Homework 2 MLE and Naive Bayes

6232035721 Saenyakorn Siangsanoh

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

$$y2 = \alpha y1 + w1$$

$$y1 = \alpha y0 + w0$$

where the noises w_0, w_1 are iid with the distribution $N(0, \sigma^2), y_0 \sim N(0, \lambda)$ 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 $p(y_2, y_1, y_0|\alpha)$

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 $p(y_n|y_{n-1}, y_{n-2}, \dots) = p(y_n|y_{n-1})$. In other words, the present is independent of the past $(y_{n-2}, y_{n-3}, \dots)$, 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.

We know that $p(y_2, y_1, y_0|\alpha) = p(y_2|y_1)p(y_1|y_0)p(y_0|\alpha)$

And

$$y_0 \sim N(0, \lambda)$$

$$y_1 \sim N(\alpha y_0, \sigma^2)$$

$$y_2 \sim N(\alpha y_1, \sigma^2)$$

So,

$$\begin{aligned} p(y_2, y_1, y_0|\alpha) &= (\frac{1}{2\sqrt{2\pi}} e^{\frac{-1}{2}\frac{(y_2-\alpha y_1)^2}{\sigma^2}})(\frac{1}{2\sqrt{2\pi}} e^{\frac{-1}{2}\frac{(y_1-\alpha y_0)^2}{\sigma^2}})(\frac{1}{2\sqrt{2\pi}} e^{\frac{-1}{2}\frac{(y_0-0)^2}{\lambda}}) \\ &= (\frac{1}{2\sqrt{2\pi}})^3 exp(\frac{-1}{2}(\frac{(y_2-\alpha y_1)^2}{\sigma^2} + \frac{(y_1-\alpha y_0)^2}{\sigma^2} + \frac{y_0^2}{\lambda}))) \end{aligned}$$

To find argmax, take $\frac{d}{d\alpha} log(p(y_2, y_1, y_0|\alpha)) = 0$

$$\frac{d}{d\alpha}(\frac{(y_2-\alpha y_1)^2}{\sigma^2}+\frac{(y_1-\alpha y_0)^2}{\sigma^2}+\frac{y_0^2}{\lambda})=0$$
$$\frac{2(-y_1)(y_2-\alpha y_1)}{\sigma}+\frac{2(-y_0)(y_1-\alpha y_0)}{\sigma}=0$$
$$(y_2y_1-\alpha y_1^2)+(y_1y_0-\alpha y_0^2)=0$$
$$\alpha(y_2^2+y_1^2)=y_2y_1+y_1y_0$$
$$\alpha=\frac{y_2y_1+y_1y_0}{y_2^2+y_1^2}$$

OT1

Consider the general case, where

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

Find the MLE given the observed price $y_{N+1},y_N,\dots,y_0$

From **T1**,

To find argmax of $p(y_{n+1},y_n,\dots,y_0|\alpha)$, take $\frac{d}{d\alpha}log(p(y_{n+1},y_n,\dots,y_0|\alpha))=0$

$$\frac{d}{d\alpha}(\frac{(y_{n+1}-\alpha y_n)^2}{\sigma^2}+\frac{(y_n-\alpha y_{n-1})^2}{\sigma^2}+\dots+\frac{y_0^2}{\lambda})=0$$
$$(y_{n+1}y_n-\alpha y_n^2)+(y_ny_{n-1}-\alpha y_{n-1}^2)+\dots+(y_1y_0-\alpha y_0^2)=0$$
$$\frac{y_{n+1}y_n+y_ny_{n-1}+\dots+y_1y_0}{y_{n+1}^2+y_n^2+\dots+y_1^2}=\alpha$$

$\alpha=\frac{y_{n+1}y_n+y_ny_{n-1}+\dots+y_1y_0}{y_{n+1}^2+y_n^2+\dots+y_1^2}$

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)

$P(x|w1)=N(5,2)$

$P(x|w2)=N(0,2)$

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.

In [957…

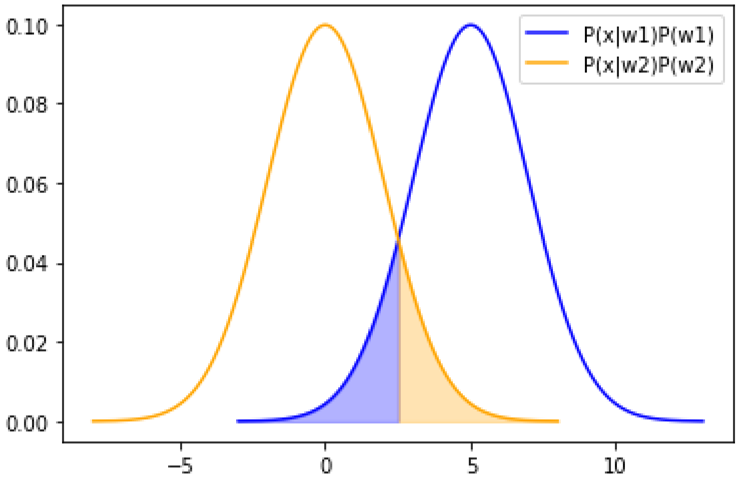
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd
```

```python
import math
from sklearn.model_selection import train_test_split
```

In [958…

```python
def plot_bayes_classifier(mu_1, mu_2, vars_1, vars_2, p_w1, p_w2, intersect_
  # Assume same variance for both classes
  x_1 = np.linspace(mu_1 - 4*vars_1, mu_1 + 4*vars_1, 10000)
  y_1 = stats.norm.pdf(x_1, mu_1, vars_1) * p_w1
  x_2 = np.linspace(mu_2 - 4*vars_2, mu_2 + 4*vars_2, 10000)
  y_2 = stats.norm.pdf(x_2, mu_2, vars_2) * p_w2

  if(intersect_x is None):
    intersect_x = (2*vars_1**2*math.log(p_w2) - 2*vars_2**2*math.log(p_w1) 
  print("intersect: x = ", intersect_x)

  # Plot the graph
  plt.plot(x_1, y_1, label='P(x|w1)P(w1)', color='blue')
  plt.fill_between(x_1[x_1 < intersect_x], y_1[x_1 < intersect_x], color='b
  plt.plot(x_2, y_2, label='P(x|w2)P(w2)', color='orange')
  plt.fill_between(x_2[x_2 > intersect_x], y_2[x_2 > intersect_x], color='o
  plt.legend()
  plt.show()

# Assume P(w1) = 0.5 and P(w2) = 0.5
vars = 2
mu_1 = 5
mu_2 = 0
p_w1 = 0.5
p_w2 = 0.5
plot_bayes_classifier(mu_1, mu_2, vars, vars, p_w1, p_w2)
```

```
intersect: x =  2.5
```



Since we assume priors are equal, that mean $P(w_1) = P(w_2)$

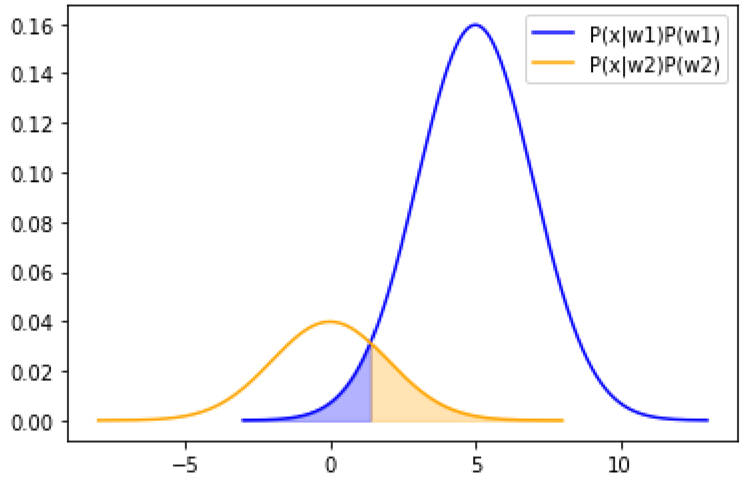
T3

What happen to the decision boundary if the cat is happy with a prior of 0.8?

In [959…

```python
vars = 2
mu_1 = 5
mu_2 = 0
p_w1 = 0.8
p_w2 = 0.2
plot_bayes_classifier(mu_1, mu_2, vars, vars, p_w1, p_w2)
```

```
intersect: x =  1.3909645111040876
```



OT2

For the ordinary case of $P(x|w1)$ = $N(\mu 1, \sigma 2)$, $P(x|w2)$ = $N(\mu 2, \sigma 2)$, $p(w1) = p(w2) = 0.5$, prove that the decision boundary is at $x = \frac{\mu_1+\mu_2}{2}$

If the student changed his model to

$P(x|w1) = N(5, 2)$

$P(x|w2) = N(0, 4)$

Answer

1. Prove $x = \frac{\mu_1+\mu_2}{2}$

   Let $y_1 = P(x|w_1) = \frac{1}{\sigma_1\sqrt{2\pi}}e^{\frac{-1}{2}\frac{(x-\mu_1)^2}{\sigma_1^2}}$

   And $y_2 = P(x|w_2) = \frac{1}{\sigma_2\sqrt{2\pi}}e^{\frac{-1}{2}\frac{(x-\mu_2)^2}{\sigma_2^2}}$

   Since they're interect, then $y_1 = y_2$ find x

   Assume $\sigma_1 = \sigma_2$, So

$$
\begin{aligned}
e^{\frac{-1}{2}\frac{(x-\mu_1)^2}{\sigma_1^2}} &= e^{\frac{-1}{2}\frac{(x-\mu_2)^2}{\sigma_2^2}} \\
\frac{-1}{2}\frac{(x-\mu_1)^2}{\sigma_1^2} &= \frac{-1}{2}\frac{(x-\mu_2)^2}{\sigma_2^2} \\
(x-\mu_1)^2 &= (x-\mu_2)^2 \\
x^2 - 2\mu_1 x + \mu_1^2 &= x^2 - 2\mu_2 x + \mu_2^2 \\
x &= \frac{\mu_1+\mu_2}{2}
\end{aligned}
$$

   Q.E.D

2. If the student changed his model to

$P(x|w1) = N(5,2)$

$P(x|w2) = N(0,4)$

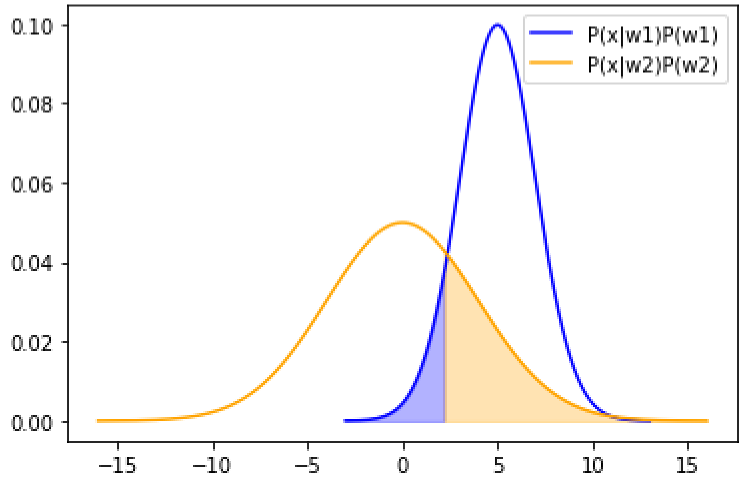
Solve $\frac{1}{2\sqrt{2\pi}}e^{\frac{-1}{2}\frac{(x-5)^2}{2^2}} = \frac{1}{4\sqrt{2\pi}}e^{\frac{-1}{2}\frac{(x-0)^2}{4^2}}$

the intersection x = $10 - \sqrt{50 + 16\log 2}$

In [960…

```
vars_1 = 2
vars_2 = 4
mu_1 = 5
mu_2 = 0
p_w1 = 0.5
p_w2 = 0.5
intersect_x = 10 - (50 + 16*math.log(2))**0.5
plot_bayes_classifier(mu_1, mu_2, vars_1, vars_2, p_w1, p_w2, intersect_x=i
```

```
intersect: x =  2.1839680854695116
```



Employee Attrition Prediction

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

https://www.kaggle.com/pavansubhashtibm-hr-analytics-attrition-dataset/home

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 $\Pi_i p(x_i|class)$ 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% of the is `Attrition == yes` and 10% of the is `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`

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

In [961…

```
# Gather training data
data_url = "https://raw.githubusercontent.com/ekapolc/pattern_2022/main/HW/
all_data = pd.read_csv(data_url)
```

In [962…

```
# Convert to codes
all_data.loc[all_data["Attrition"] == "Yes", "Attrition"] = 1.0
all_data.loc[all_data["Attrition"] == "No", "Attrition"] = 0.0
```

In [963…

```
# Drop unnecessary columns
all_data = all_data.drop(columns = ["EmployeeNumber"])
```

In [964…

```
# Separate data into training and testing
X_train, X_test, y_train, y_test = train_test_split(all_data, all_data["Att
print("All_data: ", all_data.shape)
print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
print("y_train: ", y_train.shape)
print("y_test: ", y_test.shape)
```

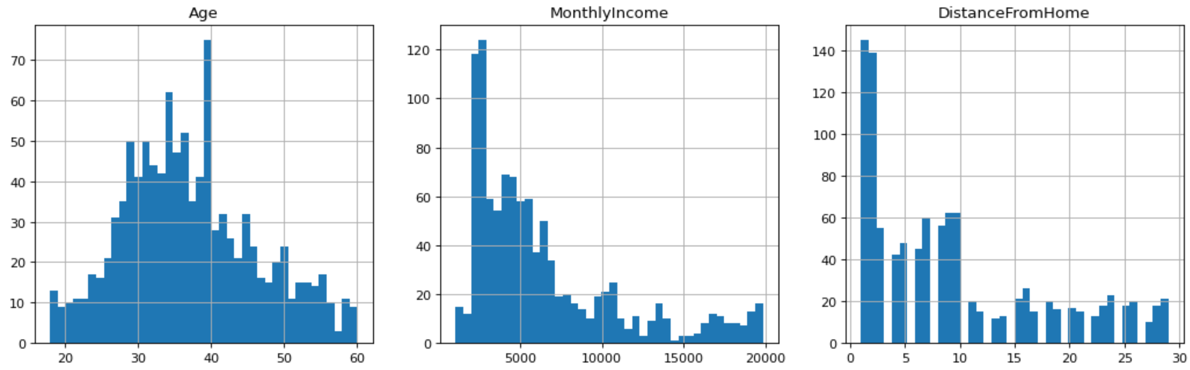
```
All_data:  (1470, 35)
X_train:  (1323, 35)
X_test:  (147, 35)
y_train:  (1323,)
y_test:  (147,)
```

T4

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

In [965…

```
bins = 40
cols = ["Age", "MonthlyIncome", "DistanceFromHome"]
plt.figure(figsize=(16, 10), dpi=80)
for i in range(len(cols)):
  col = cols[i]
  ax = plt.subplot(2, 3, i+1)
  ax.title.set_text(col)
  X_train[col].hist(bins=bins)
```



Answer

มีเพียง "DistanceFromHome" ที่มี bin ที่ไม่มีค่าอยู่ในนั้น ซึ่งคือ discretization ไม่ดี เพราะอาจจะทำให้การทำนายตอนทำ model ผิดพลาดได้

T5

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

Answer

สำหรับ Age ใช้ Gaussian ได้ แต่สำหรับ MonthlyIncome กับ DistanceFromHome น่าจะไม่ได้ (น่าจะเป็น beta distribution) และไม่น่าใช้ GMM ได้ เพราะยังไม่ค่อยชัดเจนว่ามีมากกว่า 1 Gaussian ใน histogram

In [966…

```
# Bin each values into bins
def bin_data(data: pd.Series, bins=10):
  step = (data.max() - data.min()) / bins
  bin_edge = np.block(
    [
      -np.inf,
      np.arange(data.min(), data.max(), step),
      np.inf
    ])
```

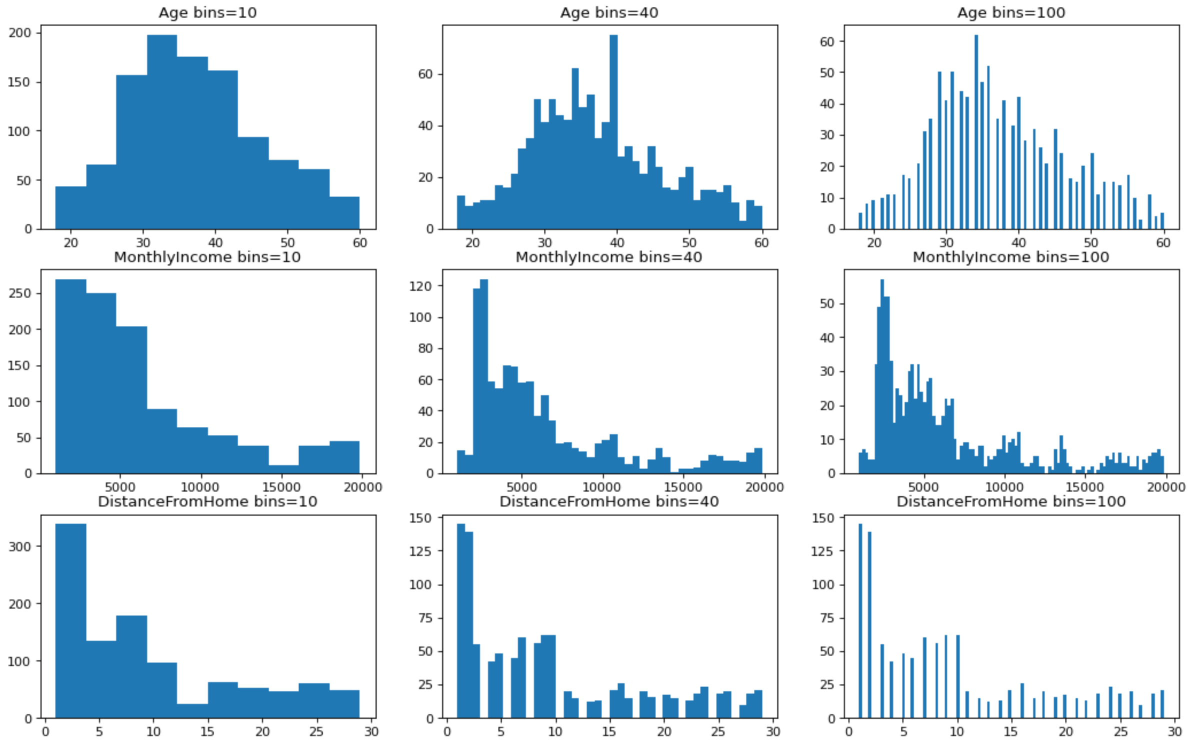
```
    data = data[~np.isnan(data)]
    inds = np.digitize(data, bin_edge)
    reault = np.bincount(inds) / np.sum(np.bincount(inds))
    return reault, bin_edge
```

T6

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

In [967…

```
bins = [10, 40, 100]
cols = ["Age", "MonthlyIncome", "DistanceFromHome"]
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(16, 10), dpi=80)
for i in range(len(bins)):
  for j in range(len(cols)):
    axs[j][i].set_title(f"{cols[j]} bins={bins[i]}")
    axs[j][i].hist(X_train[cols[j]], bins=bins[i])
```



Answer

สำหรับ Age, bins ที่เหมาะสมจะเป็น 10 เพราะสามารถบอกการกระจายข้อมูลได้ละเอียดพอดี ไม่หยาบและไม่ละเอียดจนเกินไป

สำหรับ MonthlyIncome, bins ที่เหมาะสมจะอยู่ที่ 40 เพราะข้อมูลมีการกระจายตัวมาก

สำหรับ DistanceFromHome, bins ที่เหมาะสมคือ 10 เพราะข้อมูล DistanceFromHome จะอยู่แค่ในช่วงแคบ ๆ ไม่จำเป็นต้องใช้ bins จำนวนเยอะ ๆ

T7

For the rest of the features, which one should be discretized? 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.

The MLE for the likelihood distribution of discretized histograms

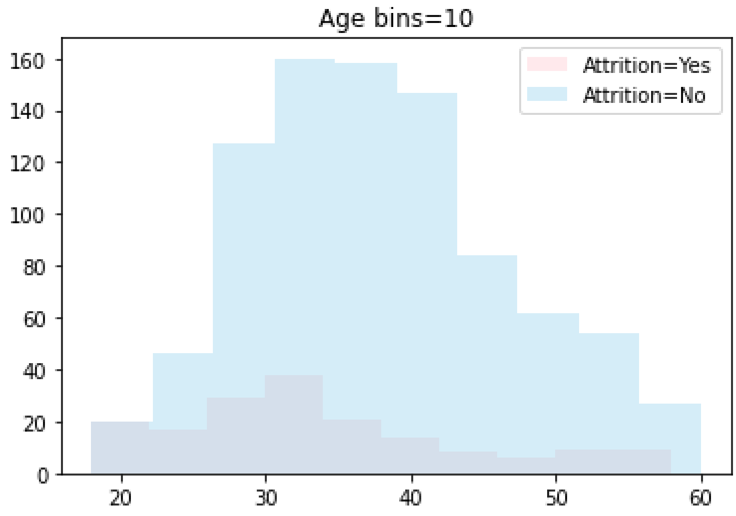
We would like to build a Naive Bayes classifier which compares the posterior $p(leave|x_i)$ against $p(stay|x_i)$. However, figuring out $p(class|x_i)$ is often hard (not true for this case). Thus, we turn to the likelihood $p(x_i|class)$, which can be derived from the discretized histograms.

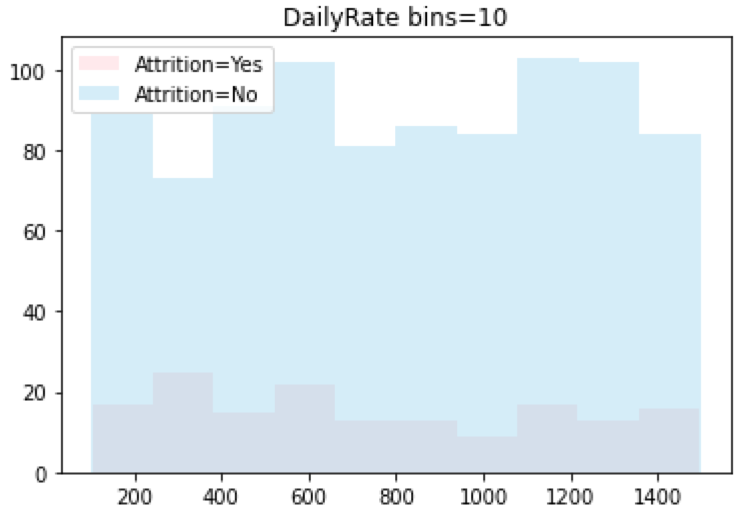
Answer

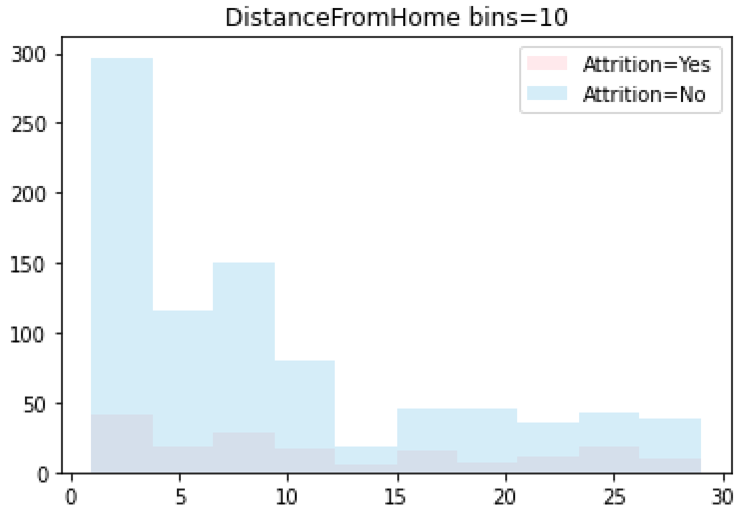
ทุก columns ที่ "น่าจะ" มีความหมายต่อ model (เกือบทุก columns) จะมีบาง columns ที่ไม่ได้ใช้ เช่น `EmployeeNumber` เพราะเป็นแค่เลขบัตรประจำตัว หรือ `EmployeeCount` ที่มี distinct value อยู่ค่าเดียว หรือ `PerformanceRating` ที่มี distinct value อยู่ 2 ค่าและไม่น่าเกี่ยวกับโจทย์

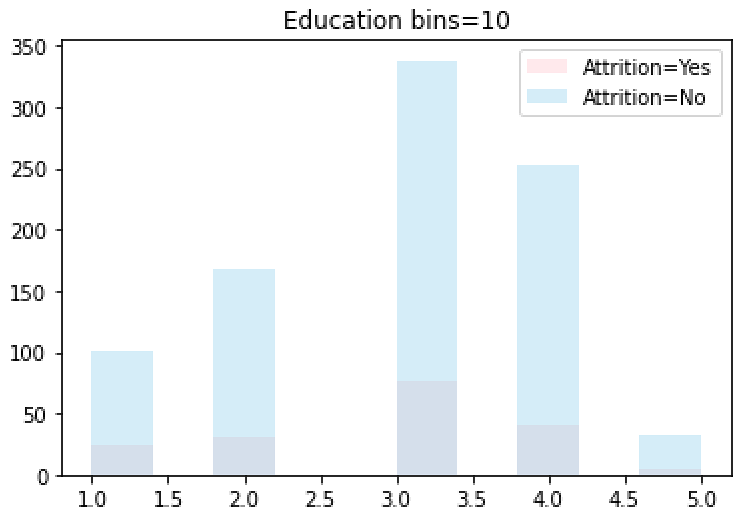
In [968…

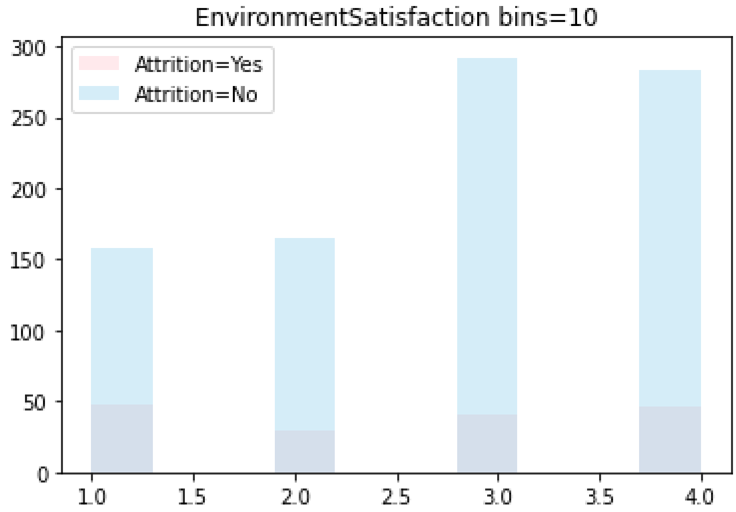
```
cols = ["Age", "DailyRate", "DistanceFromHome", "Education", "EnvironmentSa
        "JobLevel", "JobSatisfaction", "MonthlyIncome", "MonthlyRate", "Per
        "WorkLifeBalance", "YearsAtCompany", "YearsInCurrentRole", "YearsSi

X_train_leave = X_train.loc[X_train["Attrition"] == 1.0].copy()
X_train_stay  = X_train.loc[X_train["Attrition"] == 0.0].copy()
for i, col in enumerate(cols):
    plt.title(f"{col} bins=10")
    likelihoods_leave, bin_edges_leave = bin_data(X_train_leave[col], bins=
    plt.hist(X_train_leave[col], bins=10, color="pink", alpha=0.35, label="
    likelihoods_stay, bin_edges_stay = bin_data(X_train_stay[col], bins=10)
    plt.hist(X_train_stay[col],  bins=10, color="skyblue", alpha=0.35, labe
    plt.legend()
    plt.show()
```

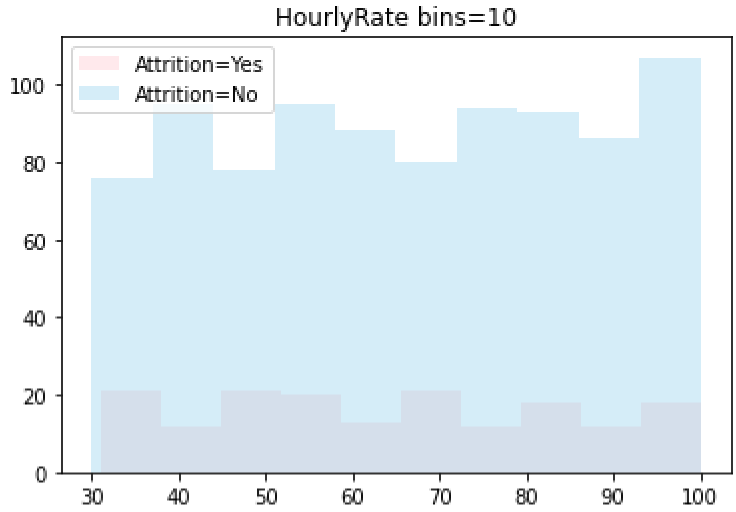


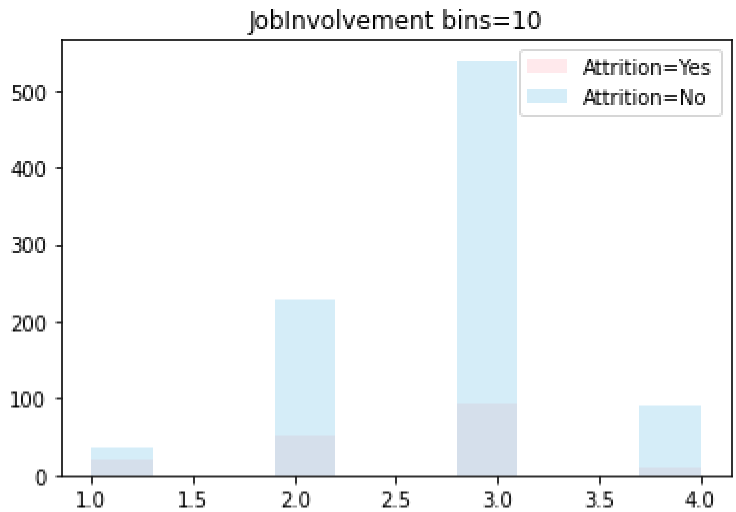

DailyRate bins=10
Attrition=Yes
Attrition=No
0
20
40
60
80
100
200
400
600
800
1000
1200
1400

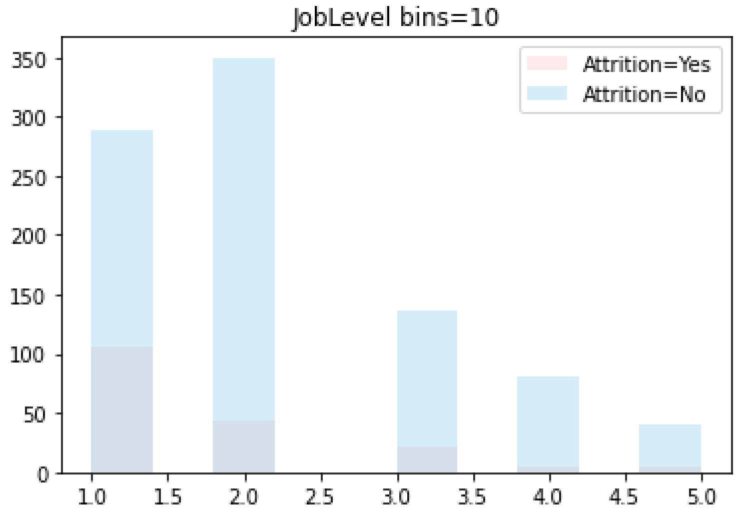

DistanceFromHome bins=10
Attrition=Yes
Attrition=No
0
50
100
150
200
250
300
0
5
10
15
20
25
30

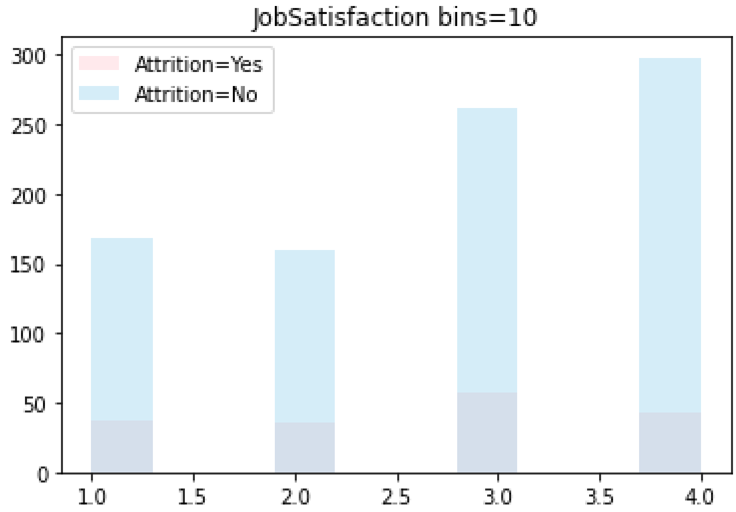

Education bins=10
Attrition=Yes
Attrition=No
0
50
100
150
200
250
300
350
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

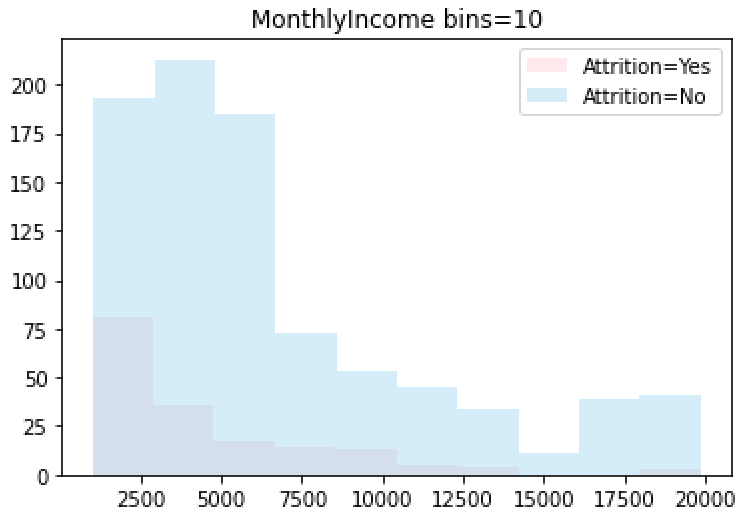

EnvironmentSatisfaction bins=10
Attrition=Yes
Attrition=No
0
50
100
150
200
250
300
1.0
1.5
2.0
2.5
3.0
3.5
4.0

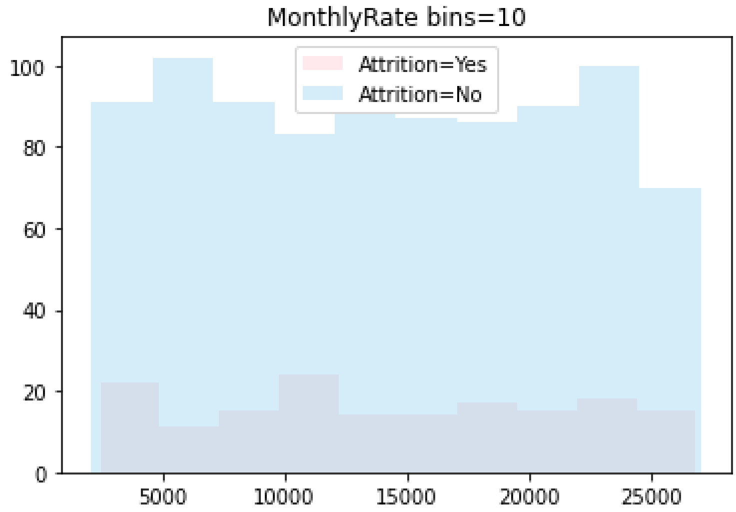

HourlyRate bins=10
Attrition=Yes
Attrition=No
0
20
40
60
80
100
30
40
50
60
70
80
90
100

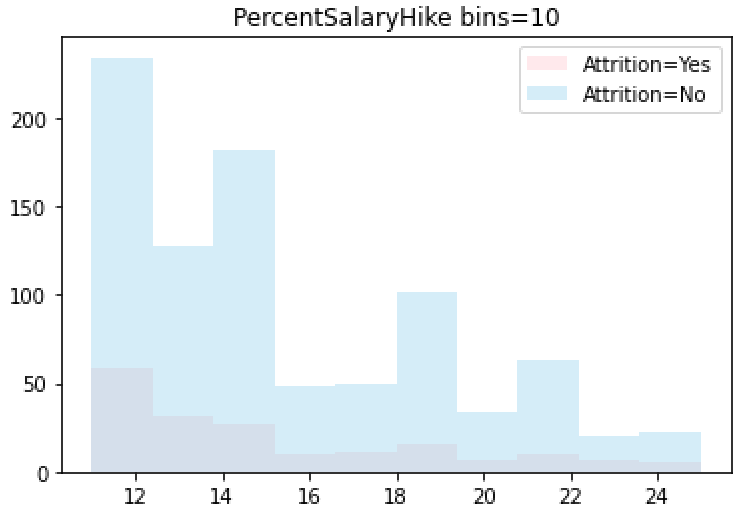

JobInvolvement bins=10
Attrition=Yes
Attrition=No
0
100
200
300
400
500
1.0
1.5
2.0
2.5
3.0
3.5
4.0

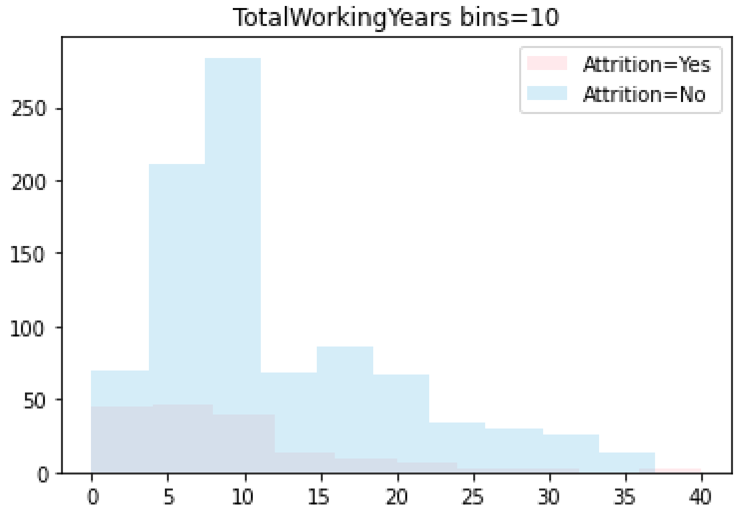

JobLevel bins=10
Attrition=Yes
Attrition=No
350
300
250
200
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

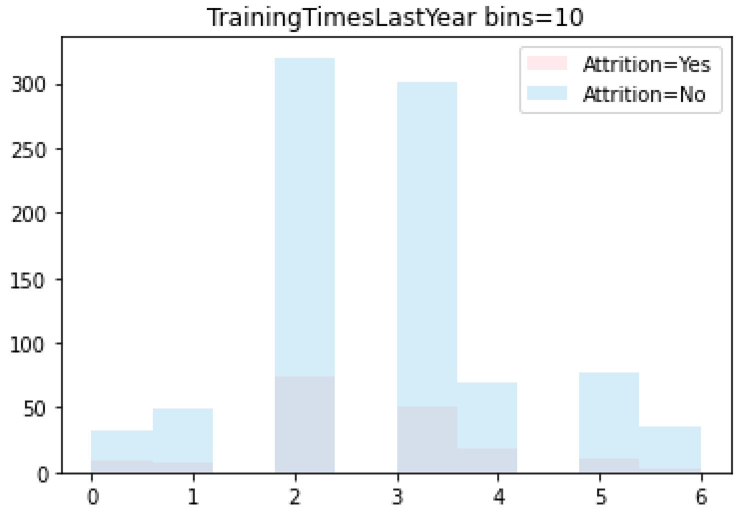

JobSatisfaction bins=10
Attrition=Yes
Attrition=No
300
250
200
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

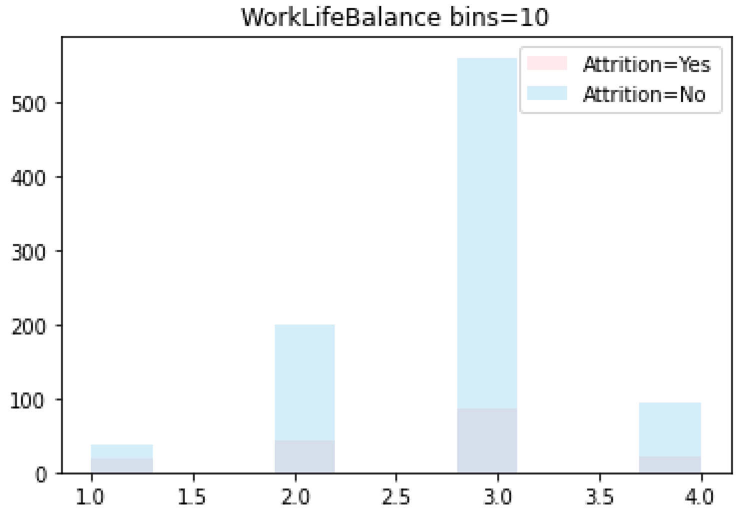

MonthlyIncome bins=10
Attrition=Yes
Attrition=No
200
175
150
125
100
75
50
25
0
2500
5000
7500
10000
12500
15000
17500
20000

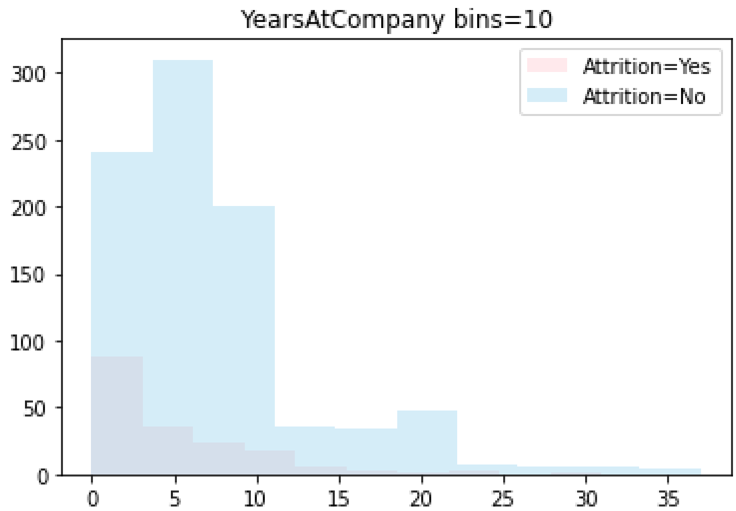

MonthlyRate bins=10
Attrition=Yes
Attrition=No
0
20
40
60
80
100
5000
10000
15000
20000
25000

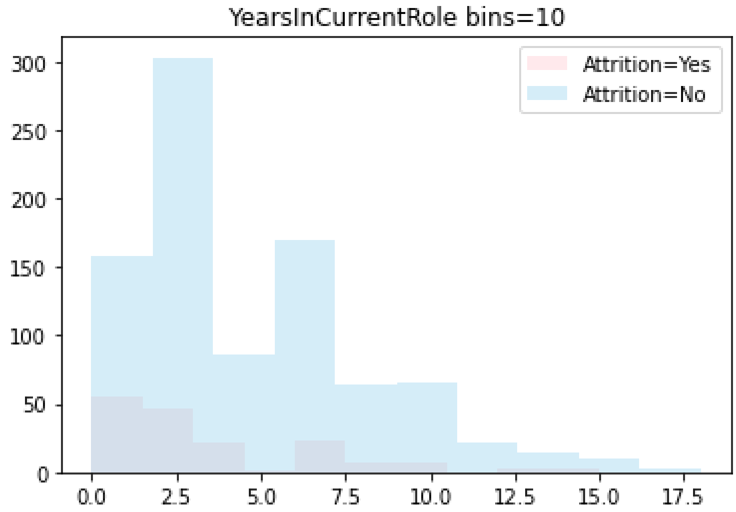

PercentSalaryHike bins=10
Attrition=Yes
Attrition=No
0
50
100
150
200
12
14
16
18
20
22
24

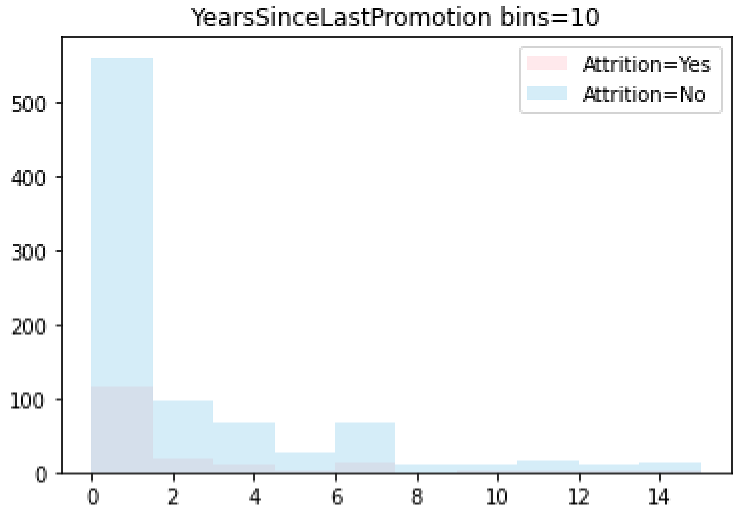

TotalWorkingYears bins=10
Attrition=Yes
Attrition=No
0
50
100
150
200
250
0
5
10
15
20
25
30
35
40

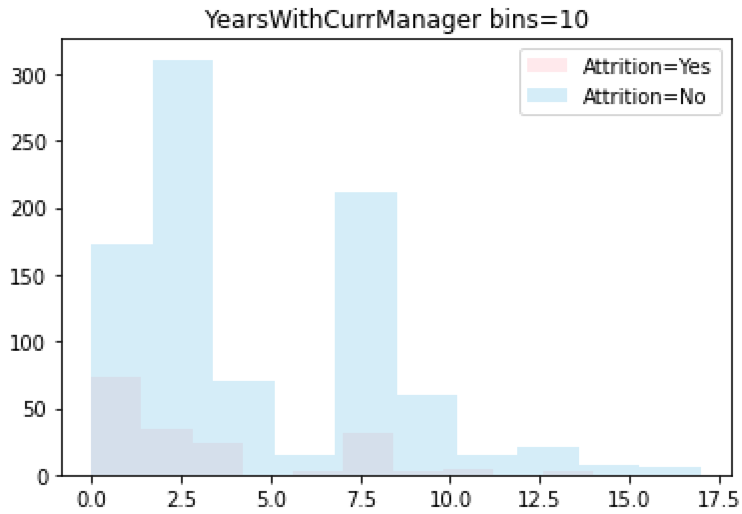

TrainingTimesLastYear bins=10
Attrition=Yes
Attrition=No


WorkLifeBalance bins=10
Attrition=Yes
Attrition=No


YearsAtCompany bins=10
Attrition=Yes
Attrition=No







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.

Answer

ใช้ Multinomial Distribution เพราะเราสามรถแสดงในรูปของ Histogram โดยแบ่งเป็นส่วน ๆ

โดย MLE ของ Multinomial Distribution คือ $p_i = \frac{x_i}{N}$ เมื่อ x_i คือ จำนวนข้อมูลใน bins นั้นและ N คือจำนวนข้อมูลทั้งหมด

T9

What is the prior distribution of the two classes?

In [969…

```
# Find prior distribution
def get_prior(df, _class=0):
  return df.loc[df["Attrition"] == _class, "Attrition"].count() / df.shape[

p_leave = get_prior(X_train, 1)
p_stay = get_prior(X_train, 0)
print("Prior distribution (Yes):", p_leave)
print("Prior distribution (No):", p_stay)
```

```
Prior distribution (Yes): 0.16099773242630386
Prior distribution (No): 0.8390022675736961
```

Naive Bayes classification

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

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

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,

$$lH(x) = logp(leave) - logp(stay) + \Sigma_{i=1}[logp(xi|leave) - logpp(xi|stay)]$$

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.

Answer

ใช้ 2 วิธีได้แก่

1. Interpolate likelihood คือถ้าสมมติว่า likelihood ตรงนี้เป็น zero count เราจะเอา bin ข้าง ๆ (ซ้ายขวา) มาเฉลี่ยกัน
2. Flooring คือถ้าใช้วิธีในข้อ 1 แล้วยังเป็น 0 อยู่ใช้แทนที่ด้วยค่าน้อย ๆ แทน (เช่น 1e-10)

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

In [970…

```
# Define Bayes classifier
class NaiveBayes:
  def __init__(self, X_train: pd.DataFrame, X_test: pd.DataFrame,
               y_train: pd.DataFrame, y_test: pd.DataFrame,
               features: list, bins: list):
    self.p_leave = get_prior(X_train, 1)
    self.p_stay = get_prior(X_train, 0)
    self.X_train = X_train.copy()
    self.X_test = X_test.copy()
    self.y_train = y_train.copy()
    self.y_test = y_test.copy()
    self.features = features.copy()
    self.bins = bins.copy()

  def train(self):
    self.likelihoods = {}
    self.bin_edges = {}
    self.mu = {}
    self.std = {}
    for _class in np.unique(self.y_train):
      self.likelihoods[_class] = {}
      self.bin_edges[_class] = {}
      self.mu[_class] = {}
      self.std[_class] = {}
      X_train_class = self.X_train.loc[self.y_train == _class]
      for i, col in enumerate(self.features):
        likelihoods, bin_edges = bin_data(X_train_class[col], bins=self.bin
        # Non parametric
        self.likelihoods[_class][col] = likelihoods
        self.bin_edges[_class][col] = bin_edges
        # Parametric (Gaussian)
        self.mu[_class][col] = X_train_class[col].mean()
        self.std[_class][col] = X_train_class[col].std()

  def interpolate_likelihood(self, likelihood, bin_index):
    if likelihood[bin_index] < 1e-6:
        lh = (likelihood[bin_index-1]+likelihood[bin_index+1])/2
    else:
      lh = likelihood[bin_index]
    if lh == 0:
      lh = 1e-20 # Flooring to avoid log(0)
    return lh

  def predict(self):
    predictions_np = []
    predictions_p = []
    for row in range(self.X_test.shape[0]):
      p_np = p_p = np.log(self.p_leave) - np.log(self.p_stay)
      for _, feature in enumerate(self.features):
        x = self.X_test.iloc[row][feature]
        if math.isnan(x):
          continue
        # Non parametric
        # Class 0
        bin_edges_leave = self.bin_edges[1][feature]
        likelihood_leave = self.likelihoods[1][feature]
        bin_index_leave = np.searchsorted(bin_edges_leave, x, side="right")
```

```
          # Class 1
          bin_edges_stay  = self.bin_edges[0][feature]
          likelihood_stay = self.likelihoods[0][feature]
          bin_index_stay = np.searchsorted(bin_edges_stay, x, side="right")
          # Interpolate likelihood
          likelihood_stay = self.interpolate_likelihood(likelihood_stay, bin_
          likelihood_leave = self.interpolate_likelihood(likelihood_leave, bi
          # Calculate log likelihood
          p_np += np.log(likelihood_leave) - np.log(likelihood_stay)
          # Parametric (Gaussian)
          likelihood_leave = stats.norm(self.mu[1][feature], scale=self.std[1
          likelihood_stay = stats.norm(self.mu[0][feature], scale=self.std[0]
          p_p += np.log(likelihood_leave) - np.log(likelihood_stay)
        predictions_np.append(p_np)
        predictions_p.append(p_p)
      df = pd.DataFrame({
        "Attrition NP": predictions_np,
        "Attrition P": predictions_p
      })
      self.prediction = df.copy()
      return df
```

In [971…

```
class Reporter:
  def __init__(self, data: np.ndarray, target: np.ndarray, threshold: float
    self.data = data.copy()
    self.target = target.copy()
    self.threshold = threshold
    self.data[self.data >  self.threshold] = 1
    self.data[self.data <= self.threshold] = 0
    self.tp = ((self.data == 1) & (self.target == 1)).sum()
    self.fp = ((self.data == 1) & (self.target == 0)).sum()
    self.tn = ((self.data == 0) & (self.target == 0)).sum()
    self.fn = ((self.data == 0) & (self.target == 1)).sum()

  def find_accuracy(self):
    predictions = self.data
    accuracy = (predictions == self.target).sum() / self.target.shape[0]
    self.accuracy = accuracy
    return accuracy

  def find_precision(self):
    denominator = self.tp + self.fp
    precision = 1e-20 if denominator == 0 else self.tp / denominator
    self.precision =  precision
    return precision

  def find_recall(self):
    denominator = self.tp + self.fn
    recall = 1e-20 if denominator == 0 else self.tp / denominator
    self.recall =  recall
    return recall

  def find_f1_score(self):
    precision = self.find_precision()
    recall = self.find_recall()
    f1_score = 2 * precision * recall / (precision + recall)
    self.f1_score = f1_score
    return f1_score

  def find_true_positive_rate(self):
    denominator = self.tp + self.fn
    tpr = 1e-20 if denominator == 0 else self.tp / denominator
    self.tpr =  tpr
```

```
    return tpr

  def find_false_positive_rate(self):
    denominator = self.fp + self.tn
    fpr = 1e-20 if denominator == 0 else self.fp / denominator
    self.fpr =  fpr
    return fpr

  def report(self):
    print("True Positive:", self.tp)
    print("False Positive:", self.fp)
    print("True Negative:", self.tn)
    print("False Negative:", self.fn)
    print("Accuracy:", self.find_accuracy())
    print("Precision:", self.find_precision())
    print("Recall:", self.find_recall())
    print("F1 Score:", self.find_f1_score())
    print("True Positive Rate:", self.find_true_positive_rate())
    print("False Positive Rate:", self.find_false_positive_rate())
```

Answer

เพื่อให้เอกสารไม่รก คำตอบของข้อ 11 จะอยู่รวมกับข้อ 12 โดย

NP จะหมายถึง Non Parametric หรือก็คือทำ Naive Bayes ด้วย Histogram หรือ Multinomial distribution

ส่วน P จะหมายถึง Parametric หรือก็คือทำ Naive Bayes โดยการ Assume ว่า features ทั้งหมดกระจายตัวแบบ normal

ใน Code ข้างบนเราจะ train ทั้ง NP และ P ไปทีเดียวเลย ใน Class `NaiveBayes` ส่วน class `Reporter` คือ Class ที่เอาไว้หา metric ต่าง ๆ จาก test set

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.

In [972…

```
features = ["Age", "MonthlyIncome", "DistanceFromHome", "MonthlyRate",
            "YearsAtCompany", "YearsSinceLastPromotion", "HourlyRate", "Per
            "TotalWorkingYears", "YearsInCurrentRole", "DailyRate", "YearsS
            "YearsWithCurrManager", "WorkLifeBalance", "TrainingTimesLastYe
            "NumCompaniesWorked", "JobSatisfaction", "JobLevel", "JobInvolv
            "EnvironmentSatisfaction", "Education"]
bins = [12, 100, 10, 55,
        4, 5, 5, 5,
```

```
        4, 4, 100, 5,
        5, 4, 3, 4,
        3, 4, 5, 4,
        4, 5]
classifier = NaiveBayes(X_train, X_test, y_train, y_test, features, bins)
classifier.train()
result = classifier.predict()

print("Answer Report (NP)")
reporter = Reporter(result["Attrition NP"].to_numpy(), y_test, threshold=0)
reporter.report()
print("True:", reporter.data.sum())
print("False:", reporter.data.shape[0]-reporter.data.sum())

print("-------")

print("Answer Report (P)")
reporter = Reporter(result["Attrition P"].to_numpy(), y_test, threshold=0)
reporter.report()
print("True:", reporter.data.sum())
print("False:", reporter.data.shape[0]-reporter.data.sum())
```

```
Answer Report (NP)
True Positive: 7
False Positive: 11
True Negative: 112
False Negative: 17
Accuracy: 0.8095238095238095
Precision: 0.3888888888888889
Recall: 0.2916666666666667
F1 Score: 0.3333333333333333
True Positive Rate: 0.2916666666666667
False Positive Rate: 0.08943089430894309
True: 18.0
False: 129.0
-------
Answer Report (P)
True Positive: 7
False Positive: 27
True Negative: 96
False Negative: 17
Accuracy: 0.7006802721088435
Precision: 0.20588235294117646
Recall: 0.2916666666666667
F1 Score: 0.2413793103448276
True Positive Rate: 0.2916666666666667
False Positive Rate: 0.21951219512195122
True: 34.0
False: 113.0
```

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.

In [973…

```
mock = np.random.randint(2, size=y_test.shape[0])

print("Random choice")
reporter = Reporter(mock, y_test, threshold=0)
reporter.report()
```

```
Random choice
True Positive: 11
False Positive: 66
True Negative: 57
False Negative: 13
Accuracy: 0.46258503401360546
Precision: 0.14285714285714285
Recall: 0.4583333333333333
F1 Score: 0.2178217821782178
True Positive Rate: 0.4583333333333333
False Positive Rate: 0.5365853658536586
```

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.

In [974…

```
print("Majority rule (NO)")
mock = np.array([0] * y_test.shape[0])
reporter = Reporter(mock, y_test)
reporter.report()
```

```
Majority rule (NO)
True Positive: 0
False Positive: 0
True Negative: 123
False Negative: 24
Accuracy: 0.8367346938775511
Precision: 1e-20
Recall: 0.0
F1 Score: 0.0
True Positive Rate: 0.0
False Positive Rate: 0.0
```

T15

Compare the two baselines with your Naive Bayes classifier.

In [975…

```
# Compare baseline model with random choice and majority rule
mock = np.random.randint(2, size=y_test.shape[0])
print("-------\nRandom choice")
reporter = Reporter(mock, y_test, threshold=0)
reporter.report()
print("-------\nMajority rule (NO)")
mock = np.array([0] * y_test.shape[0])
reporter = Reporter(mock, y_test)
reporter.report()
```

```
-------
Random choice
True Positive: 10
False Positive: 63
True Negative: 60
False Negative: 14
Accuracy: 0.47619047619047616
Precision: 0.136986301369863
Recall: 0.4166666666666667
F1 Score: 0.2061855670103093
True Positive Rate: 0.4166666666666667
False Positive Rate: 0.5121951219512195
-------
Majority rule (NO)
True Positive: 0
False Positive: 0
True Negative: 123
False Negative: 24
Accuracy: 0.8367346938775511
Precision: 1e-20
Recall: 0.0
F1 Score: 0.0
True Positive Rate: 0.0
False Positive Rate: 0.0
```

In [976…

```
## Comparing the two models NP and P
print("-------\nNB Non parametric")
reporter = Reporter(result["Attrition NP"].to_numpy(), y_test, threshold=0)
reporter.report()
print("-------\nNB parametric")
reporter = Reporter(result["Attrition P"].to_numpy(), y_test, threshold=0)
reporter.report()
```

```
-------
NB Non parametric
True Positive: 7
False Positive: 11
True Negative: 112
False Negative: 17
Accuracy: 0.8095238095238095
Precision: 0.3888888888888889
Recall: 0.2916666666666667
F1 Score: 0.3333333333333333
True Positive Rate: 0.2916666666666667
False Positive Rate: 0.08943089430894309
-------
NB parametric
True Positive: 7
False Positive: 27
True Negative: 96
False Negative: 17
Accuracy: 0.7006802721088435
Precision: 0.20588235294117646
Recall: 0.2916666666666667
F1 Score: 0.2413793103448276
True Positive Rate: 0.2916666666666667
False Positive Rate: 0.21951219512195122
```

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)

In [977…

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

predictions = classifier.predict()

# First is for NP, second is for P
_types = ["Attrition NP", "Attrition P"]
max_accuracy = [0, 0]
max_f1_score = [0, 0]
max_corresponding_threshold = [0, 0]

for t in thresholds:
  for i, _type in enumerate(_types):
    reporter = Reporter(result[_type].to_numpy(), y_test, threshold=t)
    accuracy = reporter.find_accuracy()
    f1_score = reporter.find_f1_score()
    if(accuracy > max_accuracy[i]):
      max_corresponding_threshold[i] = t

for i, _type in enumerate(_types):
  reporter = Reporter(result[_type].to_numpy(), y_test, threshold=t)
  print("Best for Class:", _type)
  print("Threshold:", max_corresponding_threshold[i])
  print("Accuracy:", reporter.find_accuracy())
  print("F1 Score:", reporter.find_f1_score())
  print("Precision:", reporter.find_precision())
  print("Recall:", reporter.find_recall())
  print("-------")
```

```
Best for Class: Attrition NP
Threshold: 4.949999999999964
Accuracy: 0.8367346938775511
F1 Score: 0.0
Precision: 1e-20
Recall: 0.0
-------
Best for Class: Attrition P
Threshold: 4.949999999999964
Accuracy: 0.8367346938775511
F1 Score: 0.0
Precision: 1e-20
Recall: 0.0
-------
```

Receiver Operating Characteristic (RoC) curve

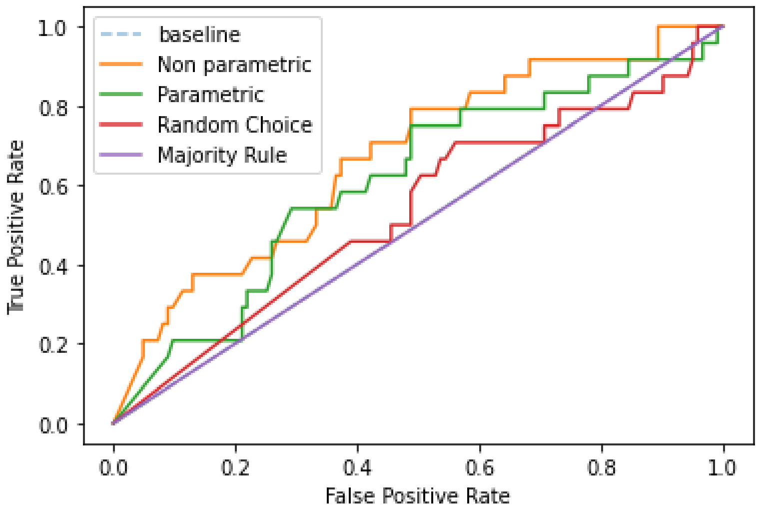
The recall rate (true positive rate) and the false alarm rate can change as we vary the threshold. The false alarm rate will deteriorate as we decrease the threshold (more false alarms). On the other hand, the recall rate will improve. This is also another trade-off machine learning practitioners need 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

T17

Plot the RoC of your classifier.

In [978…

```
# Plot RoC curve
baseline_random_choice = np.random.rand(y_test.shape[0]) * 10 - 5
baseline_majority_rule = np.array([0] * y_test.shape[0])

def plot_roc(prediction: pd.DataFrame, y_test):
  thresholds = np.arange(-60, 5, 0.05)
  plt.plot([0, 1], [0, 1], "--", label="baseline", alpha=0.5)
  datas = [
    prediction["Attrition NP"].to_numpy(),
    prediction["Attrition P"].to_numpy(),
    baseline_random_choice,
    baseline_majority_rule
  ]
  labels = ["Non parametric", "Parametric", "Random Choice", "Majority Rule
  for i, data in enumerate(datas):
    sensitivities = []
    specificities = []
    for t in thresholds:
      reporter = Reporter(data, y_test, threshold=t)
      sensitivities.append(reporter.find_true_positive_rate())
      specificities.append(reporter.find_false_positive_rate())
    plt.plot(specificities, sensitivities, label=labels[i])
  plt.xlabel("False Positive Rate")
  plt.ylabel("True Positive Rate")
  plt.legend()
  plt.show()

plot_roc(classifier.prediction.copy(), y_test)
```

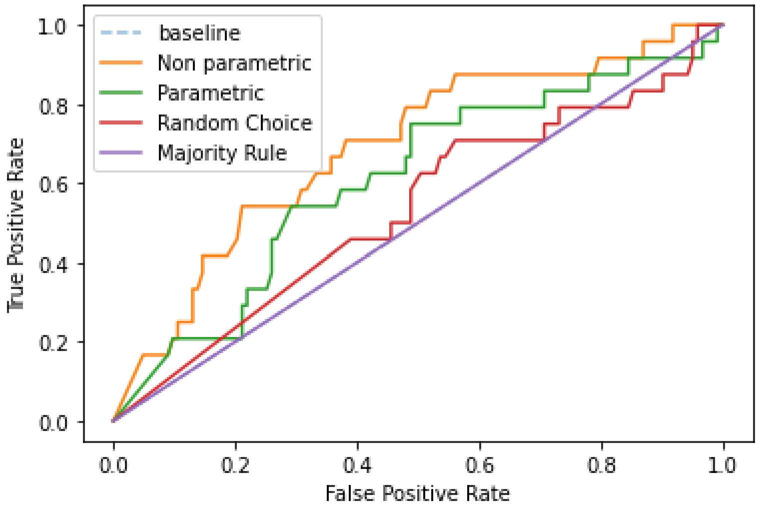


T18

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.

In [979…

```
features = ["Age", "MonthlyIncome", "DistanceFromHome", "MonthlyRate",
            "YearsAtCompany", "YearsSinceLastPromotion", "HourlyRate", "Per
            "TotalWorkingYears", "YearsInCurrentRole", "DailyRate", "YearsS
            "YearsWithCurrManager", "WorkLifeBalance", "TrainingTimesLastYe
            "NumCompaniesWorked", "JobSatisfaction", "JobLevel", "JobInvolv
            "EnvironmentSatisfaction", "Education"]
bins = [5, 5, 10, 5,
        4, 5, 5, 5,
        4, 4, 100, 5,
        5, 4, 3, 4,
        3, 4, 5, 4,
        4, 5]
classifier_2 = NaiveBayes(X_train, X_test, y_train, y_test, features, bins)
classifier_2.train()
result = classifier_2.predict()

plot_roc(result, y_test)
```



Answer

จะเห็นว่า Model แบบใหม่จะมีพื้นที่ AUC ใหญ่กว่า Model แบบเก่า ซึ่งหมายความว่า Model ใหม่นี้ดีกว่า Model แบบเก่า

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.

Answer

Submitted!

(Optional) Classifier Variance

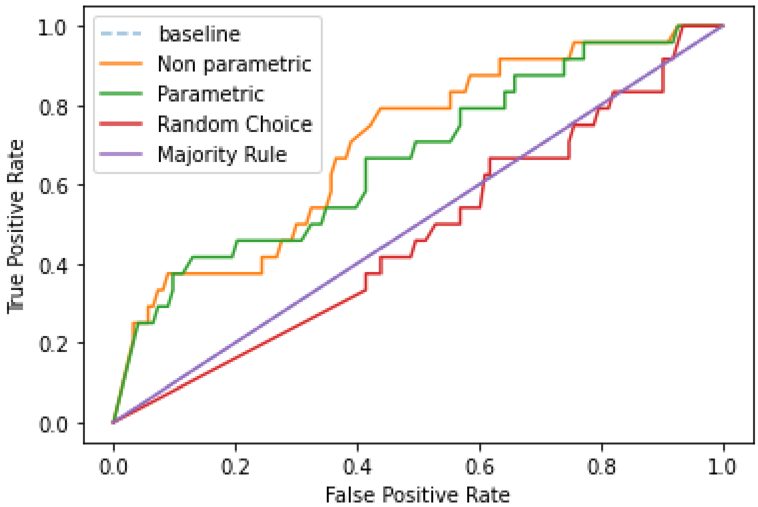
Recall, in class, we talked about the variance of a classifier as the training set changes. In this section, we will evaluate our model if we shuffle the training and test data. This will give a measure whether our recognizer is good just because we are lucky (and give statistical significance to our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

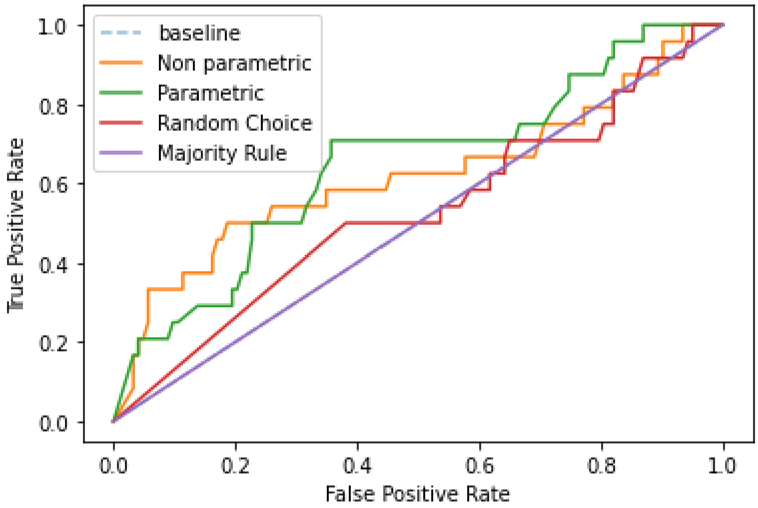
OT3

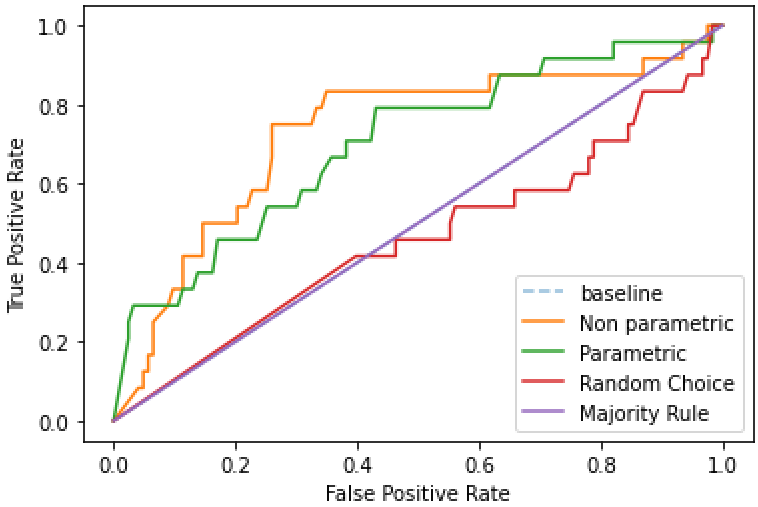
Shuffle the database, and create new test and train sets. Redo the entire training and evaluation process 10 times (each time with a new training and test set). Calculate the mean and variance of the accuracy rate.

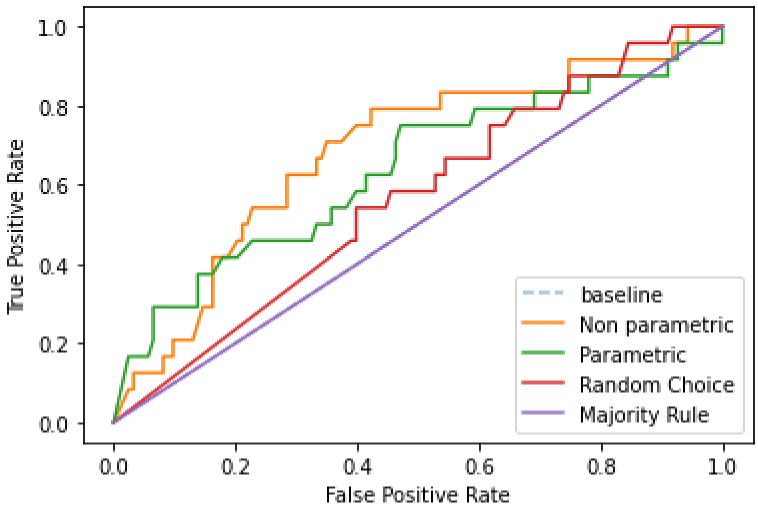
In [980…

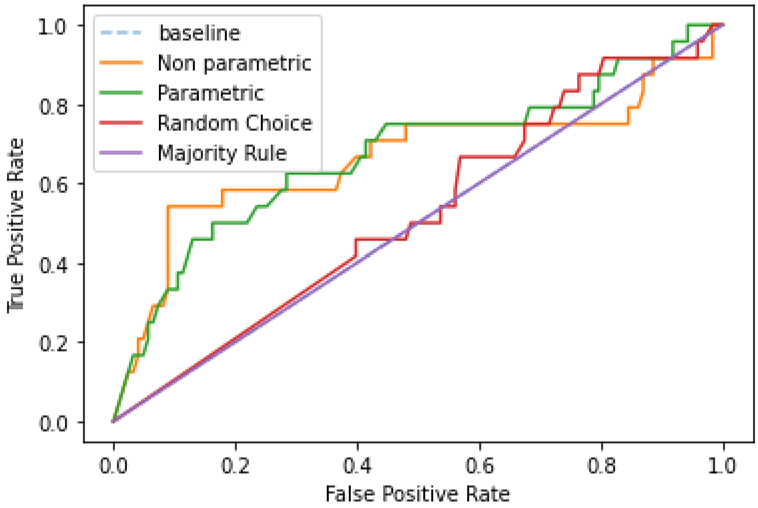
```
max_round = 10
accuracies_np = []
accuracies_p = []
features = ["Age", "MonthlyIncome", "DistanceFromHome", "MonthlyRate",
            "YearsAtCompany", "YearsSinceLastPromotion", "HourlyRate", "Per
            "TotalWorkingYears", "YearsInCurrentRole", "DailyRate", "YearsS
            "YearsWithCurrManager", "WorkLifeBalance", "TrainingTimesLastYe
            "NumCompaniesWorked", "JobSatisfaction", "JobLevel", "JobInvolv
            "EnvironmentSatisfaction", "Education"]
bins = [5, 5, 10, 5,
        4, 5, 5, 5,
        4, 4, 100, 5,
        5, 4, 3, 4,
        3, 4, 5, 4,
        4, 5]

for i in range(max_round):
  X_train, X_test, y_train, y_test = train_test_split(
          all_data, all_data["Attrition"], test_size=0.1, stratify=all_data
  classifier_x = NaiveBayes(X_train, X_test, y_train, y_test, features, bin
  classifier_x.train()
  result = classifier_x.predict()
  reporter_np = Reporter(result["Attrition NP"].to_numpy(), y_test, thresho
  reporter_p = Reporter(result["Attrition P"].to_numpy(), y_test, threshold
  accuracies_np.append(reporter_np.find_accuracy())
  accuracies_p.append(reporter_p.find_accuracy())
  plot_roc(result, y_test)

print("Average Accuracy (NP):", np.mean(accuracies_np))
print("Average Accuracy (P):", np.mean(accuracies_p))
```

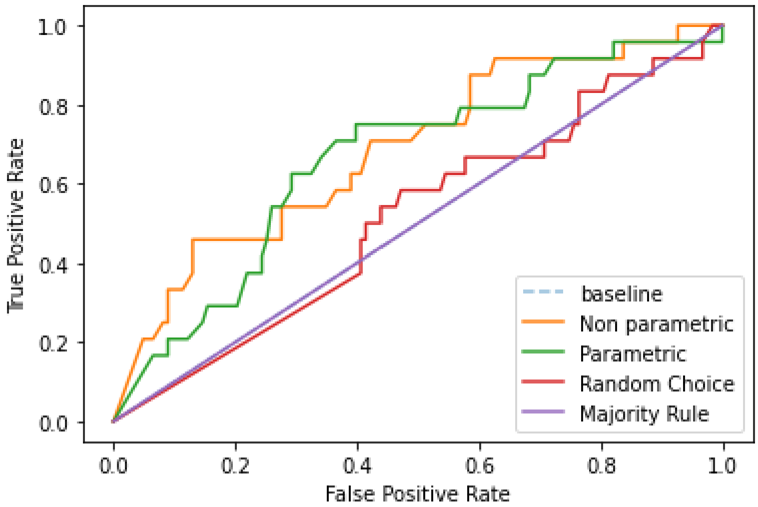


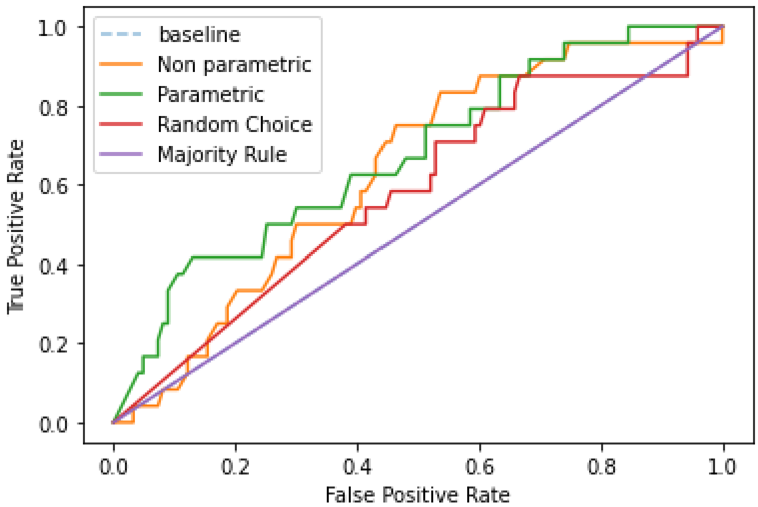

baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

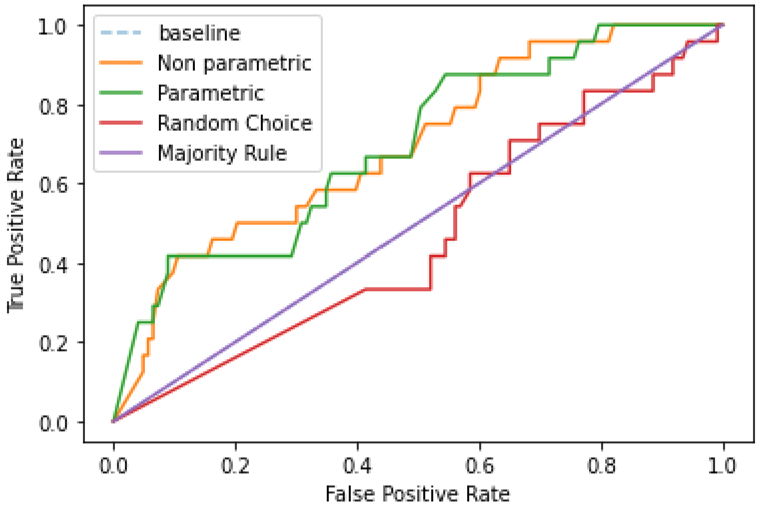

baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

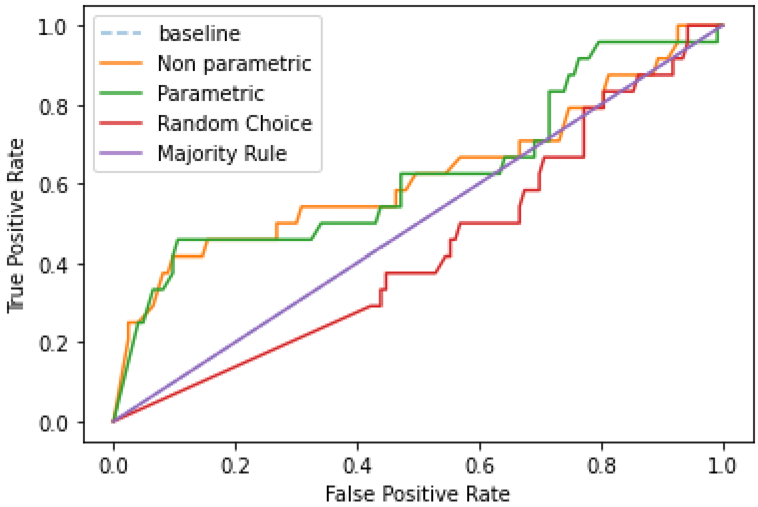

baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

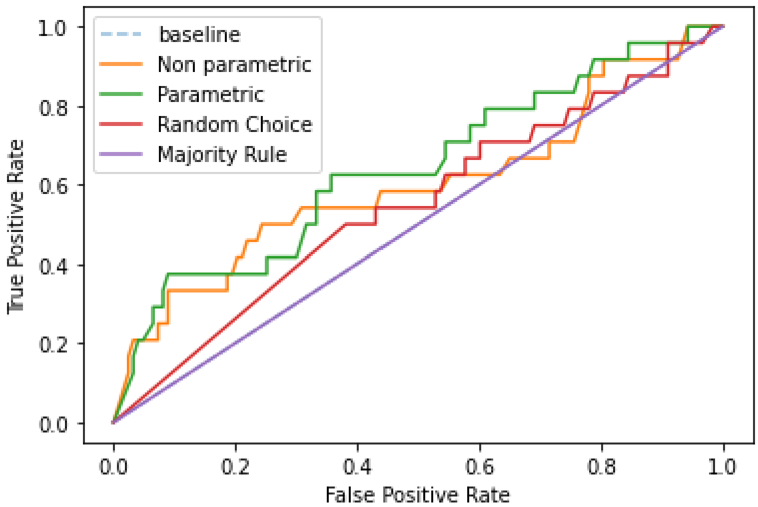

baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


baseline
Non parametric
Parametric
Random Choice
Majority Rule
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0








```
Average Accuracy (NP): 0.8095238095238095
Average Accuracy (P): 0.7523809523809523
```