Name: Phonphrm Thawatdamrongkit
Student ID: 2330802

Homework2 MLE and Naive Bayes

T1.
ANSWER

T1.

Given $y_2 = \alpha y_1 + w_1$, $y_1 = \alpha y_0 + w_0$, where w_0 and w_1 are iid $\sim N(0, \sigma^2)$, $y_0 \sim N(0, \lambda)$, λ and σ are known

→ Find MLE of the return that "α" maximize $P(y_2, y_1, y_0 | \alpha)$

$$P(y_2, y_1, y_0 | \alpha) = \frac{P(y_2, y_1, y_0, \alpha)}{P(\alpha)}$$

$$P(y_2, y_1, y_0 | \alpha) = \frac{P(y_2 | y_1, y_0, \alpha) \cdot P(y_1 | y_0, \alpha) \cdot P(y_0 | \alpha) \cdot P(\alpha)}{P(\alpha)}$$

$$P(y_2, y_1, y_0 | \alpha) = P(y_2 | y_1, y_0, \alpha) \cdot P(y_1 | y_0, \alpha) \cdot P(y_0 | \alpha)$$

Given Markov Process, $P(y_2 | y_1, y_0; \alpha) = P(y_2 | y_1; \alpha)$ → Treat as fixed?

Then,

$$P(y_2, y_1, y_0 | \alpha) = P(y_2 | y_1; \alpha) \cdot P(y_1 | y_0; \alpha) \cdot P(y_0; \alpha)$$

Since, $y_0 \sim N(0, \lambda)$ then $P(y_0; \alpha) = \frac{1}{\sqrt{2\pi\lambda}} \cdot e^{\frac{-y_0^2}{2\lambda}}$

Since, $y_1 = \underbrace{\alpha y_0}_{u} + \underbrace{w_0}_{\text{Base}}$ then $P(y_1 | y_0; \alpha) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{(y_1 - \alpha y_0)^2}{\sigma^2}\right)}$

Since, $y_2 = \underbrace{\alpha y_1}_{u} + \underbrace{\alpha_1}_{\text{Base}}$ then $P(y_2 | y_1; \alpha) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{(y_2 - \alpha y_1)^2}{\sigma^2}\right)}$

Therefore, $P(y_2, y_1, y_0 | \alpha) = \left(\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{(y_2 - \alpha y_1)^2}{\sigma^2}\right)}\right) \cdot \left(\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{(y_1 - \alpha y_0)^2}{\sigma^2}\right)}\right) \cdot \left(\frac{1}{\sqrt{2\pi\lambda}} e^{\frac{-y_0^2}{2\lambda}}\right)$

$$\text{Max } L(\alpha) = \prod_{i=1}^{n} P(y_2^i, y_1^i, y_0^i; \alpha) = \left(\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{(y_2 - \alpha y_1)^2}{\sigma^2}\right)}\right) \cdot \left(\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{(y_1 - \alpha y_0)^2}{\sigma^2}\right)}\right) \cdot \left(\frac{1}{\sqrt{2\pi\lambda}} e^{\frac{-y_0^2}{2\lambda}}\right)$$

Then we take log ! $\underset{\alpha}{\arg\min}\ l(\alpha) = -\log(\sigma\sqrt{2\pi}) - \left(\frac{(y_2 - \alpha y_1)^2}{2\sigma^2}\right) - \log(\sigma\sqrt{2\pi}) - \left(\frac{(y_1 - \alpha y_0)^2}{2\sigma^2}\right) - \log(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$

Then take derivate and set to zero : $\frac{\partial l(\alpha)}{\partial \alpha} = 0 = -\log(\sigma\sqrt{2\pi}) - \left(\frac{(y_2 - \alpha y_1)^2}{2\sigma^2}\right) - \log(\sigma\sqrt{2\pi}) - \left(\frac{(y_1 - \alpha y_0)^2}{2\sigma^2}\right) - \log(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$

$$0 = \frac{2(y_2 - \alpha y_1) \cdot y_1}{2\sigma^2} + \frac{2(y_1 - \alpha y_0) \cdot y_0}{2\sigma^2}, \text{ multiply both side by } \sigma^2$$

$$0 = y_1 y_2 - \alpha y_1^2 + y_0 y_1 - \alpha y_0^2$$

$$0 = y_1 y_2 + y_0 y_1 - \alpha(y_1^2 + y_0^2)$$

$$\alpha(y_1^2 + y_0^2) = y_1 y_2 + y_0 y_1$$

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

OT1.

ANSWER

OT1.

Let define MLE of this:

$$\arg\max_{\alpha} \mathcal{L}(\alpha) = \prod_{i=1}^{m} P(y_{n+1}, y_n, y_{n-1}, \dots, y_0 ; \alpha)$$

Take log $\rightarrow$ $\arg\min_{\alpha} I(\alpha) = \log P(y_{n+1}, y_n, y_{n-1}, \dots, y_0 ; \alpha)$

Since it is Markov Process and we know $P(y_{n+1} | y_n) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{(y_{n+1} - \alpha y_n)^2}{\sigma}\right)}$

Then we know it is true until $P(y_0) = \frac{1}{\sqrt{2\pi\lambda}} \cdot e^{\frac{-y_0^2}{2\lambda}}$

So, $\arg\min_{\alpha} I(\alpha) = \sum_{i=1}^{N} \left(-\log(\sigma\sqrt{2\pi}) - \left(\frac{(y_{i+1} - \alpha y_i)^2}{2\sigma}\right)\right) - \log(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$

Take Derivative and set to zero:

$$\frac{\partial I(\alpha)}{\partial \alpha} = 0 = \sum_{i=1}^{N} \left(-\log(\sigma\sqrt{2\pi}) - \left(\frac{(y_{i+1} - \alpha y_i)^2}{2\sigma}\right)\right) - \log(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$$

$$0 = \sum_{i=1}^{N} \frac{2(y_{i+1} - \alpha y_i) y_i}{2\sigma} = \frac{1}{\sigma} \sum_{i=1}^{N} (y_{i+1} - \alpha y_i) y_i$$

$$\sigma \cdot 0 = \sigma \cdot \frac{1}{\sigma} \sum_{i=1}^{N} (y_{i+1} - \alpha y_i) y_i$$

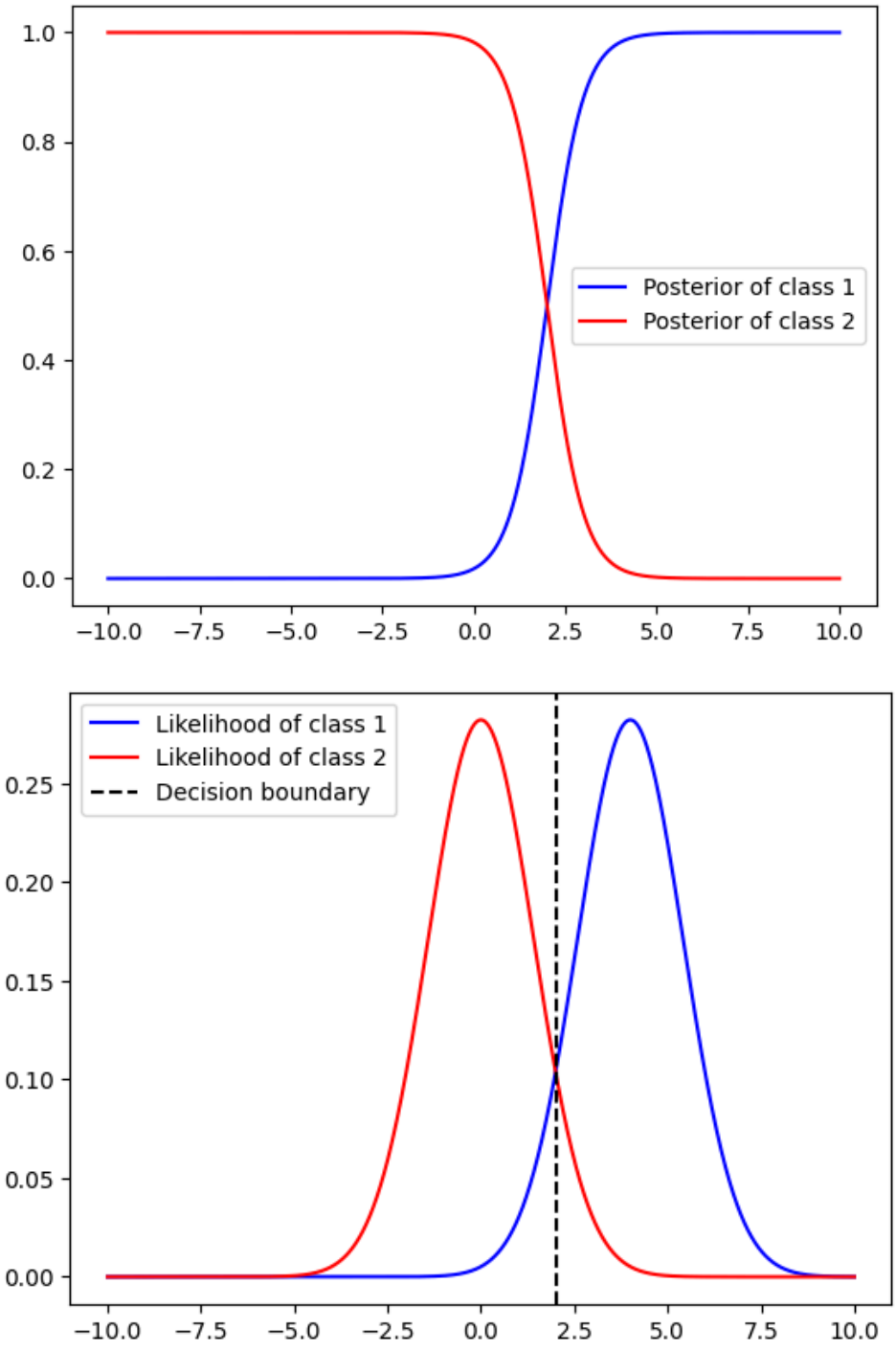
$$0 = \sum_{i=1}^{N} y_i \cdot y_{i+1} - \alpha y_i^2 = \sum_{i=1}^{N} y_i \cdot y_{i+1} - \sum_{i=1}^{N} \alpha y_i^2$$

$$\alpha \sum_{i=1}^{N} y_i^2 = \sum_{i=1}^{N} y_i \cdot y_{i+1}$$

$$\boxed{\alpha = \frac{\sum_{i=1}^{N} y_i \cdot y_{i+1}}{\sum_{i=1}^{N} y_i^2}}$$

T2.

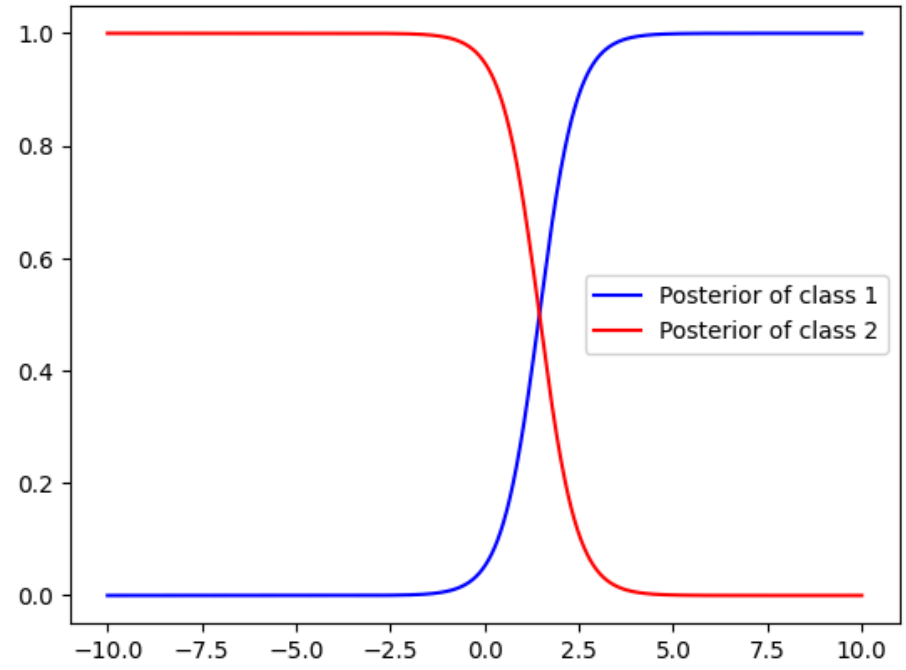
ANSWER

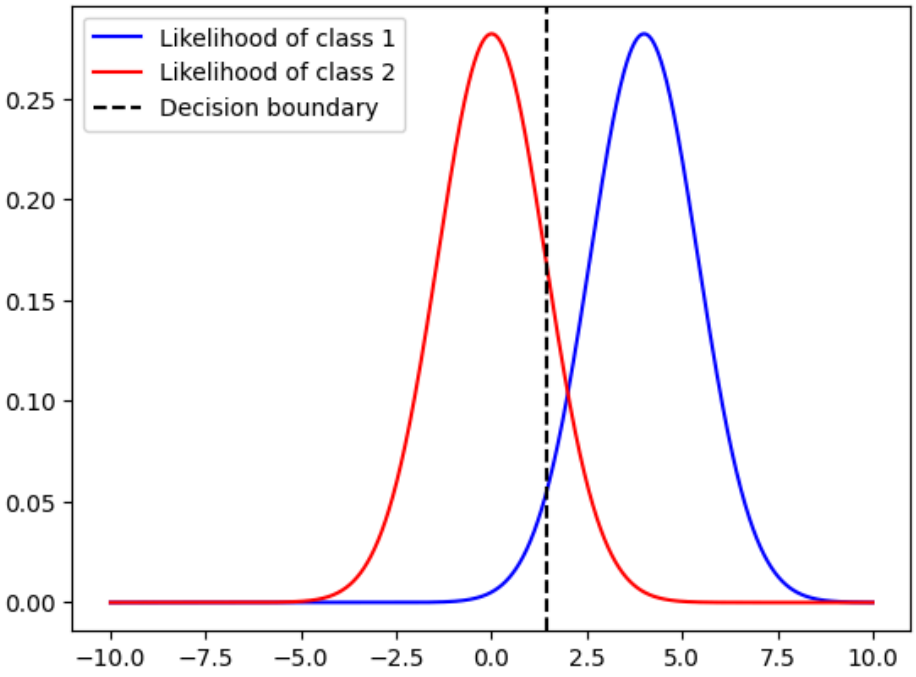


The decision boundary is at 2.000 with a likelihood ratio of 0.104.

T3.

ANSWER





The decision boundary is at 1.451 with a likelihood ratio of 0.056.

OT2.

ANSWER

OT2.

$P(X|w_1) = N(\mu_1, \sigma^2)$, $P(X|w_2) = N(\mu_2, \sigma^2)$, $P(w_1) = P(w_2) = 0.5$

Proof that decision boundary is at $X = \frac{\mu_1 + \mu_2}{2}$

Let do the Likelihood Ratio Test:

$$P(X|w_1) \cdot P(w_1) \;?\; P(X|w_2) \cdot P(w_2)$$

$$\frac{P(X|w_1)}{P(X|w_2)} \;?\; \frac{P(w_2)}{P(w_1)}$$

$$\frac{\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}} \;?\; 1 \quad \rightarrow \quad \frac{e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}}{e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}} \;?\; 1$$

Rewrite in equation, $e^{\frac{-(x-\mu_1)^2 + (x-\mu_2)^2}{2\sigma^2}} = 1$, used exp's rule.

Take "log" both sides, $\frac{(x-\mu_2)^2 - (x-\mu_1)^2}{2\sigma^2} = 0 \cdot 2\sigma^2$

$\hookrightarrow (x-\mu_2)^2 - (x-\mu_1)^2 = 0$

$\hookrightarrow x^2 - 2x\mu_2 + \mu_2^2 - x^2 + 2x\mu_1 - \mu_1^2 = 0$

$\hookrightarrow (2x - \mu_1 - \mu_2)(\mu_1 - \mu_2) = 0$ (depends on "x")

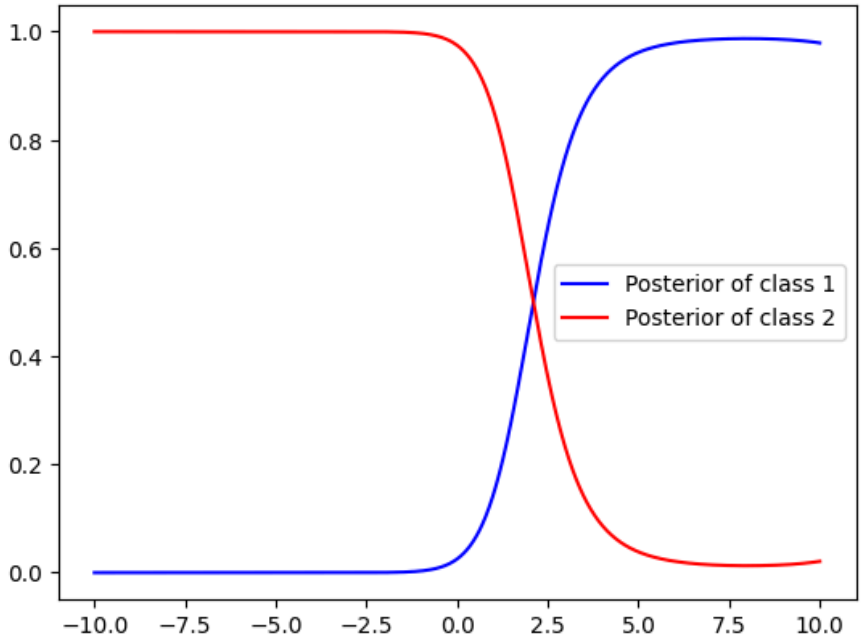
$\hookrightarrow 2x - \mu_1 - \mu_2 = 0$

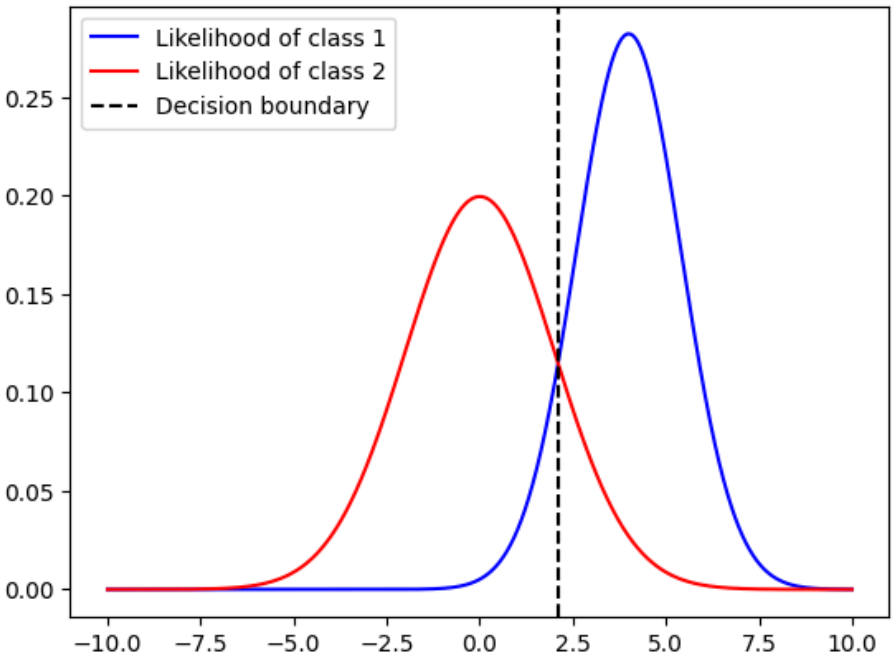
$X = \frac{\mu_1 + \mu_2}{2}$

The decision at $X = \frac{\mu_1 + \mu_2}{2}$ make the LRT = 1

OT3.

ANSWER

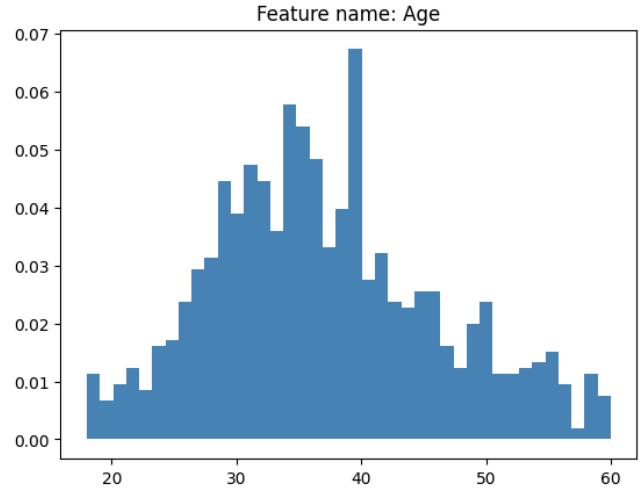


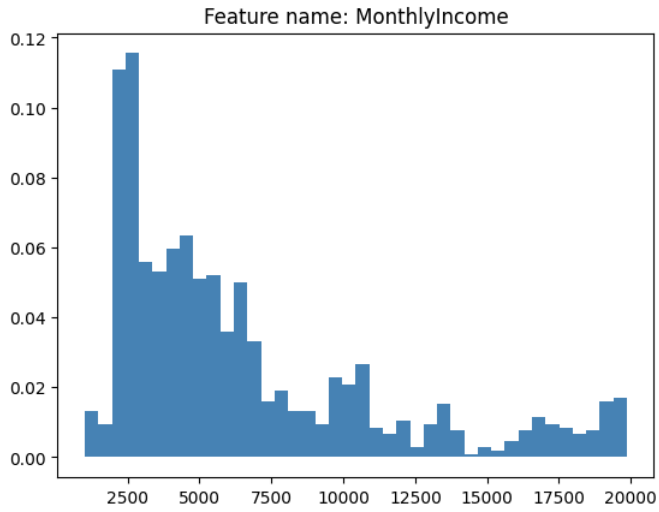


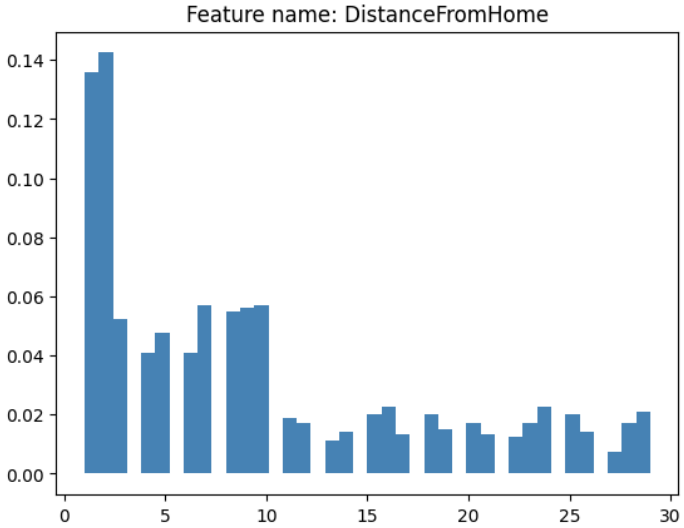
The decision boundary is at 2.103 with a likelihood ratio of 0.115.

T4.

ANSWER


Feature name: Age
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
20
30
40
50
60


Feature name: MonthlyIncome
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2500
5000
7500
10000
12500
15000
17500
20000


Feature name: DistanceFromHome
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
30

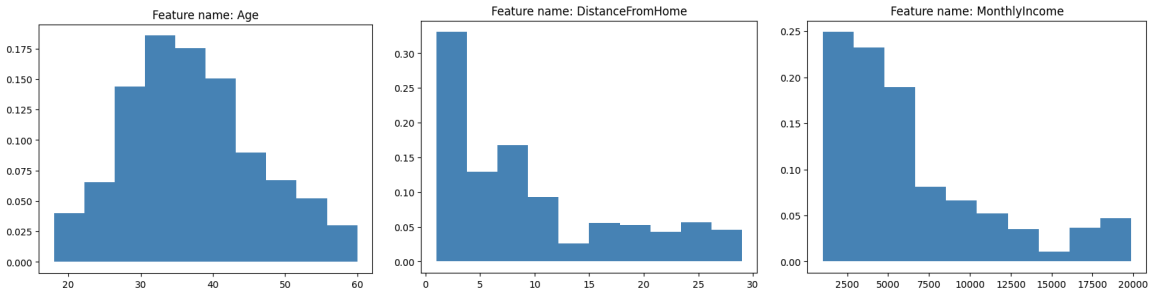
The number of zero bin are as follow:
- `Age`: 0
- `MonthlyIncome`: 0
- `DistanceFromHome`: 11

This is not a good discretization for `DistanceFromHome` because the number of zero bins is high compared to the total number of bins which is 40 and the percentage of zero bins has 27.5% of the total bins. For other features like `Age` and `MonthlyIncome`, it might be a good discretization because there are no zero bins in this setting (40 bins).

T5.

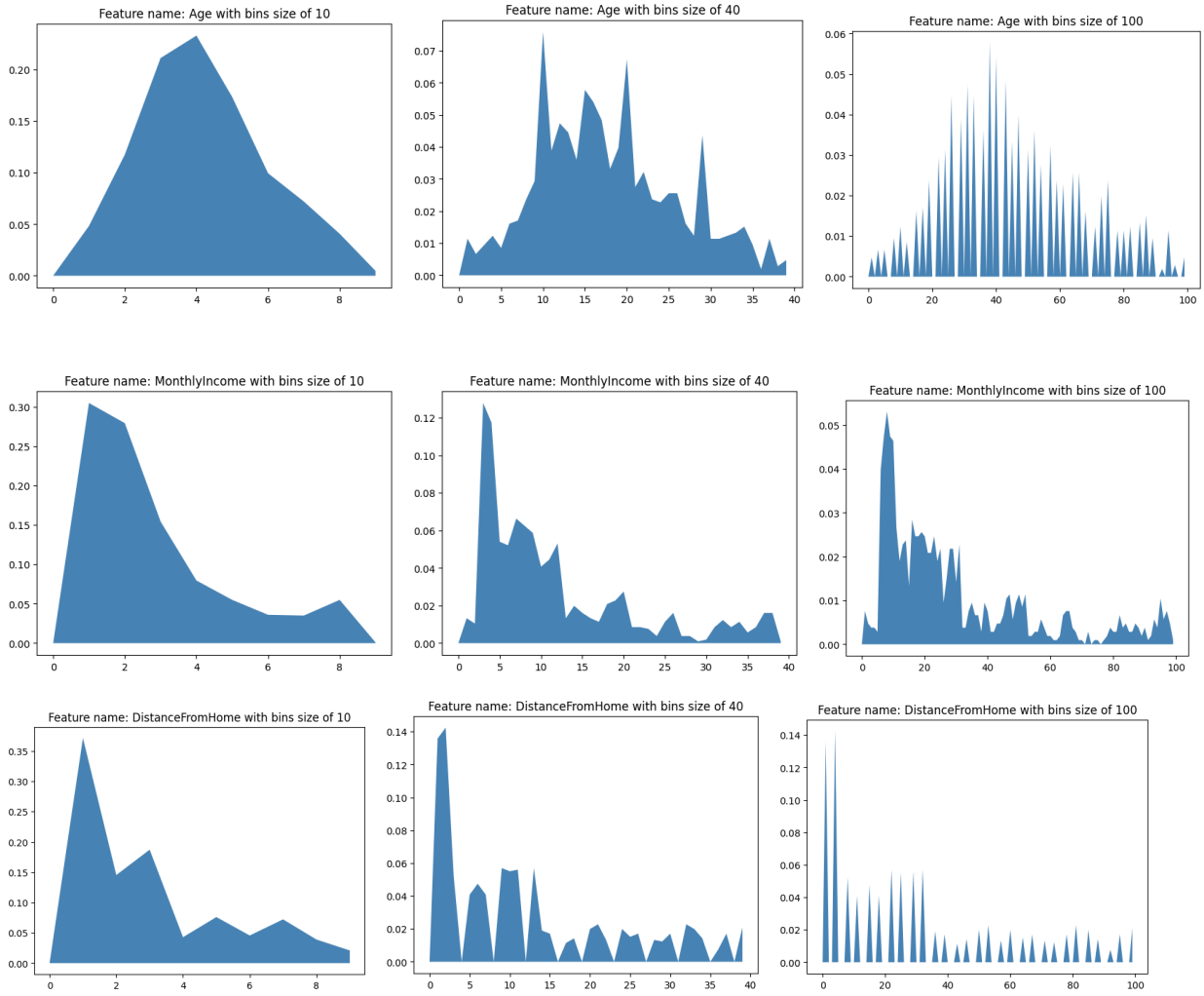
ANSWER

For `Age`, it might be possible to use Gaussian to estimate the histogram because the shape is very similar to a Gaussian distribution among other histograms. For example, `MonthlyIncome` and `DistanceFromHome` are not possible to estimate using only Gaussian. However, `MonthlyIncome` might be possible to estimate using Gaussian Mixture Model (GMM) since the shape looks right-skewed and similar to some known distribution. Lastly, the `DistanceFromHome` is not even possible to estimate using both Gaussian and GMM because its distribution is quite different from known distributions.



T6.

ANSWER



From all histograms, let's start analyze one by one:

- `Age`: The most sensible bin size in my opinion is 40 bins but where a larger bin like 100 bins causes a lot of zero bins. However, the 10 bins seems to be very small to capture the distribution of the age.
- `MonthlyIncome`: The most sensible bin size for this is 40 bins because the bins size larger than this will start to have zero bins.
- `DistanceFromHome`: The most sensible bin size for this is 10 bins which doesn't contain any zero bins.

NOTE: We don't want zero bins in our histogram when doing the estimation using probability. Therefore, the practical thing is if you have smaller data then use a smaller bin but if you can expand the bins size without increasing the zero bins then it's better to use larger bins size.

T7.

ANSWER

To select which feature should be discretized, I will try to find it. If there exist more than 10 unique values in a particular column then i will select it! However, this is just the starting point but we need to check the other factors as well after seeing the histogram. Moreover, it is likely that there will exist zero bins with features with higher numbers of unique values. But this is nothing when you have a lot of data that cover all these unique values. In this case, I will just ignore the number of unique values less than `10`.

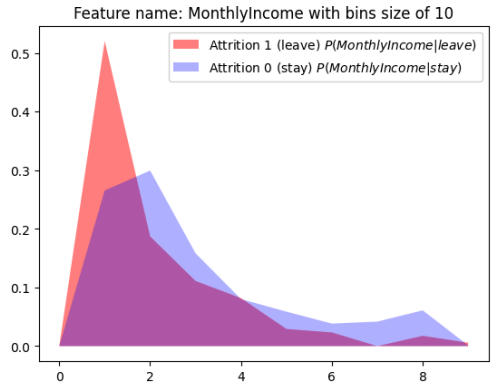
```
# Show how much uniques value in each column
cols = []
for col in df_train.columns:
  n_unique_val = len(df_train[col].unique())
  # Check if the number of unique value is less than 10
  if n_unique_val < 10:
     continue
  print(f"The number of unique value in {col} is {n_unique_val}")
  cols.append(col)

print(cols)
```

The images are in a notebook file which plots 14 features with a number of unique values greater than 10 for qualified features.

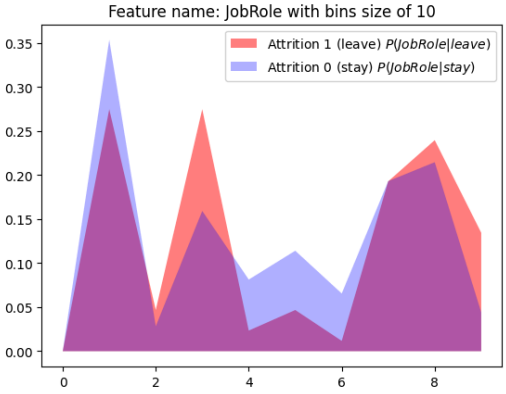
T8.

ANSWER

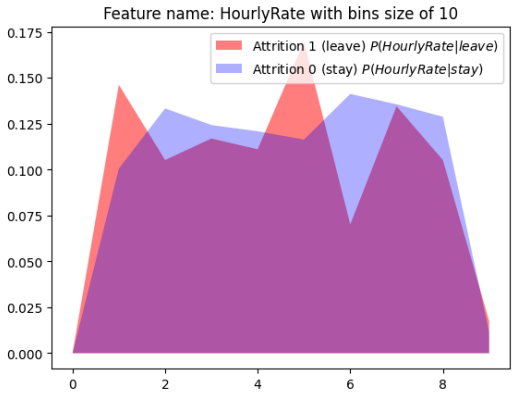
For the `MonthlyIncome`, the suitable distribution is the log-normal distribution, exponential distribution, or the gamma distribution because it is right-skewed distribution



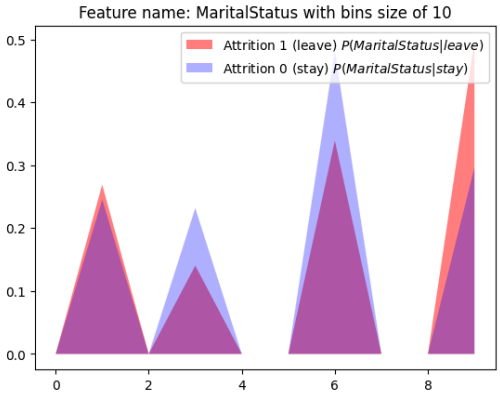
For `JobRole`, the distribution to model this histogram can be used Gaussian Mixture Model (GMM) to estimate this because there are multiple peaks in the histogram.



For `HourlyRate`, the proper distribution to model could be the Gaussian distribution with high variance or the uniform distribution (as we see from the figure).



For `MaritalStatus`, the distribution could be Gaussian Mixture Model since we can see the obvious peaks in the histogram.



T6.

Given the likelihood $P(X_i = x_i)$ which considered as our histogram distribution for feature (i). We can view this as "Multinomial Distribution". Since, given one feature in histogram we have multiple bins and we will define that X_i represents the frequency in specific bin and θ_i represents the probabily of that particular bin (same bin).

Thus, the likelihood of one feature is (Multinomial) $P(X) = \left(\frac{n!}{X_1! X_2! \cdots X_N!}\right) \cdot \theta_1^{X_1} \theta_2^{X_2} \cdots \theta_A^{X_A}$

MLE: $\arg\max_{\theta} L(\theta) = n! \cdot \prod_{i=1}^{N} \frac{\theta_i^{X_i}}{X_i!}$

$$\arg\min_{\theta} l(\theta) = \log(n!) + \sum_{i=1}^{N} X_i \log \theta_i - \sum_{i=1}^{N} X_i!$$

We introduce a constraint that $g = \sum_{i=1}^{N} \theta_i = 1$ (Sum of each bin's prob)

using Constraint Optimization (LaGrange): $\nabla f = \lambda \nabla g$

Compute ∇f: $\sum_{i=1}^{N} \frac{X_1}{\theta_i} \longrightarrow \left\langle \frac{X_1}{\theta_1}, \frac{X_2}{\theta_2}, \cdots, \frac{X_N}{\theta_N} \right\rangle$

∇g: 1, for all i $\longrightarrow \langle 1, 1, \cdots, 1 \rangle$

$$\hookrightarrow \sum_{i=1}^{N} \frac{X_i}{\theta_i} = \lambda(1) \rightarrow \sum_{i=1}^{N} \theta_i = \frac{\sum_{i=1}^{N} X_i}{\lambda}$$

$$\Rightarrow \sum_{i=1}^{N} \frac{X_i}{\lambda} = 1 \longrightarrow \lambda = \sum_{i=1}^{N} X_i$$

$$\Rightarrow \sum_{i=1}^{N} \theta_i = \frac{\sum_{i=1}^{N} X_i}{\lambda} \rightarrow \frac{X_1}{\lambda} + \frac{X_2}{\lambda} + \cdots + \frac{X_N}{\lambda} = 1$$

$$\Rightarrow \frac{X_1}{\sum X_i} + \frac{X_2}{\sum X_i} + \cdots + \frac{X_N}{\sum X_i} = 1$$

$$\therefore \theta_i = \frac{X_i}{\sum X_i}$$

T9.
ANSWER

```
n_leave = df_train[df_train["Attrition"] == 1].shape[0]
n_stay = df_train[df_train["Attrition"] == 0].shape[0]

prior_leave = n_leave / (n_leave + n_stay)
prior_stay = n_stay / (n_leave + n_stay)

print(f"The prior probability of leaving is {prior_leave:.3f} and the prior probability of staying is
{prior_stay:.3f}")
```

The prior probability of leaving is 0.161 and the prior probability of staying is 0.839

T10.
ANSWER

The method to fix this problem could be the **Flooring** which is adding the small value to the zero likelihood. This way can still help to not make the whole product term to be zero and preserve small probability(zero probability). Also, this method is quite easy to implement!

T11.
ANSWER

```
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

    # Get the features for the positive and negative classes
    pos_features = x[y == 1]
    neg_features = x[y == 0]

    n_feats = x.shape[1]

    # Calculate the params for each feature
    for i in range(n_feats):

        # Prepare features (Positive)
        pos_features_i = pos_features[:, i]
        pos_features_i = pos_features_i[~np.isnan(pos_features_i)]
        # Prepare features (Negative)
        neg_features_i = neg_features[:, i]
        neg_features_i = neg_features_i[~np.isnan(neg_features_i)]
```

```
        # Calculate max_val and min_val to be boundaries of the histogram(Positive)
        max_val_pos = max(pos_features_i)
        min_val_pos = min(pos_features_i)
        # Calculate max_val and min_val to be boundaries of the histogram(Negative)
        max_val_neg = max(neg_features_i)
        min_val_neg = min(neg_features_i)

        # Make a basis for the histogram (Positive)
        basis_pos = np.linspace(min_val_pos, max_val_pos, n_bins-1) # Subtract 1 because it start from left
edge
        # Make a basis for histogram (Negative)
        basis_neg = np.linspace(min_val_neg, max_val_neg, n_bins-1)

        # Append lower bound and upper bound for avoiding the edge case (Positive)
        basis_pos = np.append(basis_pos, 1e6)
        basis_pos = np.insert(basis_pos, 0, -1e6)
        # Append lower bound and upper bound for avoiding the edge case (Negative)
        basis_neg = np.append(basis_neg, 1e6)
        basis_neg = np.insert(basis_neg, 0, -1e6)

        # Discretize the features (Positive)
        digitized_pos = np.digitize(pos_features_i, basis_pos) # Default doesn't include right edge
        bins_pos = np.bincount(digitized_pos)[1:] # Since we don't want left edge which is [-inf, -1e6]
        # Discretize the features (Negative)
        digitized_neg = np.digitize(neg_features_i, basis_neg)
        bins_neg = np.bincount(digitized_neg)[1:]

        # Normalize the bins (Positive)
        bins_pos = bins_pos / sum(bins_pos)
        # Normalize the bins (Negative)
        bins_neg = bins_neg / sum(bins_neg)

        # Flooring the zero bins value (Positive)
        bins_pos = np.where(bins_pos == 0, 1e-6, bins_pos)
        # Flooring the zero bins value (Negative)
        bins_neg = np.where(bins_neg == 0, 1e-6, bins_neg)

        self.leave_params[i] = (bins_pos, basis_pos)
```

```
        self.stay_params[i] = (bins_neg, basis_neg)


    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

    # Loop through each feature's sample
    n_feats = x.shape[1]

    # Make a zeros array
    x_computed = np.zeros_like(x)

    for i in range(n_feats):

        # Get the bins and edges of the histogram for the feature
        bins_pos, basis_pos = self.leave_params[i]
        bins_neg, basis_neg = self.stay_params[i]

        # Check if there is null value if so drop the value from vector x_i
        x_i = x[:, i]
```

```
            nan_mask_x_i = np.isnan(x_i)
            x_i = x_i[~nan_mask_x_i]

            # Discretize the features
            digitized_pos = np.digitize(x_i, basis_pos)
            digitized_neg = np.digitize(x_i, basis_neg)

            # Get likelihood value w.r.t to bins
            likelihood_pos_i = bins_pos[digitized_pos-1]
            likelihood_neg_i = bins_neg[digitized_neg-1]

            # Calculate the difference between the likelihoods (log)
            log_likelihood_diff = np.log(likelihood_pos_i) - np.log(likelihood_neg_i)

            # Store the computed value
            x_computed[~nan_mask_x_i, i] = log_likelihood_diff
            x_computed[nan_mask_x_i, i] = 0

    # Sum for each sample and add the prior
    log_prior = np.log(self.prior_pos) - np.log(self.prior_neg)
    log_posterior = np.sum(x_computed, axis=1) + log_prior

    # Classify the sample based on the threshold in list
    y_pred = np.where(log_posterior > thresh, 1, 0).tolist()

    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

    # Get the features for the positive and negative classes
    pos_features = x[y == 1]
    neg_features = x[y == 0]

    n_feats = x.shape[1]

    # Calculate the params for each feature with assumed Gaussian distribution
    for i in range(n_feats):
        # Calculate the mean and standard deviation for each feature
        mean_pos = np.nanmean(pos_features[:, i])
        std_pos = np.nanstd(pos_features[:, i])
        mean_neg = np.nanmean(neg_features[:, i])
        std_neg = np.nanstd(neg_features[:, i])

        self.gaussian_leave_params[i] = (mean_pos, std_pos)
        self.gaussian_stay_params[i] = (mean_neg, std_neg)

    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

    # Loop through each feature's sample
    n_feats = x.shape[1]

    # Make a zeros array
    x_computed = np.zeros_like(x)

    for i in range(n_feats):

        # Get the mean and standard deviation for the feature
        mean_pos, std_pos = self.gaussian_leave_params[i]
        mean_neg, std_neg = self.gaussian_stay_params[i]

        # Check if there is null value if so drop the value from vector x_i
        x_i = x[:, i]
        nan_mask_x_i = np.isnan(x_i)
        x_i = x_i[~nan_mask_x_i]

        # Calculate the likelihood value w.r.t to Gaussian distribution
        likelihood_pos_i = stats.norm(mean_pos, std_pos).pdf(x_i)
        likelihood_neg_i = stats.norm(mean_neg, std_neg).pdf(x_i)

        # Calculate the difference between the likelihoods (log)
        log_likelihood_diff = np.log(likelihood_pos_i + 1e-9) - np.log(likelihood_neg_i + 1e-9)

        # Store the computed value
        x_computed[~nan_mask_x_i, i] = log_likelihood_diff

        # Set the value of null value to be 0
        x_computed[nan_mask_x_i, i] = 0

    # Sum for each sample and add the prior
    log_prior = np.log(self.prior_pos) - np.log(self.prior_neg)
    log_posterior = np.sum(x_computed, axis=1) + log_prior
```

```
    # Classify the sample based on the threshold in list
    y_pred = np.where(log_posterior > thresh, 1, 0).tolist()

    return y_pred
```

The **accuracy** is 0.850
The **precision** is 0.550
The **recall** is 0.458
The **F1** is 0.500
The **false positive rate** is 0.073

T12.
ANSWER

The **accuracy** is 0.810
The **precision** is 0.433
The **recall** is 0.542
The **F1** is 0.481
The **false positive rate** is 0.138

T13.
ANSWER

```
y_rand = np.random.randint(0, 2, size=len(y_test))

# Evaluate
evaluate(y_test, y_rand)
```

The **accuracy** is 0.578
The **precision** is 0.183
The **recall** is 0.458
The **F1** is 0.262
The **false positive rate** is 0.398

T14.
ANSWER

```
# Check which class appear the most in training set
unique, counts = np.unique(y_train, return_counts=True)
for u, c in zip(unique, counts):
  print(f"The class {u} appear {c} times in the training set")

# Make y_pred from majority class
majority_class = unique[np.argmax(counts)]
y_pred_majority = np.zeros(len(y_test)) + majority_class

# Evaluate
evaluate(y_test, y_pred_majority)
```

The **class 0.0** appear **1110** times in the training set
The **class 1.0** appear **213** times in the training set
The **accuracy** is 0.837
The **precision** is 0.000
The **recall** is 0.000
The **F1** is 0.000
The **false positive rate** is 0.000

T15.
ANSWER

The Naive Bayes classifier with **Histogram approach** is better than the **random choice** baseline and the **majority rule** baseline (a little bit). This is because the dataset is unbalanced so **the majority** rule will guarantee to have a high accuracy but the precision, recall, f1 and false positive rate are zero.

For the Naive Bayes classifier with **learned distributions approach** has a lower accuracy overall compared to Naive Bayes classifier with **Histogram approach**. However, this method perform better than the **random choice** but still has lower accuracy than the **majority rule**. Even though the accuracy is lower than the **majority rule** approach but the precision, recall, f1 and false positive rate are all better than the **majority rule** approach.

Thus, for the conclusion, the reason why **learned distributions approach** has lower accuracy than **Histogram approach** is because the **learned distributions approach** is assumed to be the Gaussian distribution which is not practical due to each features might not distribute as a Gaussian's fashion. However, the **Histogram approach** is more practical and can be used to model the distribution of each

feature with its own distribution but this approach still has a drawback which requires a lot of data points to fill the zero bin.

T16.
ANSWER

```
thesholds = np.arange(-5, 5, 0.05) # Vary threshold values

best_acc = 0
best_t = 1e-9
for t in thesholds:

  y_pred_t = model.predict(x_test, t)
  acc, _, _, _, _ = evaluate(y_test, y_pred_t, show_result=False)

  if acc > best_acc:
     best_acc = acc
     best_t = t

print(f"The best accuracy is {best_acc:.3f} with threshold of {best_t:.3f}")
```

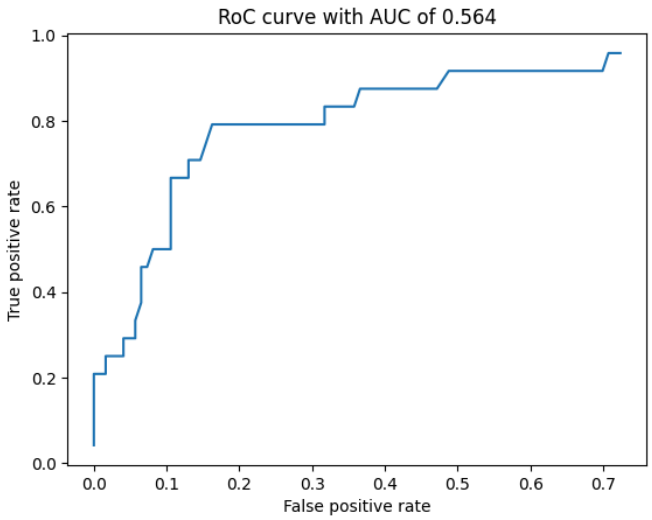
The **best accuracy** is 0.871 with **threshold** 2.300
The **best F1** is 0.604 with **threshold** -0.500

T17.
ANSWER

```
thresholds = np.arange(-5, 5, 0.05) # Vary threshold values

tpr_list = []
fpr_list = []
i = 0
for t in thresholds:

  y_pred_t = model.predict(x_test, t)
  acc_, _, tpr, _, fpr = evaluate(y_test, y_pred_t, show_result=False)

  tpr_list.append(tpr)
  fpr_list.append(fpr)
```

```
  i += 1
  if i % 20 == 0:
      print(f"Threshold {t:.3f} has TPR of {tpr:.3f} and FPR of {fpr:.3f} and accuracy of {acc_:.3f}")

# Compute the AUC
auc = -np.trapz(tpr_list, fpr_list)

plt.plot(fpr_list, tpr_list)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title(f"RoC curve with AUC of {auc:.3f}")
plt.show()
```

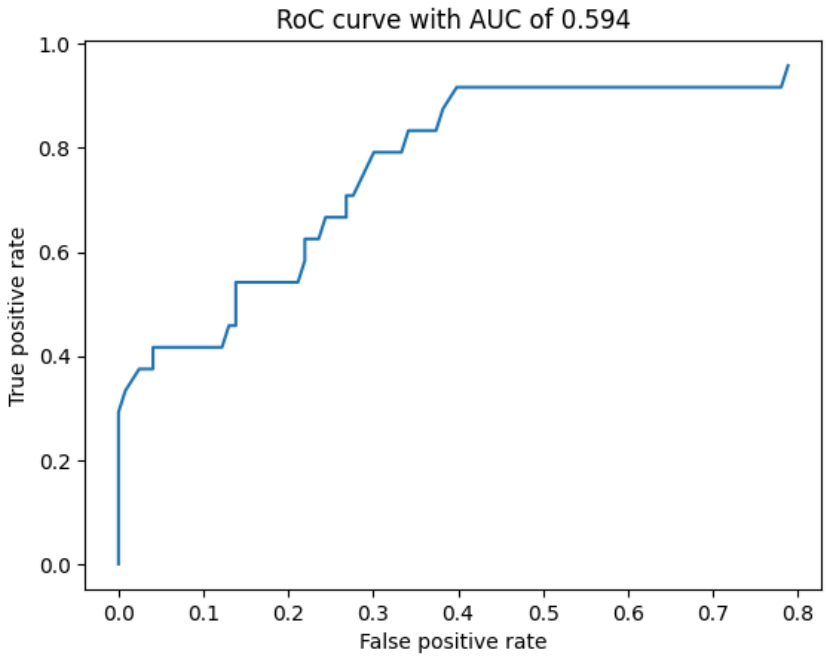
Threshold -4.050 has TPR of 0.917 and FPR of 0.545 and accuracy of 0.531
Threshold -3.050 has TPR of 0.875 and FPR of 0.398 and accuracy of 0.646
Threshold -2.050 has TPR of 0.792 and FPR of 0.244 and accuracy of 0.762
Threshold -1.050 has TPR of 0.708 and FPR of 0.138 and accuracy of 0.837
Threshold -0.050 has TPR of 0.458 and FPR of 0.073 and accuracy of 0.850
Threshold 0.950 has TPR of 0.250 and FPR of 0.024 and accuracy of 0.857
Threshold 1.950 has TPR of 0.208 and FPR of 0.008 and accuracy of 0.864
Threshold 2.950 has TPR of 0.125 and FPR of 0.000 and accuracy of 0.857
Threshold 3.950 has TPR of 0.125 and FPR of 0.000 and accuracy of 0.857
Threshold 4.950 has TPR of 0.042 and FPR of 0.000 and accuracy of 0.844



T18.

ANSWER

Threshold -4.050 has TPR of 0.917 and FPR of 0.691 with accuracy of 0.408
Threshold -3.050 has TPR of 0.917 and FPR of 0.537 with accuracy of 0.537
Threshold -2.050 has TPR of 0.792 and FPR of 0.333 with accuracy of 0.687
Threshold -1.050 has TPR of 0.542 and FPR of 0.154 with accuracy of 0.796
Threshold -0.050 has TPR of 0.417 and FPR of 0.049 with accuracy of 0.864
Threshold 0.950 has TPR of 0.250 and FPR of 0.000 with accuracy of 0.878
Threshold 1.950 has TPR of 0.167 and FPR of 0.000 with accuracy of 0.864
Threshold 2.950 has TPR of 0.042 and FPR of 0.000 with accuracy of 0.844
Threshold 3.950 has TPR of 0.000 and FPR of 0.000 with accuracy of 0.837
Threshold 4.950 has TPR of 0.000 and FPR of 0.000 with accuracy of 0.837



As I can see from the RoC curve, the number of discretization bins of 5 is better than the number of discretization bins of 10. This is because the AUC of the RoC curve of 5 bins is higher than the AUC of the RoC curve of 10 bins. Therefore, the number of discretization bins of 5 is better than the number of discretization bins of 10. However, this is only for this dataset with this particular train_test_split.

T19.
ANSWER

Submitted!
OT4.
ANSWER

```
T = 10
acc_result = []

for i in range(T):
  # Load the dataset
  (x_train, y_train), (x_test, y_test) = load_dataset()

  # Create a model
  model = SimpleBayesClassifier(n_pos = np.sum(y_train == 1), n_neg = np.sum(y_train == 0))

  # Fit data to model
  model.fit_params(x_train, y_train, n_bins=10)

  # Inference
  y_pred = model.predict(x_test)

  # Evaluate
  acc, _, _, _, _ = evaluate(y_test, y_pred, show_result=False)

  # Append the accuracy
  acc_result.append(acc)

print(f"The mean accuracy is {np.mean(acc_result):.3f} with variance of {np.var(acc_result):.6f}")
```

The **mean accuracy** is 0.845 with variance of 0.000359