nd_mmo

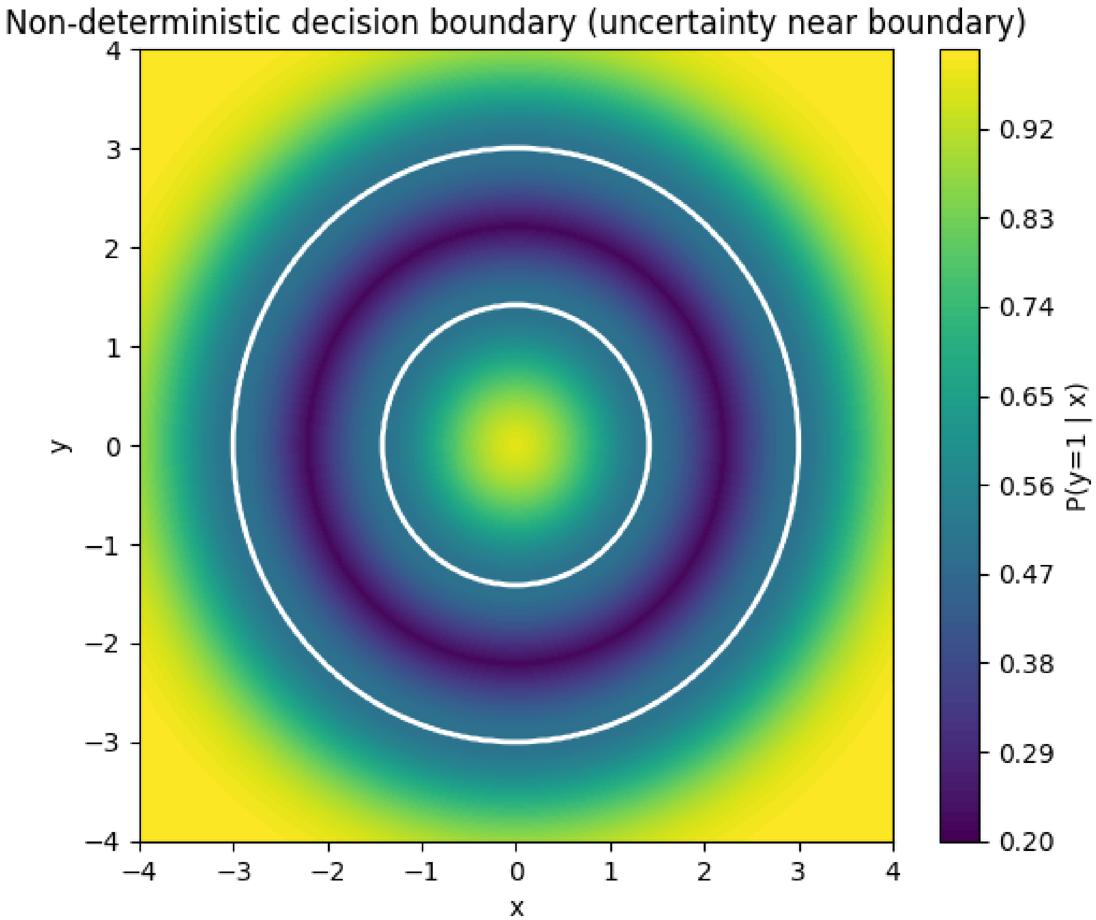
November 23, 2025

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader, random_split

from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
from pymoo.termination import get_termination
from pymoo.problems.functional import FunctionalProblem

from tqdm import tqdm


from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.inspection import DecisionBoundaryDisplay
```

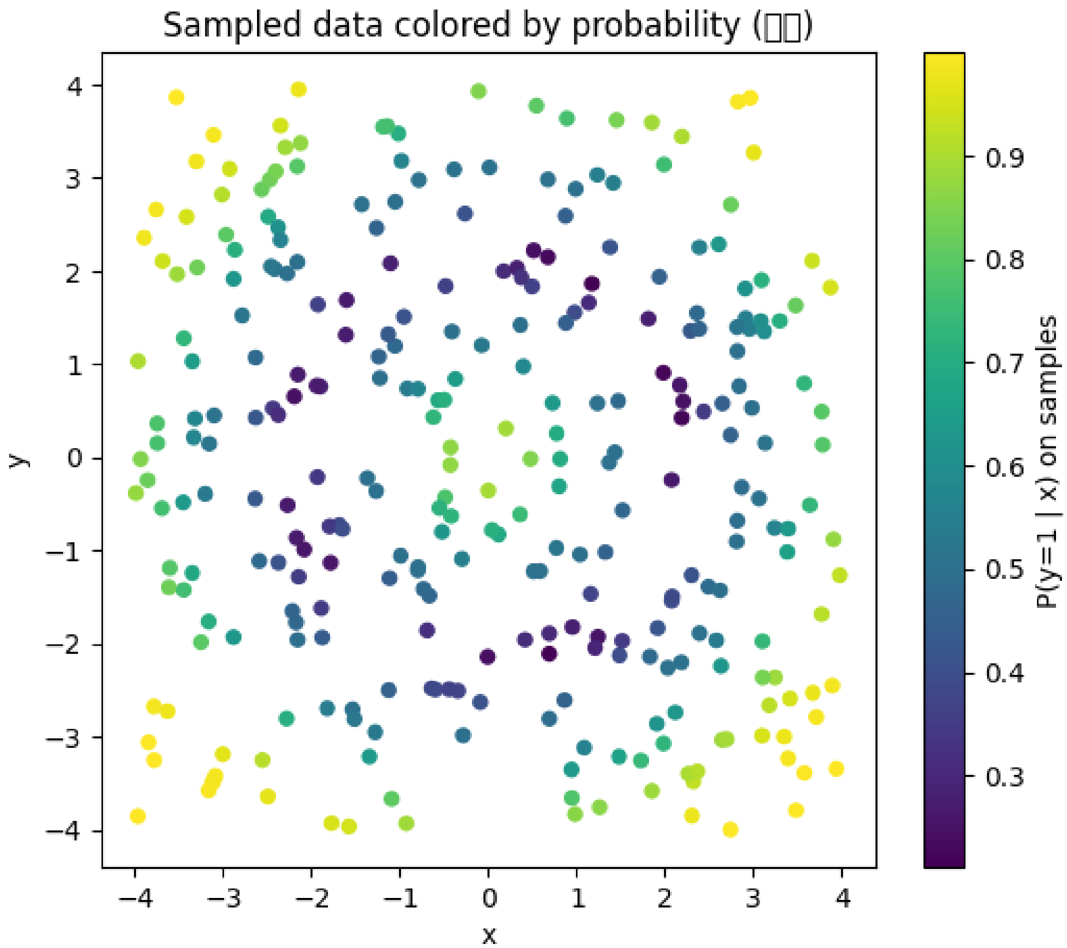
[70]:
```python
import numpy as np
import matplotlib.pyplot as plt



# -------------------------------------------
# 1) Boundary-focused probabilistic boundary
# -------------------------------------------
def decision_prob_boundary_focus(
    x, y,
    r_inner=np.sqrt(2), r_outer=3.0,
    sigma=0.85,            # width of uncertain band around boundary
    max_uncertainty=1.0    # 1.0 -> p=0.5 exactly on boundary
):
    """
    Non-deterministic P(y=1|x) with uncertainty emphasized near boundaries.

    Hard rule:
```

```
        y=1 if r<=r_inner or r>=r_outer   (inside inner disk OR outside outer␣
↪disk)
        y=0 otherwise (annulus)
    Then we add boundary uncertainty:
        closer to boundary -> p moves toward 0.5
        far away -> p -> hard label (0 or 1)
    """
    r = np.sqrt(x**2 + y**2)

    # hard/deterministic label
    base = 1.0 if (r <= r_inner or r >= r_outer) else 0.0

    # distance to nearest boundary
    dist_to_boundary = min(abs(r - r_inner), abs(r - r_outer))

    # uncertainty weight peaked at boundary (Gaussian bump)
    alpha = max_uncertainty * np.exp(-(dist_to_boundary / sigma)**2)

    # blend base toward 0.5 near boundary
    p = base * (1 - alpha) + 0.5 * alpha
    return float(np.clip(p, 0.0, 1.0))


def generate_data(n=300, prob_fn=decision_prob_boundary_focus, xlim=(-4, 4),␣
↪rng=None):
    """
    Sample X uniformly, then sample Y ~ Bernoulli(p).
    Returns X, Y, p1 (probability for class 1).
    """
    rng = np.random.default_rng() if rng is None else rng
    X = rng.uniform(xlim[0], xlim[1], size=(n, 2))
    p1 = np.array([prob_fn(x, y) for x, y in X])
    Y = rng.binomial(1, p1).astype(np.int64)
    return X, Y, p1


# -------------------------------------------
# 2) Gradient visualization of P(y=1|x)
# -------------------------------------------
def plot_probabilistic_boundary(prob_fn, xlim=(-4, 4), grid_n=450):
    xs = np.linspace(xlim[0], xlim[1], grid_n)
    ys = np.linspace(xlim[0], xlim[1], grid_n)
    xx, yy = np.meshgrid(xs, ys)

    # evaluate probability field
    pp = np.vectorize(prob_fn)(xx, yy)
```

```python
    plt.figure(figsize=(6.5, 5.5))
    #   color map showing probability
    cf = plt.contourf(xx, yy, pp, levels=80, cmap="viridis")
    plt.colorbar(cf, label="P(y=1 | x)")

    # emphasize the "closing decision boundary area" by drawing p=0.5 contour
    plt.contour(xx, yy, pp, levels=[0.5], colors="white", linewidths=2)

    plt.title("Non-deterministic decision boundary (uncertainty near boundary)")
    plt.xlabel("x"); plt.ylabel("y")
    plt.show()


# ---- run / demo ----
plot_probabilistic_boundary(decision_prob_boundary_focus)

X, Y, p1 = generate_data(300)

plt.figure(figsize=(6.5, 5.5))
plt.scatter(X[:, 0], X[:, 1], c=p1, s=25, cmap="viridis")
plt.colorbar(label="P(y=1 | x) on samples")
plt.title("Sampled data colored by probability ( )")
plt.xlabel("x"); plt.ylabel("y")
plt.show()
```



Non-deterministic decision boundary (uncertainty near boundary)
y
x
4
3
2
1
0
−1
−2
−3
−4
0.92
0.83
0.74
0.65
0.56
0.47
0.38
0.29
0.20
P(y=1 | x)



[71]:
```
# ----------------------------
# 2) Train classifier
# ----------------------------
device = "cuda" if torch.cuda.is_available() else "cpu"

dataset = TensorDataset(torch.tensor(X, dtype=torch.float32),
                        torch.tensor(Y, dtype=torch.long))

train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_ds, val_ds = random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_ds, batch_size=32, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=256, shuffle=False)


class SimpleNN(nn.Module):
```

```python
    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().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

[72]:
```python
# training loop
num_epochs = 100

bar = tqdm(range(num_epochs), desc="Training", ncols=80,␣
 ↪unit="epoch",colour="blue")
for epoch in bar:
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    bar.set_postfix({'Train Loss': epoch_loss})

# Validation
model.eval()
val_loss = 0.0
correct = 0
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
```

```
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        val_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()

val_loss /= len(val_loader.dataset)
val_accuracy = correct / len(val_loader.dataset)

bar.set_postfix({'Train Loss': epoch_loss, 'Val Loss': val_loss, 'Val Acc':␣
 ↪val_accuracy})
```

```
Training: 100%|      | 100/100 [00:02<00:00, 36.89epoch/s, Train
Loss=0.481]
```

[73]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn.inspection import DecisionBoundaryDisplay

def plot_torch_proba_boundary(
    torch_model,
    X,
    y=None,
    class_of_interest="auto",   # "auto" | None/"max" | int | "all"
    grid_resolution=300,
    padding=0.8,
    levels=100,
    cmap_single="Blues",
    scatter_kwargs=None,
    ax=None,
    device=None,
    already_prob=False,
    multiclass_cmap="tab10",    # <--- NEW: background+points share this
):
    X = np.asarray(X)
    assert X.shape[1] == 2, "X must be shape (N,2)"

    est = TorchProbaEstimator(
        torch_model,
        device=device,
        already_prob=already_prob
    )
    est.fit(X, y)
    n_classes = len(est.classes_)

    # --- Build the exact class color palette used for background regions ---
```

```python
    base_cmap = plt.cm.get_cmap(multiclass_cmap, n_classes)
    multiclass_colors = base_cmap(np.arange(n_classes))  # (n_classes,4 RGBA)

    if scatter_kwargs is None:
        scatter_kwargs = dict(
            s=28, marker="o",
            linewidths=0.9, edgecolor="k",
            alpha=0.9
        )

    if class_of_interest == "auto":
        class_of_interest = 1 if n_classes == 2 else None

    y_arr = None if y is None else np.asarray(y)

    # ---- multi-class: each class prob subplot ----
    if class_of_interest == "all" and n_classes > 2:
        if ax is None:
            fig, axes = plt.subplots(
                1, n_classes, figsize=(4.2*n_classes, 3.6),
                constrained_layout=True
            )
        else:
            axes = ax
            fig = axes[0].figure

        for k in range(n_classes):
            disp = DecisionBoundaryDisplay.from_estimator(
                est,
                X,
                response_method="predict_proba",
                class_of_interest=k,
                plot_method="contourf",
                levels=levels,
                vmin=0, vmax=1,
                cmap=cmap_single,
                ax=axes[k],
                grid_resolution=grid_resolution,
                eps=padding,
            )
            axes[k].set_title(f"Class {k} probability")
            axes[k].set(xticks=(), yticks=())

            # points colored by TRUE class using the SAME class palette
            if y_arr is not None:
                for kk in range(n_classes):
                    mask = y_arr == est.classes_[kk]
```

```
                axes[k].scatter(
                    X[mask, 0], X[mask, 1],
                    color=multiclass_colors[kk],
                    **scatter_kwargs
                )

        fig.colorbar(
            disp.surface_, ax=axes, orientation="horizontal",
            fraction=0.05, pad=0.08, label="Probability"
        )
        return fig, axes

    # ---- single plot ----
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=(5.2, 4.6))
    else:
        fig = ax.figure

    # ---- max-class regions (multi-class background) ----
    if class_of_interest is None or class_of_interest == "max":
        disp = DecisionBoundaryDisplay.from_estimator(
            est,
            X,
            response_method="predict_proba",
            class_of_interest=None,
            plot_method="contourf",
            levels=levels,
            vmin=0, vmax=1,
            ax=ax,
            grid_resolution=grid_resolution,
            eps=padding,
            multiclass_colors=multiclass_colors,  # <--- KEY LINE
        )
        ax.set_title("Max-probability class regions")
        ax.set(xticks=(), yticks=())

        # points colored by TRUE class using EXACT SAME background colors
        if y_arr is not None:
            for k in range(n_classes):
                mask = y_arr == est.classes_[k]
                ax.scatter(
                    X[mask, 0], X[mask, 1],
                    color=multiclass_colors[k],
                    **scatter_kwargs
                )

        # sklearn-like per-class mini colorbars
```

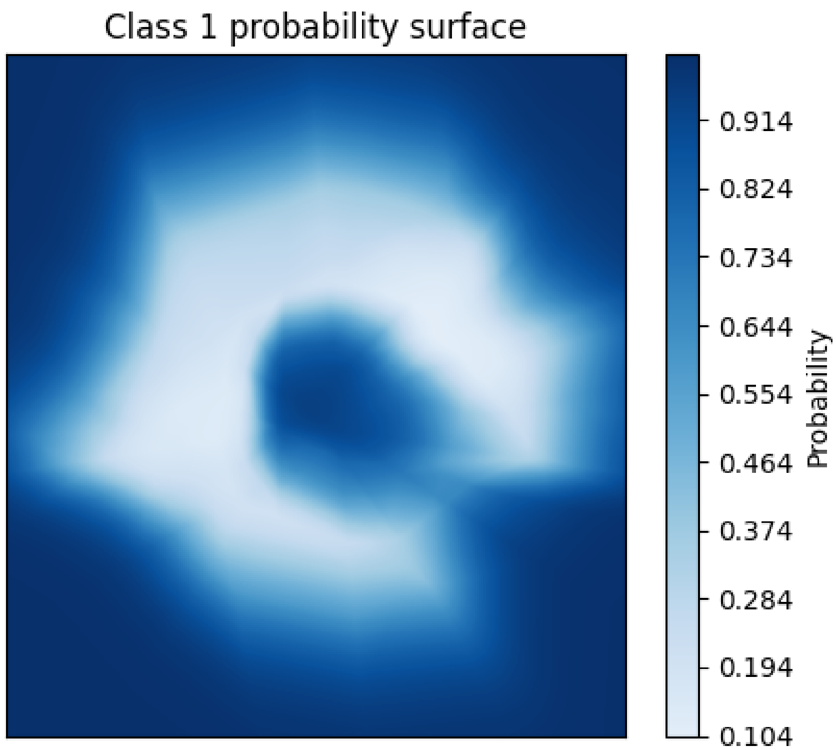
```
        max_cmaps = [s.cmap for s in disp.surface_]
        for k in range(n_classes):
            cax = fig.add_axes([0.78, 0.12 + k*0.06, 0.18, 0.04])
            fig.colorbar(
                cm.ScalarMappable(norm=None, cmap=max_cmaps[k]),
                cax=cax, orientation="horizontal"
            )
            cax.set_title(f"P(class {k})", fontsize=9)

        return fig, ax

    # ---- probability surface for a specific class (blue heatmap) ----
    k = int(class_of_interest)
    disp = DecisionBoundaryDisplay.from_estimator(
        est,
        X,
        response_method="predict_proba",
        class_of_interest=k,
        plot_method="contourf",
        levels=levels,
        vmin=0, vmax=1,
        cmap=cmap_single,
        ax=ax,
        grid_resolution=grid_resolution,
        eps=padding,
    )
    ax.set_title(f"Class {k} probability surface")
    ax.set(xticks=(), yticks=())

    # still color points by TRUE class using the SAME palette as max-mode
    if y_arr is not None:
        for kk in range(n_classes):
            mask = y_arr == est.classes_[kk]
            ax.scatter(
                X[mask, 0], X[mask, 1],
                color=multiclass_colors[kk],
                **scatter_kwargs
            )

    fig.colorbar(disp.surface_, ax=ax, label="Probability")
    return fig, ax
```

[74]:
```
# visual

plot_torch_proba_boundary(model, X,levels=500)
```

```
C:\Users\yuanl\AppData\Local\Temp\ipykernel_26632\3152812757.py:33:
MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
```

```
3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  base_cmap = plt.cm.get_cmap(multiclass_cmap, n_classes)
```

[74]:
```
(<Figure size 520x460 with 2 Axes>,
 <Axes: title={'center': 'Class 1 probability surface'}>)
```



[114]:
```
# ------------------------------
# 3) Multi-objective CF problem
# ------------------------------
def make_cf_problem(model, x_star, Y_prime, X_obs, weights):
    model.eval()

    x_star_np = x_star.detach().cpu().numpy()
    X_obs_np = X_obs.detach().cpu().numpy()
    w_np = weights.detach().cpu().numpy()
    w_np = w_np / (w_np.sum() + 1e-12)

    p = x_star_np.shape[0]

    xl = X_obs_np.min(axis=0)
    xu = X_obs_np.max(axis=0)
    feature_range = xu - xl
```

```
    feature_range[feature_range == 0] = 1.0

    target_labels = Y_prime.view(-1).long().tolist()

    def delta_G_vec(x, y):
        return np.minimum(np.abs(x - y) / feature_range, 1.0)

    # 1) Validity: distance to target in probability space
    def o1_validity(x):
        x_t = torch.from_numpy(x.astype(np.float32)).unsqueeze(0).to(device)
        with torch.no_grad():
            probs = torch.softmax(model(x_t), dim=1).squeeze(0).cpu().numpy()
        p_target = probs[target_labels].max()
        return float(1.0 - p_target)

    # 2) Similarity
    def o2_similarity(x):
        return float(delta_G_vec(x, x_star_np).mean())

    # 3) Sparsity (L0)
    EPS = 0.005
    def o3_sparsity(x):
        diff=0
        for i in range(len(x_star_np)):
            if abs(x[i] - x_star_np[i]) > EPS:
                diff += 1
        return float(diff)

    # 4) Plausibility
    def o4_plausibility(x):
        diffs = (X_obs_np - x) / feature_range
        per_sample = np.sqrt((diffs**2).mean(axis=1))
        return float((per_sample * w_np).sum())

    return FunctionalProblem(
        n_var=p,
        objs=[o1_validity, o2_similarity, o3_sparsity, o4_plausibility],
        xl=xl, xu=xu,
        elementwise=True
    )



# pick a REAL observed instance as x*
x_star = dataset.tensors[0][0].to(device)
with torch.no_grad():
```

```python
    y_star = model(x_star.unsqueeze(0)).argmax(dim=1)
Y_prime = 1 - y_star  # binary case

X_obs = dataset.tensors[0].to(device)
weights = torch.ones(len(X_obs), device=device) / len(X_obs)

problem = make_cf_problem(model, x_star, Y_prime, X_obs, weights)

algorithm = NSGA2(pop_size=200)
termination = get_termination("n_gen", 150)

# no seed => nondeterministic search
res = minimize(problem, algorithm, termination, verbose=True)

F_mmo, X_mmo = res.F, res.X
```

```
==========================================================
n_gen  |  n_eval  | n_nds  |      eps      |   indicator
==========================================================
     1 |      200 |     40 |             - |             -
     2 |      400 |     60 |  0.0074513698 |         ideal
     3 |      600 |     94 |  0.0513458059 |         nadir
     4 |      800 |    150 |  0.0451504967 |         nadir
     5 |     1000 |    200 |  0.0311961036 |         nadir
     6 |     1200 |    200 |  0.0162890534 |         nadir
     7 |     1400 |    200 |  0.0075316799 |         ideal
     8 |     1600 |    200 |  0.0118130456 |         nadir
     9 |     1800 |    200 |  0.0055258211 |         nadir
    10 |     2000 |    200 |  0.0034698238 |             f
    11 |     2200 |    200 |  0.0065894592 |         nadir
    12 |     2400 |    200 |  0.5000000000 |         ideal
    13 |     2600 |    200 |  0.0108915665 |         nadir
    14 |     2800 |    200 |  0.0034307193 |             f
    15 |     3000 |    200 |  0.0066605826 |             f
    16 |     3200 |    200 |  0.0169677274 |         nadir
    17 |     3400 |    200 |  0.0077844034 |         nadir
    18 |     3600 |    200 |  0.0043747161 |             f
    19 |     3800 |    200 |  0.0046671153 |             f
    20 |     4000 |    200 |  0.0078454757 |         nadir
    21 |     4200 |    200 |  0.0048124385 |             f
    22 |     4400 |    200 |  0.0033250719 |             f
    23 |     4600 |    200 |  0.0052012007 |             f
    24 |     4800 |    200 |  0.0047890662 |             f
    25 |     5000 |    200 |  0.0049239703 |             f
    26 |     5200 |    200 |  0.0028575481 |         nadir
    27 |     5400 |    200 |  0.0071292891 |         nadir
    28 |     5600 |    200 |  0.0052347535 |             f
    29 |     5800 |    200 |  0.0033653114 |             f
```

30	6000	200	0.0037342471	f
31	6200	200	0.0049235177	nadir
32	6400	200	0.0047452207	f
33	6600	200	0.0053395533	f
34	6800	200	0.0120562279	nadir
35	7000	200	0.0034236133	f
36	7200	200	0.0039912565	f
37	7400	200	0.0048216101	f
38	7600	200	0.0031445556	f
39	7800	200	0.0126888583	nadir
40	8000	200	0.0039170790	f
41	8200	200	0.0044806463	f
42	8400	200	0.0036271695	f
43	8600	200	0.0032980388	f
44	8800	200	0.0045974664	f
45	9000	200	0.0128973900	nadir
46	9200	200	0.0128586833	nadir
47	9400	200	0.0065780469	f
48	9600	200	0.0041116870	f
49	9800	200	0.0053098939	f
50	10000	200	0.0048270585	f
51	10200	200	0.0035304258	f
52	10400	200	0.0046016040	nadir
53	10600	200	0.0046228766	nadir
54	10800	200	0.0048943296	f
55	11000	200	0.0048947933	f
56	11200	200	0.0036914294	f
57	11400	200	0.0033226543	f
58	11600	200	0.0037516508	f
59	11800	200	0.0040519822	f
60	12000	200	0.0052028189	f
61	12200	200	0.0029925912	f
62	12400	200	0.0032377956	f
63	12600	200	0.0036506532	f
64	12800	200	0.0044730447	f
65	13000	200	0.0042951742	f
66	13200	200	0.0043840924	f
67	13400	200	0.0038819299	f
68	13600	200	0.0037165446	f
69	13800	200	0.0049938553	f
70	14000	200	0.0031738040	f
71	14200	200	0.0047548071	f
72	14400	200	0.0044246227	f
73	14600	200	0.0035528159	f
74	14800	200	0.0048461517	f
75	15000	200	0.0034655119	f
76	15200	200	0.0023737388	f
77	15400	200	0.0051952969	f

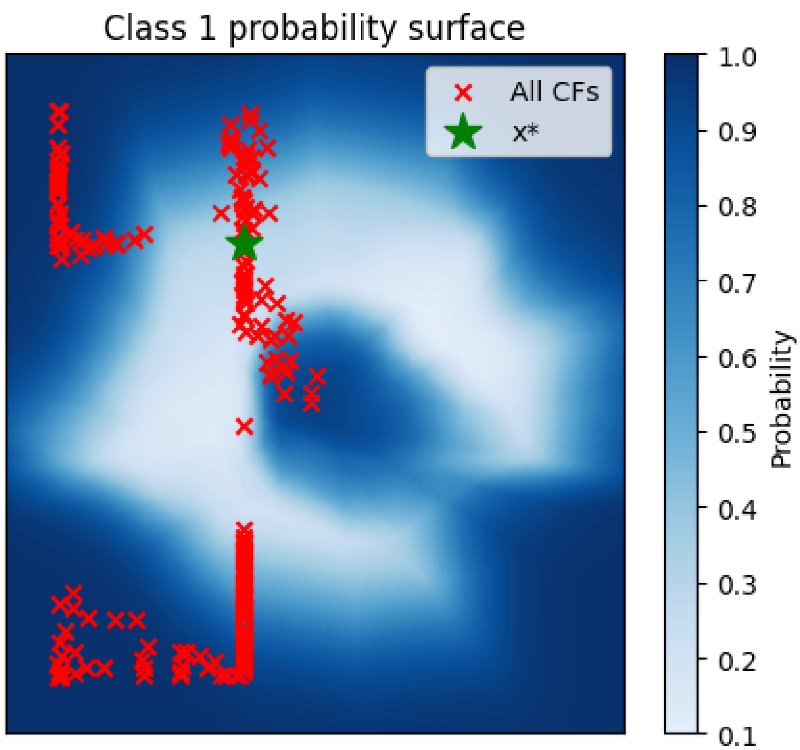
78	15600	200	0.0029768615	f
79	15800	200	0.0036218609	f
80	16000	200	0.0027665731	f
81	16200	200	0.0040260739	f
82	16400	200	0.0044229637	f
83	16600	200	0.0036679947	f
84	16800	200	0.0041204997	f
85	17000	200	0.0047870921	f
86	17200	200	0.0031880090	f
87	17400	200	0.0043575809	f
88	17600	200	0.0045959817	f
89	17800	200	0.0045771937	f
90	18000	200	0.0040913895	f
91	18200	200	0.0044173212	f
92	18400	200	0.0049210327	f
93	18600	200	0.0035746081	f
94	18800	200	0.0031885218	f
95	19000	200	0.0047175058	f
96	19200	200	0.0038728401	f
97	19400	200	0.0035890833	f
98	19600	200	0.0061153614	f
99	19800	200	0.0036474702	f
100	20000	200	0.0040699498	f
101	20200	200	0.0038524079	f
102	20400	200	0.0034879370	f
103	20600	200	0.0049641155	f
104	20800	200	0.0041284756	f
105	21000	200	0.0031381285	f
106	21200	200	0.0054959466	f
107	21400	200	0.0038063753	f
108	21600	200	0.0028959520	f
109	21800	200	0.0043060078	f
110	22000	200	0.0026290482	f
111	22200	200	0.0038648687	f
112	22400	200	0.0034890358	f
113	22600	200	0.0040045428	f
114	22800	200	0.0045793384	f
115	23000	200	0.0049557637	f
116	23200	200	0.0024040754	f
117	23400	200	0.0083459633	f
118	23600	200	0.0043890618	f
119	23800	200	0.0036891827	f
120	24000	200	0.0038066902	f
121	24200	200	0.0052014247	f
122	24400	200	0.0063508177	f
123	24600	200	0.0044058182	f
124	24800	200	0.0034870874	f
125	25000	200	0.0044639002	f

```
   126 |     25200 |     200 |  0.0043051905 |              f
   127 |     25400 |     200 |  0.0038890054 |              f
   128 |     25600 |     200 |  0.0045799123 |              f
   129 |     25800 |     200 |  0.0038052106 |              f
   130 |     26000 |     200 |  0.0042351763 |              f
   131 |     26200 |     200 |  0.0055448765 |              f
   132 |     26400 |     200 |  0.0037815549 |              f
   133 |     26600 |     200 |  0.0036317236 |              f
   134 |     26800 |     200 |  0.0042728158 |              f
   135 |     27000 |     200 |  0.0034277282 |              f
   136 |     27200 |     200 |  0.0040730193 |              f
   137 |     27400 |     200 |  0.0040445394 |              f
   138 |     27600 |     200 |  0.0043772804 |              f
   139 |     27800 |     200 |  0.0039593161 |              f
   140 |     28000 |     200 |  0.0041312695 |              f
   141 |     28200 |     200 |  0.0040233655 |              f
   142 |     28400 |     200 |  0.0046377608 |              f
   143 |     28600 |     200 |  0.0055924499 |              f
   144 |     28800 |     200 |  0.0048658302 |              f
   145 |     29000 |     200 |  0.0033648750 |              f
   146 |     29200 |     200 |  0.0038273163 |              f
   147 |     29400 |     200 |  0.0037634887 |              f
   148 |     29600 |     200 |  0.0071680760 |          nadir
   149 |     29800 |     200 |  0.0033554744 |              f
   150 |     30000 |     200 |  0.0034740292 |              f
```

[115]:
```
# -----------------------------
# 4) Post-processing & plots
# -----------------------------
# With prob-based validity, use threshold instead of ==0.
# valid_mask = F_mmo[:, 0] < 0.05  # e.g. at least 95% target prob
# valid_F = F_mmo[valid_mask]
# valid_X = X_mmo[valid_mask]

# plt.figure()
# plt.scatter(X[:, 0], X[:, 1], c=Y, label="Original data", s=15,␣
↪cmap="coolwarm")
# plt.scatter(X_mmo[:, 0], X_mmo[:, 1], c="red", marker="x", label="All CFs")
# plt.scatter(valid_X[:, 0], valid_X[:, 1], c="black", marker="x", label="Valid␣
↪CFs")
# plt.scatter(x_star[0].item(), x_star[1].item(),
#             c="green", marker="*", s=200, label="x*")
# plt.legend()
# plt.show()


fig,ax=plot_torch_proba_boundary(model, X)
```

```
ax.scatter(X_mmo[:, 0], X_mmo[:, 1], c="red", marker="x", label="All CFs")
ax.scatter(x_star[0].item(), x_star[1].item(),
           c="green", marker="*", s=200, label="x*")
ax.legend()
plt.show()
```

```
C:\Users\yuanl\AppData\Local\Temp\ipykernel_26632\3152812757.py:33:
MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in
3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()``
or ``pyplot.get_cmap()`` instead.
```

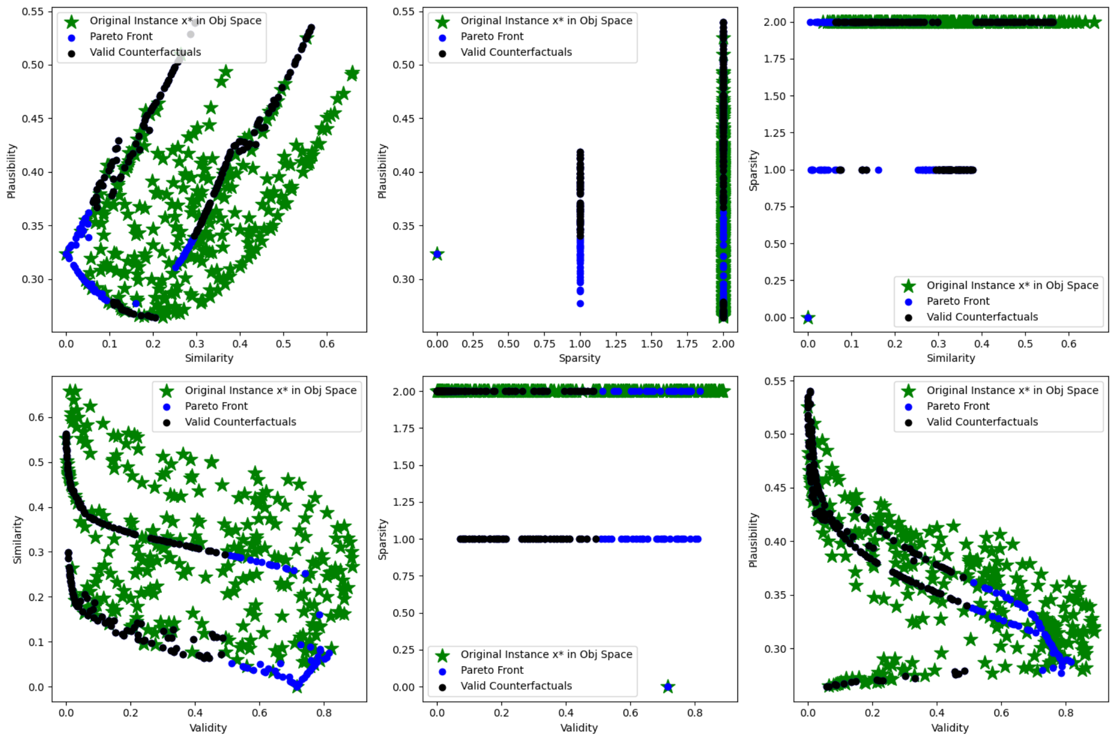


[116]:
```
valided_F_mmo=np.array([f for f in F_mmo if f[0]<=0.5])
valided_X_mmo=np.array([X_mmo[i] for i in range(len(F_mmo)) if F_mmo[i][0]<=0.
↪5])
not_valided_F_mmo=np.array([f for f in F_mmo if f[0]>=0.5])
not_valided_X_mmo=np.array([X_mmo[i] for i in range(len(F_mmo)) if␣
↪F_mmo[i][0]>=0.5])
```

```
valided_F_mmo.shape,valided_X_mmo.shape,not_valided_F_mmo.
↪shape,not_valided_X_mmo.shape
```

[116]: ((144, 4), (144, 2), (56, 4), (56, 2))

[117]:
```
# 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()
```



[118]:
```
import plotly.graph_objects as go

F_X = problem.evaluate(X)

fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=F_X[:,1], y=F_X[:,2], z=F_X[:,3],
    mode='markers',
    marker=dict(color='green', size=6, opacity=0.2),
    name='Original Instance x* in Obj Space'
))

fig.add_trace(go.Scatter3d(
    x=not_valided_F_mmo[:,1], y=not_valided_F_mmo[:,2], z=not_valided_F_mmo[:
↪,3],
    mode='markers',
    marker=dict(color='blue', size=3, opacity=0.2, symbol='x'),
    name='Pareto Front which not valid'
))

fig.add_trace(go.Scatter3d(
    x=valided_F_mmo[:,1], y=valided_F_mmo[:,2], z=valided_F_mmo[:,3],
```

```python
    mode='markers',
    marker=dict(color='red', size=5, symbol='cross'),
    name='Valid Counterfactuals in Obj Space'
))

fig.update_layout(
    scene=dict(
        xaxis_title='similarity/AU',
        yaxis_title='sparsity',
        zaxis_title='plausibility'
    ),
    width=900, height=700
)

fig.show()

#       html
fig.write_html("pareto_front_3d.html")
```

[ ]:

```python

```