    svm_constraints.append({
            'type': 'ineq','fun': unlabeled_constraint_2,'args': (X, i)
        })
    svm_constraints.append({'type': 'ineq','fun': xi_constraint,'args': (i,)
        })
    svm_constraints.append({ 'type': 'ineq','fun': zi_constraint,'args': (i,)
        })
```

[143]:
```
from scipy.optimize import minimize
result = minimize(fun=svm_target,
                  x0=theta0,
                  constraints=svm_constraints,
                  args=(X, Y),
                  method='COBYLA',
                  tol=0.0001,
                  options={'maxiter': 5000})
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[143], line 2
      1 from scipy.optimize import minimize
----> 2 result = minimize(fun=svm_target,
      3                   x0=theta0,
      4                   constraints=svm_constraints,
      5                   args=(X, Y),
      6                   method='COBYLA',
      7                   tol=0.0001,
      8                   options={'maxiter': 5000})

File␣
 ↪~\PycharmProjects\pythonProject4\venv\lib\site-packages\scipy\optimize\_minim ze.
 ↪py:716, in minimize(fun, x0, args, method, jac, hess, hessp, bounds,␣
 ↪constraints, tol, callback, options)
    713     res = _minimize_tnc(fun, x0, args, jac, bounds, callback=callback,
    714                         **options)
    715 elif meth == 'cobyla':
--> 716     res = _minimize_cobyla(fun, x0, args, constraints, callback=callbac ,
    717                            bounds=bounds, **options)
    718 elif meth == 'slsqp':
    719     res = _minimize_slsqp(fun, x0, args, jac, bounds,
    720                           constraints, callback=callback, **options)

File␣
 ↪~\PycharmProjects\pythonProject4\venv\lib\site-packages\scipy\optimize\_cobyl _py.
 ↪py:35, in synchronized.<locals>.wrapper(*args, **kwargs)
     32 @functools.wraps(func)
     33 def wrapper(*args, **kwargs):
     34     with _module_lock:
---> 35         return func(*args, **kwargs)

File␣
 ↪~\PycharmProjects\pythonProject4\venv\lib\site-packages\scipy\optimize\_cobyl _py.
 ↪py:293, in _minimize_cobyla(fun, x0, args, constraints, rhobeg, tol, maxiter,
 ↪disp, catol, callback, bounds, **unknown_options)
    290         callback(np.copy(x))
    292 info = np.zeros(4, np.float64)
```

```
--> 293 xopt, info = cobyla.minimize(calcfc, m=m, x=np.copy(x0), rhobeg=rhobeg,
    294␣
 ↪                                rhoend=rhoend, iprint=iprint, maxfun=maxfun,
    295                                  dinfo=info, callback=wrapped_callback)
    297 if info[3] > catol:
    298     # Check constraint violation
    299     info[0] = 4

File␣
 ↪~\PycharmProjects\pythonProject4\venv\lib\site-packages\scipy\optimize\_cobyla_py.
 ↪py:289, in _minimize_cobyla.<locals>.wrapped_callback(x)
    288 def wrapped_callback(x):
--> 289     if callback is not None:
    290         callback(np.copy(x))

KeyboardInterrupt:
```

[ ]:
```
theta_end = result['x']
w = theta_end[0:2]
b = theta_end[-1]
Xu= X[nb_samples - nb_unlabeled:nb_samples]
yu = -np.sign(np.dot(Xu, w) + b)
```

[ ]:
```
yu
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# Graficar los datos antes de etiquetar
ax1.scatter(X[Y == -1, 0], X[Y == -1, 1], c='blue', marker='s', label='Clase 0')
ax1.scatter(X[Y == 1, 0], X[Y == 1, 1], c='orange', marker='^', label='Clase 1')
ax1.scatter(X[Y == 0, 0], X[Y == 0, 1], c='black', marker='.', label='Sin␣
 ↪etiquetar')
ax1.set_title('Antes')
ax1.set_xlabel('X0')
ax1.set_ylabel('X1')
ax1.legend()

ax2.scatter(X[Y == -1, 0], X[Y == -1, 1], c='blue', marker='s', label='Clase 0')
ax2.scatter(X[Y == 1, 0], X[Y == 1, 1], c='red', marker='^', label='Clase 1')
ax2.scatter(Xu[yu == -1, 0], Xu[yu == -1, 1], c='blue', marker='8', label='Sin␣
 ↪clasificar Clase 0')
ax2.scatter(Xu[yu == 1, 0], Xu[yu == 1, 1], c='red', marker='v', label='Sin␣
 ↪clasificar Clase 1')
ax2.set_title('Después')
ax2.set_xlabel('X0')
ax2.set_ylabel('X1')
ax2.legend()
```

```
# Mostrar la figura
plt.show()
```

[ ]: