Homework 2

Week 2 - MLE and Naive Bayes

Patthadon Phengpinij
Collaborators. ChatGPT

1 MLE

Consider the following very simple model for stock pricing. The price at the end of each day is the price of the previous day multiplied by a fixed, but unknown, rate of return, α, with some noise, w. For a two-day period, we can observe the following sequence

$$y_2 = \alpha y_1 + w_1$$

$$y_1 = \alpha y_0 + w_0$$

where the noises w_0, w_1 are iid with the distribution $\mathcal{N}(0, \sigma^2)$, $y_0 \sim \mathcal{N}(0, \lambda)$ is independent of the noise sequence. σ^2 and λ are known, while α is unknown.

T1. Find the MLE of the rate of return α given the observed price at the end of each day, y_2, y_1, y_0. In other words, compute for the value of α that maximizes $p(y_2, y_1, y_0 \mid \alpha)$.

Hint: This is a Markov process, e.g. y_2 is independent of y_0 given y_1. In general, a process is Markov if $p(y_n \mid y_{n-1}, y_{n-2}, \ldots) = p(y_n \mid y_{n-1})$. In other words, the present is independent of the past $(y_{n-2}, y_{n-3}, \ldots)$, conditioned on the immediate past y_{n-1}. You may also find the steps of the proof for logistic regression we did in class useful.

Solution. First, we can express the joint probability $p(y_2, y_1, y_0 \mid \alpha)$ using the Markov property:

$$p(y_2, y_1, y_0 \mid \alpha) = p(y_2 \mid y_1, \alpha) \times p(y_1 \mid y_0, \alpha) \times p(y_0)$$

Next, we can write down the conditional probabilities based on the model:

1. The conditional probability $p(y_2 \mid y_1, \alpha)$ is given by the Gaussian distribution:

$$p(y_2 \mid y_1, \alpha) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_2 - \alpha y_1)^2}{2\sigma^2}\right)$$

2. Likewise, the conditional probability $p(y_1 \mid y_0, \alpha)$ is:

$$p(y_1 \mid y_0, \alpha) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_1 - \alpha y_0)^2}{2\sigma^2}\right)$$

3. The prior probability $p(y_0)$ is:

$$p(y_0) = \frac{1}{\sqrt{2\pi\lambda}} \exp\left(-\frac{y_0^2}{2\lambda}\right)$$

Thus, the joint probability becomes:

$$\begin{aligned} p(y_2, y_1, y_0 \mid \alpha) &= p(y_2 \mid y_1, \alpha) \times p(y_1 \mid y_0, \alpha) \times p(y_0) \\ &= \frac{1}{(2\pi\sigma^2)\sqrt{2\pi\lambda}} \exp\left(-\frac{(y_2 - \alpha y_1)^2 + (y_1 - \alpha y_0)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}\right) \end{aligned}$$

We want to find the value of α that maximizes this joint probability. To do this, we can take the logarithm of the joint probability (log-likelihood):

$$\log\left(p(y_2, y_1, y_0 \mid \alpha)\right) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\log(2\pi\lambda) - \frac{(y_2 - \alpha y_1)^2 + (y_1 - \alpha y_0)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}$$

To find the MLE, we take the derivative of the log-likelihood with respect to α and set it to zero:

$$\begin{aligned}
\frac{d}{d\alpha}\left[\log\left(p(y_2, y_1, y_0 \mid \alpha)\right)\right] &= -0 - 0 - \frac{1}{2\sigma^2}\left[-2y_1(y_2 - \alpha y_1) - 2y_0(y_1 - \alpha y_0)\right] - 0 \\
0 &= \frac{1}{\sigma^2}\left[(y_2 - \alpha y_1)y_1 + (y_1 - \alpha y_0)y_0\right] \\
0 &= y_2 y_1 - \alpha y_1^2 + y_1 y_0 - \alpha y_0^2 \\
\alpha(y_1^2 + y_0^2) &= y_2 y_1 + y_1 y_0 \\
\alpha &= \frac{y_2 y_1 + y_1 y_0}{y_1^2 + y_0^2}
\end{aligned}$$

Therefore, the MLE for α is:

$$\boxed{\hat{\alpha} = \frac{y_2 y_1 + y_1 y_0}{y_1^2 + y_0^2}}$$

OT1. Consider the general case, where

$$y_{n+1} = \alpha y_n + w_n,\ n = 0, 1, 2, \ldots$$

Find the MLE given the observed prices $y_{N+1}, y_N, \ldots, y_0$.

Solution. Just like in the two-day case, we can express the joint probability using the Markov property:

$$p(y_{N+1}, y_N, \ldots, y_0 \mid \alpha) = p(y_{N+1} \mid y_N, \alpha) \times p(y_N \mid y_{N-1}, \alpha) \times \ldots \times p(y_1 \mid y_0, \alpha) \times p(y_0)$$

Next, we can write down the conditional probabilities based on the model:

1. The conditional probability $p(y_{n+1} \mid y_n, \alpha)$ is given by the Gaussian distribution:

$$p(y_{n+1} \mid y_n, \alpha) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2}\right)$$

2. The prior probability $p(y_0)$ is:

$$p(y_0) = \frac{1}{\sqrt{2\pi\lambda}} \exp\left(-\frac{y_0^2}{2\lambda}\right)$$

Thus, the joint probability becomes:

$$\begin{aligned}
p(y_{N+1}, y_N, \ldots, y_0 \mid \alpha) &= \prod_{n=0}^{N} p(y_{n+1} \mid y_n, \alpha) \times p(y_0) \\
&= \frac{1}{(2\pi\sigma^2)^{(N+1)/2}\sqrt{2\pi\lambda}} \exp\left(-\sum_{n=0}^{N} \frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}\right)
\end{aligned}$$

We want to find the value of α that maximizes this joint probability. To do this, we can take the logarithm of the joint probability (log-likelihood):

$$\log\left(p(y_{N+1}, y_N, \ldots, y_0 \mid \alpha)\right) = -\frac{N+1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\log(2\pi\lambda) - \sum_{n=0}^{N} \frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}$$

To find the MLE, we take the derivative of the log-likelihood with respect to α and set it to zero:

$$\begin{aligned}
\frac{d}{d\alpha}\left[\log\left(p(y_{N+1}, y_N, \ldots, y_0 \mid \alpha)\right)\right] &= -\sum_{n=0}^{N} \frac{1}{2\sigma^2}\left[-2(y_{n+1} - \alpha y_n)y_n\right] \\
0 &= \sum_{n=0}^{N} \frac{1}{\sigma^2}(y_{n+1} - \alpha y_n)y_n \\
0 &= \sum_{n=0}^{N} y_{n+1}y_n - \alpha \sum_{n=0}^{N} y_n^2 \\
\alpha \sum_{n=0}^{N} y_n^2 &= \sum_{n=0}^{N} y_{n+1}y_n \\
\alpha &= \frac{\sum_{n=0}^{N} y_{n+1}y_n}{\sum_{n=0}^{N} y_n^2}
\end{aligned}$$

Therefore, the MLE for α in the general case is:

$$\boxed{\hat{\alpha} = \frac{\sum_{n=0}^{N} y_{n+1}y_n}{\sum_{n=0}^{N} y_n^2}}$$

2 Simple Bayes Classifier

A student in Pattern Recognition course had finally built the ultimate classifier for cat emotions. He used one input features: the amount of food the cat ate that day, x (Being a good student he already normalized x to standard Normal). He proposed the following likelihood probabilities for class 1 (happy cat) and 2 (sad cat)

$$P(x \mid w_1) = \mathcal{N}(4, 2)$$

$$P(x \mid w_2) = \mathcal{N}(0, 2)$$

Figure 1: The sad cat and the happy cat used in training.

T2. Plot the posteriors values of the two classes on the same axis. Using the likelihood ratio test, what is the decision boundary for this classifier? Assume equal prior probabilities.

Solution. First, we can calculate the posterior probabilities for each class using Bayes' theorem:

$$P(w_1 \mid x) = \frac{P(x \mid w_1)P(w_1)}{P(x)}$$
$$P(w_2 \mid x) = \frac{P(x \mid w_2)P(w_2)}{P(x)}$$

Assume equal prior probabilities:

$$P(w_1) = P(w_2)$$

The likelihood ratio test compares the posterior probabilities:

$$\frac{P(w_1 \mid x)}{P(w_2 \mid x)} = \frac{P(x \mid w_1)P(w_1)}{P(x \mid w_2)P(w_2)}$$
$$\frac{P(w_1 \mid x)}{P(w_2 \mid x)} = \frac{P(x \mid w_1)}{P(x \mid w_2)}$$

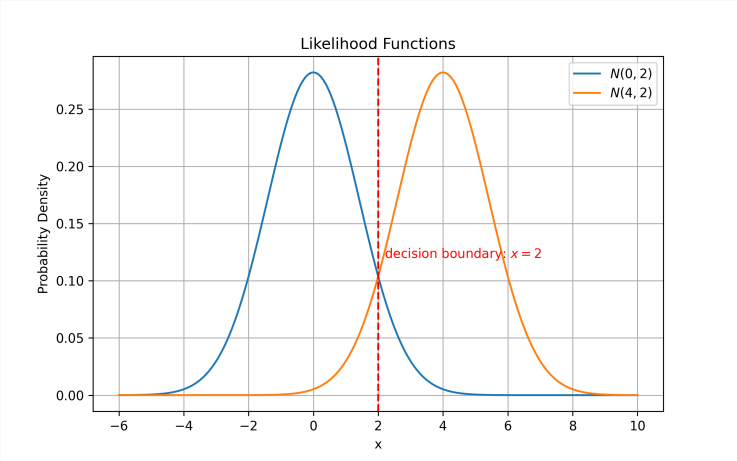
To find the decision boundary, we set the likelihood ratio equal to 1:

$$\frac{P(x \mid w_1)}{P(x \mid w_2)} = 1$$
$$P(x \mid w_1) = P(x \mid w_2)$$

Substituting the Gaussian likelihoods:

$$\frac{1}{\sqrt{2\pi \cdot 2}} \exp\left(-\frac{(x-4)^2}{2 \cdot 2}\right) = \frac{1}{\sqrt{2\pi \cdot 2}} \exp\left(-\frac{x^2}{2 \cdot 2}\right)$$
$$\exp\left(-\frac{(x-4)^2}{4}\right) = \exp\left(-\frac{x^2}{4}\right)$$
$$-\frac{(x-4)^2}{4} = -\frac{x^2}{4}$$
$$(x-4)^2 = x^2$$
$$x = 2$$

Thus, the decision boundary for this classifier is at $\boxed{x = 2}$.
Using Python, we can plot the posterior probabilities for both classes:



T3. What happen to the decision boundary if the cat is happy with a prior of 0.75?

Solution. With the new prior probabilities:

$$P(w_1) = 0.75 \text{ and } P(w_2) = 0.25$$

The likelihood ratio test now becomes:

$$\begin{aligned}
\frac{P(w_1 \mid x)}{P(w_2 \mid x)} &= \frac{P(x \mid w_1)P(w_1)}{P(x \mid w_2)P(w_2)} \\
\frac{P(w_1 \mid x)}{P(w_2 \mid x)} &= \frac{P(x \mid w_1) \cdot (0.75)}{P(x \mid w_2) \cdot (0.25)} \\
\frac{P(w_1 \mid x)}{P(w_2 \mid x)} &= 3\left(\frac{P(x \mid w_1)}{P(x \mid w_2)}\right)
\end{aligned}$$

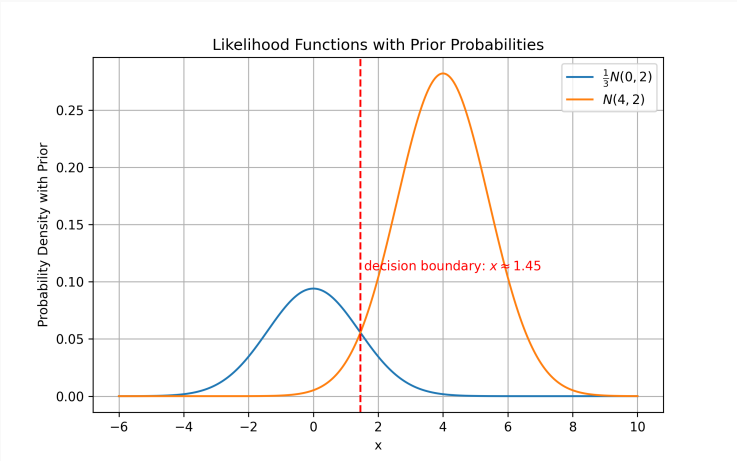
To find the decision boundary, we set the likelihood ratio equal to 1:

$$\begin{aligned}
3\left(\frac{P(x \mid w_1)}{P(x \mid w_2)}\right) &= 1 \\
P(x \mid w_1) &= \frac{1}{3}P(x \mid w_2)
\end{aligned}$$

Substituting the Gaussian likelihoods:

$$\begin{aligned}
\frac{1}{\sqrt{2\pi \cdot 2}} \exp\left(-\frac{(x-4)^2}{2 \cdot 2}\right) &= \frac{1}{3} \times \frac{1}{\sqrt{2\pi \cdot 2}} \exp\left(-\frac{x^2}{2 \cdot 2}\right) \\
\exp\left(-\frac{(x-4)^2}{4}\right) &= \frac{1}{3} \times \exp\left(-\frac{x^2}{4}\right) \\
-\frac{(x-4)^2}{4} &= \ln\left(\frac{1}{3}\right) - \frac{x^2}{4} \\
(x-4)^2 &= x^2 + 4\ln(3) \\
x &= 2 - \frac{\ln(3)}{2} \\
x &\approx 1.45
\end{aligned}$$

Thus, the decision boundary for this classifier is at $\boxed{x \approx 1.45}$.
Using Python, we can plot the posterior probabilities for both classes:



This shows that the decision boundary has shifted to the left due to the increased prior probability of the happy cat class (as shown by the orange line in the figure above).

OT2. For the ordinary case of $P(x \mid w_1) = \mathcal{N}(\mu_1, \sigma^2)$, $P(x \mid w_2) = \mathcal{N}(\mu_2, \sigma^2)$, $p(w_1) = p(w_2) = 0.5$, prove that the decision boundary is at $x = \frac{\mu_1+\mu_2}{2}$.

Proof. For the ordinary case, we have:

$$P(x \mid w_1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)$$

$$P(x \mid w_2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right)$$

Using Bayes' theorem, the posterior probabilities for each class are:

$$P(w_1 \mid x) = \frac{P(x \mid w_1)P(w_1)}{P(x)}$$
$$P(w_2 \mid x) = \frac{P(x \mid w_2)P(w_2)}{P(x)}$$

Because the equality of prior probabilities:

$$P(w_1) = P(w_2) = 0.5$$

The likelihood ratio test compares the posterior probabilities:

$$\frac{P(w_1 \mid x)}{P(w_2 \mid x)} = \frac{P(x \mid w_1)P(w_1)}{P(x \mid w_2)P(w_2)}$$
$$\frac{P(w_1 \mid x)}{P(w_2 \mid x)} = \frac{P(x \mid w_1)}{P(x \mid w_2)}$$

To find the decision boundary, we set the likelihood ratio equal to 1:

$$\frac{P(x \mid w_1)}{P(x \mid w_2)} = 1$$
$$P(x \mid w_1) = P(x \mid w_2)$$

Substituting the Gaussian likelihoods:

$$\begin{aligned}
\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) \\
\exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) &= \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) \\
-\frac{(x-\mu_1)^2}{2\sigma^2} &= -\frac{(x-\mu_2)^2}{2\sigma^2} \\
(x-\mu_1)^2 &= (x-\mu_2)^2 \\
0 &= (x-\mu_2)^2 - (x-\mu_1)^2 \\
0 &= ((x-\mu_2) - (x-\mu_1))\,((x-\mu_2) + (x-\mu_1)) \\
0 &= (\mu_1-\mu_2)\,(2x - (\mu_1+\mu_2)) \\
0 &= (2x - (\mu_1+\mu_2)) \ ; \text{because } \mu_1 \neq \mu_2 \\
x &= \frac{\mu_1+\mu_2}{2}
\end{aligned}$$

Thus, the decision boundary for this classifier is at $\boxed{x = \frac{\mu_1+\mu_2}{2}}$.

OT3. If the student changed his model to

$$P(x \mid w_1) = \mathcal{N}(4, 2)$$

$$P(x \mid w_2) = \mathcal{N}(0, 4)$$

Plot the posteriors values of the two classes on the same axis. What is the decision boundary for this classifier? Assume equal prior probabilities.

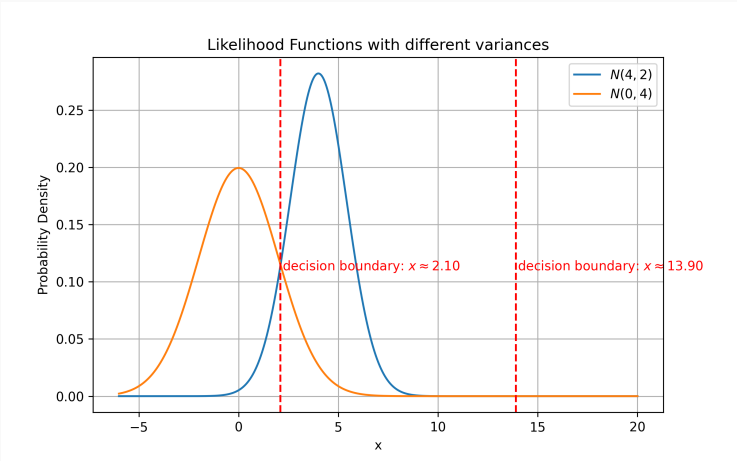
Solution. Like the previous problems, we can calculate the posterior probabilities for each class using Bayes' theorem, assume the equality of prior probabilities, we know that:

$$\frac{P(x \mid w_1)}{P(x \mid w_2)} = 1$$
$$P(x \mid w_1) = P(x \mid w_2)$$

Substituting the Gaussian likelihoods:

$$\frac{1}{\sqrt{2\pi \cdot 2}} \exp\left(-\frac{(x-4)^2}{2 \cdot 2}\right) = \frac{1}{\sqrt{2\pi \cdot 4}} \exp\left(-\frac{x^2}{2 \cdot 4}\right)$$
$$\exp\left(-\frac{(x-4)^2}{4}\right) = \frac{1}{\sqrt{2}} \times \exp\left(-\frac{x^2}{8}\right)$$
$$-\frac{(x-4)^2}{4} = -\frac{1}{2}\ln(2) - \frac{x^2}{8}$$
$$(x-4)^2 = \frac{x^2}{2} + 2\ln(2)$$
$$x^2 - 8x + 16 = \frac{x^2}{2} + 2\ln(2)$$
$$\frac{x^2}{2} - 8x + (16 - 2\ln(2)) = 0$$
$$x = \frac{8 \pm \sqrt{64 - 2(16 - 2\ln(2))}}{1}$$
$$x \approx 2.10, 13.90$$

Thus, the decision boundary for this classifier is at $\boxed{x \approx 2.10, 13.90}$.
Using Python, we can plot the posterior probabilities for both classes:



3 Employee Attrition Prediction

Figure 2: Pictures of ~~employees~~ cats not used in this dataset.

In this part of the homework, we will work on employee attrition prediction using data from Kaggle IBM HR Analytics Employee Attrition & Performance.

The data

For each employee, 34 features are provided. We will use these features to predict each employee attrition e.g whether the employee will leave the company (`yes` for leaving, `no` for staying)

Notable features are:

- **Education:** 1 'Below College', 2 'College', 3 'Bachelor', 4 'Master', 5 'Doctor'.
- **Environment Satisfaction:** 1 'Low', 2 'Medium', 3 'High', 4 'Very High'.
- **Job Involvement:** 1 'Low', 2 'Medium', 3 'High', 4 'Very High'.
- **Job Satisfaction:** 1 'Low', 2 'Medium', 3 'High', 4 'Very High'.
- **Performance Rating:** 1 'Low', 2 'Good', 3 'Excellent', 4 'Outstanding'.
- **Relationship Satisfaction:** 1 'Low', 2 'Medium', 3 'High', 4 'Very High'.
- **WorkLifeBalance:** 1 'Bad', 2 'Good', 3 'Better', 4 'Best'.

The database

First let's look at the given data file `hr-employee-attrition-with-null.csv`. Load the data using pandas. Use `describe()` and `head()` to get a sense of what the data is like. Our target of prediction is Attrition. Other columns are our input features.

Data cleaning

There are many missing values in this database. They are represented with `NaN`. In the previous homework, we filled the missing values with the mean, median, or mode values. That is because classifiers such as logistic regression cannot deal with missing feature values. However, for the case of Naive Bayes which we will use in this homework compares $\prod_i p(x_i \mid class)$ and treat each x_i as independent features. Thus, if a feature i is missing, we can drop that term from the comparison without having to guess what the missing feature is. First, convert the yes and no in this data table to 1 and 0. Then, we have to convert each categorical feature to number.

```
all.loc[all["Attrition"] == "no", "Attrition"] = 0.0
all.loc[all["Attrition"] == "yes", "Attrition"] = 1.0
for col in cat_cols:
    all[col] = pd.Categorical(all[col]).codes
```

We will also drop the employee numbers.

```
all = all.drop(columns = "EmployeeNumber")
```

There is no standard rule on how much data you should segment into as training and test set. But for now let's use 90% training 10% testing. Select 10% from the "Attrition == yes" and 10% from the "Attrition == no" as your testing set, `test_set`. Then, use the rest of the data as your training set, `train_set`.

Histogram discretization

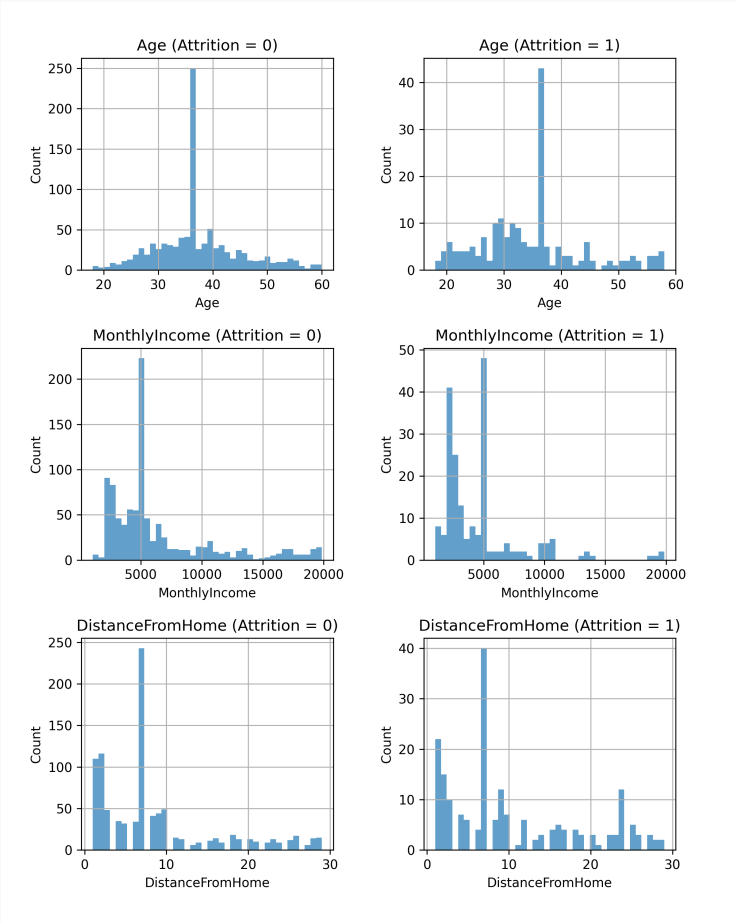
In class, we learned that in order to create a Bayes Classifier we first need to estimate the posterior or likelihood probability distributions. The simplest way to estimate probability distributions is via histograms. To do histogram estimation, we divide the entire data space into a finite number of bins. Then, we count how many data points are there in each bin and normalize using the total number of data points (so that the probability sums to 1). Since we are grouping a continuous valued feature into a finite number of bins, we can also call this process, discretization.

The following code create a histogram of a column `col` from `train_set`.

```
train_col_no_nan = train_set[~np.isnan(train_set[col])]
# remove NaN values

# bin the data into 40 equally spaced bins
# hist is the count for each bin
# bin_edge is the edge values of the bins
hist, bin_edge = np.histogram(train_col_no_nan, 40)

# make sure to import matplotlib.pyplot as plt
# plot the histogram
plt.fill_between(bin_edge.repeat(2)[1:-1],hist.repeat(2),
    facecolor='steelblue')
plt.show()
```

T4. Observe the histogram for `Age`, `MonthlyIncome` and `DistanceFromHome`. How many bins have zero counts? Do you think this is a good discretization? Why?

Solution. Using the provided code, we can plot the histograms for the features `Age`, `MonthlyIncome`, and `DistanceFromHome`. After plotting the histograms, we can count the number of bins with zero counts.



The number of bins with zero counts for each feature is as follows:

- `Age`: 0 (Attrition = 0) + 2 (Attrition = 1) bins with zero counts
- `MonthlyIncome`: 0 (Attrition = 0) + 14 (Attrition = 1) bins with zero counts
- `DistanceFromHome`: 11 (Attrition = 0) + 11 (Attrition = 1) bins with zero counts

Having bins with zero counts indicates that there are ranges of values for these features that are not represented in the training data. This can be problematic for a histogram-based model, as it may lead to poor generalization when making predictions on new data. If a new data point falls into a bin with zero counts, the model will not have any information to estimate the probability for that feature value.

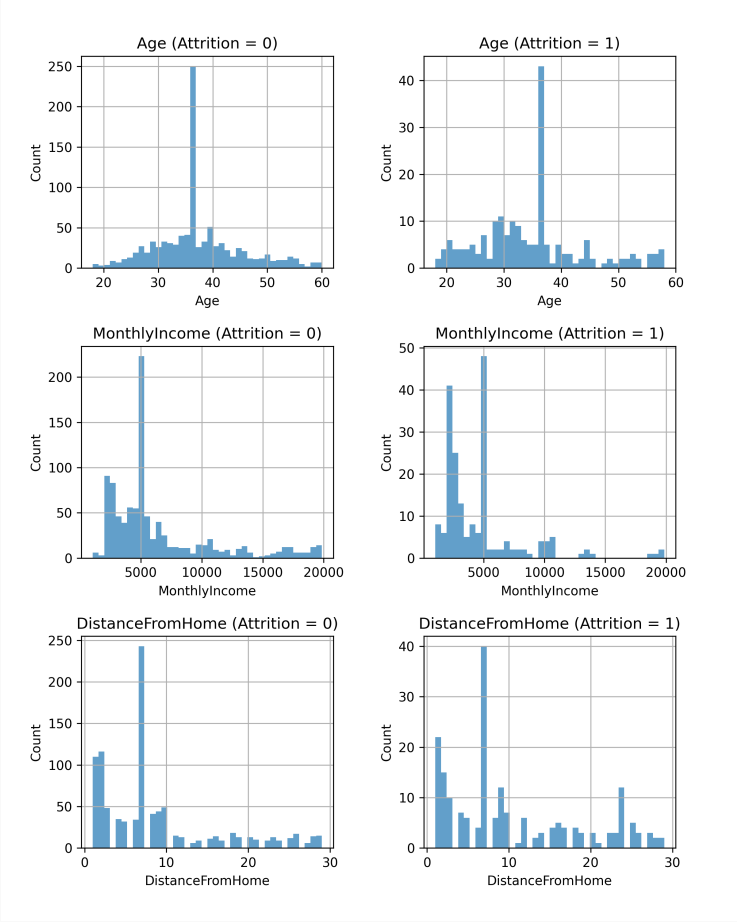
Therefore, this discretization may not be ideal, especially for features like `MonthlyIncome` and `DistanceFromHome`, where a significant number of bins have zero counts. To improve the discretization, we could consider using a different number of bins or employing techniques such as Gaussian Mixture Models (GMMs) to better capture the underlying distribution of the data.

T5. Can we use a Gaussian to estimate this histogram? Why? What about a Gaussian Mixture Model (GMM)?

Solution. To determine whether we can use a Gaussian to estimate the histogram for the features `Age`, `MonthlyIncome`, and `DistanceFromHome`, we need to analyze the shape of the histograms.

A Gaussian distribution is characterized by its bell-shaped curve, which is symmetric around the mean. If the histogram of a feature closely resembles this bell-shaped curve, then it may be appropriate to use a Gaussian to estimate the distribution of that feature.

Upon examining the histograms:



- `Age`: The histogram appears to be roughly bell-shaped, suggesting that a Gaussian distribution could be a reasonable approximation.
- `MonthlyIncome`: The histogram is skewed and does not resemble a bell-shaped curve, indicating that a Gaussian may not be suitable for this feature.
- `DistanceFromHome`: The histogram is also skewed and does not resemble a bell-shaped curve, suggesting that a Gaussian may not be appropriate for this feature either.

A Gaussian Mixture Model (GMM) is a more flexible approach that can model complex distributions by combining multiple Gaussian components. If the histogram of a feature shows multiple peaks or is skewed, a GMM can better capture the underlying distribution.

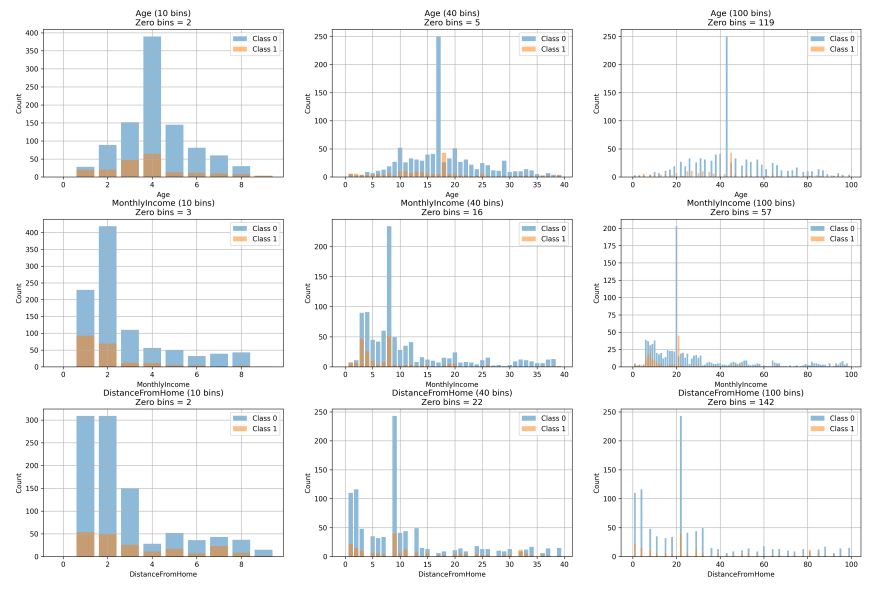
The above discretization equally segments the space into equally spaced bins. This is the best method to segment if you know nothing about the data. Still, doing so may leave us with many bins with zero counts when we have too little data. To prevent this issue, we might assume that the distribution of our data is Normal then draw the probabilities of each data point from this distribution instead. We will do this later. For now, do

1. First set the number of bins to 10 for `Age`, `MonthlyIncome` and `DistanceFromHome`. **Make numbers of bin a parameter as we will change this later.**
2. Bin each values in the training set into bins using the function `np.digitize`, then count the number in each bins using `np.bincount`. Be careful with the maximum and minimum values, your first bin should cover `-inf`, and your final bin should cover `inf`, so that you can handle test data that might be outside of the minimum and maximum values.

You do not need to submit anything for this task. You might want to make this a function so that you can change the number of bins.

T6. Now plot the histogram according to the method described above (with 10, 40, and 100 bins) and show 3 plots each for `Age`, `MonthlyIncome`, and `DistanceFromHome`. Which bin size is most sensible for each features? Why?

Solution. Using the method described, we can plot the histograms for the features `Age`, `MonthlyIncome`, and `DistanceFromHome` with different bin sizes (10, 40, and 100 bins).



After analyzing the histograms with different bin sizes, we can determine the most sensible bin size for each feature:

- `Age`: A bin size of 10 seems most sensible as it captures the overall distribution without being too granular.
- `MonthlyIncome`: A bin size of 40 appears to be the most appropriate, as it provides a good balance between detail and generalization, capturing the skewness of the distribution.
- `DistanceFromHome`: A bin size of 40 also seems most sensible, as it captures the distribution well without being too sparse.

T7. For the rest of the features, which one should be discretized in order to be modeled by histograms? What are the criteria for choosing whether we should discretize a feature or not? Answer this and discretize those features into 10 bins each. In other words, figure out the `bin_edge` for each feature, then use `digitize()` to convert the features to discrete values.

Solution. The solution goes here.

The MLE for the likelihood distribution of discretized histograms

We would like to build a Naive Bayes classifier which compares the posterior $p(leave \mid x_i)$ against $p(stay \mid x_i)$. However, figuring out $p(class \mid x_i)$ is often hard (not true for this case). Thus, we turn to the likelihood $p(x_i \mid class)$, which can be derived from the discretized histograms.

T8. What kind of distribution should we use to model histograms? (Answer a distribution name) What is the MLE for this likelihood distribution? (Describe how to do the MLE). Plot the likelihood distributions of `MonthlyIncome`, `JobRole`, `HourlyRate`, and `MaritalStatus` for different Attrition values.

Hint: In class we talk about how a fair coin can be modeled using the Bernoulli distribution. A histogram is very similar to a dice in the sense that the outcome is a set of possibilities.

Solution. The solution goes here.

T9. What is the prior distribution of the two classes?

Solution. The solution goes here.

Naive Bayes classification

We are now ready to build our Naive Bayes classifier. Which makes a decision according to

$$H(x) = \frac{p(leave)}{p(stay)} \prod_{i=1} \frac{p(x_i \mid leave)}{p(x_i \mid stay)}$$

If $H(x)$ is larger than 1, then classify it as `leave`. If $H(x)$ is smaller than 1, then classify it as `stay`.

Note we often work in the log scale to prevent floating point underflow. In other words,

$$lH(x) = \log\left(p(leave)\right) - \log\left(p(stay)\right) + \sum_{i=1} \left[\log\left(p(x_i \mid leave)\right) - \log\left(p(x_i \mid stay)\right)\right]$$

If $lH(x)$ is larger than 0, then classify it as leave. If $lH(x)$ is smaller than 0, then classify it as stay.

T10. If we use the current Naive Bayes with our current Maximum Likelihood Estimates, we will find that some $P(x_i|attrition)$ will be zero and will result in the entire product term to be zero. Propose a method to fix this problem.

Solution. The solution goes here.

T11. Implement your Naive Bayes classifier. Use the learned distributions to classify the `test_set`.

Don't forget to allow your classifier to handle missing values in the test set. Report the overall Accuracy. Then, report the Precision, Recall, and F score for detecting attrition. See Lecture 1 for the definitions of each metric.

> **Solution.** The solution goes here.

Probability density function

Now, instead of using histogram discretization, we will assume that our features are normally distributed. In other words, for certain feature types, $P(x_i|attrition)$ is now Normally distributed. By doing so, we can estimate the mean and standard deviation for each feature and compute the probability of each test feature by using the Gaussian probability density function instead. You can do this by calling:

```
scipy.stats.norm(mean, std).pdf(feature_value)
```

T12. Use the learned distributions to classify the `test_set`. Report the results using the same metric as the previous question.

> **Solution.** The solution goes here.

Baseline comparison

In machine learning, we need to be able to evaluate how good our model is. We usually compare our model with a different model and show that our model is better. Sometimes we do not have a candidate model to evaluate our method against. In this homework, we will look at two simple baselines, the random choice, and the majority rule.

T13. The random choice baseline is the accuracy if you make a random guess for each test sample. Give random guess (50% leaving, and 50% staying) to the test samples. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the random choice baseline.

> **Solution.** The solution goes here.

T14. The majority rule is the accuracy if you use the most frequent class from the training set as the classification decision. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the majority rule baseline.

> **Solution.** The solution goes here.

T15. Compare the two baselines with your Naive Bayes classifier.

> **Solution.** The solution goes here.

Threshold finding

In practice, instead of comparing $lH(x)$ against 0, we usually compare against a threshold, t. We can change the threshold so that we maximize the accuracy, precision, recall, or F score

(depending on which measure we want to optimize).

T16. Use the following threshold values

```
t = np.arange(-5,5,0.05)
```

find the best accuracy, and F score (and the corresponding thresholds)

Solution. The solution goes here.

Receiver Operating Characteristic (RoC) curve

The recall rate (true positive rate) and the false alarm rate can change as we vary the threshold. The false alarm rate will deteriorate as we decrease the threshold (more false alarms). On the other hand, the recall rate will improve. This is also another trade-off machine learning practitioners need to consider. If we plot the false alarm vs recall as we vary the threshold (false alarm as the x-axis and recall as the y-axis), we get a plot called the "Receiver operating characteristic (RoC) curve." The RoC curve illustrates the performance of a binary classifier (Will this person leave? Will this person survive the Titanic? yes or no) as the threshold is varied. An example RoC curve is shown below

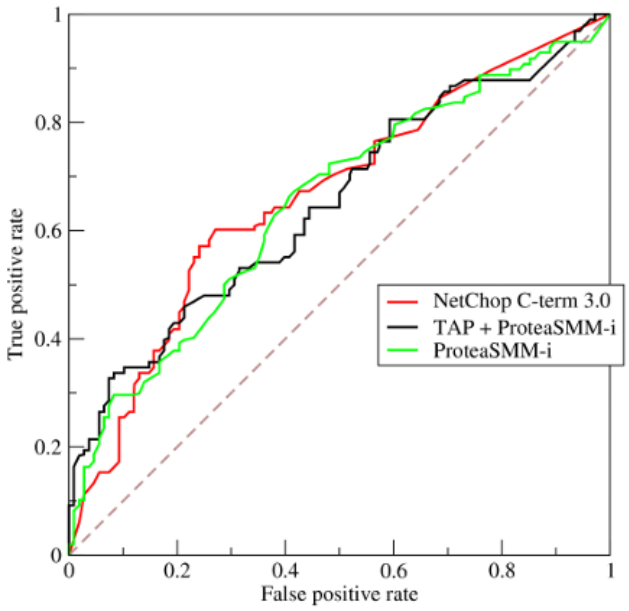


Figure 3: An example RoC curve. Source wikipedia

T17. Plot the RoC of your classifier.

Solution. The solution goes here.

T18. Change the number of discretization bins to 5. What happens to the RoC curve? Which discretization is better? The number of discretization bins can be considered as a hyperparameter, and must be chosen by comparing the final performance.

Solution. The solution goes here.

T19. Submit your code (.py or .ipynb) on mycourseville. If you've made it this far, **congratulations!** you've just created simple models that can help HR deal with one of their biggest problems. Simple, isn't it? This is a real world task with real implications, and I personally have been approached by big companies to help with this.

Solution. The solution goes here.

Classifier Variance

Recall, in class, we talked about the variance of a classifier as the training set changes. In this section, we will evaluate our model if we shuffle the training and test data. This will give a measure whether our recognizer is good just because we are lucky (and give statistical significance to our experiments).

OT4. Shuffle the database, and create new test and train sets. Redo the entire training and evaluation process 10 times (each time with a new training and test set). Calculate the mean and variance of the accuracy rate.

Solution. The solution for the **OT** problem goes here.

APPENDIX 1

Pattern HW2 Starter Code

Pattern_HW2_student_2026

January 30, 2026

1 Employee Attrition Prediction

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Read CSV File

[2]:
```
df = pd.read_csv("hr-employee-attrition-with-null.csv")
```

1.2 Dataset Statistic

[3]:
```
df.describe()
```

[3]:

	Unnamed: 0	Age	DailyRate	DistanceFromHome	Education	\
count	1470.000000	1176.000000	1176.000000	1176.00000	1176.000000	
mean	734.500000	37.134354	798.875850	9.37500	2.920918	
std	424.496761	9.190317	406.957684	8.23049	1.028796	
min	0.000000	18.000000	102.000000	1.00000	1.000000	
25%	367.250000	30.000000	457.750000	2.00000	2.000000	
50%	734.500000	36.000000	798.500000	7.00000	3.000000	
75%	1101.750000	43.000000	1168.250000	15.00000	4.000000	
max	1469.000000	60.000000	1499.000000	29.00000	5.000000	

	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	\
count	1176.0	1176.000000	1176.000000	1176.000000	
mean	1.0	1031.399660	2.733844	65.821429	
std	0.0	601.188955	1.092992	20.317323	
min	1.0	1.000000	1.000000	30.000000	
25%	1.0	494.750000	2.000000	48.000000	
50%	1.0	1027.500000	3.000000	66.000000	
75%	1.0	1562.250000	4.000000	84.000000	
max	1.0	2068.000000	4.000000	100.000000	

	JobInvolvement	…	RelationshipSatisfaction	StandardHours	\
count	1176.000000	…	1176.000000	1176.0	
mean	2.728741	…	2.694728	80.0	

```
std          0.705280  …                1.093660            0.0
min          1.000000  …                1.000000           80.0
25%          2.000000  …                2.000000           80.0
50%          3.000000  …                3.000000           80.0
75%          3.000000  …                4.000000           80.0
max          4.000000  …                4.000000           80.0

       StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
count       1176.000000        1176.000000            1176.000000
mean           0.752551          11.295068               2.787415
std            0.822550           7.783376               1.290507
min            0.000000           0.000000               0.000000
25%            0.000000           6.000000               2.000000
50%            1.000000          10.000000               3.000000
75%            1.000000          15.000000               3.000000
max            3.000000          40.000000               6.000000

       WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
count      1176.000000     1176.000000         1176.000000
mean          2.770408        7.067177            4.290816
std           0.705004        6.127836            3.630901
min           1.000000        0.000000            0.000000
25%           2.000000        3.000000            2.000000
50%           3.000000        5.000000            3.000000
75%           3.000000       10.000000            7.000000
max           4.000000       37.000000           18.000000

       YearsSinceLastPromotion  YearsWithCurrManager
count              1176.000000           1176.000000
mean                  2.159014              4.096939
std                   3.163524              3.537393
min                   0.000000              0.000000
25%                   0.000000              2.000000
50%                   1.000000              3.000000
75%                   2.250000              7.000000
max                  15.000000             17.000000

[8 rows x 27 columns]
```

[4]:
```
df.head()
```

[4]:
```
   Unnamed: 0   Age Attrition     BusinessTravel  DailyRate  \
0           0  41.0       Yes      Travel_Rarely        NaN
1           1   NaN        No                NaN      279.0
2           2  37.0       Yes                NaN     1373.0
3           3   NaN        No  Travel_Frequently     1392.0
4           4  27.0        No      Travel_Rarely      591.0
```

```
              Department  DistanceFromHome  Education EducationField  \
0                    NaN               1.0        NaN  Life Sciences
1  Research & Development              NaN        NaN  Life Sciences
2                    NaN               2.0        2.0            NaN
3  Research & Development              3.0        4.0  Life Sciences
4  Research & Development              2.0        1.0        Medical

   EmployeeCount  …  RelationshipSatisfaction  StandardHours  \
0            1.0  …                       1.0           80.0
1            1.0  …                       4.0            NaN
2            1.0  …                       NaN           80.0
3            NaN  …                       3.0            NaN
4            1.0  …                       4.0           80.0

  StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  WorkLifeBalance  \
0              0.0                8.0                    0.0              NaN
1              1.0               10.0                    NaN              3.0
2              0.0                7.0                    3.0              NaN
3              NaN                8.0                    3.0              NaN
4              1.0                6.0                    NaN              3.0

  YearsAtCompany  YearsInCurrentRole YearsSinceLastPromotion  \
0            6.0                 NaN                     0.0
1           10.0                 NaN                     NaN
2            NaN                 0.0                     NaN
3            8.0                 NaN                     3.0
4            2.0                 2.0                     2.0

   YearsWithCurrManager
0                   NaN
1                   7.0
2                   0.0
3                   0.0
4                   NaN

[5 rows x 36 columns]
```

1.3 Feature transformation

[5]:
```python
df.loc[df["Attrition"] == "no", "Attrition"] = 0.0
df.loc[df["Attrition"] == "yes", "Attrition"] = 1.0

string_categorical_col = [
    "Department", "Attrition", "BusinessTravel",
```

```
    "EducationField", "Gender", "JobRole",
    "MaritalStatus", "Over18", "OverTime"
]

# ENCODE STRING COLUMNS TO CATEGORICAL COLUMNS
for col in string_categorical_col:
    # INSERT CODE HERE
    df[col] = df[col].astype("category").cat.codes

# HANDLE NULL NUMBERS
# INSERT CODE HERE
df = df.fillna(df.median())
df = df.loc[:, ~df.columns.isin(["EmployeeNumber", "Unnamed: 0",␣
 ↪"EmployeeCount", "StandardHours", "Over18"])]
```

1.3.1 Spliting data into train and test

[6]:
```
from sklearn.model_selection import train_test_split
```

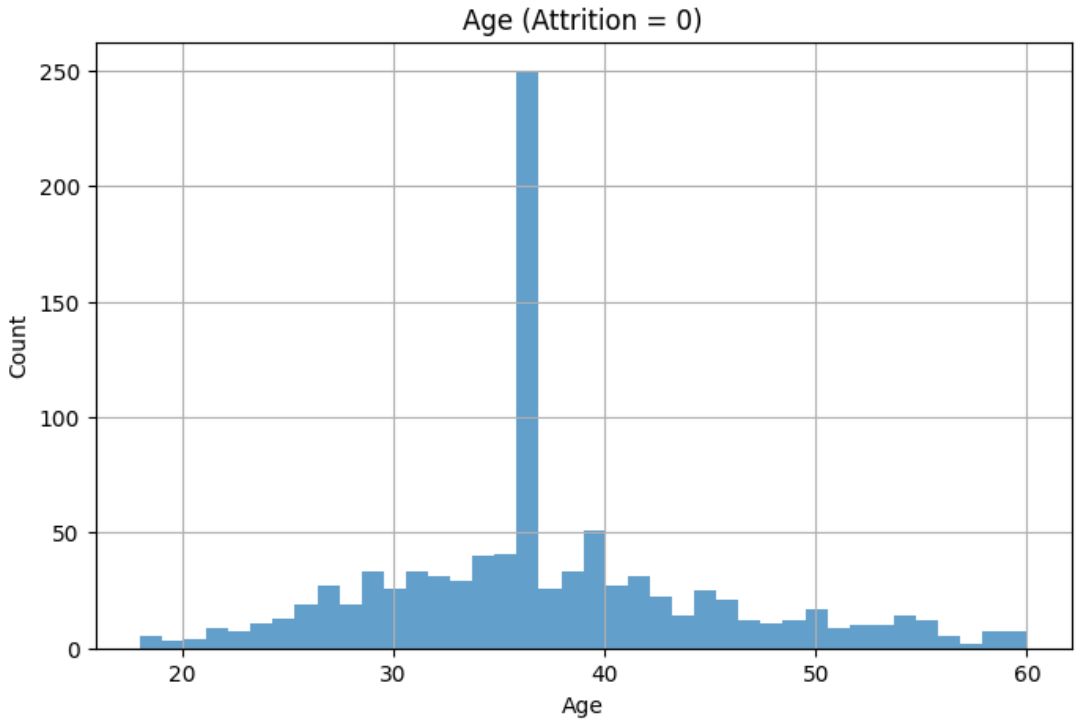
[7]:
```
df_train, df_test = train_test_split(df, test_size=0.2, random_state=42)
```

1.3.2 Display histogram of each feature

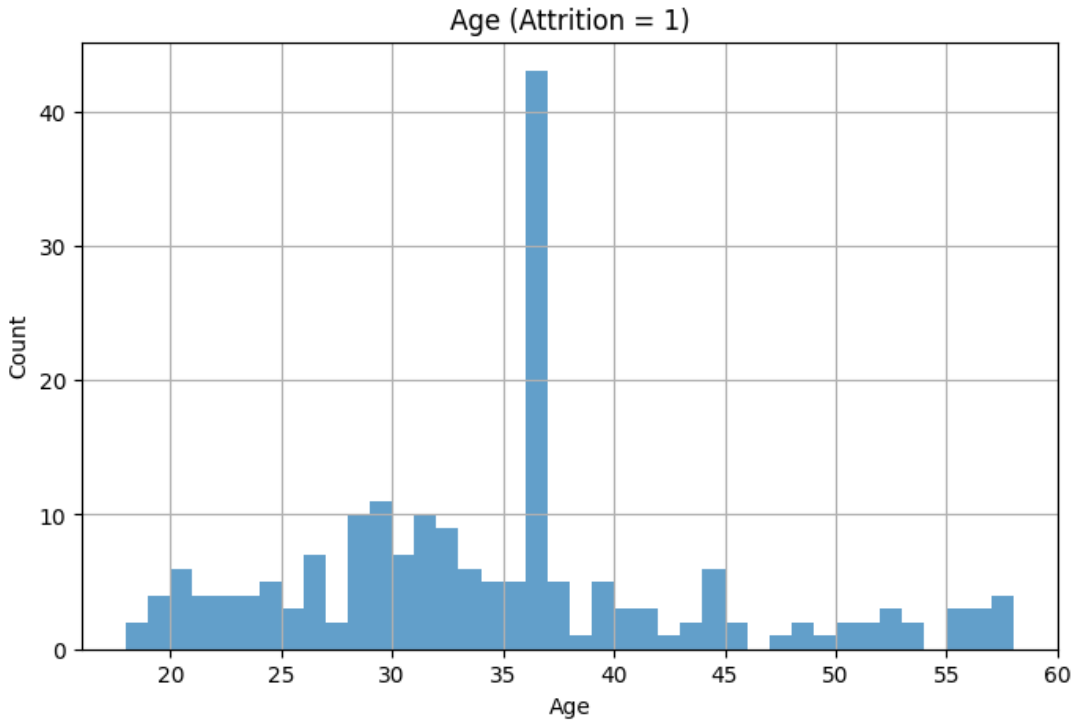
[8]:
```
def display_histogram(df, col_name, cls, n_bin = 40):
    # INSERT CODE HERE
    # Filter data by Attrition == cls
    data = df[df["Attrition"] == cls][col_name]

    plt.figure(figsize=(8, 5))

    counts, bins, _ = plt.hist(
        data,
        bins=n_bin,
        alpha=0.7
    )

    zero_bins = np.sum(counts == 0)

    plt.xlabel(col_name)
    plt.ylabel("Count")
    plt.title(f"{col_name} (Attrition = {cls})")
    plt.grid(True)

    plt.show()

    print(f"Number of bins with zero counts: {zero_bins}")
```

1.3.3 Problem T4

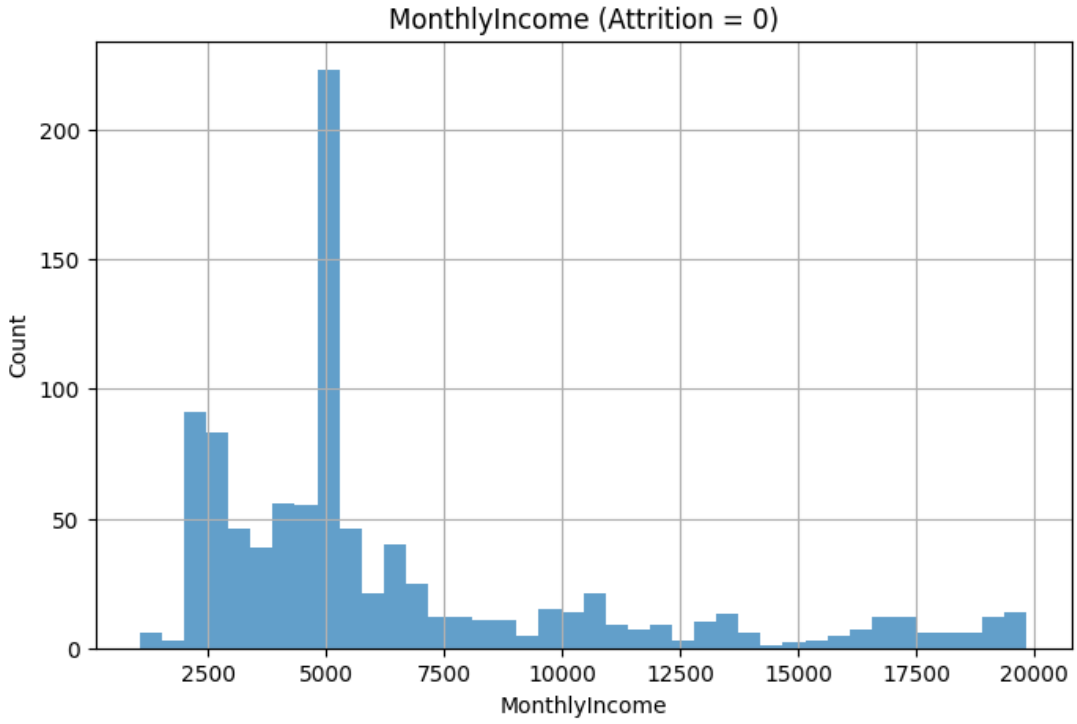
[9]:
```
display_histogram(df_train, "Age", 0)
display_histogram(df_train, "Age", 1)

display_histogram(df_train, "MonthlyIncome", 0)
display_histogram(df_train, "MonthlyIncome", 1)

display_histogram(df_train, "DistanceFromHome", 0)
display_histogram(df_train, "DistanceFromHome", 1)
```



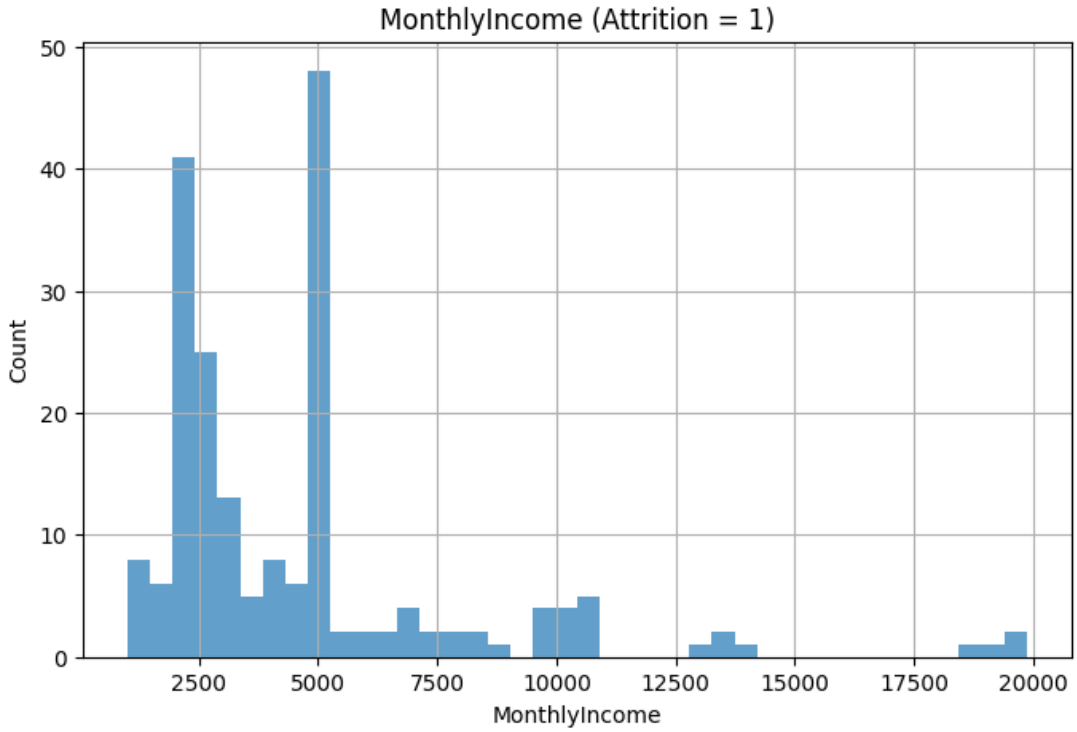

```
Number of bins with zero counts: 0
```



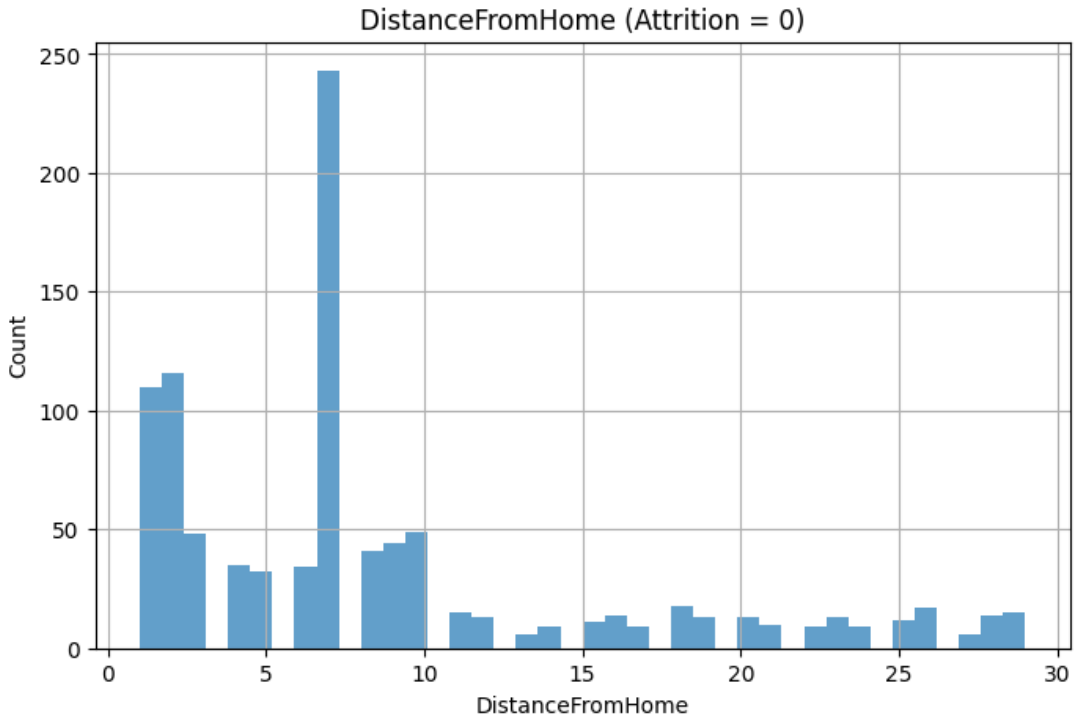

```
Number of bins with zero counts: 2
```



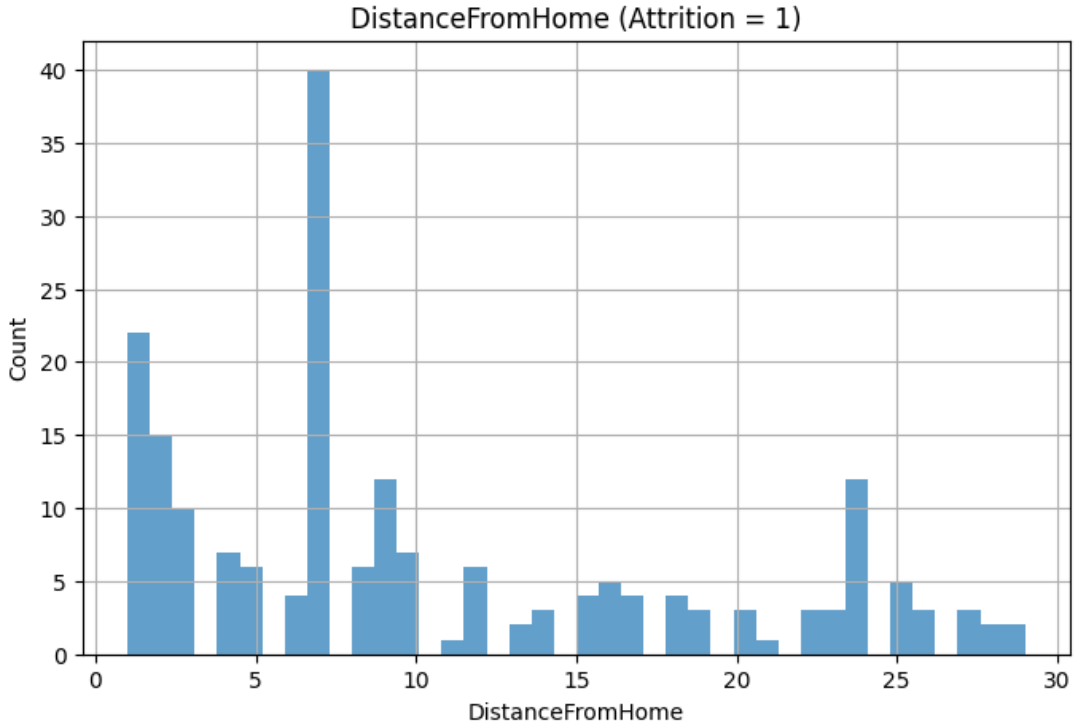
```
Number of bins with zero counts: 0
```




```
Number of bins with zero counts: 14
```



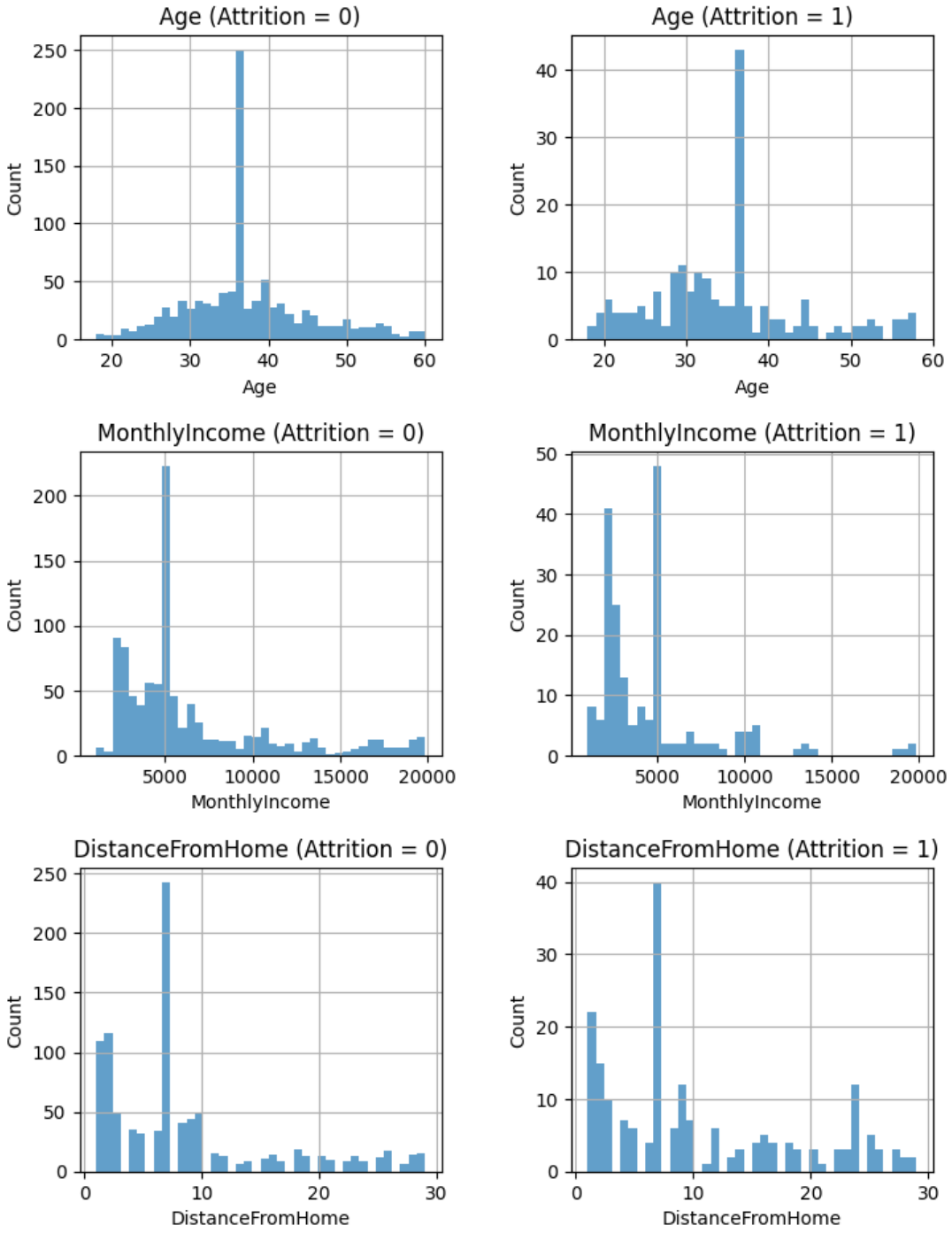

```
Number of bins with zero counts: 11
```



```
Number of bins with zero counts: 11
```

[10]:
```python
# Make better plots
fig, axes = plt.subplots(3, 2, figsize=(8, 10))
fig.tight_layout(pad=4.0)

col_names = ["Age", "MonthlyIncome", "DistanceFromHome"]
attrition_classes = [0, 1]

for i, col_name in enumerate(col_names):
    for j, cls in enumerate(attrition_classes):
        ax = axes[i, j]
        data = df_train[df_train["Attrition"] == cls][col_name]

        counts, bins, _ = ax.hist(
            data,
            bins=40,
            alpha=0.7
        )

        zero_bins = np.sum(counts == 0)

        ax.set_xlabel(col_name)
        ax.set_ylabel("Count")
        ax.set_title(f"{col_name} (Attrition = {cls})")
        ax.grid(True)

        print(f"Number of bins with zero counts in {col_name} (Attrition =␣
 ↪{cls}): {zero_bins}")

plt.savefig("../images/p4.png", dpi=300)
plt.show()
```

```
Number of bins with zero counts in Age (Attrition = 0): 0
Number of bins with zero counts in Age (Attrition = 1): 2
Number of bins with zero counts in MonthlyIncome (Attrition = 0): 0
Number of bins with zero counts in MonthlyIncome (Attrition = 1): 14
Number of bins with zero counts in DistanceFromHome (Attrition = 0): 11
Number of bins with zero counts in DistanceFromHome (Attrition = 1): 11
```


Age (Attrition = 0)
Count
Age
Age (Attrition = 1)
Count
Age
MonthlyIncome (Attrition = 0)
Count
MonthlyIncome
MonthlyIncome (Attrition = 1)
Count
MonthlyIncome
DistanceFromHome (Attrition = 0)
Count
DistanceFromHome
DistanceFromHome (Attrition = 1)
Count
DistanceFromHome

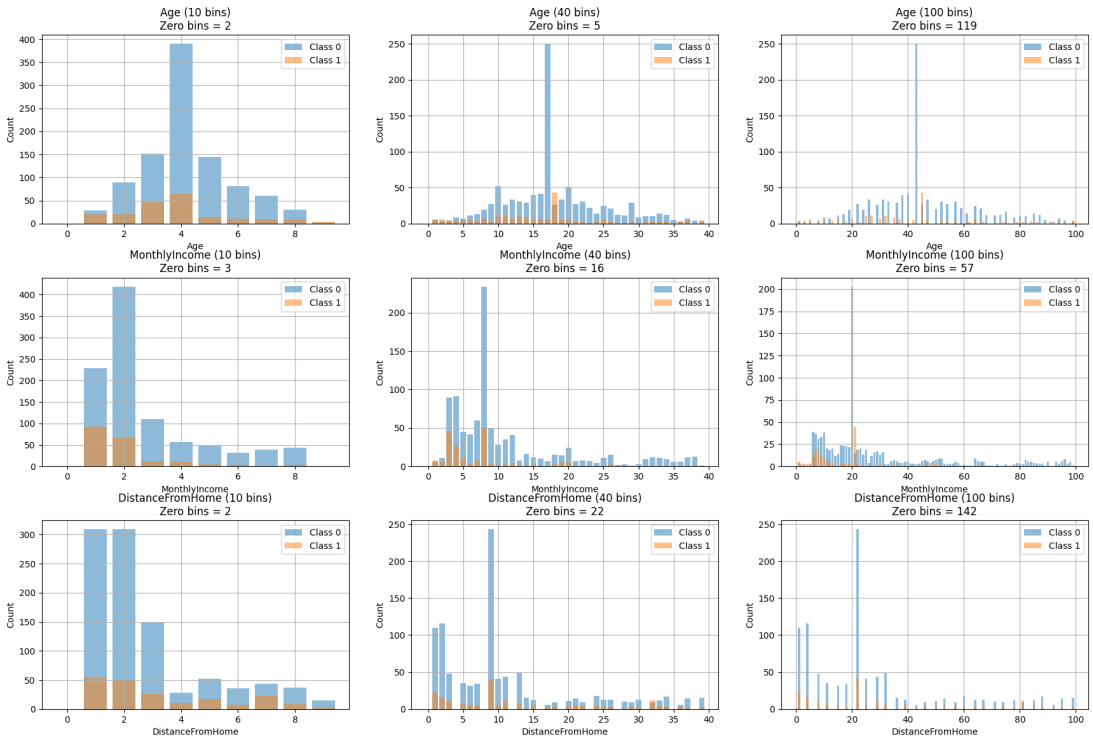
1.3.4 Problem T6

[11]:
```python
def discretize_feature(values, n_bins):
    min_val = np.min(values)
    max_val = np.max(values)

    # Internal bin edges (exclude -inf, +inf)
    bin_edges = np.linspace(min_val, max_val, n_bins - 1)

    # Assign bins
    bin_indices = np.digitize(values, bin_edges)

    # Count samples per bin
    counts = np.bincount(bin_indices, minlength=n_bins)

    return counts, bin_edges
```

[12]:
```python
# Make better plots
fig, axes = plt.subplots(3, 3, figsize=(18, 12))
fig.tight_layout(pad=4.0)

col_names = ["Age", "MonthlyIncome", "DistanceFromHome"]
bin_number_list = [10, 40, 100]
attrition_classes = [0, 1]

for i, col_name in enumerate(col_names):
    for j, n_bin in enumerate(bin_number_list):
        ax = axes[i, j]

        total_zero_bins = 0

        for cls in attrition_classes:
            data = df_train[df_train["Attrition"] == cls][col_name].values

            counts, bin_edges = discretize_feature(data, n_bin)

            # Plot as bar chart (manual histogram)
            ax.bar(
                range(n_bin),
                counts,
                width=0.8,
                alpha=0.5,
                label=f"Class {cls}"
            )

            zero_bins = np.sum(counts == 0)
            total_zero_bins += zero_bins
```

```
        ax.set_xlabel(col_name)
        ax.set_ylabel("Count")
        ax.set_title(f"{col_name} ({n_bin} bins)\nZero bins =␣
↪{total_zero_bins}")
        ax.legend()
        ax.grid(True)

        print(
            f"Zero-count bins in {col_name} (bins={n_bin}): {total_zero_bins}"
        )

plt.savefig("../images/p6.png", dpi=300)
plt.show()
```

```
Zero-count bins in Age (bins=10): 2
Zero-count bins in Age (bins=40): 5
Zero-count bins in Age (bins=100): 119
Zero-count bins in MonthlyIncome (bins=10): 3
Zero-count bins in MonthlyIncome (bins=40): 16
Zero-count bins in MonthlyIncome (bins=100): 57
Zero-count bins in DistanceFromHome (bins=10): 2
Zero-count bins in DistanceFromHome (bins=40): 22
Zero-count bins in DistanceFromHome (bins=100): 142
```



1.3.5 T7. For the rest of the features, which one should be discretized in order to be modeled by histograms? What are the criteria for choosing whether we should discretize a feature or not? Answer this and discretize those features into 10 bins each. In other words, figure out the bin edge for each feature, then use digitize() to convert the features to discrete values

1.3.6 T8. What kind of distribution should we use to model histograms? (Answer a distribution name) What is the MLE for the likelihood distribution? (Describe how to do the MLE). Plot the likelihood distributions of MonthlyIncome, JobRole, HourlyRate, and MaritalStatus for different Attrition values.

1.3.7 T9. What is the prior distribution of the two classes?

1.3.8 T10. If we use the current Naive Bayes with our current Maximum Likelihood Estimates, we will find that some P (x i |attrition) will be zero and will result in the entire product term to be zero. Propose a method to fix this problem.

1.3.9 T11. Implement your Naive Bayes classifier. Use the learned distributions to classify the test set. Don't forget to allow your classifier to handle missing values in the test set. Report the overall Accuracy. Then, report the Precision, Recall, and F score for detecting attrition. See Lecture 1 for the definitions of each metric.

[13]:
```
from SimpleBayesClassifier import SimpleBayesClassifier
```

```
Traceback (most recent call last):

  File ~/study/cedt-2-2/2110573-pattern/CEDT-2110573-Pattern-Recognition/.venv/
↪lib/python3.10/site-packages/IPython/core/interactiveshell.py:3579 in run_cod
    exec(code_obj, self.user_global_ns, self.user_ns)

  Cell In[13], line 1
    from SimpleBayesClassifier import SimpleBayesClassifier

  File ~/study/cedt-2-2/2110573-pattern/CEDT-2110573-Pattern-Recognition/
↪homework-2/code/SimpleBayesClassifier.py:22
    self.n_pos =
               ^

SyntaxError: invalid syntax
```

[ ]:
```
data_train = df_train.to_numpy()
data_test = df_test.to_numpy()
```

[ ]:
```
x_train =
y_train =

x_test =
y_test =
```

[ ]:
```
model = SimpleBayesClassifier(n_pos = , n_neg = )
```

[ ]:
```
def check_prior():
    """
    This function designed to test the implementation of the prior probability␣
 ↪calculation in a Naive Bayes classifier.
    Specifically, it checks if the classifier correctly computes the prior␣
 ↪probabilities for the
    negative and positive classes based on given input counts.
    """

    # prior_neg = 5/(5 + 5) = 0.5 and # prior_pos = 5/(5 + 5) = 0.5
    assert (SimpleBayesClassifier(5, 5).prior_pos, SimpleBayesClassifier(5, 5).
 ↪prior_neg) == (0.5, 0.5)

    assert (SimpleBayesClassifier(3, 5).prior_pos, SimpleBayesClassifier(3, 5).
 ↪prior_neg) ==
    assert (SimpleBayesClassifier(0, 1).prior_pos, SimpleBayesClassifier(0, 1).
 ↪prior_neg) ==
    assert (SimpleBayesClassifier(1, 0).prior_pos, SimpleBayesClassifier(1, 0).
 ↪prior_neg) ==

check_prior()
```

[ ]:
```
model.fit_params(x_train, y_train)
```

[ ]:
```
def check_fit_params():

    """
    This function is designed to test the fit_params method of a␣
 ↪SimpleBayesClassifier.
    This method is presumably responsible for computing parameters for a Naive␣
 ↪Bayes classifier
    based on the provided training data. The parameters in this context is bins␣
 ↪and edges from each histogram.
    """

    T = SimpleBayesClassifier(2, 2)
    X_TRAIN_CASE_1 = np.array([
        [0, 1, 2, 3],
```

```
        [1, 2, 3, 4],
        [2, 3, 4, 5],
        [3, 4, 5, 6]
    ])
    Y_TRAIN_CASE_1 = np.array([0, 1, 0, 1])
    STAY_PARAMS_1, LEAVE_PARAMS_1 = T.fit_params(X_TRAIN_CASE_1, Y_TRAIN_CASE_1)

    print("STAY PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"BINS : {STAY_PARAMS_1[f_idx][0]}")
        print(f"EDGES : {STAY_PARAMS_1[f_idx][1]}")
    print("")
    print("LEAVE PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"BINS : {LEAVE_PARAMS_1[f_idx][0]}")
        print(f"EDGES : {LEAVE_PARAMS_1[f_idx][1]}")

check_fit_params()
```

[ ]:
```
y_pred = model.predict(x = x_test)
```

[ ]:
```
def evaluate(y_true, y_pred, show_result = True):

  return accuracy, precision, recall, F1, fpr
```

[ ]:
```
evaluate(y_test, y_pred)
```

1.3.10 T12. Use the learned distributions to classify the test set. Report the results using the same metric as the previous question.

[ ]:
```
model.fit_gaussian_params(x_train, y_train)
```

[ ]:
```
def check_fit_gaussian_params():

    """
    This function is designed to test the fit_gaussian_params method of a␣
↪SimpleBayesClassifier.
    This method is presumably responsible for computing parameters for a Naive␣
↪Bayes classifier
    based on the provided training data. The parameters in this context is mean␣
↪and STD.
    """

    T = SimpleBayesClassifier(2, 2)
```

```
    X_TRAIN_CASE_1 = np.array([
        [0, 1, 2, 3],
        [1, 2, 3, 4],
        [2, 3, 4, 5],
        [3, 4, 5, 6]
    ])
    Y_TRAIN_CASE_1 = np.array([0, 1, 0, 1])
    STAY_PARAMS_1, LEAVE_PARAMS_1 = T.fit_gaussian_params(X_TRAIN_CASE_1,␣
↪Y_TRAIN_CASE_1)

    print("STAY PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"Mean : {STAY_PARAMS_1[f_idx][0]}")
        print(f"STD. : {STAY_PARAMS_1[f_idx][1]}")
    print("")
    print("LEAVE PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"Mean : {LEAVE_PARAMS_1[f_idx][0]}")
        print(f"STD. : {LEAVE_PARAMS_1[f_idx][1]}")

check_fit_gaussian_params()
```

[ ]:
```
y_pred = model.gaussian_predict(x_test)
```

[ ]:
```
evaluate(y_test, y_pred)
```

1.3.11 T13 : The random choice baseline is the accuracy if you make a random guess for each test sample. Give random guess (50% leaving, and 50% staying) to the test samples. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the random choice baseline.

1.3.12 T14. The majority rule is the accuracy if you use the most frequent class from the training set as the classification decision. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the majority rule baseline.

1.3.13 T15. Compare the two baselines with your Naive Bayes classifier.

1.3.14 T16. Use the following threshold values

$ t = np.arange(-5,5,0.05) $ ### find the best accuracy, and F score (and the corresponding thresholds)

1.3.15 T17. Plot the RoC of your classifier.

1.3.16 T18. Change the number of discretization bins to 5. What happens to the RoC curve? Which discretization is better? The number of discretization bins can be considered as a hyperparameter, and must be chosen by comparing the final performance.

APPENDIX 2

SimpleBayesClassifier

SimpleBayesClassifier

January 30, 2026

1 Simple Bayes Classifier

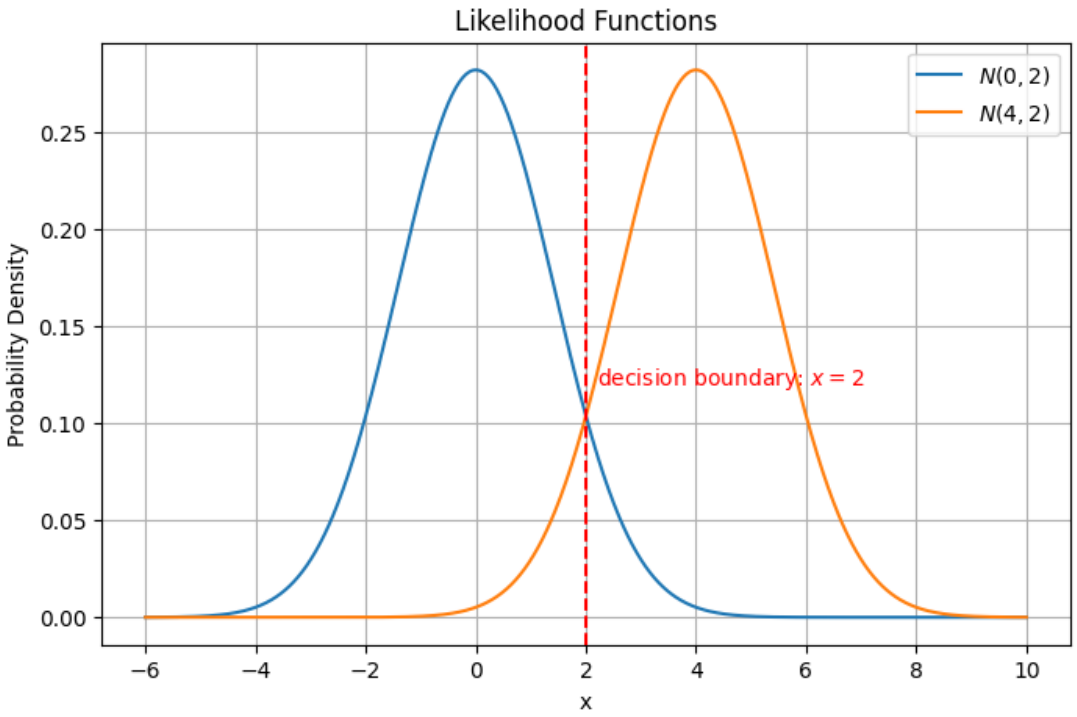
1.1 Import Libraries

[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()
```

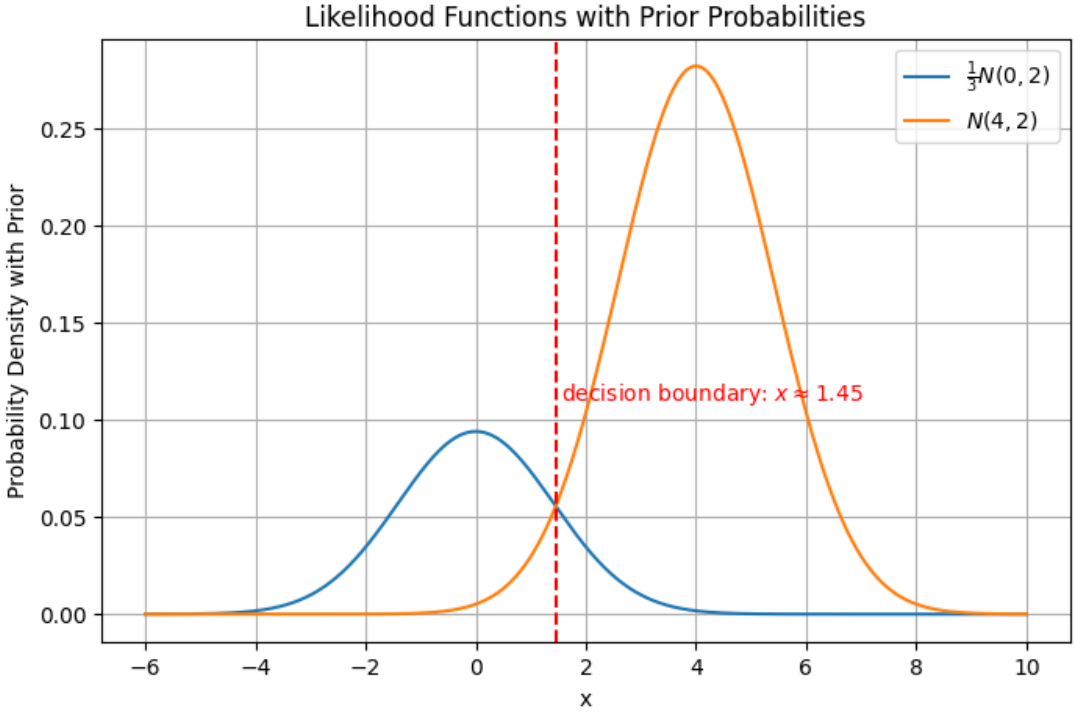


---

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"$\frac{1}{3} N(0, 2)$")
```

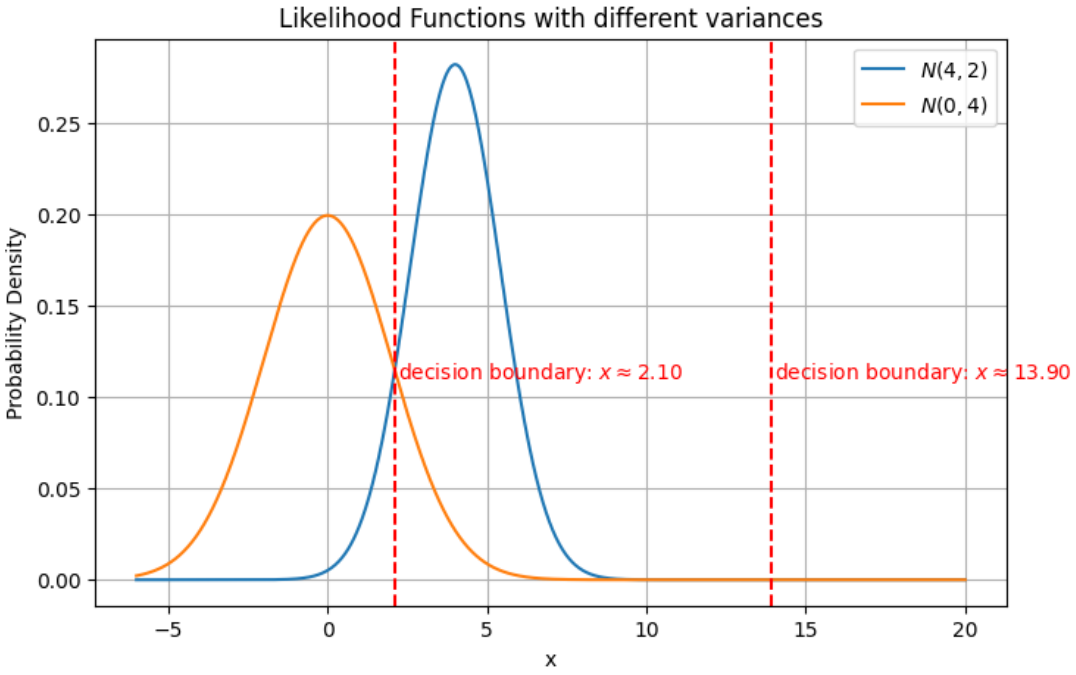
```
plt.plot(x, p_w1, label=r"$N(4, 2)$")

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

[ ]:
```python
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
```

```python
    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
```
