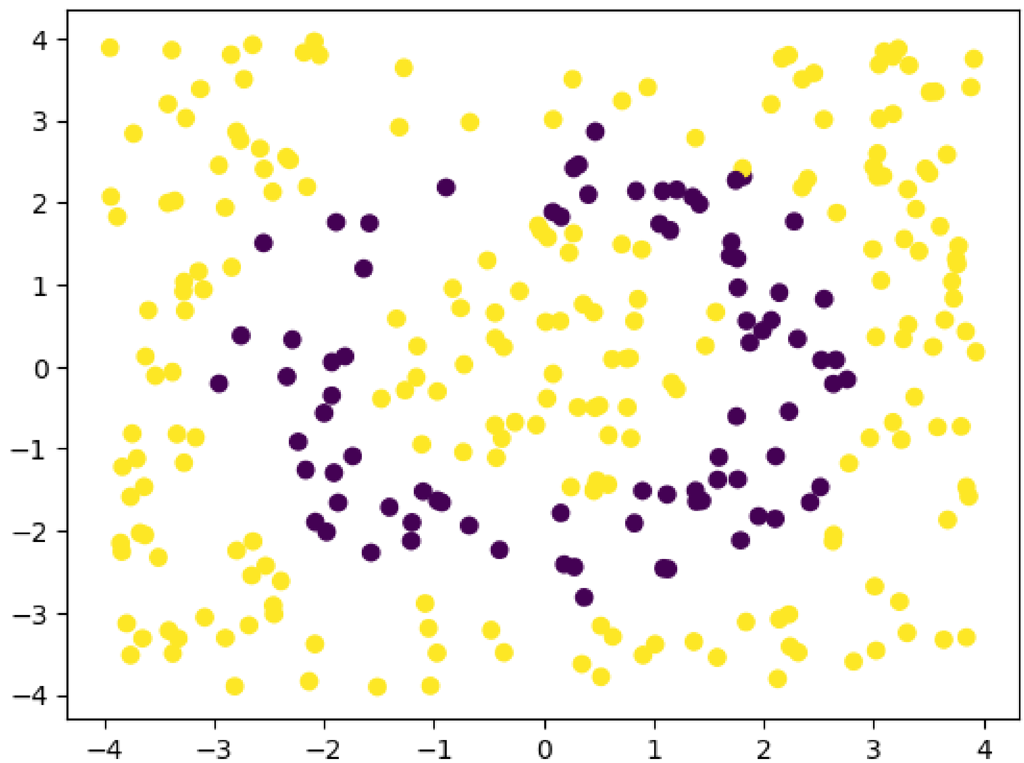
mmo

November 23, 2025

[1]:
```
import numpy as np, matplotlib.pyplot as plt, torch
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
from pymoo.termination import get_termination
from pymoo.visualization.scatter import Scatter
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.termination import get_termination
from pymoo.optimize import minimize
from pymoo.visualization.scatter import Scatter
from pymoo.problems.functional import FunctionalProblem
```

[2]:
```
def decison_bound (x,y):
    return x**2 + y**2 <=3 or x**2 + y**2 >=9
def generate_data (n=100, bound=decison_bound):
    X = np.random.uniform(-4,4,(n,2))
    Y = np.array([1 if bound(x[0],x[1]) else 0 for x in X])
    return X,Y

X,Y = generate_data(300)
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()
```

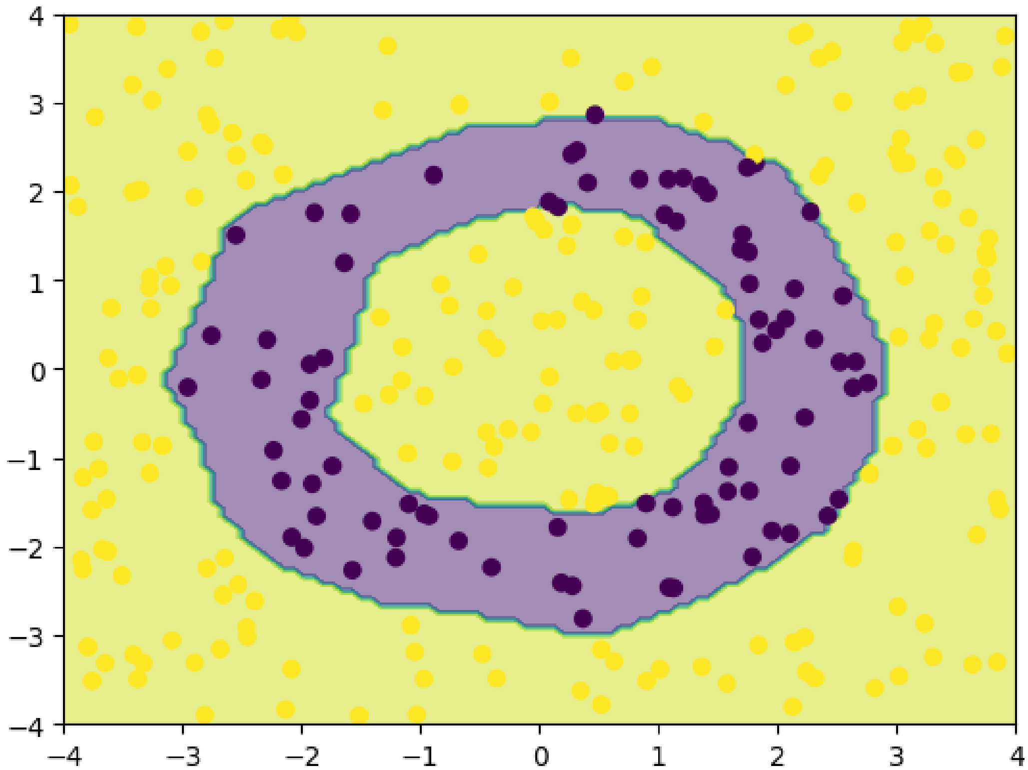


[3]:
```
from torch.utils.data import TensorDataset, DataLoader
from torch import nn, optim
dataset = TensorDataset(torch.tensor(X).float(), torch.tensor(Y).long())
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
class SimpleNN (nn.Module):
    def __init__ (self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2,16),
            nn.ReLU(),
            nn.Linear(16,16),
            nn.ReLU(),
            nn.Linear(16,16),
            nn.ReLU(),
            nn.Linear(16,16),
            nn.ReLU(),
            nn.Linear(16,2)

        )
    def forward (self,x):
        return self.net(x)
model = SimpleNN()
```

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
for epoch in range(100):
    for xb, yb in dataloader:
        pred = model(xb)
        loss = criterion(pred, yb)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
# Visualize decision boundary
xx, yy = np.meshgrid(np.linspace(-4,4,100), np.linspace(-4,4,100))
grid = torch.tensor(np.c_[xx.ravel(), yy.ravel()]).float()
with torch.no_grad():
    Z = model(grid).argmax(dim=1).numpy().reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.5)
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()
```

```
Epoch 0, Loss: 0.7300
Epoch 10, Loss: 0.6859
Epoch 20, Loss: 0.5873
Epoch 30, Loss: 0.4619
Epoch 40, Loss: 0.1041
Epoch 50, Loss: 0.1040
Epoch 60, Loss: 0.0803
Epoch 70, Loss: 0.1154
Epoch 80, Loss: 0.0108
Epoch 90, Loss: 0.0676
```



[46]:
```
def make_cf_problem(model, x_star, Y_prime, X_obs, weights):

    model.eval()

    # -------    numpy     pymoo   numpy  -------
    x_star_t = x_star.detach().cpu()
    X_obs_t = X_obs.detach().cpu()
    Y_prime_t = Y_prime.detach().cpu()
    w_t = weights.detach().cpu()

    x_star_np = x_star_t.numpy()            # shape: (p,)
    X_obs_np = X_obs_t.numpy()              # shape: (k, p)
    w_np = w_t.numpy()                      # shape: (k,)
    p = x_star_np.shape[0]                  #     p

    #          min/max
    xl = X_obs_np.min(axis=0)
    xu = X_obs_np.max(axis=0)

    feature_range = X_obs_np.max(axis=0) - X_obs_np.min(axis=0)
    feature_range[feature_range == 0] = 1.0     #     0
```

```python
def delta_G_vec(x, y):
    # normalized |x - y| in [0, 1]
    return np.minimum(np.abs(x - y) / feature_range, 1.0)

# ------- 4    elementwise   1   x -------

# 1) Validity o1(f(x), Y') = 0 if f(x) in Y', else inf_{y' in Y'} |f(x) - y'|
#    f(x)     argmax Y'
#       0/1
def o1_validity(x):
    x_t = torch.from_numpy(x.astype(np.float32)).unsqueeze(0)  # shape: (1, p)
    with torch.no_grad():
        logits = model(x_t)            # shape: (1, num_classes)
        y_hat_class = int(logits.argmax(dim=1).item())  #

    # Y_prime_t      1D
    target_labels = Y_prime_t.view(-1).long().tolist()

    # formal definition:
    # if f(x)   Y' -> 0
    # else inf_{y' Y'} |f(x) - y'|,   0/1   => 1
    if y_hat_class in target_labels:
        return 0.0
    else:
        return 1.0

# 2) Similarity  x*
def o2_similarity(x):
    return float(delta_G_vec(x, x_star_np).mean())
EPS = 1e-6    # or something like 1e-8

def o3_sparsity(x):
    # ||x - x*||_0 = number of coordinates that changed
    diff = np.abs(x - x_star_np) > EPS
    return float(diff.sum())

def o4_plausibility(x):
    diffs = (X_obs_np - x) / feature_range            # (k, p)
    per_sample = np.sqrt((diffs ** 2).mean(axis=1))   # RMS distance per sample
    w_np_norm = w_np / w_np.sum()
    return float((per_sample * w_np_norm).sum())

objs = [o1_validity, o2_similarity, o3_sparsity, o4_plausibility]
```

```
    problem = FunctionalProblem(
    n_var=p,
    objs=objs,
    xl=xl,
    xu=xu,
    elementwise=True   #       x
)

    return problem
```

[61]:
```
for x, y in dataloader:
    x_star = torch.tensor([-3.0, -3.0])        # original feature vector, tensor shape (p,)
    y_star = model(x_star.unsqueeze(0)).argmax(dim=1)          # original label, tensor scalar
    Y_prime = 1 - y_star   # target label = opposite class, still a tensor
    break

X_obs = dataset.tensors[0]                             # (k, p)
weights = torch.ones(len(X_obs)) / len(X_obs)     # normalized weights

# model:      PyTorch
# x_star:     shape (p,)
# Y_prime:    shape (n_y,)    tensor([1])
# X_obs:      shape (k, p)
# weights:    shape (k,)    weights.sum() == 1

problem = make_cf_problem(model, x_star, Y_prime, X_obs, weights)
print(f'problem: \n{problem}\nx_star: {x_star} dtype={x_star.dtype}\ny_star: {y_star} dtype={y_star.dtype}\nY_prime: {Y_prime} dtype={Y_prime.dtype}\n')
```

```
problem:
# name: FunctionalProblem
# n_var: 2
# n_obj: 4
# n_ieq_constr: 0
# n_eq_constr: 0

x_star: tensor([-3., -3.]) dtype=torch.float32
y_star: tensor([1]) dtype=torch.int64
Y_prime: tensor([0]) dtype=torch.int64
```

[62]:
```
# 3.    NSGA-II
algorithm = NSGA2(pop_size=200)

# 4.
```

```python
termination = get_termination("n_gen", 150)

# 5.
res = minimize(
    problem,
    algorithm,
    termination,
    seed=1,
    verbose=True
)

F_mmo = res.F   # shape (n_points, 4)  - 4
X_mmo = res.X   #    x counterfactual
```

```
==========================================================
n_gen  |  n_eval  | n_nds  |      eps       |  indicator
==========================================================
     1 |      200 |     24 |              - |            -
     2 |      400 |     38 |  0.0161086118 |            f
     3 |      600 |     56 |  0.0283300066 |        ideal
     4 |      800 |     84 |  0.2918553079 |        nadir
     5 |     1000 |    112 |  0.0161482289 |        ideal
     6 |     1200 |    155 |  0.0029280554 |            f
     7 |     1400 |    192 |  0.0016162887 |            f
     8 |     1600 |    200 |  0.0031595497 |            f
     9 |     1800 |    200 |  0.0236131650 |        nadir
    10 |     2000 |    200 |  0.0025127633 |        ideal
    11 |     2200 |    200 |  0.0008021693 |            f
    12 |     2400 |    200 |  0.0016311007 |            f
    13 |     2600 |    200 |  0.0020713096 |            f
    14 |     2800 |    200 |  0.0028266040 |            f
    15 |     3000 |    200 |  0.0007672678 |            f
    16 |     3200 |    200 |  0.0012367182 |            f
    17 |     3400 |    200 |  0.0014434369 |            f
    18 |     3600 |    200 |  0.0017924985 |            f
    19 |     3800 |    200 |  0.0020773360 |            f
    20 |     4000 |    200 |  0.0021984108 |            f
    21 |     4200 |    200 |  0.0022855866 |            f
    22 |     4400 |    200 |  0.0024667144 |            f
    23 |     4600 |    200 |  0.0027248130 |        ideal
    24 |     4800 |    200 |  0.0010185664 |            f
    25 |     5000 |    200 |  0.0012098361 |            f
    26 |     5200 |    200 |  0.0014922058 |            f
    27 |     5400 |    200 |  0.0616817970 |        nadir
    28 |     5600 |    200 |  0.0005739704 |            f
    29 |     5800 |    200 |  0.0008859201 |            f
    30 |     6000 |    200 |  0.0012630944 |            f
    31 |     6200 |    200 |  0.0016250452 |            f
```

32	6400	200	0.0017435239	f
33	6600	200	0.0018781713	f
34	6800	200	0.0020403752	f
35	7000	200	0.0021949866	f
36	7200	200	0.0021579430	f
37	7400	200	0.0023719035	f
38	7600	200	0.0024793777	f
39	7800	200	0.0024438316	f
40	8000	200	0.0025453579	f
41	8200	200	0.0006518537	f
42	8400	200	0.0011084829	f
43	8600	200	0.0013903594	f
44	8800	200	0.0020027985	f
45	9000	200	0.0021865842	f
46	9200	200	0.1499866316	nadir
47	9400	200	0.0005469937	f
48	9600	200	0.0012034132	f
49	9800	200	0.0013799873	f
50	10000	200	0.1748723706	nadir
51	10200	200	0.0007089263	f
52	10400	200	0.0012279786	f
53	10600	200	0.0014311530	f
54	10800	200	0.0018282268	f
55	11000	200	0.0019844647	f
56	11200	200	0.0021402982	f
57	11400	200	0.0028467112	f
58	11600	200	0.0005277270	f
59	11800	200	0.0010995733	f
60	12000	200	0.0015174655	f
61	12200	200	0.0018576030	f
62	12400	200	0.0021367447	f
63	12600	200	0.0024039789	f
64	12800	200	0.0025045061	f
65	13000	200	0.0005676409	f
66	13200	200	0.0011288650	f
67	13400	200	0.0013698472	f
68	13600	200	0.0016698545	f
69	13800	200	0.0020714754	f
70	14000	200	0.0020013490	f
71	14200	200	0.0020073841	f
72	14400	200	0.0021773045	f
73	14600	200	0.0025943011	f
74	14800	200	0.0006942584	f
75	15000	200	0.0012709383	f
76	15200	200	0.0020286453	f
77	15400	200	0.0022007383	f
78	15600	200	0.0018613290	f
79	15800	200	0.0027663059	f

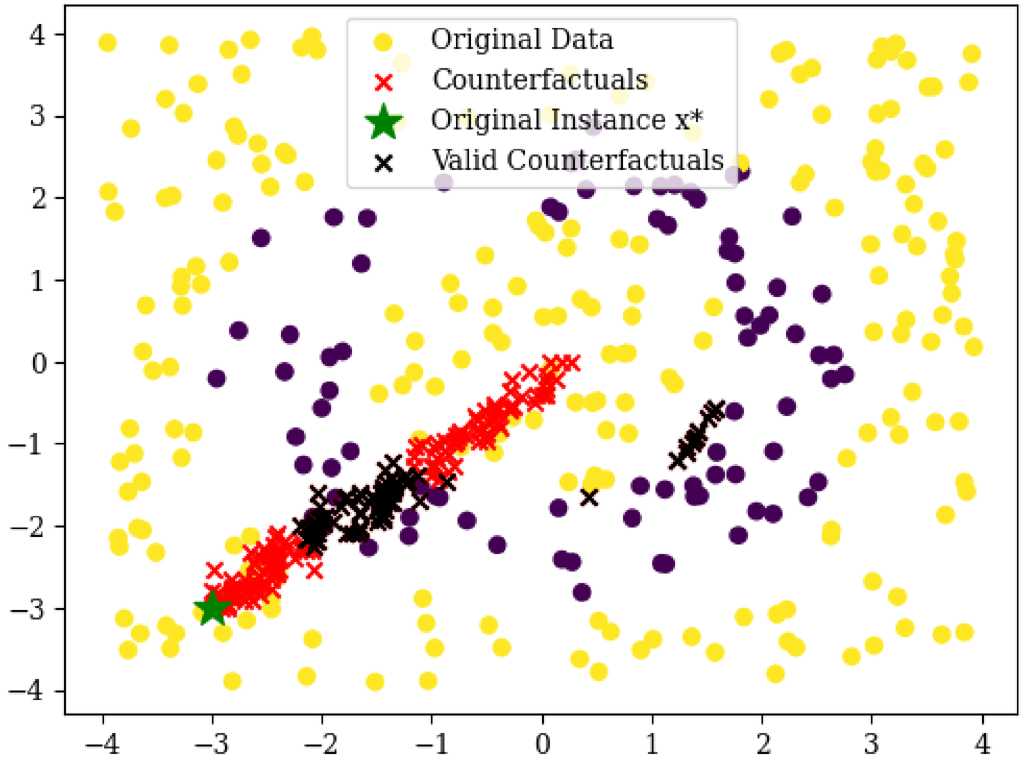
80	16000	200	0.0007024758	f
81	16200	200	0.0012642045	f
82	16400	200	0.0015793193	f
83	16600	200	0.0020380773	f
84	16800	200	0.0019854444	f
85	17000	200	0.0024173950	f
86	17200	200	0.0023310783	f
87	17400	200	0.0024011083	f
88	17600	200	0.0023468347	f
89	17800	200	0.0024014122	f
90	18000	200	0.0025506875	f
91	18200	200	0.0006558593	f
92	18400	200	0.0012192895	f
93	18600	200	0.0016064577	f
94	18800	200	0.0015203262	f
95	19000	200	0.0016269841	f
96	19200	200	0.0018887636	f
97	19400	200	0.0019557499	f
98	19600	200	0.0021626475	f
99	19800	200	0.0021709597	f
100	20000	200	0.0024227786	f
101	20200	200	0.0024543355	f
102	20400	200	0.0024662093	f
103	20600	200	0.0025881959	f
104	20800	200	0.0006869396	f
105	21000	200	0.0010990010	f
106	21200	200	0.0014809536	f
107	21400	200	0.0016570267	f
108	21600	200	0.0018601334	f
109	21800	200	0.0020965129	f
110	22000	200	0.0020427781	f
111	22200	200	0.0022907762	f
112	22400	200	0.0024092551	f
113	22600	200	0.0025829932	f
114	22800	200	0.0007899281	f
115	23000	200	0.0013524006	f
116	23200	200	0.0017630601	f
117	23400	200	0.0021071434	f
118	23600	200	0.0021589078	f
119	23800	200	0.0023120683	f
120	24000	200	0.0025575761	f
121	24200	200	0.0007593238	f
122	24400	200	0.0016386373	f
123	24600	200	0.0018383481	f
124	24800	200	0.0022710158	f
125	25000	200	0.0025229971	f
126	25200	200	0.0005995640	f
127	25400	200	0.0011227219	f

```
   128 |     25600 |      200 |  0.0013707302 |             f
   129 |     25800 |      200 |  0.0017976421 |             f
   130 |     26000 |      200 |  0.0020612335 |             f
   131 |     26200 |      200 |  0.0022007110 |             f
   132 |     26400 |      200 |  0.0021280011 |             f
   133 |     26600 |      200 |  0.0023599920 |             f
   134 |     26800 |      200 |  0.0023482060 |             f
   135 |     27000 |      200 |  0.0022551189 |             f
   136 |     27200 |      200 |  0.0022732140 |             f
   137 |     27400 |      200 |  0.0023231268 |             f
   138 |     27600 |      200 |  0.0025053903 |             f
   139 |     27800 |      200 |  0.0006328135 |             f
   140 |     28000 |      200 |  0.0011412854 |             f
   141 |     28200 |      200 |  0.0015436657 |             f
   142 |     28400 |      200 |  0.0020454520 |             f
   143 |     28600 |      200 |  0.0018502364 |             f
   144 |     28800 |      200 |  0.0021361437 |             f
   145 |     29000 |      200 |  0.0020407645 |             f
   146 |     29200 |      200 |  0.0024657630 |             f
   147 |     29400 |      200 |  0.0031008839 |             f
   148 |     29600 |      200 |  0.0007825100 |             f
   149 |     29800 |      200 |  0.0017450532 |             f
   150 |     30000 |      200 |  0.0020029516 |             f
```

[109]:
```
valided_F_mmo=np.array([f for f in F_mmo if f[0]==0])
valided_X_mmo=np.array([X_mmo[i] for i in range(len(F_mmo)) if F_mmo[i][0]==0])
not_valided_F_mmo=np.array([f for f in F_mmo if f[0]!=0])
not_valided_X_mmo=np.array([X_mmo[i] for i in range(len(F_mmo)) if F_mmo[i][0]!=0])
not_patero_front_X=np.array([X_obs[i] for i in range(len(X_obs)) if i not in res.opt.get("idx")])  # observed instances not in pareto front
```

[110]:
```
# visualize the conterfactuals candidates in original data space
plt.figure()
plt.scatter(X[:,0],X[:,1],c=Y, label='Original Data')
plt.scatter(X_mmo[:,0], X_mmo[:,1], marker='x', c='red', label='Counterfactuals')
plt.scatter(x_star[0], x_star[1], marker='*', c='green', s=200, label='Original Instance x*')
# color black for valid counterfactuals
plt.scatter(valided_X_mmo[:,0], valided_X_mmo[:,1], marker='x', c='black', label='Valid Counterfactuals')
plt.legend()
```

[110]:
```
<matplotlib.legend.Legend at 0x204943d4310>
```

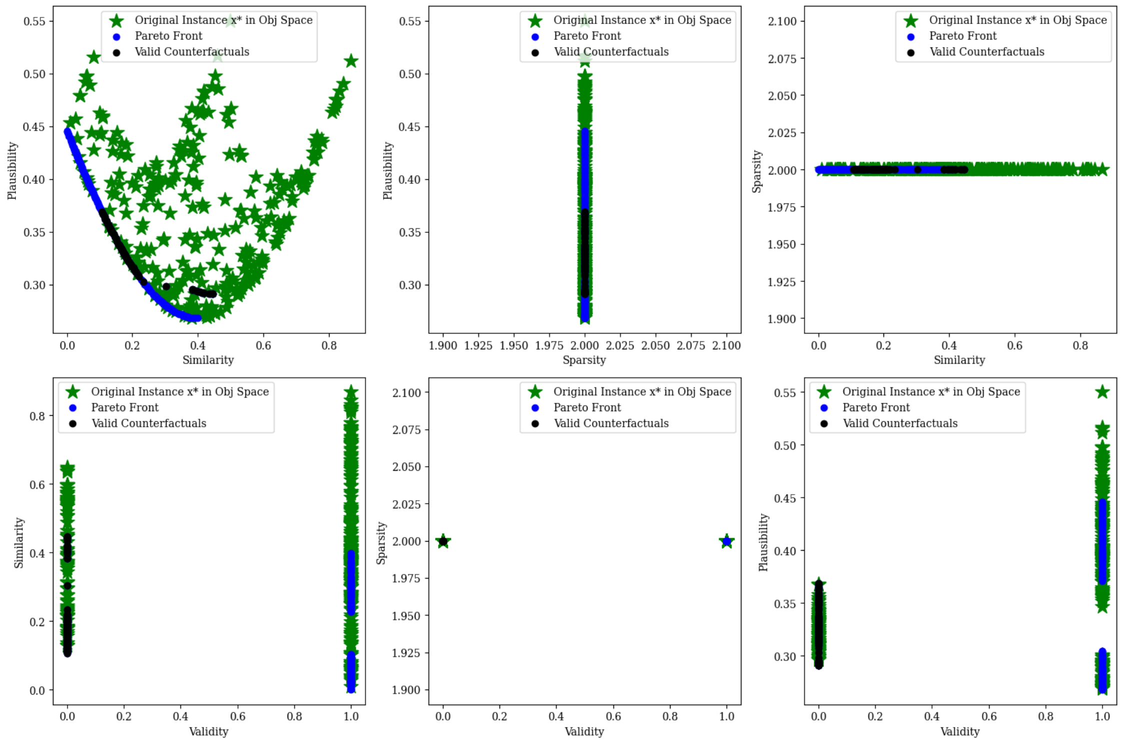


[111]:
```
# visualize the Pareto front in objective space
fig, ax = plt.subplots(2, 3, figsize=(15, 10))

# Create pairs of objective indices to plot
obj_pairs = [(1, 3), (2, 3), (1, 2), (0, 1), (0, 2), (0, 3)]
obj_labels = ['Validity', 'Similarity', 'Sparsity', 'Plausibility']

for i, (obj_x, obj_y) in enumerate(obj_pairs):
    row = i // 3
    col = i % 3
    ax[row, col].scatter(problem.evaluate(X)[:, obj_x],
                         problem.evaluate(X)[:, obj_y],
                         marker='*', c='green', s=200, label='Original Instance␣
↪x* in Obj Space')
    ax[row, col].scatter(F_mmo[:, obj_x], F_mmo[:, obj_y], c='blue',␣
↪label='Pareto Front')
    ax[row, col].scatter(valided_F_mmo[:, obj_x], valided_F_mmo[:, obj_y],␣
↪c='black', label='Valid Counterfactuals')
    ax[row, col].set_xlabel(obj_labels[obj_x])
    ax[row, col].set_ylabel(obj_labels[obj_y])
    ax[row, col].legend()
```

```
plt.tight_layout()
plt.show()
```

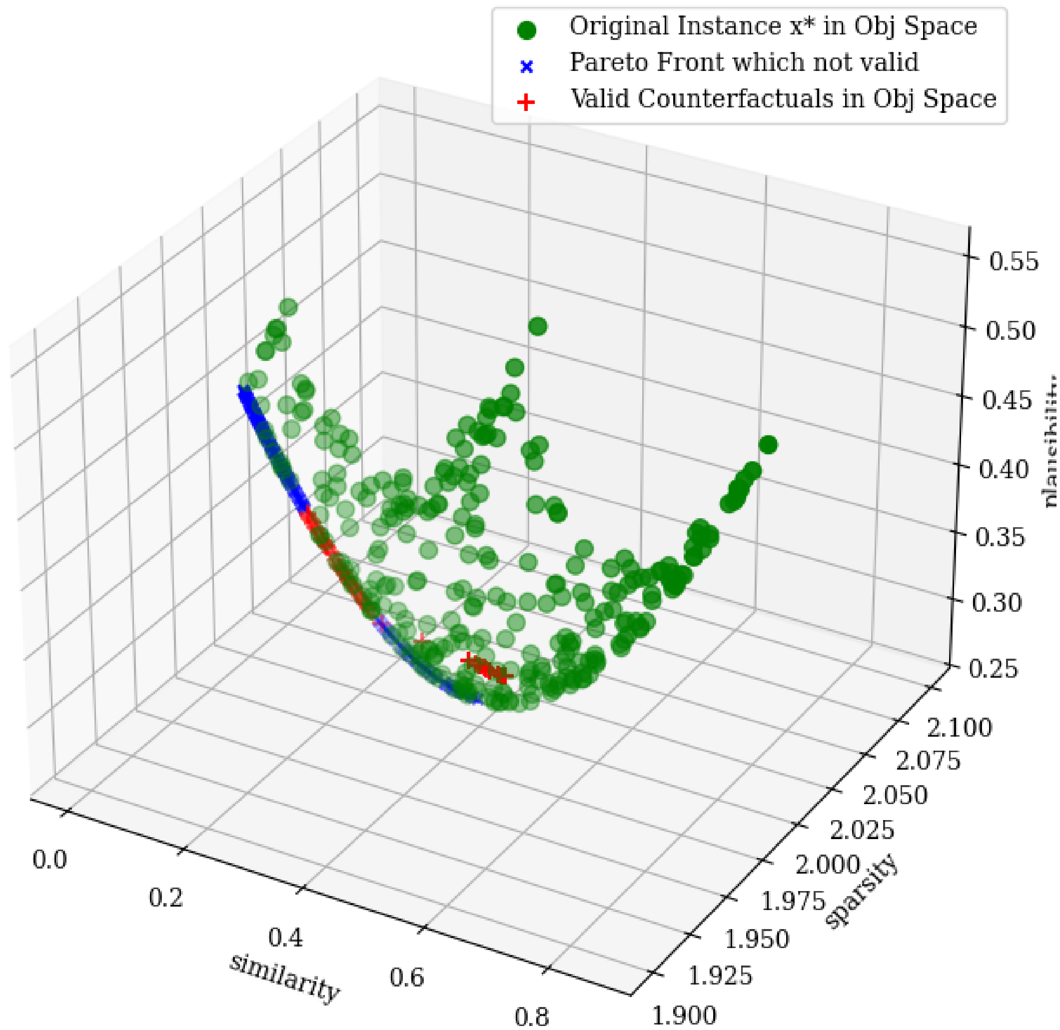


[114]:

```
# visualize the Pareto front in 3D objective space
plt.figure(figsize=(10, 8))
from mpl_toolkits.mplot3d import Axes3D
ax = plt.axes(projection='3d')

ax.scatter(problem.evaluate(X)[:,1],
            problem.evaluate(X)[:,2],
            problem.evaluate(X)[:,3],
            marker='.', c='green', s=200, label='Original Instance x* in Obj␣
 ↪Space')
ax.scatter(not_valided_F_mmo[:,1], not_valided_F_mmo[:,2], not_valided_F_mmo[:
 ↪,3], c='blue', label='Pareto Front which not valid', marker='x', s=20)
# color black for valid counterfactuals
ax.scatter(valided_F_mmo[:,1], valided_F_mmo[:,2], valided_F_mmo[:,3], c='red',␣
 ↪label='Valid Counterfactuals in Obj Space', marker='+', s=50)
ax.set_xlabel('similarity')
ax.set_ylabel('sparsity')
ax.set_zlabel('plausibility')
```

```
plt.legend()
plt.show()
```



[12]:
```
poi=np.array([[0,0]])
problem.evaluate(poi), model(torch.tensor(poi).float().unsqueeze(0)).
 ↪argmax(dim=1), Y_prime
```

[12]:
```
(array([[0.        , 0.20342529, 2.        , 0.26868186]]),
 tensor([[0, 0]]),
 tensor(1))
```