```python
import random as rnd
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats


class SimpleBayesClassifier:

    def __init__(self, n_pos, n_neg):
        """
        Initializes the SimpleBayesClassifier with prior probabilities.

        Parameters:
        n_pos (int): The number of positive samples.
        n_neg (int): The number of negative samples.

        Returns:
        None: This method does not return anything as it is a constructor.
        """

        self.n_pos = n_pos
        self.n_neg = n_neg
        self.prior_pos = n_pos / (n_pos + n_neg)
        self.prior_neg = n_neg / (n_pos + n_neg)

    def fit_params(self, x, y, n_bins=10):
        """
        Computes histogram-based parameters for each feature in the dataset.

        Parameters:
        x (np.ndarray): The feature matrix, where rows are samples and columns are features.
        y (np.ndarray): The target array, where each element corresponds to the label of a sample.
        n_bins (int): Number of bins to use for histogram calculation.

        Returns:
        (stay_params, leave_params): A tuple containing two lists of tuples,
        one for 'stay' parameters and one for 'leave' parameters.
        Each tuple in the list contains the bins and edges of the histogram for a feature.
        """

        self.stay_params = []
        self.leave_params = []

        # INSERT CODE HERE
        for i in range(x.shape[1]):  # for each attributes in x_train
            stay = x[y == 0, i]
            stay = stay[~np.isnan(stay)]
            Shist, Sedges = np.histogram(stay, n_bins)
            Sedges[0] = -np.inf
            Sedges[-1] = np.inf
            Shist = Shist / np.sum(Shist)
            self.stay_params.append((Shist, Sedges))

            leave = x[y == 1, i]
            leave = leave[~np.isnan(leave)]
            Lhist, Ledges = np.histogram(leave, n_bins)
            Ledges[0] = -np.inf
            Ledges[-1] = np.inf
            Lhist = Lhist / np.sum(Lhist)
            self.leave_params.append((Lhist, Ledges))

        return self.stay_params, self.leave_params

    def predict(self, x, thresh=0):
        """
        Predicts the class labels for the given samples using the non-parametric model.

        Parameters:
        x (np.ndarray): The feature matrix for which predictions are to be made.
        thresh (float): The threshold for log probability to decide between classes.

        Returns:
        result (list): A list of predicted class labels (0 or 1) for each sample in the feature matrix.
        """

        y_pred = []

        # INSERT CODE HERE
        init = np.log(self.prior_pos) - np.log(self.prior_neg)

        for i in range(x.shape[0]):
            lH = init
            for j in range(x.shape[1]):
                if np.isnan(x[i][j]):
                    continue
                leave_index = (
                    np.searchsorted(self.leave_params[j][1], x[i][j], side="right") - 1
                )
                stay_index = (
                    np.searchsorted(self.stay_params[j][1], x[i][j], side="right") - 1
                )

                leave_value = self.leave_params[j][0][leave_index]
                if leave_value == 0:
                    leave_value += 1e-6
                stay_value = self.stay_params[j][0][stay_index]
                if stay_value == 0:
                    stay_value += 1e-6

                lH += np.log(leave_value) - np.log(stay_value)

            if lH > thresh:
                y_pred.append(1)
            else:
                y_pred.append(0)

        return y_pred

    def fit_gaussian_params(self, x, y):
        """
        Computes mean and standard deviation for each feature in the dataset.

        Parameters:
        x (np.ndarray): The feature matrix, where rows are samples and columns are features.
        y (np.ndarray): The target array, where each element corresponds to the label of a sample.

        Returns:
        (gaussian_stay_params, gaussian_leave_params): A tuple containing two lists of tuples,
        one for 'stay' parameters and one for 'leave' parameters.
        Each tuple in the list contains the mean and standard deviation for a feature.
        """

        self.gaussian_stay_params = [(0, 0) for _ in range(x.shape[1])]
        self.gaussian_leave_params = [(0, 0) for _ in range(x.shape[1])]

        # INSERT CODE HERE
        for i in range(x.shape[1]):  # for each feature: calculate the parameter
            stay = x[y == 0, i]
            stay = stay[~np.isnan(stay)]
            stay_mean = np.mean(stay)
            stay_std = max(np.std(stay), 1e-6)
            self.gaussian_stay_params[i] = (stay_mean, stay_std)

            leave = x[y == 1, i]
            leave = leave[~np.isnan(leave)]
            leave_mean = np.mean(leave)
            leave_std = max(np.std(leave), 1e-6)
            self.gaussian_leave_params[i] = (leave_mean, leave_std)

        return self.gaussian_stay_params, self.gaussian_leave_params

    def gaussian_predict(self, x, thresh=0):
        """
        Predicts the class labels for the given samples using the parametric model.

        Parameters:
        x (np.ndarray): The feature matrix for which predictions are to be made.
        thresh (float): The threshold for log probability to decide between classes.

        Returns:
        result (list): A list of predicted class labels (0 or 1) for each sample in the feature matrix.
        """

        y_pred = []

        # INSERT CODE HERE
        for i in range(x.shape[0]):
            predict = np.log(self.prior_pos) - np.log(self.prior_neg)
            for j in range(x.shape[1]):
                if np.isnan(x[i][j]):
                    continue

                log_leave = max(
                    stats.norm(
                        self.gaussian_leave_params[j][0],
                        self.gaussian_leave_params[j][1],
                    ).pdf(x[i][j]),
                    1e-9,
                )
                log_stay = max(
                    stats.norm(
                        self.gaussian_stay_params[j][0], self.gaussian_stay_params[j][1]
                    ).pdf(x[i][j]),
                    1e-9,
                )

                predict += np.log(log_leave) - np.log(log_stay)

            if predict > thresh:
                y_pred.append(1)
            else:
                y_pred.append(0)

        return np.array(y_pred)

```