

# 보고서

## Naïve Bayes Classification

|       |            |     |            |
|-------|------------|-----|------------|
| 제출일   | 2023.11.22 | 전 공 | 휴먼지능정보공학전공 |
| 과 목   | 인공지능개론     | 학 번 | 202115047  |
| 담당교수님 | 이지향 교수님    | 이 름 | 김희재        |

# 목차

## 1. 서론

- 1.1 지도 학습
- 1.2 확률 기반 학습 Naïve Bayes

## 2. 본론

- 2.1 Fraud data의 분석 및 예측
  - 2.1.1 접근
  - 2.1.2 구현 및 실험, 결과
- 2.2 Social Network Ads data의 분석 및 예측
  - 2.2.1 접근
  - 2.2.2 구현 및 실험, 결과
- 2.3 Naïve Bayes scratch 코드 구현
  - 2.3.1 접근
  - 2.3.2 구현 및 실험, 결과

## 3. 결론

- 3.1 고찰

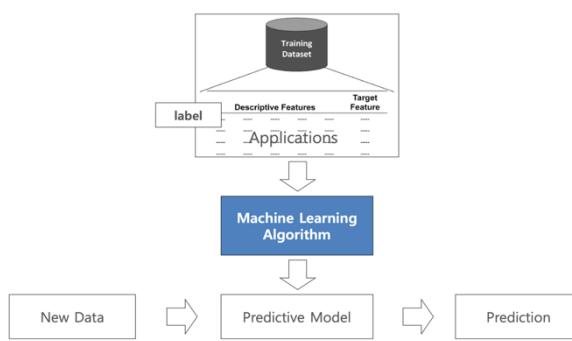
## 4. 참고문헌

# 1. 서론

## 1.1 지도 학습

인공지능이란 인간의 간섭 없이 인식-추론-행동이 가능한 계산 기계에 대한 학문을 의미한다. 즉, 인공지능이 직접 인식하고 추론한 결과로 행동하는 것을, 사람의 간섭 없이 할 수 있게 하는 것이 과학자들의 역할이고, 본 과목인 인공지능개론에서는 수학적 기반에 기초한 전통적 기계학습 알고리즘을 배운다.

기계학습을 더 자세하게 설명한다면, 과거의 데이터로부터 학습하여 정확한 예측을 하도록 스스로 예측 모델을 만드는 컴퓨터 알고리즘이라고 할 수 있다. 크게 지도학습(supervised learning)과 비지도학습(unsupervised learning) 그리고 강화학습(reinforcement learning)으로 나뉘는데, 본 보고서에서 다루는 Naïve bayes 모델은 학습하기 위해 수집한 데이터들의 정답을 알고 있는 지도학습에 속한다.



왼쪽은 지도학습의 단계를 간단하게 설명하고 있는 그림이다. 우선 학습하기 위해 수집한 Label(정답)이 있는 데이터들을 machine learning algorithm에 적용하여 predictive model(예측 모델)을 만든다. 그 이후 해당 예측 모델에 모델이 처음 보는 새로운 데이터를 넣게 되면 모델은 그 과거의 데이터로부터 학습한 것을 기반으로 새로운 데이터의 정답을 예측하게 될 것이다.

Naïve bayes 모델도 위와 같은 패턴으로 데이터를 학습하고 예측한다.

기계학습이 과거의 데이터로부터 학습하여, 새로운 데이터를 예측하는 형태라는 것은 파악이 되었다. 그걸 수행하는 구체적인 과정에 대해 알아보았다.

### 1) 문제 정의

우선 기계학습을 통해 해결하고자 하는 문제, 알고 싶은 정답을 정의해야 한다. 교수님께서 주신 데이터처럼 대출의 상환 여부가 될 수도 있고, 실생활과 밀접한 미래의 날씨도 그 대상이 될 수도 있다.

### 2) 데이터 수집

두 번째로는 모델에 학습시킬 데이터를 수집해야 한다. 이 때 신뢰할 수 있고, 모델 학습에 도움이 되는 양질의 데이터를 수집하는 것이 중요하다.

### 3) 데이터 준비

수집한 데이터를 처리하고 모델에 학습시킬 준비를 하여야 한다. 값이 존재하지 않거나, 일반적인 데이터와 다른 양상을 띠는 결측치를 대치하거나 제거하고, 중복 레코드 또한 제거하고, 데이터의 값을 정규화하는 등의 과정이 필요하다.

#### 4) 데이터 분리

데이터를 훈련 세트, 검증 세트, 테스트 세트로 분할하여 나눠 학습할 준비가 필요하다. 훈련 세트와 테스트 세트로 나누는 이유는 모델이 학습을 얼마나 잘했나 평가하기 위해 훈련할 때 사용하지 않은 새로운 데이터를 제시하기 위해서이다. 검증 세트의 경우, 모델을 학습시키며 신경을 많이 써야하는 과적합을 피하기 위해서 분할하는 데이터이다. 우리가 수집하는 데이터는 한정적일 수 밖에 없고, 아무리 데이터를 수집하고 준비할 때 최대한 정제되도록 정규화하거나 상관성을 지워도 데이터의 벤런스가 깨져 있는 경우가 다반사이다. 이를 모델이 충실히 학습하다 보면 과적합이 발생하며, 과적합이 발생하면 모델은 훈련한 데이터셋에서만 좋은 결과를 내고 정작 기계학습의 목표인 새로운 데이터를 예측하는 데에 있어서는 안 좋은 성과를 낼 수 밖에 없다. 이를 위해 검증 세트를 분리하여 훈련 와중에도 테스트 세트와 같은 역할을 하는 데이터를 둘으로써 과적합을 피할 수 있다. 검증 세트를 분리하는 방법에는 여러가지 방법이 있는데, 본 보고서에서는 K-fold cross validation 을 사용하였다. 이는 해당 코드에서 더 자세하게 설명하도록 하겠다.

#### 5) 모델 학습

4 번에서 분리한 훈련 데이터 세트를 사용해 모델을 훈련하는 핵심 과정이다.

#### 6) 후보 모델 평가

역시 4 번에서 분리한 테스트 세트와 검증 세트를 사용하여 모델을 평가하고, 모델 성능을 더 높일 방법이 있다면 고려하는 단계이다. 평가는 여러 수치들로 할 수 있는데, 본 보고서에서는 정확도, 정밀도, 재현율, F1 score, 민감도, 특이도, ROC 커브 를 사용하였다. 이 역시 해당 코드에서 더 자세하게 설명하도록 하겠다.

#### 7) 모델 배포와 성능 모니터링

6 번 까지의 과정으로 모델을 만들고 선택한 이후 이를 제품으로 배포하고 그 이후 지속적으로 모니터링하는 과정이다. 이 과정은 협업이나 실제 연구에서 진행하며 본 보고서에서는 다루지 않는 단계이다.

### 1.2 확률 기반 학습 Naïve Bayes

본 보고서에서는 확률에 기반을 둔 모델인 Naïve Bayes 모델을 사용하여 학습하였다. 이 모델이 기반을 둔 베이즈 정리부터 설명하겠다.

## Bayes' Theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

베이즈 정리는 위와 같은 식으로 정리 가능하다. 위에서 설명한 예측 모델에 빗대어 이 식을 설명하자면, 우리가 구하고 싶은 것은 어떤 사건  $Y$  가 일어났을 때  $X$  의 확률, 이는  $P(X|Y)$ 로 표현이 가능하다.  $P(X|Y)$ 는 그의 역 조건부확률인  $X$  가 일어났을 때  $Y$ 의 확률  $P(Y|X)$ 와  $X$  가 일어날 확률  $P(X)$ 를 곱하고, 이를  $Y$  가 일어날 확률인  $P(Y)$ 의 확률로 나눈 식과 동치이다. 이 때  $P(X|Y)$ 는 사후확률(prior),  $P(Y|X)$ 는 likelihood(우도),  $P(X)$ 는 prior(사전확률)이라고 한다. 이러한 기본적인 베이즈 정리의 식을 예측모델에 넣는 방식으로 생각해보면 아래와 같이 생각해볼 수 있다.

만약 데이터에  $m$  개의 column(descriptive feature)이 있고, target feature (label, 정답) levels( $t$ )개 만큼 존재할 있다고 하자. 우리가 알고 싶은 prior는 column 데이터가  $q[1], q[2], \dots, q[m]$  과 같을 때의 정답이 target feature levels( $t$ )개 중 어느 것일 확률이 제일 높은 것이냐 이다. 이는 아래와 같은 식으로 표현할 수 있고, 데이터를 가장 잘 맞출 확률을 높이자는 뜻의 MAP (Maximum a Posterior) 에 해당한다.

### Bayesian MAP<sup>1</sup> Prediction Model

$$\begin{aligned} M_{MAP}(q) &= \underset{l \in levels(t)}{\operatorname{argmax}} P(t = l | q[1], \dots, q[m]) \\ &= \underset{l \in levels(t)}{\operatorname{argmax}} \frac{P(q[1], \dots, q[m] | t = l) \times P(t = l)}{P(q[1], \dots, q[m])} \end{aligned}$$

생각해보면, 우도와 사전확률을 제외하고 분모의 belief 부분은 일어나는 사실들의 확률을 기재해둔 부분이므로, 변하지 않는 숫자라 예측에 제외해도 전혀 상관이 없다. 그래서 아래와 같은 식을 표현할 수 있다.

### Bayesian MAP Prediction Model (without normalization)

$$M_{MAP}(q) = \underset{l \in levels(t)}{\operatorname{argmax}} P(q[1], \dots, q[m] | t = l) \times P(t = l)$$

#### Chain Rule

$$\begin{aligned} P(q[1], \dots, q[m]) &= \\ &P(q[1]) \times P(q[2]|q[1]) \times \\ &\dots \times P(q[m]|q[m-1], \dots, q[2], q[1]) \end{aligned}$$

위 식을 계산하려고 보면  $P(q[1], q[2], \dots, q[m] | t = l)$  부분이 계산하기에 참 답답하다. 우선 조건부 확률이고 각 데이터가 왼쪽과 같이 쌓여가는 형태의 식(chain rule)이라 컴퓨터로서는 계산할 양이 너무 많아지는 것이다. 그리고 가장 큰 문제가 있는데, 바로 과적합이다. 각 column 이 쌓여가며 의존적으로 계산되어 학습데이터에 너무 충실해 버린 나머지 과적합이 되어 새로운 데이터를 객관적으로 잘 예측해내지 못할것이다. 이를 해결하고자 위 식에 조건부 독립을 도입한 것이 바로 Naïve Bayes 이다.

## Without conditional independence

Chain rule

$$P(X, Y, Z|W) = P(X|W) \times P(Y|X, W) \times P(Z|Y, X, W) \times P(W)$$

## With conditional independence

$$P(X, Y, Z|W) = \underbrace{P(X|W)}_{Factor1} \times \underbrace{P(Y|W)}_{Factor2} \times \underbrace{P(Z|W)}_{Factor3} \times \underbrace{P(W)}_{Factor4}$$

원래 체인룰에 입각하여 계산되어 주어진 데이터에 과적합 되었던 기존의 식에 조건부 독립을 적용하면 위와 같은 식으로 바뀐다. Label 이 주어진 상황에서는 column 들이 전부 독립적이라고, 조건부로 독립을 허용해준 식이다. 이와 같이 계산하게 되면 계산양도 현저히 줄어 학습 시간이 단축되고, 조건부로 독립을 허용해주어 좀 더 naïve 하게, 정확하진 않을지라도 새로운 데이터에 대해 올바른 예측을 수행할 수 있게 된다. 이렇게 수업시간에 naïve bayes 모델의 이론에 대해 자세히 배웠고, 본론에서는 교수님께서 주신 데이터와 task 에 대해 수행하는 코드를 작성하였다.

## 2. 본론

### 2.1. Fraud data 의 분석 및 예측

#### 2.1.1. 접근

코드 작성에 앞서, 주어진 fraud data 를 먼저 살펴보고, 교수님께서 주신 코드를 전부 이해하는 걸 우선시하였다. 우선 주어진 fraud data 를 먼저 관찰하였다.

`fraud_data`

| ID | History | CoApplicant | Accommodation | Fraud |       |
|----|---------|-------------|---------------|-------|-------|
| 1  | current | none        | own           | TRUE  |       |
| 2  | paid    | none        | own           | FALSE |       |
| 3  | paid    | none        | own           | FALSE |       |
| 4  | paid    | guarantor   | rent          | TRUE  |       |
| 5  | arrears | none        | own           | FALSE |       |
| 6  | arrears | none        | own           | TRUE  |       |
| 7  | current | none        | own           | FALSE |       |
| 8  | arrears | none        | own           | FALSE |       |
| 9  | current | none        | rent          | FALSE |       |
| 10 | none    | none        | own           | TRUE  |       |
| 11 | current | coapplicant | own           |       | FALSE |
| 12 | current | none        | own           |       | TRUE  |
| 13 | current | none        | rent          |       | TRUE  |
| 14 | paid    | none        | own           |       | FALSE |
| 15 | arrears | none        | own           |       | FALSE |
| 16 | current | none        | own           |       | FALSE |
| 17 | arrears | coapplicant | rent          |       | FALSE |
| 18 | arrears | none        | free          |       | FALSE |
| 19 | arrears | none        | own           |       | FALSE |
| 20 | paid    | none        | own           |       | FALSE |

1 행을 제외한 나머지 행에는 각 데이터가 들어있고 각 열은 column에 따라 데이터가 분류되어 있다. 1 열은 데이터의 ID가 1부터 20까지 sorting되어있으며 이는 데이터의 번호를 indexing해둔 것으로 학습시킬 때 넣지 않는다. 데이터의 개수가 20개로 매우 적지만, 예측을 naïve하게 하는 naïve bayes 모델에게는 문제가 되지 않는다. 물론 많을수록 좋기도 하겠지만, 각각의 column을 조건부 독립적으로 여기기에 새로운 데이터를 계산할 때 크게 문제가 되지 않는다. History 열은 current, paid, arrears, none으로 구성되어 있다. 이는 신용 이력에 관한 column으로 각 상환 중, 상환 완료, 연체, 대출이력 없음을 뜻한다. CoApplicant 열은 none, guarantor, coapplicant으로 구성되어 있고 대출 신청자가 신청과 관련해 보증인이나 공동 신청자가 있는지를 나타내는 것으로, 각 없음, 보증인, 공동 신청을 뜻한다. Accommodation 열은 own, rent, free로 구성되어 있고 주거상태를 의미하는데 각 주택을 소유, 주택을 임대, 자유 주거를 뜻한다. 마지막 Fraud 열은 마지막에 위치하며 True와 False로 구성되어 있는 것으로 보아 정답 label, target feature 데이터임을 알 수 있고, 사기가 True인지 False인지를 나타낸다.

즉 우리는 History, CoApplicant, Accommodation 세개의 descriptive feature(column)을 보고 이러한 데이터를 소유한 신청인이 사기일지 아닐지를 예측하는 모델을 이 데이터로서 만들 수 있다.

이렇게 데이터를 간단하게 관찰한 후에, 교수님의 코드를 이해하려 코드를 읽어보았다. 우선 본인은 인공지능 관련 수업은 처음이지만, 여름방학 때 인공지능에 대해 공부하고 싶어 온라인 자료들을 보면 맛보기 공부를 가볍게 했어서 다양한 라이브러리와 함수들이 아예 초면은 아니라 이해에 도움이 크게 되었다. 그러나 충실히 코드를 읽고 작성하였다.

```
# Import Library of Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets

import pandas as pd
import numpy as np
```

우선 코드 작성에 필요한 라이브러리들을 import하신 걸 볼 수 있었다. Scikit-learn library에서 제공하는 GaussianNB (Gaussian Naïve Bayes) 모델을 사용하여 코드를 작성하고, datasets의 경우 dataset 처리할 때 사용할 것 같다고 생각하였다. 하지만 이후 코드에 주석을 달아두었지만 from sklearn import datasets의 경우 iris data와 같이 기계학습 공부에 널리 사용되는 데이터들을 불러오기 쉽게 scikit learn에서 만들어 둔 라이브러리로, fraud data를 사용하는 본 문제에서는 사용하지 않은 라이브러리이다. Pandas와 Numpy의 경우 파이썬에서 대표적으로, 유용하게 사용되는 두 라이브러리인데, Pandas는 dataframe이라는 데이터구조를 제공하여 데이터를 분석 및 사용 할 때 유용하게 사용하고, numpy의 경우 배열 처리를 쉽게 하고 여러 함수를 제공하여 (예를 들면 분산 계산) 편리하게 코드를 작성할 수 있는 라이브러리이다.

```

def replace(df):
    df = df.replace(['paid', 'current', 'arrears'], [2, 1, 0])
    df = df.replace(['none', 'guarantor', 'coapplicant'], [0, 1, 1])
    df = df.replace(['coapplicant'], [1])
    df = df.replace(['rent', 'own'], [0, 1])
    df = df.replace(['False', 'True'], [0, 1])
    df = df.replace(['none'], [float('NaN')])
    df = df.replace(['free'], [-1])
    return df
df = pd.read_csv('./fraud_data.csv')
res = replace(df)

```

그 다음은 replace()라는 함수를 정의하신 걸 볼 수 있었다. 우선 이름이 replace 이고 매개변수와 반환값이 df (dataframe 의 약자)인 것으로 보아 데이터를 매개변수로 받아 어떠한 처리를 하고 그 데이터를 반환함을 알 수 있다. 그래서 함수 내부 코드를 보니, 위에 데이터를 분석하며 작성하였던 column 들의 값들이었고, 각 column 마다의 문자들을 숫자로 변경하여 컴퓨터가 데이터 계산을 하기에 쉽게 처리한 것임을 알았다. 단순하게 생각해보아도 문자로는 나이브 베이즈 계산을 하기에 헛갈리고 쉽지 않을 것이다.

함수 밑의 코드는 pandas 를 이용해 데이터를 dataframe 변수인 df 에 불러오고 res 에는 replace 함수로 숫자처리한 데이터를 넣은 것을 알 수 있다.

좀 더 자세히 보자면, guarantor 와 coapplicant 는 왜 둘다 같은 숫자인 1 로 mapping 되었을까? 이는 둘 다 사기가 아님을 인증할 수 있는 수단으로 같게 보아서 그런 것 같다. Coapplicant 는 왜 두번 적혀있을까? 이 점은 아무리 생각해보아도 잘 모르겠다... 그럼 none 은 왜 NaN 으로 적혀있을까? 이는 coapplicant column 의 none 을 0 으로 NaN 으로 바꾸는 게 아니라 혹시 모르게 있을 결측치를 뜻한다. 결측치는 missing data 로 현실에서 데이터를 수집하다보면 데이터의 일부를 얻지 못하거나 누락되어 없는 데이터를 뜻한다. 이미 두번째 줄에서 none 을 0 으로 바꾸었으니 6 번째 줄을 수행할 때는 결측치만 NaN 으로 바꿔줄 것이다. 이렇게 함으로써 나중에 결측치들을 삭제 처리할 수 있는데, 본 데이터에는 결측치가 없으므로 삭제해주는 코드가 따로 없다.

```

history = # 무엇이 들어갈까요?
coapplicant = # 무엇이 들어갈까요?
accommodation = # 무엇이 들어갈까요?

```

다음 데이터에는 이렇게 무엇이 들어갈지를 생각해 보아야 했는데, 아마 위에서 처리한 데이터를 각 column 별로 나누어 변수에 넣어야 하는 것 같았다. 이 함수는 사실 2 번 문제의 코드를 분석하다 보고 알게 되었는데 바로 아래가 그 함수를 알게 된 2 번의 코드이다.

```

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

```

교수님께서 2 번 문제에 필요한 데이터를 똑같이 pandas 라이브러리를 사용해 불러오신 후 iloc 함수를 사용해 X 와 y 에 각 데이터를 열에 따라 나눠주시는 코드를 보고 이 iloc 함수를 사용하면 history, coapplicant, accommodation 도 나눌 수 있음을 깨달았다. Values 를 붙여주는 이유에 대해서는 무슨 차이인지 코드에 각각을 print 해서 알아보았는데, 아래와 같다.

```
Age   EstimatedSalary
0      19            19000
1      35            20000
2      26            43000
3      27            57000
4      19            76000
..    ...
395    46          41000
396    51          23000
397    50          20000
398    36          33000
399    49          36000
[400 rows x 2 columns]
[[ 19  19000]
 [ 35  20000]
 [ 26  43000]
 [ 27  57000]
 [ 19  76000]
 [ 27  58000]
 [ 27  84000]
 [ 32  150000]
 [ 25  33000]
 [ 35  65000]
 [ 26  80000]
 [ 26  52000]]
```

왼쪽은 iloc 만으로 나눴을 때의 출력된 값이고 오른쪽은 values 를 하고 난 후 출력된 형태이다. 물론 이는 fraud data 가 아닌 2 번 문제에서 사용할 ads 데이터이지만 데이터의 형태가 계산하기 쉽게 인덱스는 없애고 value 들만 배열형태로 저장된 것을 볼 수 있다.

```
X = # history, coapplicant, accommodation 을 이용하여 학습 데이터를 구축하시오.
Y = # 무엇이 들어갈까요?
```

위 코드에 해당하는 코드도 위에서 본 2 번 코드를 통해, 그리고 미리 했던 데이터 분석을 통해 감을 잡았다. X에는 적어 주신 것처럼 학습할 데이터들을, descriptive feature들을 위에서 나눴으니 다시 합치면 되고, Y에는 정답 label 데이터들, target feature 들이 들어갈 것이다. 이는 마지막 열의 데이터니 iloc 함수를 써서 마지막 열만 데이터에 넣어주면 될 것이다.

```
model = GaussianNB()
model.fit(X, Y)
predicted = model.predict([[2, 0, 0]])
#pred_prob = model.predict_proba(X)

print (predicted)
```

이후에 코드에서는 GaussianNB()로 sklearn 에서 제공하는 모델을 모델 변수에 넣고, 이를 이용해 fit(학습)하고 predict(예측)하고 이를 출력하는 코드를 볼 수 있었다. 주석 처리된 Pred\_prob 은 prob 가 쓰인 걸 보아 확률을 어떠한 함수를 써서 받고 이 또한 출력하는 코드임을 알 수 있다.

여기서 왜 GaussianNB()를 사용하는 것일까? Scikit learn 에서는 Gaussian 말고도 Bernoulli, Multinomial Naïve Bayes 모델이 존재하였다. 검색해보니 GaussianNB 는 연속적 데이터에 적용, BernoulliNB 는 이진 데이터에 적용, MultinomialNB 는 카운트(count; e.g. 문장에 나타난 단어의 횟수) 데이터에 적용한다고 한다. BernoulliNB 의 경우는 각 클래스의 특성 중 0 이 아닌 것이 몇 개인지 세는 알고리즘으로, 본 데이터처럼 한 column 의 세개 이상의 데이터가 있는 경우 적합하지 않다. MultimnomialNB 의 경우 count 데이터에 적용한다고 하니 이 역시 이번 데이터에 적합하지 않아 이번 과제에 GaussianNB()를 사용한 것 같다는 결론에 도달했다. 또한 Gaussian 분포 자체가 정규분포로 데이터를 naïve 하게 예측하기 가장 좋은 형태라는 생각도 개인적으로 하였다.

그리고 1 번 문제를 보면

(1) Fraud\_data.csv 데이터를 활용하여, 다음 케이스에 대해 결과를 확인합니다.

- (history, co-applicant, accommodation) = ([paid, none, rent], [paid, guarantor, rent], [arrears, guarantor, rent], [arrears, guarantor, own], [arrears, coapplicant, own])

- Classification 결과와 결과에 따른 확률 값을 제시합니다.

이러한 task 를 내주셨는데, 우선 제시해주신 case 들을 위 코드처럼 숫자로 대입하여 예측 결과를 얻으면 될 것 같고, 결과와 결과에 따른 확률 값은 pred\_proba 와 predicted 변수처럼 함수에서 반환값으로 받아와서 출력하면 됨을 인지하였다.

### 2.1.2 구현 및 실험, 결과

아래부터는 구현한 코드를 캡처한 것이다.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 29 15:52:28 2016

@author: JeeHang Lee
@date: 20160926
@description: This is an example code showing how to use Naive Bayes
    implemented in scikit-learn.
"""

# scikit-learn에서 제공하는 Gaussian Naive Bayes 모델 사용 위해 import
from sklearn.naive_bayes import GaussianNB

# scikit-learn에서 제공하는 다양한 dataset이 있는 라이브러리지만 본 코드에서 사용되지 않았다.
from sklearn import datasets

# pandas가 제공하는 2차원 데이터구조인 dataframe을 사용하기 위해 pandas import
import pandas as pd

# 배열과 행렬을 처리하기 위해 numpy import
import numpy as np

# 상관행렬 그레프를 그리기 위해 import
import seaborn as sns
```

우선 라이브러리를 위와 같이 import 하였다. 추가한 seaborn library 의 경우 그레프를 그릴 수 있도록 도와주는 Matplotlib 과 같은 라이브러리인데 코드를 작성하다가 상관행렬 그레프를 그려보고 싶어 추가하였다.

```

# 데이터 전처리를 위한 replace 함수를 정의한다.
# 데이터를 숫자로 치환한다.

def replace(df):
    df = df.replace(['paid', 'current', 'arrears'], [2, 1, 0])
    df = df.replace(['none', 'guarantor', 'coapplicant'], [0, 1, 1])
    df = df.replace(['coapplicant'], [1])
    df = df.replace(['rent', 'own'], [0, 1])
    df = df.replace(['False', 'True'], [0, 1])
    df = df.replace(['none'], [float('NaN')])
    df = df.replace(['free'], [-1])
    return df

# fraud_data.csv 파일을 df 변수에 불러옴
df = pd.read_csv('./fraud_data.csv')

# 위에서 정의한 replace 함수로 데이터 전처리 진행
res = replace(df)

```

앞에서 살펴봄과 같이 Data 를 불러오고, replace 함수로 숫자 처리하는 과정이다.

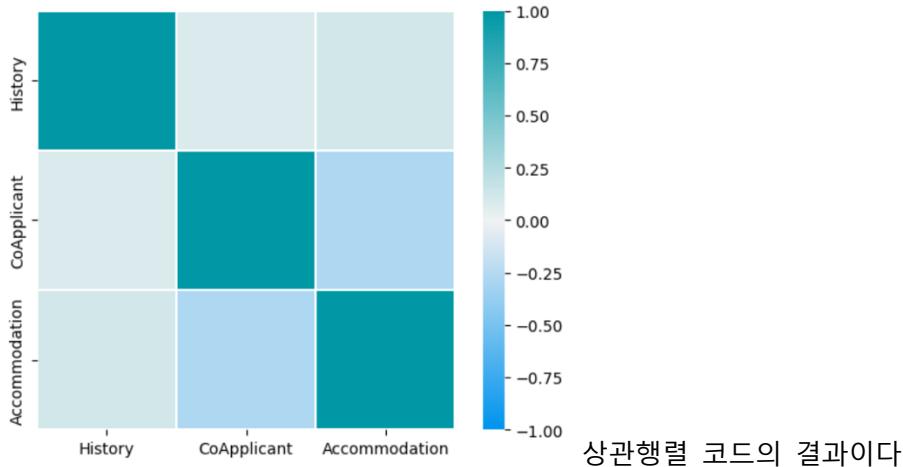
그리고 나서는 상관행렬 그래프를 그려보았다. 이는 naïve bayes scratch 코드를 작성해보고 싶어 검색한 유튜브에서 초반 데이터 분석 부분에서 사용 후 설명해준 행렬인데, 중요한 것 같아 1 번 문제에도 적용하여 상관행렬을 그려 보았다. 상관 행렬은 각 column 들끼리의 상관계수를 구해 그래프로 그려 각 column 들끼리 얼마나 상관이 있나 확인할 수 있는 그래프이다. 상관계수의 범위는 -1 에서부터 1 인데, 1에 가까울수록 양의 상관, 0 이면 무상관, -1 이면 음의 상관을 뜻한다. 상관계수의 숫자가 높을수록 서로 상관이 있다는 건데, 물론 나이브 베이즈 모델에서는 각 만 column 이 독립적이라고 생각하긴 하지만, 애초에 데이터에서부터 각 column 이 독립적이고, 객관적이라 새로운 데이터를 더 잘 설명할 수 있는 다양한 column 들로 구성되어 있으면 당연히 모델의 성능이 올라갈 것이다. 그래서 이 행렬을 이용해서 데이터를 골라서 학습할 수도 있고 어떠한 column 때문에 데이터가 잘 학습되지 않는 것인지도 알 수 있다.

```

# 상관행렬
# 이 행렬은 column들간의 상관을 행렬로 보여주는 그래프이다.
# 숫자가 1에 가까울수록 양의 상관, -1에 가까울수록 음의 상관, 0이면 무상관이다.
# 이 경우 모든 column을 학습하여 진행하기에 괜찮지만 혹시 column을 선택하여 학습하거나,
# 어떤 column이 학습에 도움이 안 되는지 알기 위해서는 상관행렬을 보는 것이 좋다.

corr = res.iloc[:,1:-1].corr(method="pearson")
cmap = sns.diverging_palette(230,200,250,50,center='light',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-1, cmap=cmap, square=True, linewidths=.2)

```



상관행렬 코드의 결과이다.

본 문제에서는 데이터를 전부 사용하긴 했지만, 데이터를 파악함에 있어 중요한 것 같아 상관행렬을 코드로 작성하여 확인해보았고, CoApplicant 와 Accommodation 이 상관이 0.4 에 근접한 색깔로 가장 높긴 했지만 아주 높은 수준은 아님으로 판단되었다.

Pearson 은 Pearson 상관계수 공식을 사용하였다는 뜻이고, 함수를 통해 색깔과 옆의 colormap (cmap) 설정, 범위를 1 부터 -1 까지 설정 등을 할 수 있었다.

```
# fraud_data의 column으로 데이터를 각각 나눠 추출한다.
# 각 1열의 전부, 2열의 전부, 3열의 전부이므로 아래와 같은 코드를 적을 수 있다.

history = res.iloc[:,1]
coapplicant = res.iloc[:,2]
accommodation = res.iloc[:,3]
```

이는 위에서 코드 분석할 때 분석한 것처럼 코드를 작성하였다.

iloc 함수의 [ : ,1] 은 무엇을 뜻하나면, 행렬 표시처럼 생각하면 간단한데, 이를 [행, 열]로 생각하여 모든 행 그리고 1 열 전부를 추출한 것으로 생각하면 된다.

|    |   |    |   |    |    |
|----|---|----|---|----|----|
| 0  | 1 | 0  | 0 | 0  | 1  |
| 1  | 2 | 1  | 0 | 1  | 1  |
| 2  | 2 | 2  | 0 | 2  | 1  |
| 3  | 2 | 3  | 1 | 3  | 0  |
| 4  | 0 | 4  | 0 | 4  | 1  |
| 5  | 0 | 5  | 0 | 5  | 1  |
| 6  | 1 | 6  | 0 | 6  | 1  |
| 7  | 0 | 7  | 0 | 7  | 1  |
| 8  | 1 | 8  | 0 | 8  | 0  |
| 9  | 0 | 9  | 0 | 9  | 1  |
| 10 | 1 | 10 | 1 | 10 | 1  |
| 11 | 1 | 11 | 0 | 11 | 1  |
| 12 | 1 | 12 | 0 | 12 | 0  |
| 13 | 2 | 13 | 0 | 13 | 1  |
| 14 | 0 | 14 | 0 | 14 | 1  |
| 15 | 1 | 15 | 0 | 15 | 1  |
| 16 | 0 | 16 | 1 | 16 | 0  |
| 17 | 0 | 17 | 0 | 17 | -1 |
| 18 | 0 | 18 | 0 | 18 | 1  |
| 19 | 2 | 19 | 0 | 19 | 1  |

Name: History, dtype: int64    Name: CoApplicant, dtype: int64    Name: Accommodation, dtype: int64

History 와 CoApplicant, Accommodation 을 출력해보면 위와 같이 잘 각 20 개의 데이터 모두 잘 나눠진 것을 볼 수 있다. 아래 코드에서 데이터를 합친 후 학습을 진행할 거라 values 를 붙이지 않았다.

```
# 입력데이터 X에는 바로 위에서 나눠 추출한 history, coapplicant, accommodation을 넣어준다.
# axis = 0 은 세로로 데이터가 합쳐지는 것이고 axis = 1 은 가로로 데이터가 합쳐지는 것
# 우리는 가로의 데이터가 한 쌍이므로 axis = 1
# .values는 DataFrame을 scikit-learn 모델에서 예상하는 형식인 NumPy 배열로 변환

X = pd.concat([history, coapplicant, accommodation], axis=1).values

# 정답 label이 들어가는 Y에는 data의 가장 마지막 열을 넣어준다. (True, False 와 같은 정답이 담김)
Y = res.iloc[:, -1].values
```

X 에는 위에서 나눈 descriptive feature 들을 합쳐 X 에 넣고, 정답 label 은 마지막 열만 분리하여 Y 에 넣는 코드를 작성하면 된다. Concat 함수의 경우 구글에 'pandas 데이터 합치기' 를 검색하여 알게 된 함수인데, 위와 같이 배열에 나눈 데이터들을 적어주고, axis 는 0 이면 세로로 데이터가 길게 합쳐지는 것이고 1 이면 가로로 데이터가 옆으로 합쳐지는 것인데, 우리는 가로의 데이터 한 쌍으로 학습을 진행시키므로 axis=1 을 설정해주었다. 역시 학습을 위해 values 로 value 값만 남긴다. Y 의 경우 iloc 함수를 사용하였다. [:,-1]로 모든 행 그리고 마지막 열만 선택하여 주었다.

```
# Gaussian Naive Bayes 모델 생성
model = GaussianNB()

# 입력 데이터와 정답 데이터 넣고 학습
model.fit(X, Y)
```

Scikit learn 에서 제공하는 GaussianNB() 모델로 학습을 진행하는 코드이다.

```
# case마다의 결과를 출력하기 위한 함수

def print_result(case):

    # dictionary로 교수님이 데이터를 replace하셨듯이 해당 숫자를 case에 매칭하는 딕셔너리를 만들었다.
    category = {'paid': 2, 'current': 1, 'arrears': 0,
                'none': 0, 'guarantor': 1, 'coapplicant': 1,
                'rent': 0, 'own': 1}

    # 입력값을 dictionary 이용해서 숫자로 치환
    h, c, a = map(category.get, case.split(", "))

    # Classification 결과
    predicted = model.predict([[h,c,a]])
    # 결과에 따른 확률 값
    pred_prob = model.predict_proba([[h,c,a]])

    print("-"*80)
    print(f"\nCase : (history, co-applicant, accommodation) = [{case}]\n")
    print(f"Classification 결과 : {predicted[0]}")
    print(f"결과에 따른 확률 값 : False = {pred_prob[0][0]:.3f} | True = {pred_prob[0][1]:.3f}\n")
    print("-"*80, end="\n\n")
```

```

# 교수님께서 주신 케이스에 대해 결과를 확인할 때 함수를 이용하기 위해 각 case를 case 리스트 안에 넣었다.
# 그러면 나중에 케이스를 입력받아 그 값을 리스트에 저장해서 확률을 구하기 쉽다.

case = ["paid, none, rent",
        "paid, guarantor, rent",
        "arrears, guarantor, rent",
        "arrears, guarantor, own",
        "arrears, coapplicant, own"]

# 5개의 case에 대해 printResult 함수 실행
for i in range(5):
    print_result(case[i])

```

위 두 코드는 교수님께서 주신 케이스들에 대해 classification 결과를 구하고, 그 결과에 따른 확률 값을 구하는 코드이다.

처음에는 주어진 코드처럼 2,0,0 .. 과 같이 숫자를 바로 predict 함수와 predict\_proba 함수에 넣어 나열하여 코드를 작성하였다. 하지만 이를 간단하게 함수화 하고 싶다는 생각이 들었고, 위와 같은 코드를 작성하게 되었다.

우선 case 배열에, 주어진 case들을 문자열로 저장하였다. 이에 대해 for 문을 5 번 돌려 print\_result() 함수로 classification 결과와 확률값을 출력하였다. 이렇게 함수화 했을 시의 이점은 개인적인 만족일수도 있지만 만약 특정 케이스에 대한 결과를 알고 싶은 경우가 생기면, 직접 해당 숫자를 찾아 2,0,0 과 같은 방식으로 입력하기보다 우리가 파악하기 쉬운 문자로 입력을 하면 컴퓨터가 알아서 문자열을 숫자로 처리 후 학습 결과를 출력해줘 간편히 결과를 얻을 수 있다. 물론 본 보고서에서는 입력이 따로 필요하지 않아 입력 코드를 작성하진 않았는데, 간단하게 case.append(input()) 과 같은 코드로 입력을 받을 수 있다.

그래서 print\_result 함수 내부를 들여다 보면 category 딕셔너리로 각 column 값에 대해 위해 전처리 해둔 것 처럼 숫자를 매칭해 두었고, 입력 값을 get 과 split()함수를 이용해서, h, c, a 변수에 나눠 저장하였다. 이후 h, c, a 변수를 predict 함수와 predict\_prob 함수에 넣어 출력하여 결과값을 얻었다. Pred\_prob 의 경우 2 열로 구성된 데이터를 반환하는데 이는 검색해보니 0 열에 데이터가 False(0)일 확률 그리고 1 열에 데이터가 True(1) 확률을 반환해주는 함수였다. 그래서 위와 같이 [0][0], [0][1] 로 나누어 False 일때, True 일 때의 결과를 각각 출력해주었다. 이는 앞서 숫자 처리된 결과가 sorting 되어 각 column 에 순서대로 나오는 것이라고 한다.

아래는 위 코드에 대한 결과 값이며 1 번 문제의 답이다.

---

---

```
Case : (history, co-applicant, accommodation) = [paid, none, rent]
```

```
Classification 결과 : False  
결과에 따른 확률 값 : False = 0.782 | True = 0.218
```

---

---

```
Case : (history, co-applicant, accommodation) = [paid, guarantor, rent]
```

```
Classification 결과 : False  
결과에 따른 확률 값 : False = 0.681 | True = 0.319
```

---

---

```
Case : (history, co-applicant, accommodation) = [arrears, guarantor, rent]
```

```
Classification 결과 : False  
결과에 따른 확률 값 : False = 0.614 | True = 0.386
```

---

---

```
Case : (history, co-applicant, accommodation) = [arrears, guarantor, own]
```

```
Classification 결과 : False  
결과에 따른 확률 값 : False = 0.583 | True = 0.417
```

---

---

```
Case : (history, co-applicant, accommodation) = [arrears, coapplicant, own]
```

```
Classification 결과 : False  
결과에 따른 확률 값 : False = 0.583 | True = 0.417
```

---

모든 case 에서 False 가 나온 것을 알 수 있다. 마지막 두 case 의 경우 근소한 차이인 듯 해서 데이터가 더 많았으면 또 다른 결과가 나오지 않았을까 하는 아쉬움이 있다. 결국 모든 case 에 대해 Fraud 예측은 False 로 나와 다섯개의 case 모두 현재 예측으로는 사기가 아닌 것으로 판단되었다.

## 2.2 Social Network Ads data의 분석 및 예측

### 2.2.1 접근

Fraud data와 마찬가지로 Social Network Ads data에 대해서도 우선 데이터를 관찰한 후 교수님께서 주신 코드를 분석하여 접근 해보도록 하겠다.

Social\_Network\_Ads

| User ID  | Gender | Age | EstimatedSalary | Purchased |
|----------|--------|-----|-----------------|-----------|
| 15624510 | Male   | 19  | 19000           | 0         |
| 15810944 | Male   | 35  | 20000           | 0         |
| 15668575 | Female | 26  | 43000           | 0         |
| 15603246 | Female | 27  | 57000           | 0         |
| 15804002 | Male   | 19  | 76000           | 0         |
| 15728773 | Male   | 27  | 58000           | 0         |
| 15598044 | Female | 27  | 84000           | 0         |
| 15694829 | Female | 32  | 150000          | 1         |
| 15600575 | Male   | 25  | 33000           | 0         |
| 15727311 | Female | 35  | 65000           | 0         |

데이터가 엄청 많아서 처음 10개 부분만 캡쳐 해왔다. 데이터의 개수는 총 400개였는데, 이는 코드에서 dataset을 불러온 이후 print(len(dataset))을 해보면 알 수 있다. 우선 데이터 열부터 관찰해보도록 하겠다. User ID는 indexing을 위한 정보이고 이는 그저 인덱싱을 위한 번호로, 학습에 넣지 않을 것이다. Gender, Age, EstimatedSalary의 경우 데이터가 학습하게 될 descriptive features에 해당한다. Gender의 경우 성별이며 남성과 여성, Male과 Female 두 가지로 나뉜다. Age의 경우 나이이므로

숫자데이터가 다양하게, 아마 1~100까지 있을 것이다. EstimatedSalary의 경우 연봉을 나타내며 다양한 숫자로 분포하고 있다. Purchased의 경우 결제 여부를 0, 1 bool형으로 나타낸듯 하다. 이는 곧 정답 데이터, lable, target feature이다.

데이터의 제목인 Social Network Ads 와 정답데이터가 Purchased 유무임에 따라 이 데이터로는 Gender, Age, EstimatedSalary가 이러한 사람들이 Social Network Ads 를 보고 제품을 구매할지 안 할지를 예측할 수 있을 것 같다.

이 데이터에서 Gender의 경우 문자열이므로 1번에서 했던 것 처럼 숫자로 mapping 해주면 될 것 같다. Age, EstimatedSalary의 경우 이미 숫자이므로 변경이 필요 없을 것이라고 간단하게 생각했지만, 교수님의 코드를 보고 과제를 수행한 결과 두 column은 서로 너무 다른 숫자이기에 표준화라는 과정을 거치는 것을 알게 되었다. 이는 코드 부분에서 더 자세하게 설명하도록 하겠다.

교수님께서 주신 코드를 분석해보도록 하겠다.

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
```

```

from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

```

우선 코드에 필요한 여러 library들이 import 되어있다. 1번과 겹치는 라이브러리 빼고 관찰해보면 train\_test\_split는 데이터를 train과 test로 split 할 때 필요한 것으로 보인다. StandardScaler의 경우 표준화에 필요한 라이브러리 같다. Confusion\_matrix, accuracy\_score, classification\_report, roc\_curve, roc\_auc\_score 모두 성능 평가 할 때 쓰이는 라이브러리들인 것 같다. 위 라이브러리를 모두 사용하면 2번 문제를 수행하면 될 것 같다.

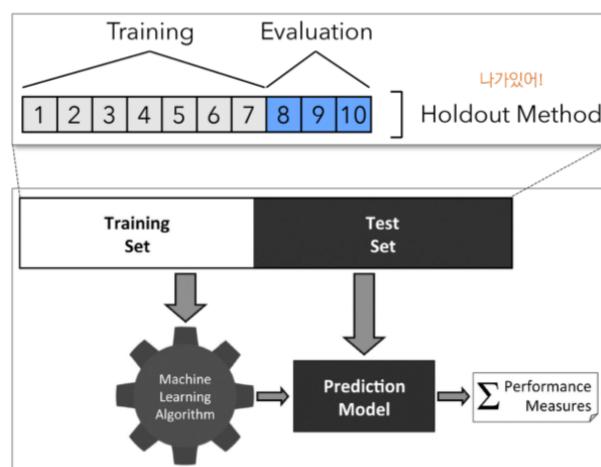
```

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 0)

```

Dataset을 pandas 라이브러리로 불러오고, 이를 iloc함수를 이용해 descriptive features와 target feature로 나누어 주었다. 그 아래 코드는 교수님이 주석으로 달아주신 것 처럼 train data와 test data를 split하는 함수로 test size를 코드와 같이 20%로 설정할 수 있다. 이렇게 train data와 test data로 데이터를 나누는 것을 hold out 한다고 한다. Hold out은 서론에서 설명한 것처럼 overfitting이 되지 않도록 설정해두는 것이고, 이로써 학습의 효율이 올라간다. 위 코드에서 validation data로까지 나누었다면 그것은 3-way hold out 이다.



Hold out에 대해서 배운 slide에서 가져온 그림이다.

Random state는 random seed를 지정해주어 랜덤하지만, 같은 숫자가 나오도록 설정한 것이다. 컴퓨터의 random은 완전 random이 아니고 seed에 따라 숫자 패턴이 정해져있기 때문이다. 만약 완전 random으로 하고 싶다면 seed를 현재시각으로 설정해주면 되는데, 여러 방식으로 성능을 비교하며 학습시킬 때에는 비교를 쉽게 하기 위해 random seed를 동일한 숫자로 지정해준다.

```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

이 부분은 데이터를 feature scaling 해주는 부분이다. 이를 표준화한다고 하며, StandardScaler.fit\_transform() 함수는 데이터의 평균과 표준편차를 0, 1로 설정하여 데이터를 표준화해준다. 표준화 식은  $(x - \text{평균}) / (\text{표준편차})$ 이다.

```
# Training the Naive Bayes model on the Training set
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
ac = accuracy_score(y_test,y_pred)
```

데이터를 GaussianNB 모델로 학습하고, 예측한 이후 accuracy를 정답 데이터, 예측 데이터, 라이브러리 함수를 이용해 구하는 코드이다.

```
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
lr_probs = classifier.predict_proba(y_test)
→ lr_probs = classifier.predict_proba(X_test)[:, 1]
lr_auc = roc_auc_score(y_test, lr_probs)
```

위 코드는 정답 데이터와 예측 데이터로 혼동행렬 (confusion matrix)을 만들고 roc curve를 그리고 auc score를 구하기 위해 라이브러리 함수를 이용하는 코드이다.

Predict\_proba 함수는 예측 후의 확률을 반환하는데, 이는 1번에서 각 결과의 확률을 반환할 때 사용했던 함수이기도 하다. 근데 이 코드에서 예측 후의 반환된 lr\_probs에는 결과가 0일 때와 1일 때 모두 반환되는데, roc curve를 그리고 auc score를 그리기 위해서는 positive를 예측한 배열을 반환 받아야 한다. Roc curve가 TPR 과 FPR 즉, positive로 예측한 것 중 잘 예측한 것과 잘못 예측한 것을 이용해 그리는 그래프이기도 하고, roc\_auc\_score의 두번째 매개변수로 들어가야 할 것도 positive를 예측한 배열이므로 코드를 위와 같이 [:,1] 을 추가해서 변경해주어야 한다.

```
# calculate roc curves
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
plt.show()
```

이 부분은 라이브러리 함수를 사용하여 roc\_curve를 그리는 코드이다. Auc score를 구했을 때와 마찬가지로 정답 데이터와 positive라고 예측한 배열을 roc\_curve 함수에 넣으면 fpr, tpr과 threshold를 반환하는데 threshold는 사용하지 않을 값이라 변수에 담지 않고 \_ 처리해주었다. 그 다음 fpr과 tpr을 이용하여 그래프를 그리는 코드였다.

## 2.2.2 구현 및 실험, 결과

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 13 12:05:37 2021

@author: jeehang

acknowledgement: https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/
"""

# 배열과 행렬을 처리하기 위해 numpy import
import numpy as np

# 그래프 그리기 위해 matplotlib.pyplot import
import matplotlib.pyplot as plt

# pandas가 제공하는 2차원 데이터구조인 dataframe을 사용하기 위해 pandas import
import pandas as pd

# confusion matrix 그리기 위해 seaborn import
import seaborn as sns

# 데이터를 train, test 데이터로 각각 나누려고 import
from sklearn.model_selection import train_test_split

# 데이터 전처리 위해 가져온 라이브러리
from sklearn.preprocessing import StandardScaler

# scikit-learn에서 제공하는 Gaussian Naive Bayes 모델 사용 위해 import
from sklearn.naive_bayes import GaussianNB

# confusion matrix 그리고, accuray 구하고, 평가지표 확인하기 위해 import
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# precision_score, recall_score, f1_score 함수 사용 위해 import
from sklearn.metrics import precision_score, recall_score, f1_score

# roc_curve, roc_auc_score 함수 사용 위해 import
from sklearn.metrics import roc_curve, roc_auc_score

# 결과 시각화 그림
from mlxtend.plotting import plot_decision_regions
```

우선 코드를 작성하기 위해 import한 라이브러리들이다. 교수님께서 적어 주신 것 이외에도 그 래프를 그리기 위해 seaborn 라이브러리를 가져오고, 성능평가를 위해 precision\_score, recall\_score, f1\_score을 가져오고, 최종 결과를 시각화하고 싶어 Plot\_decision\_regions라는 라이브러리를 가져왔다.

```
# dataset을 가져왔다.
dataset = pd.read_csv('Social_Network_Ads.csv')

# X, y를 각각 column들과 label로 데이터를 나눴다.

# iloc[:, [2,3]]의 뜻은 모든 행, 열은 2,3열만 이라는 뜻이다.
X = dataset.iloc[:, [2, 3]].values

# 모든 행, 마지막 열만
y = dataset.iloc[:, -1].values
```

Dataset을 가져오고 iloc 함수로 descriptive feature들과 target feature를 나눈 코드이다. 근데 보면 X에 모든 행과 1열을 제외한 2,3열만 가져온 것을 볼 수 있다. 1열은 Gender 데이터인데 왜 제외한 것일까?

```
# 혹시 교수님께서 Gender을 빼고 데이터를 학습하라고 적어주신 것이 상관계수에 의한 것인지 궁금해서
# Gender의 Male, Female 을 0, 1로 치환하여 상관행렬을 그려보았다.
```

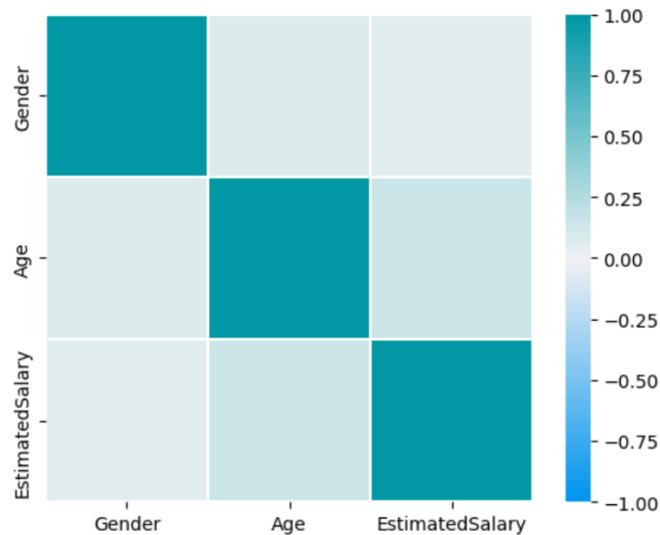
```
dataset = dataset.replace(['Male', 'Female'], [0, 1])
```

```
# 상관계수에 큰 차이가 없어 학습에 넣어 진행시켜도 괜찮을 것 같다는 판단을 하였다.
```

```
corr = dataset.iloc[:,1:-1].corr(method="pearson")
```

```
cmap = sns.diverging_palette(230, 200, 250, 50, center='light', as_cmap=True)
```

```
sns.heatmap(corr, vmax=1, vmin=-1, cmap=cmap, square=True, linewidths=.2)
```



혹시 상관이 커서 제외하신 건 아닌가 해서 이번 과제를 수행하면서, scratch 코드로 작성하기 위해 본 유튜브로부터 배운 상관행렬을 그려 관찰해보았다. 관찰하기 위해 replace함수를 이용하여 Gender의 Male과 Female을 0과 1로 mapping 한 후 진행하였다. 그러나 Gender 데이터의 상관계수가 그렇게 크지 않아 학습에 넣어 진행시켜도 괜찮을 것 같다는 판단을 했다. 왜 제외하신 걸까 과제를 하면서 계속 생각해보았는데 시각화를 2차원으로 쉽게 하기 위해 제외하신 것 같다는 생각도 하였고, 숫자 데이터로만 과제를 수행하는 경험을 위해 제외하신 건가라는 생각도 하였다.

```
# train_test_split함수를 이용하여 데이터를 나누었다.
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Train\_test\_split 함수를 이용하여 데이터를 각각 학습데이터와 테스트데이터로 나눈 코드이다. Hold out 을 적용한 코드라고 할 수 있다.

```

# 표준화 이전 데이터 시각화 (X_train만 함)

# 한 행에 두 개의 히스토그램을 그리기 위해 subplot 생성
fig, axes = plt.subplots(1, 2, figsize=(18, 6), sharey=True)

# 첫번째 subplot에 X_train의 첫번째 열에 대한 히스토그램 빨간색으로 그림
sns.histplot(X_train[:,0], ax=axes[0], kde=True, color='r')

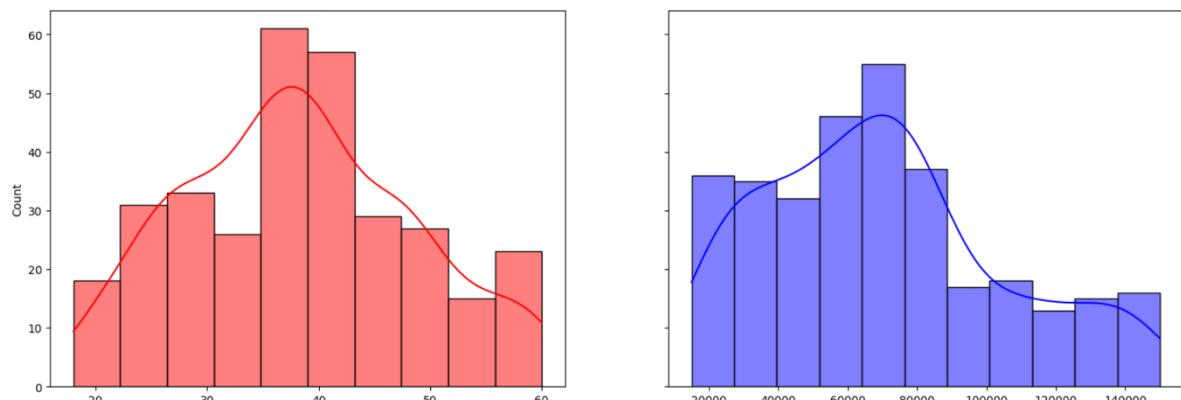
# 두번째 subplot에 X_train의 두번째 열에 대한 히스토그램 파란색으로 그림
sns.histplot(X_train[:,1], ax=axes[1], kde=True, color='b')

# 제목을 달아주었다.
plt.suptitle("Before Feature Scaling (X_train)", size = 20)

```

원래 train\_test\_split 코드 바로 다음은 숫자 데이터를 표준화하는 코드였다. 근데 과제를 수행하면서, 특히 뒷부분에 scratch 코드로 작성하기 위해 본 유튜브에서 데이터를 시각화해서 관찰하는 것이 중요하다는 가르침을 주었다. 게다가 표준화 하기 전과 후의 데이터가 어떻게 바뀌는지도 궁금하여 히스토그램을 그려 관찰해보기로 하였다. Seaborn 라이브러리를 사용하였으며 아래는 위 코드를 돌려 표준화 전 데이터를 히스토그램으로 시각화한 그래프이다. 왼쪽은 Age이고 오른쪽은 EstimatedSalary 그래프인데, 숫자의 크기 차이가 큰 걸 볼 수 있다. 두 그래프 모두 완벽한 정규분포는 아니지만 이 정도면 학습을 하기에 준수한 형태로 분포하고 있다는 생각도 아래 시각화한 그래프를 보고 깨달았다.

Before Feature Scaling (X\_train)



```

# 데이터에 표준화 처리를 하였다.
# StandardScaler.fit_transform() 함수는 데이터의 평균과 표준편차를 계산 한 후 각각 0, 1로 바꾸어 표준화해준다.
# 표준화 식은 (x-평균)/(표준편차)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

이후 데이터의 StandardScaler 함수로 표준화 처리를 해주고 표준화 이후의 히스토그램을 그려 보았다.

```

# 표준화 이후 데이터 시각화
# 양측 그래프 모두 0을 중심으로 분포하는 것을 확인할 수 있다.
# 값 차이가 엄청 많이 나던 것들이 표준화 이후 비슷한 숫자가 된 것도 확인할 수 있다.

# 한 행에 두 개의 히스토그램을 그리기 위해 subplot 생성
fig, axes = plt.subplots(1, 2, figsize=(18, 6), sharey=True)

# 첫번째 subplot에 X_train의 첫번째 열에 대한 히스토그램 빨간색으로 그림
sns.histplot(X_train[:,0], ax=axes[0], kde=True, color='r')

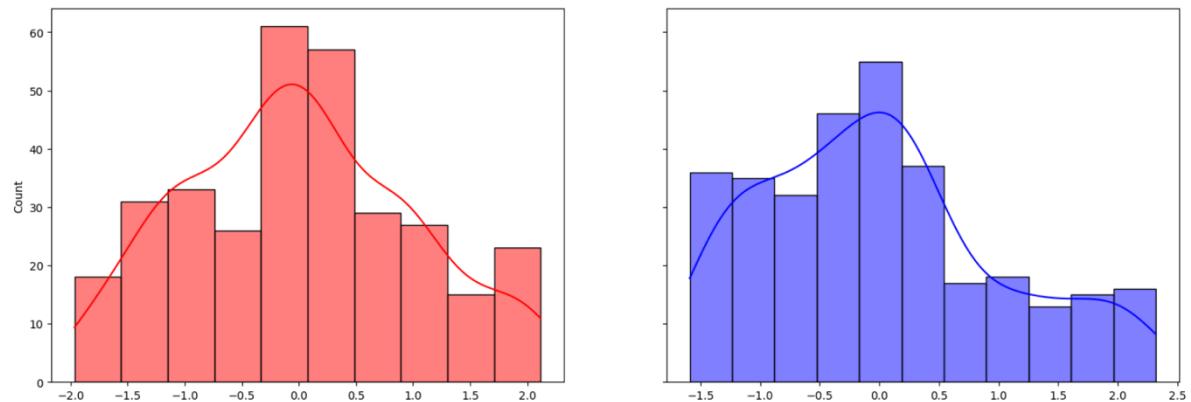
# 두번째 subplot에 X_train의 두번째 열에 대한 히스토그램 파란색으로 그림
sns.histplot(X_train[:,1], ax=axes[1], kde=True, color='b')

# 제목을 달아주었다.
plt.suptitle("After Feature Scaling (X_train)", size = 20)

```

코드는 표준화 이전 그래프와 동일하다.

After Feature Scaling (X\_train)



표준화 이후의 데이터의 히스토그램을 그린 그래프이다. 두 그래프 모두 데이터를 더 수집하거나 줄이거나 한건 아니라 분포는 그대로 유지하고 있고, 평균이 0이며 숫자 범위가 -2.0부터 2.0, -1.5부터 2.5로 비슷해진 걸 볼 수 있다. 왜 데이터를 이렇게 표준화하는 것일까? 이 역시 구글에 검색해보았는데 아주 이해하기 쉬운 예시를 찾았다. 100점 만점인 수학 시험과 990점 만점인 토익 점수를 이용해 같은 숫자지만 너무 다른 두 지표로 성적을 비교하는 것은 매우 애매하고 잘 와닿지 않을 것이다. 이를 표준화 해주어 비슷한 숫자로 만들어 주어 비교하면 비교하는 데 수월 할 것이므로 데이터에 표준화 처리를 해 주는 것이다.

```

# train 데이터 학습
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# test 데이터를 얼마나 잘 예측하는지 classifier에 test 데이터 넣어보고 정확도 확인
y_pred = classifier.predict(X_test)
ac = accuracy_score(y_test,y_pred)

print("-"*30)
print(f"\nhold out accuracy : {ac}\n")
print("-"*30)

```

hold out accuracy : 0.9125

위는 GaussianNB() 모델로 학습하고, 예측한 후 정확도를 출력한 코드이다.

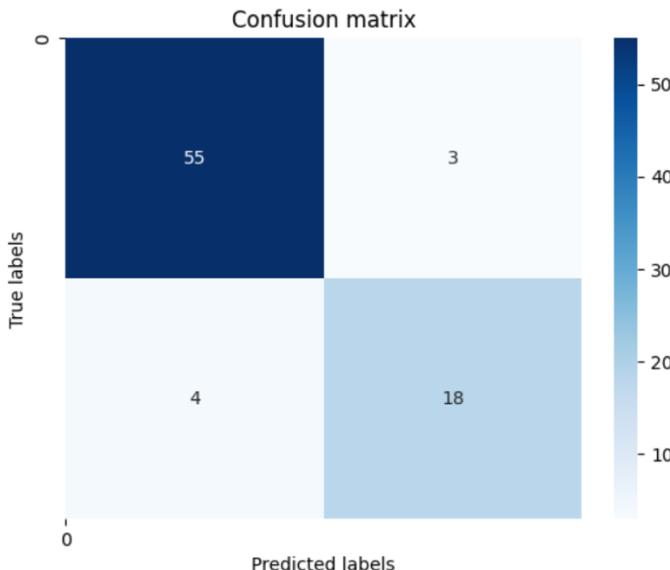
```

# test 데이터에 대한 confusion matrix
# TP, TN 위치가 y축의 위가 0이라 우리가 배운 모양이랑 바꿔어 있음
cm = confusion_matrix(y_test, y_pred)

# confusion matrix 시각화를 위해 heatmap 그리기
sns.heatmap(cm, annot=True, cmap='Blues')
plt.title('Confusion matrix')

# x축, y축 눈금 설정
ticks = np.arange(1)
plt.xticks(ticks, ticks)
plt.yticks(ticks, ticks)
# y축, x축 레이블 추가
plt.ylabel('True labels')
plt.xlabel('Predicted labels')
plt.show()

```



위는 confusion matrix 함수에 정답 테스트 데이터와 예측 데이터를 넣어 confusion matrix 를 반환 받고, 이를 이용하여 시각화한 코드이다. Sns.heatmap 속성에 annot=True를 하여 행렬 값들이 그래프에 같이 나타나도록 하였다. 위에서 정확도가 0.9 이상이 나오기도 했고, TP와 TN 비율이 높은 걸로 봐서 예측을 어느정도 잘한 듯 하다.

```
# classification_report()함수를 이용한 precision, recall, f1-score 구하기
print("*"*60)
print("\n",classification_report(y_test, y_pred),"\n")
print("*"*60)
```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.93      | 0.95   | 0.94     | 58      |
| 1            | 0.86      | 0.82   | 0.84     | 22      |
| accuracy     |           |        | 0.91     | 80      |
| macro avg    | 0.89      | 0.88   | 0.89     | 80      |
| weighted avg | 0.91      | 0.91   | 0.91     | 80      |

위는 classification\_report 함수에 정답 테스트 데이터와 예측 데이터를 입력하여 precision, recall, f1-score, support를 받은 코드이다. Support는 실제 데이터의 수를 의미한다고 한다. 0 label은 58개 1은 22개 임을 뜻한다. Macro avg는 평균에 평균을 내는 단순 평균을 내고 샘플 개수의 불균형을 고려하지 않는 지표라고 한다. Weighted avg는 샘플 개수의 불균형을 고려하여 그만큼 weight를 준 후 평균을 고려하는 지표이다. 그래서 둘의 값이 같은 평균인데 조금 다른 것을 볼 수 있다.

```
# Precision, Recall은 다른 함수로 구해보고
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1score = f1_score(y_test, y_pred)

# Sensitivity=recall
# Specificity는 confusion 행렬을 이용해 식으로 구하기
sensitivity = recall
specificity = cm[0,0]/(cm[1,0]+cm[0,0])

print("*"*30)
print(f"\nPrecision : {precision:.3f}")
print(f"Recall : {recall:.3f}")
print(f"Sensitivity : {sensitivity:.3f}")
print(f"Specificity : {specificity:.3f}")
print(f"F1-score : {f1score:.3f}\n")
print("*"*30)
```

위는 다른 함수로 precision, recall, f1score를 구하였고, sensitivity는 recall 과 같으니 그대로 적은 후 specificity는 위에서 구한 confusion matrix의 값을 이용해서 수업시간에 배운 것 처럼 구해 출력하였다.

---

```

Precision : 0.857
Recall : 0.818
Sensitivity : 0.818
Specificity : 0.932
F1-score : 0.837

```

---

코드에 따른 출력값이다.

```

# GaussianNB 모델을 사용하여 테스트 데이터의 정답 label 0, 1 중 1에 속할 확률 → positive 예측 확률 배열
lr_probs = classifier.predict_proba(X_test)[:, 1]

# ROC curve의 AUC score 계산
# 정답 라벨 (y_test)와 positive 예측 확률 배열 (lr_probs)가 들어감
lr_auc = roc_auc_score(y_test, lr_probs)

# ROC curve를 그리기 위해 fpr, tpr 계산
fpr, tpr, _ = roc_curve(y_test, lr_probs)

# roc curve 그래프 출력
plt.plot(fpr, tpr, marker='.', label="ROC curve")

# random guessing 선 추가 (여기애 그래프가 가까우면 모델 성능 최악)
plt.plot([0,1], [0,1], color = 'gray', label = 'random guessing', linestyle="--")

# perfect performance 선 추가 (여기애 그래프가 가까우면 이상적인 모델 성능)
plt.plot([0,1], [1,1], color = 'gray', label = 'perfet performance', linestyle=":")
plt.plot([0,0], [0,1], color = 'gray', linestyle=":")

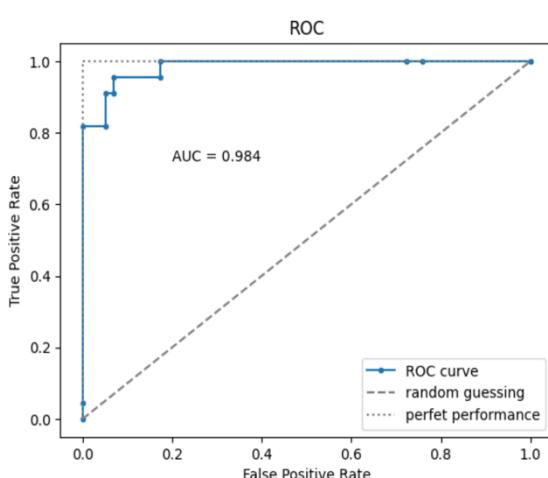
# 범례 추가
plt.legend()

# AUC 출력
plt.text(0.20,0.72,r'AUC = %0.3f'%(lr_auc),fontsize = 10)

# x축, y축 label
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

# 그래프 제목
plt.title('ROC')
plt.show()

```

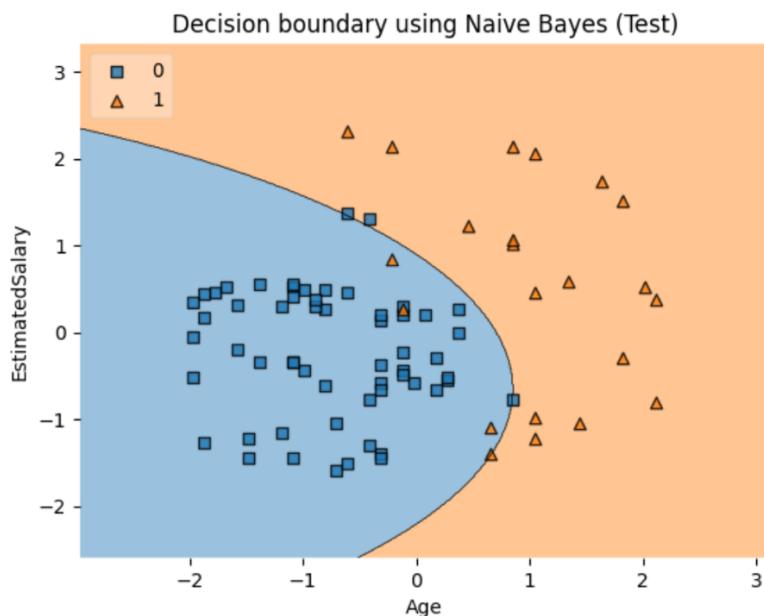


위에서 코드 분석하며 적었던 것 처럼 1이라고, positive라고 예측한 확률의 배열은 반환받아야 하므로 predict\_proba 함수로부터 `[:,1]` 1열만 반환받아 이를 이용하여 auc score를 구하고 ROC curve를 그린 코드이다. AUC score는 ROC 그래프의 아래 부분의 넓이를 숫자로 나타낸 수치이며 0.5일때는 random guessing 선과 같아 모델의 성능이 둘 중 하나를 찍었을 때와 같이 아주 나쁘게 나온 것이고, 1에 가까우면 거의 완벽하게 예측을 해낸 것이다. 본 모델에서 AUC score는 0.984로 나왔는데 예측을 잘 한 것 같다.

```
# 테스트 결과 시각화
```

```
plot_decision_regions(X_test, y_test, classifier, legend = 2)
plt.title("Decision boundary using Naive Bayes (Test)")
plt.xlabel("Age")
plt.ylabel("EstimatedSalary")
```

지금까지의 코드로 데이터를 얼마나 잘 예측하였는지, 테스트 데이터를 `plot_decision_regions` 함수에 넣어 시각화 해보았다. 0은 파란색 네모 데이터이고 1은 주황색 세모 데이터이다. 우리가 나눈 바운더리가 그래프에 나타난 것처럼 나누어져 있다. 그래프를 자세히 보면 바운더리를 벗어난 데이터가 아주 조금 있는데, 이외에는 전부 다 잘 예측되어 나누어진 것 같다. Descriptive feature들을 age와 EstimatedSalary 두개만 사용하여 간단한 코드로 2차원 데이터의 시각화를 구현할 수 있었다.



```

# KFold 교차 검증 k=10
# KFold를 하기 위해 sklearn에서 KFold import
from sklearn.model_selection import KFold

# KFold 객체 생성하고 k는 10으로 설정
kfold = KFold(n_splits=10)
accuracy = []

# 반복 횟수 초기화
n = 0

print("-" * 60)

for train_idx, test_idx in kfold.split(X):
    n += 1

    # 훈련 데이터와 검증 데이터 분류하기
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # GaussianNB() 사용하여 모델 훈련하고 예측
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    # accuracy 배열에 accuracy 계산 후 append
    accuracy.append(accuracy_score(y_test, y_pred))

print(f"\n{n}번째 KFold")
print(f"정확도 : {accuracy[n-1]:.3f} | 학습데이터 크기 : {X_train.shape[0]} | 검증데이터 크기 : {X_test.shape[0]}")

# accuracy 평균
print("\n\n예측 정확도 평균 : {:.3f}".format(sum(accuracy)/n_iter))

# accuracy 분산
print("예측 정확도 분산 : {:.3f}\n".format(np.var(accuracy)))
print("-" * 60)

```

우선 sklearn에서 제공하는 KFold 라이브러리를 가져왔다. Kfold 객체를 생성한 이후 10-fold를 하기 위해 n\_splits를 10으로 설정해주고, for문을 돌려 주었다. Kfold.split 함수는 descriptive features를 매개변수로 받으면 데이터의 개수에 따라 10번을 train index와 test index로 나누어준다. 이를 for문의 반복횟수로 정의하여 반복문을 돌릴 수 있다. (10번 돌아갈 것이다) 그리고 나서는, 함수에서 반환해준 index들로 훈련 데이터와 검증 데이터를 분류할 수 있다.

```

[ 40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57
 58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75
 76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93
 94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273
274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291
292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309
310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327
328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345
346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363
364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399]

[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]

```

옆은 이해하기 쉽도록 첫번째 kfold의 train index와 test index를 출력해본 결과 인데 아래의 test index만큼이 train index에 존재하지 않는 걸 알 수 있다. 비교적 간단한 알고리즘으로 Kfold 데이터가 나누어지는 것을 관찰하였다. 그렇게 나눈 데이터를 모델에 넣어 훈련하고 예측하였고, 시행할 때마다의 accuracy를 출력도 하고 리스트에 저장하여 나중에 평균과 분산을 구하여 출력할 수 있도록 하였다.

---

```
1번째 KFold
정확도 : 0.825 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

2번째 KFold
정확도 : 0.925 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

3번째 KFold
정확도 : 0.950 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

4번째 KFold
정확도 : 0.925 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

5번째 KFold
정확도 : 0.925 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

6번째 KFold
정확도 : 0.700 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

7번째 KFold
정확도 : 0.925 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

8번째 KFold
정확도 : 0.725 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

9번째 KFold
정확도 : 0.950 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

10번째 KFold
정확도 : 0.725 | 학습데이터 크기 : 360 | 검증데이터 크기 : 40

예측 정확도 평균 : 0.857
예측 정확도 분산 : 0.010
```

---

이는 위 코드의 결과이며 10-fold cross validation의 결과와 각 accuacy의 평균과 분산까지 나타낸 코드이다. Accuracy 평균이 0.857로 위에서 hold-out 방법으로 진행하여 구했던 accuracy인 0.9125 보다 많이 낮은 수치임을 알 수 있다.

수업 때 배운 내용대로라면 validation data 를 썼을 때 정확도가 더 높아야 하는데 왜 그런걸까?

곰곰이 생각해봤는데, 정확도로 모델을 평가함에 있어, test 데이터가 아닌 validation data로 모델을 평가해 왔음을 깨달았다. 그 이후 코드를 아래와 같이 수정하였다.

```
# KFold 교차 검증 k=10
# KFold를 하기 위해 sklearn에서 KFold import
from sklearn.model_selection import KFold

# KFold 객체 생성하고 k는 10으로 설정
kfold = KFold(n_splits=10)
fold_accuracy = []
accuracy = []

# 반복 횟수 초기화
n = 0

print("-"*60)

for train_idx, test_idx in kfold.split(X_train):
    n += 1

    # 훈련 데이터와 검증 데이터 분류하기
    # train data의 이름이 똑같아 train_data_2로 명시하였다..
    X_train_2, X_validation = X_train[train_idx], X_train[test_idx]
    y_train_2, y_validation = y_train[train_idx], y_train[test_idx]

    # GaussianNB() 사용하여 train data와 validation data로 모델 훈련하고 예측
    classifier.fit(X_train_2, y_train_2)
    y_pred_v = classifier.predict(X_validation) # validation 중 예측

    # test 데이터 예측
    y_pred = classifier.predict(X_test) # test 데이터 예측

    # fold_accuracy 배열에 validation 진행 중의 accuracy 계산 후 append
    fold_accuracy.append(accuracy_score(y_validation, y_pred_v))

    # accuracy 배열에 test 예측의 accuracy 계산 후 append
    accuracy.append(accuracy_score(y_test, y_pred))

    print(f"\n{n}번째 KFold")
    print(f"검증 정확도 : {fold_accuracy[n-1]:.3f} | 학습데이터 크기 : {X_train.shape[0]} | 검증데이터 크기 : {X_test.shape[0]}")
    print(f"테스트 정확도 : {accuracy[n-1]:.3f}")

# accuracy 평균
print("\n\n검증 정확도 평균 : {:.5f}".format(sum(fold_accuracy)))
print("예측 정확도 평균 : {:.5f}".format(sum(accuracy)))

# accuracy 분산
print("\n\n검증 정확도 분산 : {:.5f}".format(np.var(fold_accuracy)))
print("예측 정확도 분산 : {:.5f}\n".format(np.var(accuracy)))
print("-"*60)
```

기존에 나눈 train data와 test data 중 train data를 새로운 train data와 validation data로 나누었고, 이를 학습 후 예측한 데이터에 test data를 넣어 데이터를 예측해보았다. 그리고 나서 검증 중의 정확도의 test data로 구한 정확도를 모두 구하여 평균과 분산을 구해보았다.

---

1번째 KFold  
검증 정확도 : 0.875 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.938

2번째 KFold  
검증 정확도 : 0.719 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

3번째 KFold  
검증 정확도 : 0.812 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.938

4번째 KFold  
검증 정확도 : 0.875 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

5번째 KFold  
검증 정확도 : 0.812 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

6번째 KFold  
검증 정확도 : 0.844 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.912

7번째 KFold  
검증 정확도 : 0.875 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

8번째 KFold  
검증 정확도 : 0.969 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

9번째 KFold  
검증 정확도 : 0.969 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.938

10번째 KFold  
검증 정확도 : 1.000 | 학습데이터 크기 : 320 | 검증데이터 크기 : 80  
테스트 정확도 : 0.925

검증 정확도 평균 : 8.75000  
테스트 정확도 평균 : 9.27500

---

검증 정확도 분산 : 0.00664  
테스트 정확도 분산 : 0.00006

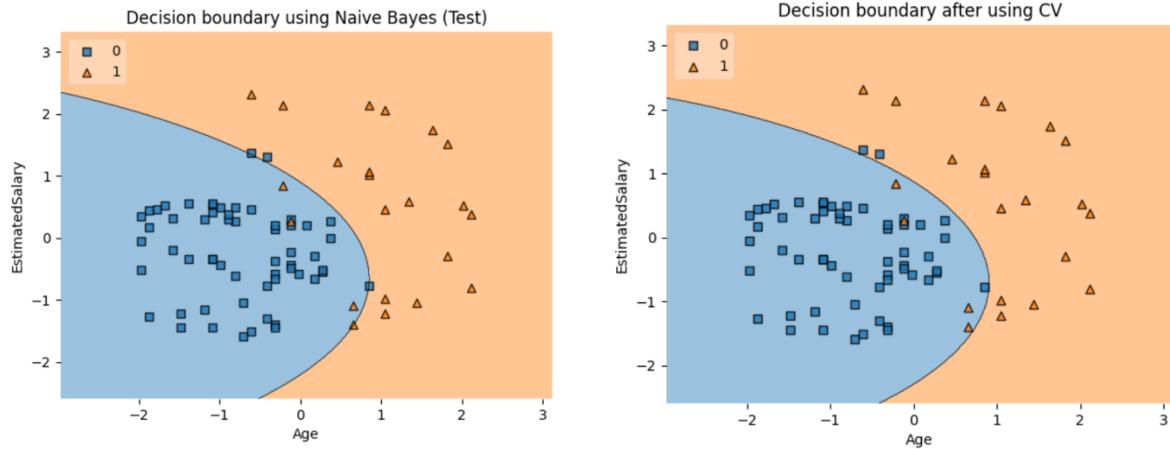
그 결과 위와 같이 정말 아름다운 결과가 나왔다. 테스트 정확도가 9.27500으로 기준에 구했던 hold out의 정확도인 0.9125 보다 0.01 이상 상승한 정확도가 나왔다. Kfold 를 잘못 이해하고 코드를 잘못 짜고 있었던 것이다. 분산 또한 0.00006으로 아주 낮게 나와 Kfold 내내 높은 정확도를 유지하며 테스트가 진행되었음을 알 수 있었다.

보고서를 쓰며 자세히 들여다보고 생각한 결과 찾아낸 오류였는데, 이를 깨닫게 되어 너무 행복했다.

10-fold cross validation 이후 데이터 분류에 대한 시각화 코드를 한번 더 작성해보았다. 코드는 위와 동일하다.

```
# 10-fold validation 이후 테스트 결과 시각화
```

```
plot_decision_regions(X_test, y_test, classifier, legend = 2)
plt.title("Decision boundary after using CV")
plt.xlabel("Age")
plt.ylabel("EstimatedSalary")
```



왼쪽이 hold-out만 했을 때의 시각화 오른쪽이 cross validation 이후의 시각화이다. 정말 미세하지만 바운더리가 조금 변한 것을 경계에 있는 데이터들의 위치에 따라 확인할 수 있다.

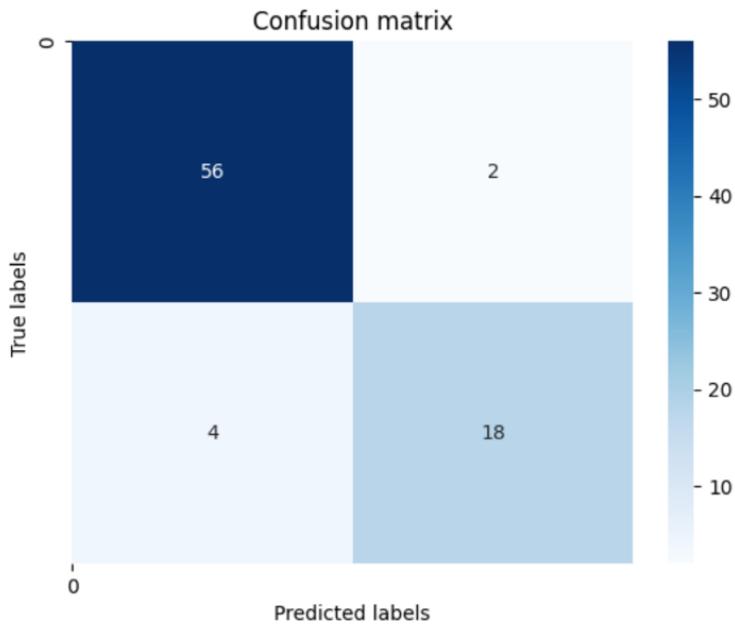
과제를 다 진행한 이후에 2번 데이터에서 제외되었던 Gender 데이터를 포함하여 학습, 예측하고 성능평가를 진행하면 어떤 결과가 나올 지 궁금해서 gender만 추가하여 코드를 복사하여 작성 후 실행해보았다.

```
X = dataset.iloc[:, [1, 2, 3]].values  
  
y = dataset.iloc[:, -1].values  
  
# Gender 의 데이터들은 문자열이므로 숫자로 치환해준다.  
dataset = dataset.replace(['Male', 'Female'], [0, 1])  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)  
  
sc = StandardScaler()  
X_train = sc.fit_transform(X_train)  
X_test = sc.transform(X_test)  
  
classifier = GaussianNB()  
classifier.fit(X_train, y_train)  
  
y_pred = classifier.predict(X_test)  
ac = accuracy_score(y_test,y_pred)  
  
print("-"*30)  
print(f"\nhold out accuracy : {ac}\n")  
print("-"*30)  
  
cm = confusion_matrix(y_test, y_pred)  
  
sns.heatmap(cm, annot=True, cmap='Blues')  
plt.title('Confusion matrix')  
  
ticks = np.arange(1)  
plt.xticks(ticks, ticks)  
plt.yticks(ticks, ticks)  
  
plt.ylabel('True labels')  
plt.xlabel('Predicted labels')  
plt.show()  
  
precision = precision_score(y_test, y_pred)  
recall = recall_score(y_test, y_pred)  
f1score = f1_score(y_test, y_pred)  
sensitivity = recall  
specificity = cm[0,0]/(cm[1,0]+cm[0,0])  
  
print("-"*30)  
print(f"\nPrecision : {precision:.3f}")  
print(f"Recall : {recall:.3f}")  
print(f"Sensitivity : {sensitivity:.3f}")  
print(f"Specificity : {specificity:.3f}")  
print(f"F1-score : {f1score:.3f}\n")  
print("-"*30)
```

---

```
hold out accuracy : 0.925
```

---



```
Precision : 0.900  
Recall : 0.818  
Sensitivity : 0.818  
Specificity : 0.933  
F1-score : 0.857
```

---

Gender를 포함하고 성능 평가를 하니, cross validation을 하지 않았음에도 다음과 같은 변화가 있었다.

Accuracy : 0.9125 -> 0.925

Precision : 0.857 -> 0.900

Recall, sensitivity : 0.818 -> 0.818

Specificity : 0.932 -> 0.933

F1-score : 0.837 -> 0.857

AUC score : 0.984 -> 0.983

Recall, sensitivity, AUC score를 제외한 모든 성능 평가 지표들의 값이 상승한 걸 볼 수 있었다.

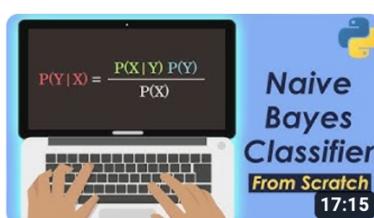
Descriptive feature가 늘어나 데이터가 많아지니 조금이지만 더 예측을 잘하게 되어 여러 지표들에서 긍정적인 결과가 나온다는 것을 알 수 있었다.

## 2.3 Naïve Bayes scratch 코드 구현

### 2.3.1 접근

2.1과 2.2에서 과제를 수행하며 사용한 scikit learn의 GaussianNB() 모델을 라이브러리를 사용하지 않고 구현해보는 경험을 갖고 싶었다.

처음에는 Scikit-learn에서 제공하는 공식 코드 문서를 보았지만 너무 어렵고 복잡하여, 간단하게 본인도 이해하고 적어볼 수 있는 코드가 있지 않을까 하고 검색을 하다가 아래와 같은 유튜브 링크와 작성자가 올려놓은 깃허브를 발견하였다. 이 두 링크에서는 정말 많은 도움을 받았기에 참고문헌에도 물론 있지만 추가로 언급한다.



### Naïve Bayes Classifier in Python

#### (from scratch!)

Normalized Nerd • 조회수 6.3만회

Let's create a Naive Bayes classifier with barebone NumPy and Pandas! You'll learn how to deal with continuous features and other...

그리고 추가적으로 더 구현하고 싶은 부분들이 있어, 위 유튜브와 함께 ChatGPT도 참고하여 Naïve Bayes 모델을 scratch로 작성해보았다.

Naïve Bayes 모델 이외에도 confusion 행렬과 그에 따른 precision, recall, sensitivity, specificity 또한 간단하지만 라이브러리 없이 구현해보았다.

### 2.3.2 구현 및 실험, 결과

```
# 배열과 행렬을 처리하기 위해 numpy import
import numpy as np

# 그래프 그리기 위해 matplotlib.pyplot import
import matplotlib.pyplot as plt

# pandas가 제공하는 2차원 데이터구조인 dataframe을 사용하기 위해 pandas import
import pandas as pd

# confusion matrix 그리기 위해 seaborn import
import seaborn as sns

# 데이터를 train, test 데이터로 각각 나누려고 import
from sklearn.model_selection import train_test_split

# 데이터 전처리 위해 가져온 라이브러리
from sklearn.preprocessing import StandardScaler

# accuracy 구하기 위해 import
from sklearn.metrics import accuracy_score

# roc_curve 그리고 auc 구하기 위해 import
from sklearn.metrics import roc_curve, auc

# 결과 시각화 그림
from mlxtend.plotting import plot_decision_regions
```

우선 코드 작성에 필요한 라이브러리들을 import 하였고, GaussianNB()와 confusion matrix 관련 라이브러리는 import 하지 않았다. Social\_Network\_ads 데이터를 이용해 2번과 동일한 과정으로 진행할 것이다.

NaiveBayes class를 아래와 같이 작성하여 모델을 구현하였다.

```
# scikit-learn에서 제공하는 모델 없이 naive bayes 구현하기

class NaiveBayes:
    def fit(self, X, y):
        # 데이터셋의 행과 열 개수를 가져옴
        n_samples, n_features = X.shape

        # 정답 label에서 class를 가져옴
        self._classes = np.unique(y)

        # 바로 위에서 가져온 class의 길이를 n_classes 변수에 저장
        n_classes = len(self._classes)

        # 평균, 분산, 사전 확률을 계산하기 위해 우선 각각의 크기에 0으로 가득 차 있는 배열 만들기
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        # 평균, 분산, 사전 확률 계산해서 배열에 저장
        for idx, c in enumerate(self._classes):
            # 현재 class c에 해당하는 sample 선택 (y == c)
            X_c = X[y == c]
            # 평균, 분산, 사전 확률을 numpy method로 계산
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)
```

우선 fit 함수를 작성하여 데이터로 모델을 학습하는 코드를 작성하였다. 사용자가 데이터 학습을 위해 객체변수명.fit()을 하면 이 함수로 오게 된다. 우선 descriptive features인 X의 shape을 n\_samples와 n\_features로 반환 받아, 데이터셋의 행과 열 개수를 변수로 가져온다. X.shape을 하면 행렬의 행과 열의 개수가 반환된다. self.\_classes 변수에는 np.unique 함수로 정답 label의 class를 가져온다. Social\_network\_ads data의 경우 0과 1이 반환될 것이다. 그리고 그 class의 개수를 n\_classes에 반환해 변수에 저장해 둔다.

그 이후 학습을 하며 평균, 분산, 사전 확률을 계산하기 위해 np.zeros 함수를 이용하여 위에서 구한 각각의 크기에 맞게 0으로 가득 차 있는 배열을 만들어준다. 그 다음 class에 해당하는 sample을 선택하여 (0==0일때, 1==1일때) X\_c 변수에 넣어주고 이의 평균, 분산, 사전 확률을 계산하여 위에서 만든 배열에 넣어 저장해준다.

```
def _pdf(self, class_idx, x):
    # fit() 함수에서 만들었던 _mean 배열에서 해당 class의 평균을 가져와 변수에 넣음
    mean = self._mean[class_idx]

    # fit() 함수에서 만들었던 _var 배열에서 해당 class의 분산을 가져와 변수에 넣음
    var = self._var[class_idx]

    # 주어진 클래스에 대한 확률 밀도 함수 계산
    numerator = np.exp(-((x - mean) ** 2) / (2 * var))
    denominator = np.sqrt(2 * np.pi * var)

    # 확률 밀도 함수값 반환
    return numerator / denominator
```

$$\frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

이 함수는 아래 코드에서 posterior를 구할 때 사용할 확률 밀도 함수값 반환 코드이다. GaussianNB 모델을 만드는 것이므로 평균과 분산을 구하여 정규분포의 확률밀도함수에 넣어 계산하여 그 값을 반환한다. 확률밀도함수의 식은 오른쪽과 같다.

```

def _calculate_posteriors(self, x):
    posteriors = []

    # 각 클래스에 대한 사후 확률 계산
    # idx : 현재 class의 index | c : class label
    for idx, c in enumerate(self._classes):

        # 현재 class에 대한 사전 확률의 log를 계산하여 prior 변수에 저장
        # 매우 작은 확률을 보다 효과적으로 처리하기 위해 수행
        # 확률의 로그를 취하는 것은 특히 매우 작은 확률이나 확률의 곱을 다룰 때 수치 계산에서 일반적인 기술이라고 함
        prior = np.log(self._priors[idx])

        # 주어진 데이터 x에 대한 확률 밀도 함수의 로그 계산 -> likelihood 계산하는 부분
        posterior = np.sum(np.log(self._pdf(idx, x)))

        # likelihood에 prior를 더함 (사후 = 사전 + 사후)
        # 로그 연산이므로 곱하게 아니라 더한것임
        # 이것은 MAP estimation
        posterior += prior

        # 위 계산 값을 posteriors 배열에 append
        posteriors.append(posterior)

    return posteriors

```

이는 전체 데이터에 대한 posterior를 계산하여 배열에 저장한 후 반환하는 함수이다. 바로 아래 작성되어 있는 predict 함수와 predict\_proba 함수에서 이 함수를 사용한다. Prior와 likelihood를 계산한 후 이 둘을 곱하여 posterior를 구하는 베이즈 정리와 동일한 식을 사용한다. 단 하나 다른 점은 log를 씌워 계산한다는 것인데, 이는 매우 작은 확률을 보다 효과적으로 처리하기 위한 것으로 확률에 로그를 취하는 것은 매우 작은 확률이나 확률의 곱을 다룰 때 수치 계산에서 일반적인 기술이라고 한다. 이 경우 확률을 구하는 것이고 매우 작은 숫자이니 곱해서 수치가 더 작아지지 않게 log를 씌워 계산하도록 하였다.

```

def predict(self, X):
    # 각각의 입력 데이터에 대해 _calculate_posteriors() 함수로 class 예측
    y_pred = [self._classes[np.argmax(self._calculate_posteriors(x))] for x in X]

    # 예측 데이터 반환
    return np.array(y_pred)

def predict_proba(self, X):
    # 각 입력 데이터에 대해 _calculate_posteriors() 함수를 사용하여 클래스 확률 예측
    posteriors = [self._calculate_posteriors(x) for x in X]

    # 소프트맥스 함수를 사용하여 확률 계산
    # 소프트맥스 함수를 사용하는 이유는
    # 일단 np.exp()의 경우 posteriors에 현재 log가 씌워져있기 때문에 해주는 것이고
    exp_posteriors = np.exp(posteriors)

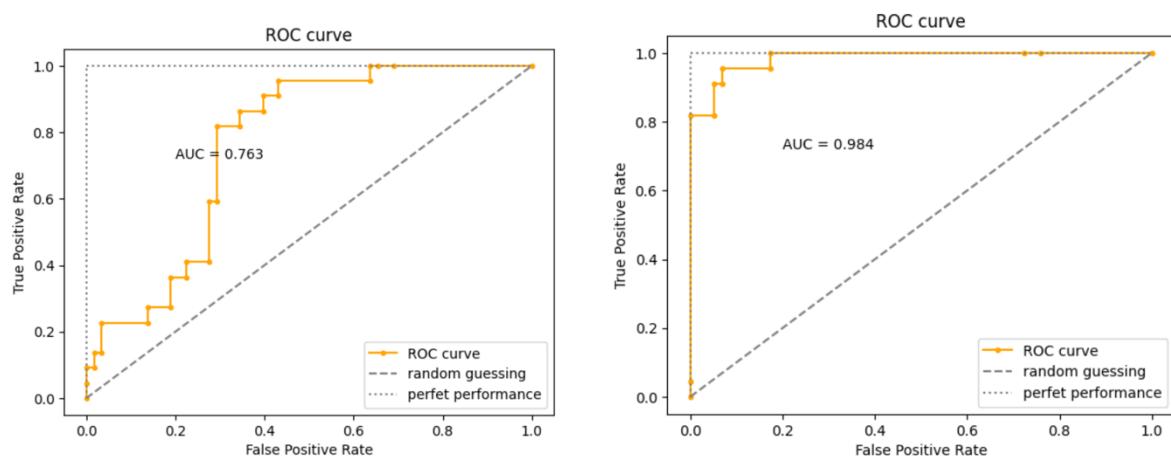
    # 여기는 확실하는 모르겠지만, 이는 소프트맥스로 확률값을 정규화하는 부분인데
    # 이 부분이 빠지면 auc값이 0.7대로 확률이 아주 낮아진다.
    # 이 부분이 들어가면 auc값이 이전에 library를 사용해서 한 것과 같게 나온다.
    # 확실하지는 않지만.. 정규화를 해줌으로써 확률값이 안정되어 학습이 더 잘 된 것 같다.
    probabilities = exp_posteriors / np.sum(exp_posteriors, axis=1, keepdims=True)

    return probabilities

```

이는 scikit learn에서처럼 predict 함수와 Predict\_proba 함수를 구현해 둔 코드이다. Predict 함수의 경우 간단하게 \_calculate\_posteriors() 함수로부터 posterior들의 배열을 받아와서 전체 데이터에 대해 각각 argmax를 진행하여 classification 결과를 y\_pred에 저장하여 반환하면 된다.

Predict\_proba 함수의 경우가 정말 복잡하고 이해하기 다소 어려운 내용이 담긴 함수였다. 이 부분은 코드를 참고한 유튜브에는 없었던 코드이고, scratch로 구현한 Naïve Bayes 모델을 이용해 ROC curve를 그리고 싶어 예측 이후 class 별 확률을 반환해주는 predict proba 함수가 꼭 필요하여, ChatGPT에 물어봐서 참고하여 작성한 코드이다. 우선 역시 self\_calculate\_posteriors() 함수를 사용하여 posterior 값들을 변수에 저장하고, 이는 log값으로 적혀있으니까 np.exp() 함수로 log를 벗겨준다. 그 다음 부분이 ChatGPT가 알려주기를 소프트맥스 활성화 함수로 확률값을 정규화하는 부분이다. 소프트 맥스가 뭔지 전혀 몰랐는데, 검색해보니 입력받은 값을 출력으로 0~1사이의 값으로 모두 정규화하며 출력 값들의 총합은 항상 1이 되는 특성을 가진 함수라고 한다. 주로 딥러닝에 사용되는데 왜 사용된건지 모르겠어서 해당 부분을 빼고 코드를 돌린 결과 이후 코드에서 돌린 AUC 값이 0.763으로 급격하게 성능이 낮아졌다.



왼쪽은 소프트맥스 함수를 제외하고 구한 AUC 값, 오른쪽은 소프트맥스 함수를 포함하고 구한 AUC 값이다. 그래서 이를 ChatGPT에게 또 물어보니 소프트맥스 함수를 적용하지 않고도 모델 학습은 시킬 수 있지만, 소프트맥스 함수를 적용하지 않을 경우 모델의 출력이 일반적으로 로짓 (logit) 형태로 나오며, 확률로 해석하기 어렵다고 한다. 또한, 확률 값의 크기 차이가 크게 나면서 수치적으로 불안정할 수 있어 이를 해결하고자 소프트맥스 함수를 사용하는 것이 일반적인 접근 방식이라고 설명해주었다. 아래는 해당 답변이다.

따라서 `np.sum(exp\_posteriors, axis=1, keepdims=True)`는 소프트맥스 함수를 통해 얻은 확률들을 정규화하기 위해 사용되며, 각 입력 데이터에 대한 클래스 확률의 합이 1이 되도록 조정합니다. 이는 다중 클래스 분류에서 각 클래스에 속할 확률을 상대적으로 비교하기 쉽게 하기 위함입니다.

현재

You

정규화를 안하면 학습이 아예 안돼?

현재

ChatGPT

정규화를 하지 않으면 학습이 안된다고 말씀드린 것은 아닙니다. 정규화는 모델의 안정성과 수렴 속도에 영향을 미칠 수 있지만, 모델이 학습되지 않는다는거나 하는 문제가 발생하는 것은 아닙니다.

정규화는 모델의 출력을 확률로 해석하기 쉽게 만들어주고, 순실 함수의 수렴에 도움을 줄 수 있습니다. 특히 다중 클래스 분류에서는 클래스 간의 상대적인 크기를 고려하는 데 도움이 됩니다. 그러나 학습이 전혀 안되는 것은 아니며, 소프트맥스 함수를 적용하지 않더라도 모델은 학습할 수 있습니다.

다만, 소프트맥스 함수를 적용하지 않을 경우 모델의 출력이 일반적으로 로짓(logit) 형태로 나오며, 확률로 해석하기 어렵습니다. 또한, 확률 값의 크기 차이가 크게 나면서 수치적으로 불안정할 수 있습니다. 이를 해결하고자 소프트맥스 함수를 사용하는 것이 일반적인 접근 방식입니다.

따라서 정규화는 모델의 안정성과 해석력을 향상시키는데 도움을 주지만, 학습이 안된다거나 하는 문제는 정규화와는 별개일 것입니다.



이해하기 어려웠지만, 소프트맥스 함수를 적용하여 log를 써뒀다 벗겨 수치적으로 불안정한 확률을 안정화 시켜 모델 학습의 성능을 높여 이와 같은 코드를 작성해주었다고 이해하였다. 여기까지만 나아가 Naïve Bayes class의 코드이다.

```

# dataset을 가져왔다.
dataset = pd.read_csv('Social_Network_Ads.csv')

# X, y를 각각 column들과 label로 데이터를 나눴다.
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# train_test_split함수를 이용하여 데이터를 나누었다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# 데이터에 표준화 처리를 하였다.
# StandardScaler.fit_transform()함수는 데이터의 평균과 표준편차를 계산한 후 표준화한다.
# 표준화 식은  $(x - \text{평균}) / (\text{표준편차})$ 

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# nb 객체 생성 후 훈련, 예측
nb = NaiveBayes()
nb.fit(X_train, y_train)
predictions = nb.predict(X_test)

print("-"*30)

# scratch로 짠 Naive Bayes 모델의 accuracy 출력
print("\nNaive Bayes 정확도 :", accuracy_score(y_test, predictions))
print()
print("-"*30)
-----
```

Naive Bayes 정확도 : 0.9125

-----

위는 2번과 동일한 코드로, 모델 부분만 바꾸어 학습을 해보았는데, scikit-learn에서 제공하는 Naïve Bayes 모델과 동일한 accuracy가 계산되었다.

같은 조건에서의 성능평가를 하기 위해 random\_state seed도 동일하게 0으로 고정해주었다.

```

# TP, TN 위치가 y축의 위가 0이라 우리가 배운 모양이랑 바꿔어 있음

# confusion_matrix 역시 scikit learn 없이 짜보기

def confusion_matrix(y_test, predictions):
    TP = 0
    FP = 0
    FN = 0
    TN = 0

    for i in range(len(predictions)):

        # True Positives : 예측 True, 정답 True
        if int(predictions[i]) == 1 and int(y_test[i]) == 1:
            TP += 1

        # False Positives : 예측 True, 정답 False
        elif int(predictions[i]) == 1 and int(y_test[i]) == 0:
            FP += 1

        # False Negatives : 예측 False, 정답 True
        elif int(predictions[i]) == 0 and int(y_test[i]) == 1:
            FN += 1

        # True Negatives : 예측 False, 정답 False
        elif int(predictions[i]) == 0 and int(y_test[i]) == 0:
            TN += 1

    return TP, FP, FN, TN

TP, FP, FN, TN = confusion_matrix(y_test, predictions)

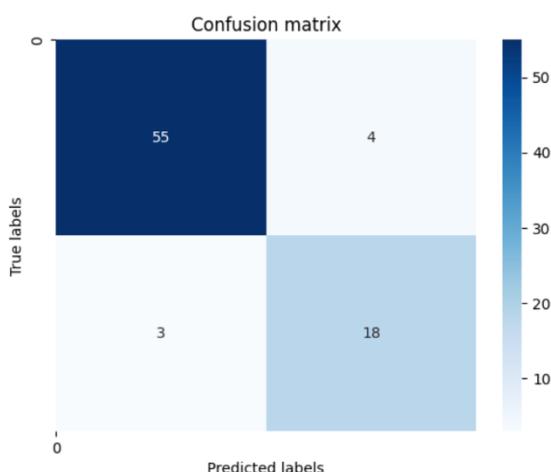
# cm을 혼동행렬로 사용
cm = np.array([[TN, FN], [FP, TP]])

# confusion matrix 시각화 (annot=True는 숫자를 그래프에 나타내는 것)
sns.heatmap(cm, annot=True, cmap='Blues')
plt.title('Confusion matrix')

# x축, y축 눈금 설정
ticks = np.arange(1)
plt.xticks(ticks, ticks)
plt.yticks(ticks, ticks)

# y축, x축 레이블 추가
plt.ylabel('True labels')
plt.xlabel('Predicted labels')
plt.show()

```



위는 confusion matrix를 scratch로 구현한 코드로, 혼동행렬 표를 바탕으로 작성하면 되는 코드이기에 크게 어렵지 않았다. TP와 TN 위치만 그 그래프의 0의 위치에 맞게 고려하여 예측 결과를 전부 반복문에 돌려 구하면 된다. 혼동행렬 역시 2번 문제와 동일하게 나왔다.

```

# 혼동행렬 값으로 precision, recall, sensitivity, specificity 계산 후 출력
print("-"*60)

precision = TP / (TP + FP)
recall = TP / (TP + FN)
sensitivity = TP / (TP + FN)
specificity = TN / (FP + TN)

print(f"\nPrecision : {precision:.3f}")
print(f"Recall : {recall:.3f}")
print(f"Sensitivity : {sensitivity:.3f}")

print(f"Specificity : {specificity:.3f}")
print(f"F1-score : {2 * (precision*recall)/(precision+recall):.3f}\n")

print("-"*60)
-----
```

```

Precision : 0.857
Recall : 0.818
Sensitivity : 0.818
Specificity : 0.948
F1-score : 0.837
-----
```

위에서 구한 혼동행렬 값으로 precision, recall, sensitivity, specificity를 계산하여 출력한 값이다.  
0| 역시 2번과 같은 결과가 나왔다.

```

# 위에서 만든 모델을 사용하여 테스트 데이터의 정답 label 0, 1 중 1에 속할 확률 -> positive 예측 확률 배열
probabilities = nb.predict_proba(X_test)[:,1]

# ROC curve로 fpr, tpr 구하기

fpr, tpr, _ = roc_curve(y_test, probabilities)

# auc() 이용해 auc 계산
roc_auc = auc(fpr, tpr)

# roc curve 그래프 출력
plt.plot(fpr, tpr, color = "orange", marker='.', label="ROC curve")

# random guessing 선 추가 (여기에 그래프가 가까우면 모델 성능 최악)
plt.plot([0,1], [0,1], color = 'gray', label = 'random guessing', linestyle="--")

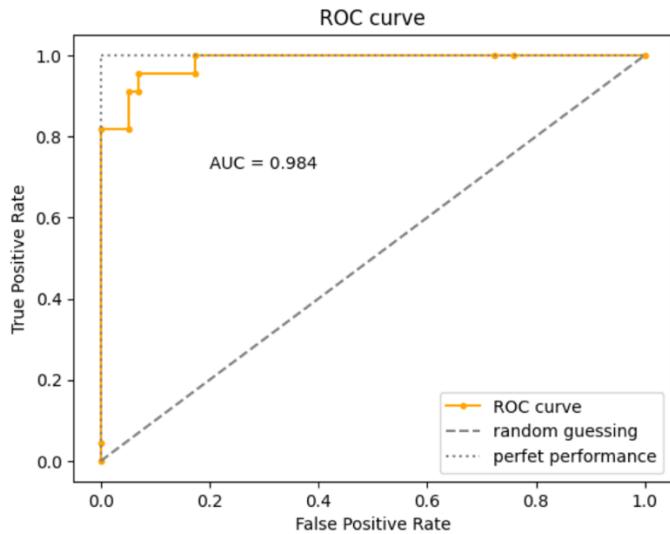
# perfect performance 선 추가 (여기에 그래프가 가까우면 이상적인 모델 성능)
plt.plot([0,1], [1,1], color = 'gray', label = 'perfet performance', linestyle=":")
plt.plot([0,0], [0,1], color = 'gray', linestyle=":")

# 범례 추가
plt.legend()

# AUC 출력
plt.text(0.20, 0.72, r'AUC = %0.3f'%(roc_auc), fontsize = 10)

# x축, y축 label
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

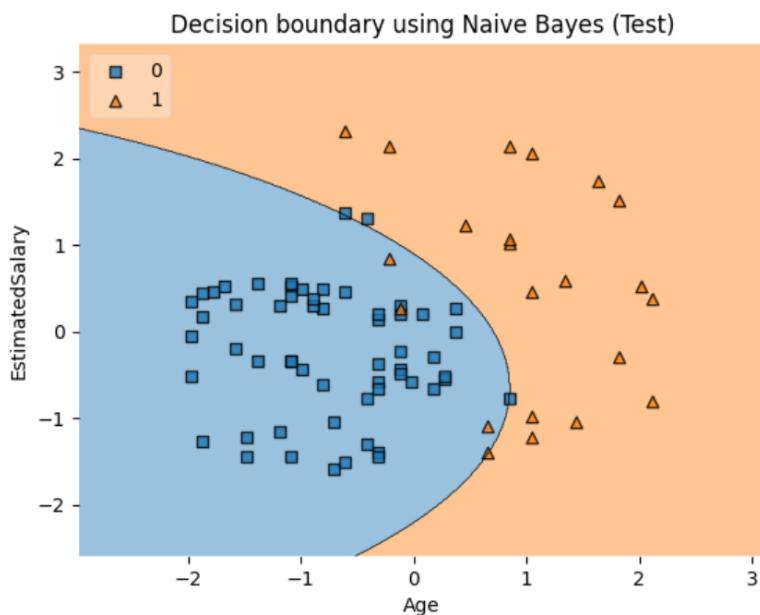
# 그래프 제목
plt.title('ROC curve')
plt.show()
```



앞서 작성한 NaïveBayes 객체의 predict\_proba 함수와 roc curve 함수, auc score 함수를 이용해 ROC curve를 그린 코드이다. 이 또한 2번과 동일하게 출력되었다.

```
# scratch code로 학습한 test 데이터의 결과 시각화
```

```
plot_decision_regions(X_test, y_test, clf = nb, legend = 2)
plt.title("Decision boundary using Naive Bayes (Test)")
plt.xlabel("Age")
plt.ylabel("EstimatedSalary")
```



시각화도 위와 같이 해보았고, 2번과 동일한 결과가 출력되었다.

### 3. 결론

#### 3.1 고찰

이렇게 Naïve Bayes에 관한 과제를 나름 최선을 다해서 작성해보았다. 처음에 코드를 봤을 때는 감도 잘 안 오고 너무 어려워보였지만, 배운 것을 바탕으로, 그리고 주어진 데이터와 코드를 바탕으로 차분하게 관찰하고 하나씩 해결해가니 이것저것 추가하고 싶은 코드들이 많아져 열심히 코드를 작성해보았다. 매 task를 해결할 때마다 재밌었지만 특히 10-fold cross validation을 진행 할 때 작성한 코드에 오류를 찾아 수정하여 성능이 올라간 부분에서는 인공지능 모델링의 참재미를 깨달은 느낌이었다. 그 동안 이론 공부만 해오고 코드를 맛보기로 몇 번 읽어본 상태에서는 인공지능이 어렵기만 하고, 다양한 라이브러리에 어색한 함수들로 가득하여 범접하기 어려운 느낌이 있었다. 그러나 이렇게 직접 작성해보고 보고서를 적으며 자세히 들여다보고 생각해보니 정말 배우는 점이 많았고 한 과제로 조금은 성장한 듯한 생각이 들었다. 더욱 더 성장하는 개발자가 되기 위해서는 바닥부터 하나씩 차근차근 공부해 나가는 게 정말 중요하구나를 더욱 절실히 깨닫게 된 소중한 과제였다.

## 4. 참고문헌

- 0) 인공지능개론 수업 자료
- 1) 데이터 예측을 위한 머신러닝 (존 캘러히, 브라이언 맥 네미, 이퍼 다시 / 에이콘)
- 2) 한권으로 다지는 머신러닝 & 딥러닝 with 파이썬 (알베르토 아르타산체스, 프라틱 조시 / 한빛미디어)
- 3) 누구나 파이썬 통계분석 (타니아이 히로키 / 한빛아카데미)
- 4) 데이터 전처리 기초  
<https://datascienceschool.net/03%20machine%20learning/02.01%20데이터%20전처리%20기초.html>
- 5) pandas.DataFrame.values 설명  
<https://runebook.dev/ko/docs/pandas/reference/api/pandas.DataFrame.values>
- 6) pandas.DataFrame.values 설명  
<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.values.html#pandas.DataFrame.values>
- 7) 데이터프레임 합치기  
[https://yganalyst.github.io/data\\_handling/Pd\\_12/](https://yganalyst.github.io/data_handling/Pd_12/)
- 8) 나이브베이즈 모델 설명  
<https://tensorflow.blog/파이썬-머신러닝/2-3-4-나이브-베이즈-분류기/>
- 9) sklearn.naive\_bayes.GaussianNB 공식문서  
[https://scikit-learn.org/stable/modules/generated/sklearn.naive\\_bayes.GaussianNB.html#sklearn.naive\\_bayes.GaussianNB.predict\\_proba](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.predict_proba)
- 10) 데이터 표준화  
<https://communities.sas.com/t5/SAS-Tech-Tip/SAS-활용-노하우-데이터-표준화-Standardization/ta-p/841526>
- 11) classification\_report() 함수  
<https://wikidocs.net/193994>
- 12) Cross-validation: evaluating estimator performance 공식문서  
[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)
- 13) K-fold cross validation  
<https://continuous-development.tistory.com/entry/MLDL-python-을-통한-교차검증-k-Fold-stratifiedkFold>
- 14) K-fold cross validation  
<https://velog.io/@jee-9/Scikit-learn-교차-검증-crossvalscore> Cross-validation-KFold-Stratified-KFold-
- 15) Kfold cross validation  
<https://dacon.io/en/codeshare/4546>
- 16) confusion\_matrix() 함수

<https://wikidocs.net/194464>

17) Precision, Recall, and F1 Score

<https://proclusacademy.com/blog/practical/precision-recall-f1-score-sklearn/>

18) Calculating Sensitivity and Specificity in Python

<https://stainfer.com/204-4-2-calculating-sensitivity-and-specificity-in-python/>

19) ROC curve, AUC score

[https://continuous-development.tistory.com/entry/MLDL-python-으로-구현하는-ROC\\_곡선과-AUC](https://continuous-development.tistory.com/entry/MLDL-python-으로-구현하는-ROC_곡선과-AUC)

20) ROC curve, AUC score

<https://zephyrus1111.tistory.com/425>

21) Naive Bayes Scratch code, 상관행렬, 히스토그램

[https://github.com/Suji04/ML\\_from\\_Scratch/blob/master/naive%20bayes.ipynb](https://github.com/Suji04/ML_from_Scratch/blob/master/naive%20bayes.ipynb)

22) Naive Bayes Scratch code

[https://youtu.be/3I8oX3OUL6I?si=7Bn5qiXpZ\\_D7UBch](https://youtu.be/3I8oX3OUL6I?si=7Bn5qiXpZ_D7UBch)

23) Naive Bayes Scratch code

<https://chat.openai.com/>

24) sklearn.preprocessing.StandardScaler

<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

25) feature scaling

<https://dacon.io/codeshare/4526>

26) plot\_decision\_regions 시각화

<https://www.theclickreader.com/naive-bayes-classifier/>

27) 활성화 함수 소프트맥스(Softmax)

<https://blog.naver.com/wideeyed/221021710286>