SimpleBayesClassifier

January 30, 2026

1 Simple Bayes Classifier

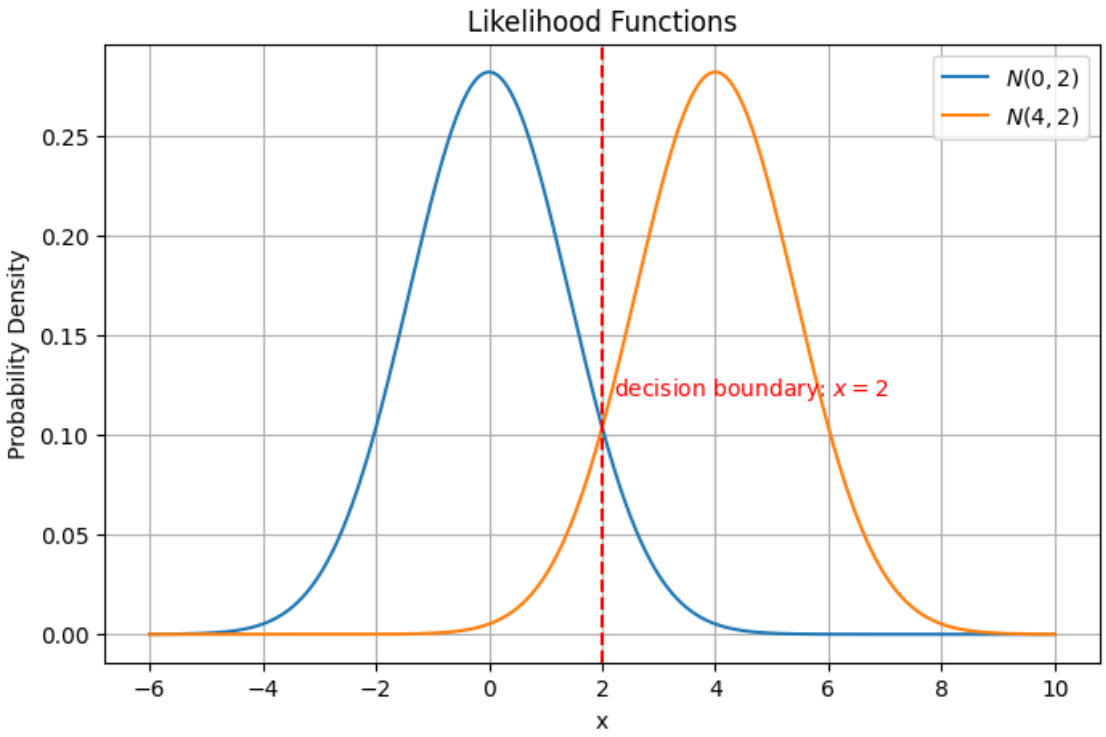
1.1 Import Libraries

[2]:
```
import random as rnd
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from scipy import stats
```

1.2 Problem T2

[15]:
```
# Define x range
x = np.linspace(-6, 10, 1000)

# Parameters
mu1, mu2 = 0, 4
sigma = np.sqrt(2)

# Gaussian PDFs
p_w2 = stats.norm.pdf(x, mu1, sigma)  # N(0, 2)
p_w1 = stats.norm.pdf(x, mu2, sigma)  # N(4, 2)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(x, p_w2, label=r"$N(0, 2)$")
plt.plot(x, p_w1, label=r"$N(4, 2)$")

plt.xlabel("x")
plt.ylabel("Probability Density")
plt.title("Likelihood Functions")
plt.legend()
plt.grid(True)

plt.axvline(x=2, color="red", linestyle="dashed")
```

```
plt.text(2.2, 0.12, r"decision boundary: $x = 2$", color="red")

plt.savefig("../images/p2.png", dpi=300)
plt.show()
```

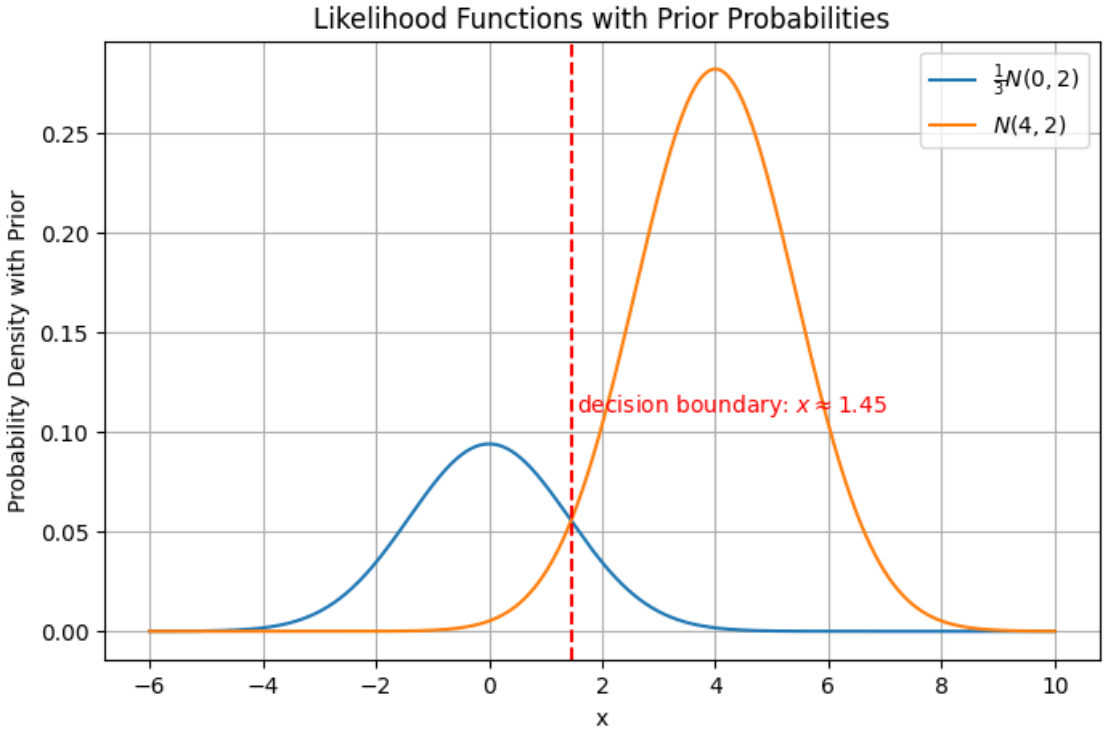


---

2 Problem T3

[19]:
```
# Define x range
x = np.linspace(-6, 10, 1000)

# Parameters
mu1, mu2 = 0, 4
sigma = np.sqrt(2)

# Gaussian PDFs
p_w2 = (1.0 / 3.0) * stats.norm.pdf(x, mu1, sigma)  # 1/3 N(0, 2)
p_w1 = stats.norm.pdf(x, mu2, sigma)                # N(4, 2)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(x, p_w2, label=r"$\frac{1}{3} N(0, 2)$")
```

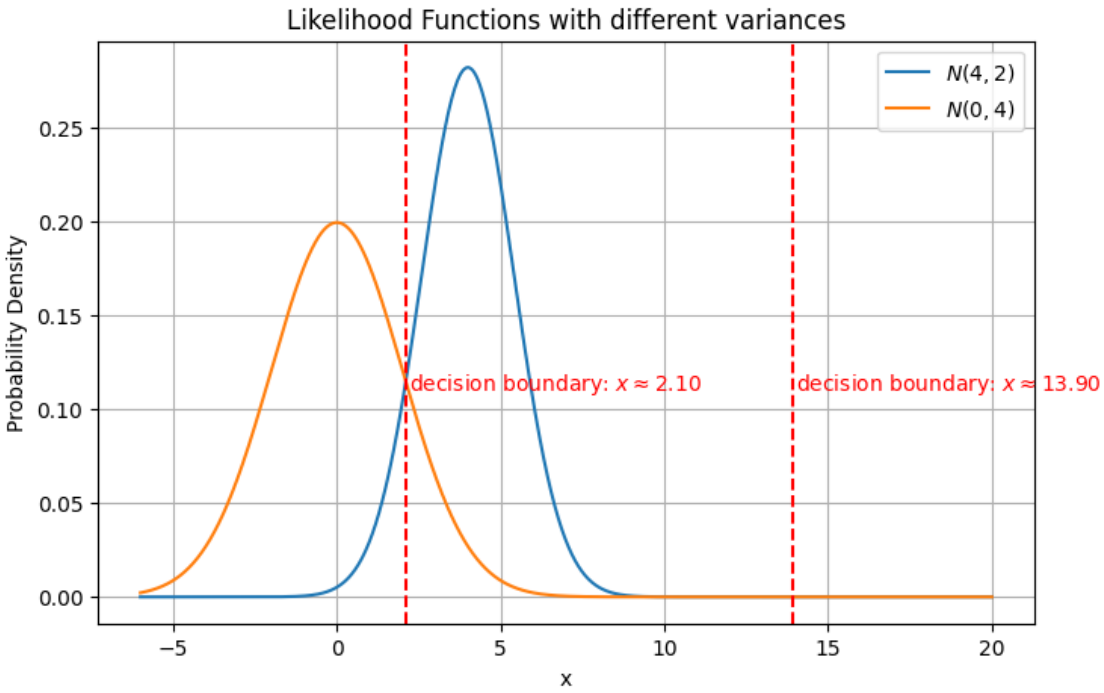
```
plt.plot(x, p_w1, label=r"$N(4, 2)$")

plt.xlabel("x")
plt.ylabel("Probability Density with Prior")
plt.title("Likelihood Functions with Prior Probabilities")
plt.legend()
plt.grid(True)

boundary = 2 - (np.log(3) / 2.0)
plt.axvline(x=boundary, color="red", linestyle="dashed")
plt.text(boundary + 0.1, 0.11, r"decision boundary: $x \approx 1.45$",␣
 ↪color="red")

plt.savefig("../images/p3.png", dpi=300)
plt.show()
```



2.1 Problem OT3

[ ]:
```
# Define x range
x = np.linspace(-6, 20, 1000)
```

```python
# Parameters
mu1, mu2 = 4, 0
sigma1, sigma2 = np.sqrt(2), 2

# Gaussian PDFs
p_w2 = stats.norm.pdf(x, mu1, sigma1)  # N(4, 2)
p_w1 = stats.norm.pdf(x, mu2, sigma2)  # N(0, 4)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(x, p_w2, label=r"$N(4, 2)$")
plt.plot(x, p_w1, label=r"$N(0, 4)$")

plt.xlabel("x")
plt.ylabel("Probability Density")
plt.title("Likelihood Functions with different variances")
plt.legend()
plt.grid(True)

boundary1, boundary2 = 2.10, 13.90

plt.axvline(x=boundary1, color="red", linestyle="dashed")
plt.text(boundary1 + 0.1, 0.11, r"decision boundary #1: $x \approx 2.10$",␣
 ↪color="red")

plt.axvline(x=boundary2, color="red", linestyle="dashed")
plt.text(boundary2 + 0.1, 0.11, r"decision boundary #2: $x \approx 13.90$",␣
 ↪color="red")

plt.savefig("../images/po3.png", dpi=300)
plt.show()
```



2.2 Template class: SimpleBayesClassifier

[ ]:
```
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 = 1
        self.n_neg = 1
        self.prior_pos = 1
        self.prior_neg = 1
```

```
  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 = [(None, None) for _ in range(x.shape[1])]
      self.leave_params = [(None, None) for _ in range(x.shape[1])]

      # INSERT CODE HERE

      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
```

```
        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

        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

        return y_pred
```
