Solution 1: Cost Curves

(a) We can simply retrieve the FPRs and TPRs from the table (or matrix) and plot the curves.

```
import numpy as np
import matplotlib.pyplot as plt

# The first column is FPR, the second column is TPR
FPR_TPR_1 = np.array(
    [[0.0, 0.00],
     [0.1, 0.60],
     [0.2, 0.75],
     [0.3, 0.825],
     [0.4, 0.85],
     [0.5, 0.875],
     [0.6, 0.90],
     [0.7, 0.925],
     [0.8, 0.950],
     [0.9, 0.975],
     [1.0, 1.0]])

FPR_TPR_2 = np.array(
    [[0.0, 0.00],
     [0.1, 0.2],
     [0.2, 0.4],
     [0.3, 0.6],
     [0.4, 0.8],
     [0.5, 0.925],
     [0.6, 0.96],
     [0.7, 0.98],
     [0.8, 0.99],
     [0.9, 0.995],
     [1.0, 1.00]]
)


def draw_roc_curves(fpr_tpr_1: np.ndarray, fpr_tpr_2: np.ndarray) -> None:
    fig, ax = plt.subplots(1, 1, figsize=(8, 7))
    ax.plot(fpr_tpr_1[:, 0], fpr_tpr_1[:, 1], marker=’o’, label=’Classifier 1’)
    ax.plot(fpr_tpr_2[:, 0], fpr_tpr_2[:, 1], marker=’o’, label=’Classifier 2’)

    ax.set_xlabel(’False Positive Rate’)
    ax.set_ylabel(’True Positive Rate’)
    ax.grid(’on’)

    plt.legend()
    plt.savefig(’roc_curves.pdf’, bbox_inches=’tight’)
```

The ROC curves are shown in Figure 1

(b) The cost is computed as

$$\rho_{MCE} = (FNR - FPR) \cdot \pi_{+} + FPR$$

For each (FPR, TPR) point in the ROC space, we draw a line of (π_{+}, ρ_{MCE}).

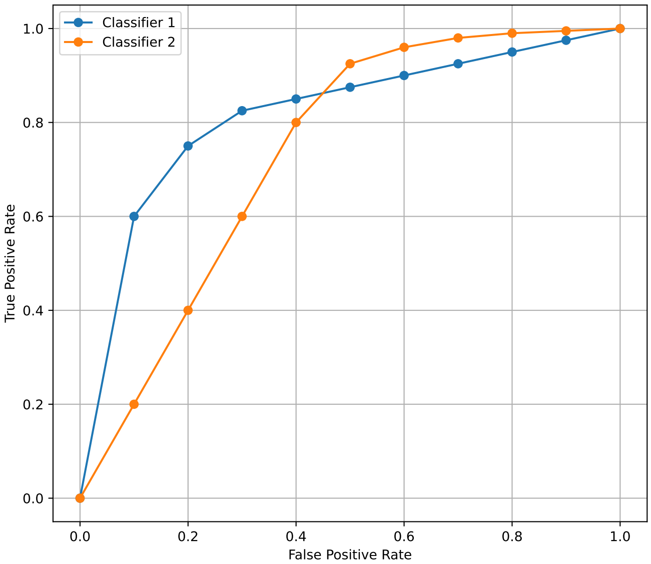


Figure 1: ROC Curves.

```
def draw_cost_curves(fpr_tpr_1: np.ndarray, fpr_tpr_2: np.ndarray) -> None:
    fnr_1 = 1 - fpr_tpr_1[:, 1]
    fnr_2 = 1 - fpr_tpr_2[:, 1]
    # Probability of positive
    # Assume P = 50 points in [0.0, 1.0]
    pos_probs = np.linspace(0.0, 1.0, 50)

    # Compute the coefficient (FNR - FPR)
    # Shape after expansion: (N, 1), where N is the number of points in ROC space.
    fnr_minus_fpr_1 = np.expand_dims(fnr_1 - fpr_tpr_1[:, 0], 1)
    fnr_minus_fpr_2 = np.expand_dims(fnr_2 - fpr_tpr_2[:, 0], 1)

    # Costs shape: (N, P)
    costs_1 = fnr_minus_fpr_1 * np.expand_dims(pos_probs, 0) + np.expand_dims(
   fpr_tpr_1[:, 0], 1)
    costs_2 = fnr_minus_fpr_2 * np.expand_dims(pos_probs, 0) + np.expand_dims(
   fpr_tpr_2[:, 0], 1)
    # Point-wise minimum across cost lines.
    cost_curve_1 = costs_1.min(0)
    cost_curve_2 = costs_2.min(0)

    # Find the cross point of two cost curves. The following operation assumes that
    # cust_curve_1 is lower than cost_curve_2 in the beginning of the curves.
    cross_point_ind = np.flatnonzero(cost_curve_1 > cost_curve_2)[0]

    fig, ax = plt.subplots(1, 1, figsize=(8, 7))
    ax.plot(pos_probs, cost_curve_1, label='Classifier 1')
    ax.plot(pos_probs, cost_curve_2, label='Classifier_2')
    ax.vlines(
        pos_probs[cross_point_ind],
        ymin=0.0,
        ymax=cost_curve_1[cross_point_ind],
        linestyles='dashed',
        colors=['k'],
    )
    ax.set_xlabel('Probability of Positive')
    ax.set_ylabel('Error Rate')

    plt.legend()
    plt.savefig('cost_curves.pdf', bbox_inches='tight')
```

```
if __name__ == '__main__':
    draw_roc_curves(FPR_TPR_1, FPR_TPR_2)
    draw_cost_curves(FPR_TPR_1, FPR_TPR_2)
```

Then, we obtain Figure 2.

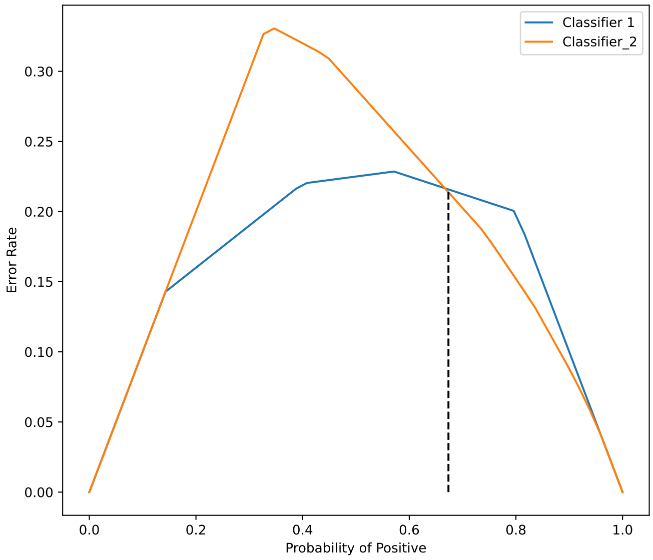


Figure 2: Cost Curves.

Solution 2: Tomek Links

(a)

```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import make_classification
from sklearn.metrics import pairwise_distances


def find_tomek_links(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    """Find Tomek Links in samples.

    Args:
        x: Data samples with shape (num_samples, num_features).
        y: Binary class labels with shape (num_samples,).
           1 means positive class, 0 means negative class.

    Returns:
        An array with shape (num_samples, ) with binary values,
        for which 1 means that the corresponding sample
        belongs to a Tomek link, while 0 means that the
        sample does not belong to any Tomek link.
    """

    num_samples = x.shape[0]
    dist = pairwise_distances(x)

    # Compute the min. pairwise distance between samples. Note that the distance
    # to each sample itself is 0 and trivial, so this case should be excluded when
    # computing the pairwise distance. To this end, we employ masked arrays,
    # which can be constructed by applying binary masks to the original array.
    # In the masks, 1 means excluding the corresponding samples when performing
    # operations e.g. min, argmin, etc.

    mask = np.eye(num_samples, dtype=bool)
    masked_dist = np.ma.array(dist, mask=mask)
    nearest_j_to_i = masked_dist.argmin(axis=1)
    # Check if the nearest neighbor of j is also i.
    # If yes, they are mutually nearest.
    nearest_i_to_j = np.zeros_like(nearest_j_to_i)
    nearest_i_to_j[nearest_j_to_i] = np.arange(num_samples)

    is_mutually_closest = nearest_i_to_j == nearest_j_to_i
    # Tomek links also requires the pair to have different labels
    is_diff_y = y != y[nearest_j_to_i]
    is_tomek_sample = np.logical_and(is_mutually_closest, is_diff_y)

    return is_tomek_sample
```

In this function, nearest_i_to_j and nearest_j_to_i are used to determine if i-th sample and j-th sample are mutually closest. Figure 3 shows an example of the process.

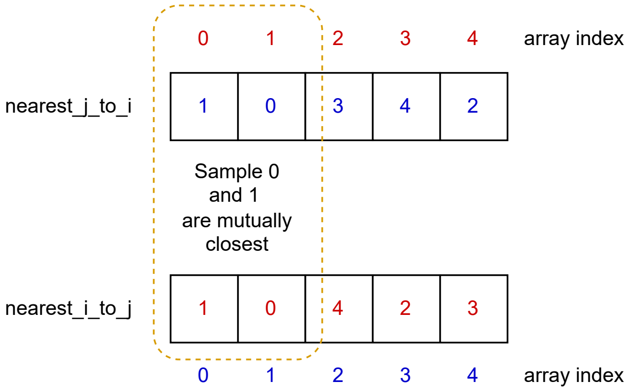


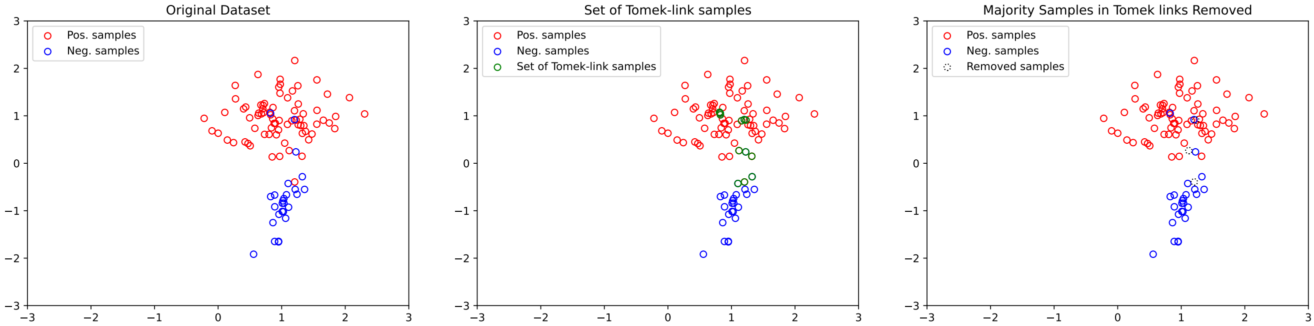
Figure 3: Determining the Mutually Closest Pairs. The first two elements are consistent in both arrays. This is interpreted as that the 0-th sample and the 1-th sample are mutually closest.

(b)
```
def find_kept_samples(
        x: np.ndarray,
        y: np.ndarray,
        is_tomek_sample: np.ndarray
) -> np.ndarray:
    """Given the binary array indicating which sample belongs to a Tomek link. Find
     out which sample should be kept. The sample should be removed if it belongs to
     a Tomek link and it is majority class.

    Args:
        x: Data samples with shape (num_samples, num_features).
        y: Binary class labels with values 0 or 1.
        is_tomek_sample: A binary array with shape (num_samples,),
            in which 1 means that the sample belongs to a Tomek link.

    Returns:
        A binary array with shape (num_samples,), in which 1 means that the
        corresponding sample should be kept, otherwise it should be removed.
    """
    num_samples = x.shape[0]
    to_be_kept = np.ones((num_samples, ), dtype=bool)

    # Define which class is majority
    num_pos_samples = y.sum()
    num_neg_samples = num_samples - num_pos_samples
    if num_pos_samples >= num_neg_samples:
        y_major = 1
    else:
        y_major = 0

    # Only remove the sample in a Tomek link if it is majority class
    is_major_tomek_sample = np.logical_and(is_tomek_sample, y == y_major)
    to_be_kept[is_major_tomek_sample] = False

    return to_be_kept
```

(c)

```
def run_experiment() -> None:
    # Generate data
    random_state = np.random.RandomState(11)
    x, y = make_classification(
        100,
        n_features=2,
        n_redundant=0,
        n_clusters_per_class=1,
        flip_y=0.02,
        class_sep=1.0,
        weights=[0.25, 0.75],
        random_state=random_state)

    # Plot the original dataset
    fig, ax = plt.subplots(1, 1)
    x_pos = x[y == 1]
    x_neg = x[y == 0]
    ax.scatter(x_pos[:, 0], x_pos[:, 1], facecolors='none', edgecolors='r', label='
   Pos. samples')
    ax.scatter(x_neg[:, 0], x_neg[:, 1], facecolors='none', edgecolors='b', label='
   Neg. samples')
    ax.set_title('Original Dataset')
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)
    plt.legend()
    plt.savefig('original_dataset.pdf', bbox_inches='tight')
    plt.clf()

    # Find out which samples belong to Tomek links.
    is_tomek_link = find_tomek_links(x, y)

    fig, ax = plt.subplots(1, 1)
    x_tomek = x[is_tomek_link]
    ax.scatter(x_pos[:, 0], x_pos[:, 1], facecolors='none', edgecolors='r', label='
   Pos. samples')
    ax.scatter(x_neg[:, 0], x_neg[:, 1], facecolors='none', edgecolors='b', label='
   Neg. samples')
    ax.scatter(
        x_tomek[:, 0],
        x_tomek[:, 1],
        facecolors='none',
        edgecolors='g',
        label='Set of Tomek-link samples')
    ax.set_title('Set of Tomek-link samples')
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)
    plt.legend()
    plt.savefig('tomek_links.pdf', bbox_inches='tight')
    plt.clf()

    # Identify which samples should be kept and which should be removed
    to_be_kept = find_kept_samples(x, y, is_tomek_link)
    fig, ax = plt.subplots(1, 1)
    x_removed = x[~to_be_kept]
    x, y = x[to_be_kept], y[to_be_kept]
    x_pos, x_neg = x[y == 1], x[y == 0]
    ax.scatter(x_pos[:, 0], x_pos[:, 1], facecolors='none', edgecolors='r', label='Pos. samples')
    ax.scatter(x_neg[:, 0], x_neg[:, 1], facecolors='none', edgecolors='b', label='Neg. samples')
    ax.scatter(
        x_removed[:, 0],
        x_removed[:, 1],
        facecolors='none',
        edgecolor='k',
        linestyle='dotted',
        label='Removed samples')
    ax.set_title('Majority Samples in Tomek links Removed')
    ax.set_xlim(-3, 3)
    ax.set_ylim(-3, 3)
    plt.legend()
    plt.savefig('subsampled_dataset.pdf', bbox_inches='tight')
    plt.clf()


if __name__ == '__main__':
    run_experiment()
```

After running the script, we obain the following figures:



(a) Original Dataset.

(b) Tomek Links.

(c) Subsampled Dataset.