문제1. Fraud_data.csv 데이터를 활용하여 = ([paid, none, rent], [paid, guarantor, rent], [arrears, guarantor, rent], [arrears, guarantor, own], [arrears, coapplicant, own])의 케이스에 대해서 결과를 확인

1번문제 접근

이 문제를 해결하기 위해서는 naïve bayes의 개념에 대해 알아야한다. naive bayes정리에 대해 알려면 먼저 bayes정리가 무엇인지 알아야 한다.

bayes정리란 조건부 확률 P(A|B) (사건 B가 발생한 경우 A의 확률)의 추정이 P(A ∩B)와 P(B)에 기반을 두어야 한다는 정리이다.

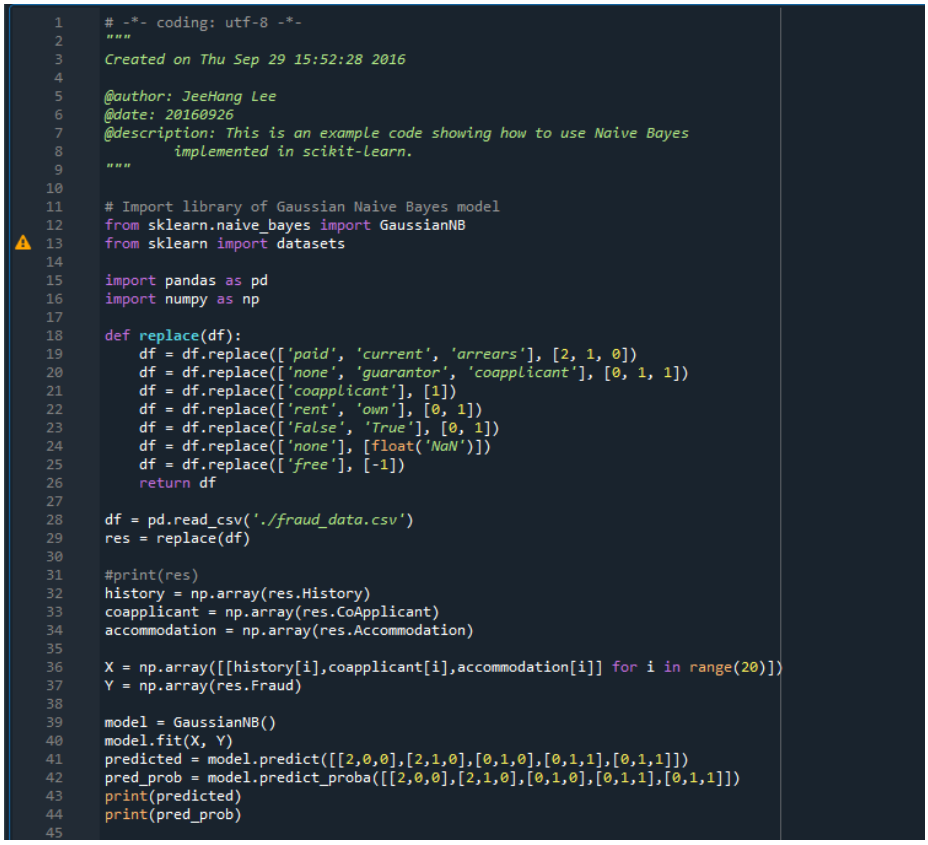
$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

naïve bayes에서 naive는 순진하다 라는 뜻이다. 이는 data set의 모든 feature들이 독립적이라는 가정을 한다고 볼 수 있다. 여러 feature들, 예를 들어 iris 꽃 품종 분류 에서 꽃받침의 너비가 길이보다 꽃 품종을 결정하는데 더 중요한 feature가 될 수 있겠지만 naïve bayes 에서는 이런 사실을 무시한다.

naive bayes로 task를 수행하는 방법은 다음과 같다.

1. 각각의 feaute들을 하나의 조건으로 보고 그때의 결과값으로 하나의 dataset을 만든다. 문제를 예로들면 History가 current고 CoApplicant가 none이며 Accommodation이 own일 때 Fraud가 true일 확률 …
2. 각각의 feature들을 파이썬의 numpy 라이브러리를 사용해서 array메소드로 list화 시킨다.
3. X에는 feature들을 Y에는 각각의 feature가 주어졌을때의 label을 할당한다.
4. GaussianNB를 통해 model.fit(X,Y)로 주어진 dataset을 학습 시킨다
5. 원하는 입력데이터(feature)를 모델에 넣어보고 예측된 결과를 확인한다.

1번문제 구현



```
# -*- 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.
"""

# 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

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)

#print(res)
history = np.array(res.History)
coapplicant = np.array(res.CoApplicant)
accommodation = np.array(res.Accommodation)

X = np.array([[history[i],coapplicant[i],accommodation[i]] for i in range(20)])
Y = np.array(res.Fraud)

model = GaussianNB()
model.fit(X, Y)
predicted = model.predict([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]])
pred_prob = model.predict_proba([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]])
print(predicted)
print(pred_prob)

```

(주어진 코드 부분은 생략)

history = np.array(res.History) //history에 res.History를 list형식으로 입력

coapplicant = np.array(res.CoApplicant) //coapplicant에 res.CoApplicant을 list형식으로 입력

accommodation = np.array(res.Accommodation) //accommodation에 res.Accommodation을 list형식으로 입력

X = np.array([[history[i],coapplicant[i],accommodation[i]] for i in range(20)]) //X에 history, coapplicant, accommodation에 입력된 값들을 차례로 입력

Y = np.array(res.Fraud) //Y에 label 값인 Fraud를 feature들에 맞게 순서대로 입력(numpy.array 사용)

model = GaussianNB()

model.fit(X, Y)

predicted = model.predict([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]]) //문제에서 주어진 조건들에 대한 결과값 예측

pred_prob = model.predict_proba([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]]) //문제에서 주어진 조건들에 대한 결과값의 확률

print(predicted) //문제에서 주어진 조건들을 예측하여 나온 결과값들을 출력

print(pred_prob) //문제에서 주어진 조건들에 대한 결과값들의 확률들을 출력

1번문제 실험(데이터)

```
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
```

그림 1 (csv파일)

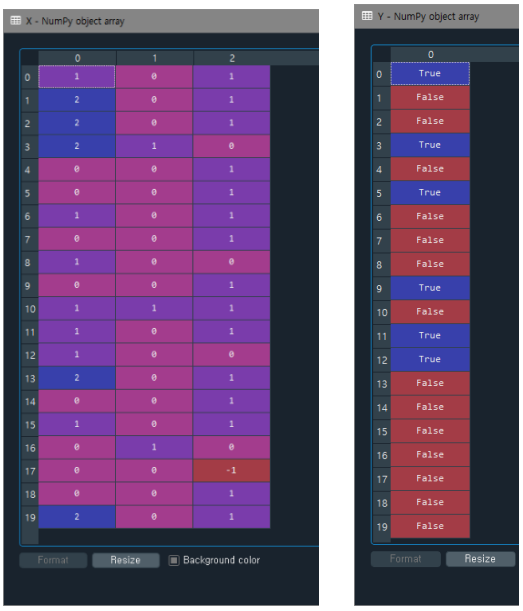


그림 2 (X, Y)

그림1 과 같은 형식으로 되어있는 csv파일을 History feature의 paid를 2로, current를 1로, arrears를 0으로 둠. CoApplicant feature의 none을 0으로, guarantor를 1로, coapplicant를 1로 둠. Accommodation feature의 rent를 0으로, own을 1로, free를 -1로 둠. Fraud label의 true를 1로, false를 0으로 둠

주어진 csv파일의 data들을 전부 숫자로 바꾼 후에 numpy의 array를 사용하여 하나의 data set을 작성후에 anaconda의 Variable explorer를 사용하여 X와 Y에 알맞은 값들이 들어갔는지 그림2로 확인.

1번문제 실험(실험 내용)

```
model = GaussianNB()
model.fit(X, Y)
predicted = model.predict([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]])
pred_prob = model.predict_proba([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]])
print(predicted)
print(pred_prob)
```

문제에서 결과를 확인해야 하는 케이스인 ([paid, none, rent], [paid, guarantor, rent], [arrears, guarantor, rent], [arrears, guarantor, own], [arrears, coapplicant, own])을 replace 함수를 통해 변환하면 각각 ([[2,0,0],[2,1,0],[0,1,0],[0,1,1],[0,1,1]])이 된다. 이를 model.predict에 넣어서 각각의 결과값을 도출 후 확인하고, model.predict_proba에 넣어서 결과값이 도출될 때의 각각의 확률을 알아본다.

1번문제 결과

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Administrator/Desktop/ML_middle/naive_bayes_todo.py', wdir='C:/Users/
Administrator/Desktop/ML_middle')
[False False False False False]
[[0.78185378 0.21814622]
 [0.68131802 0.31868198]
 [0.61436973 0.38563027]
 [0.58257789 0.41742211]
 [0.58257789 0.41742211]]

In [2]:
```

[paid, none, rent] (2,0,0) 일때 결과는 False이며 그때의 확률은 0.78185378
[paid, guarantor, rent] (2,1,0) 일때 결과는 False이며 그때의 확률은 0.68131802
[arrears, guarantor, rent] (0,1,0) 일때 결과는 False이며 그때의 확률은 0.61436973
[arrears, guarantor, own] (0,1,1) 일때 결과는 False이며 그때의 확률은 0.58257789
[arrears, coapplicant, own] (0,1,1) 일때 결과는 False이며 그때의 확률은 0.58257789

실험결과 주어진 조건들에 대한 결과가 모두 False가 나왔다. 이때 두번째 조건인 paid, guarantor, rent를 제외한 나머지 조건들은 모두 기존 dataset에 있지 않은 결과이기 때문에 코딩을 통해 만들어진 model이 스스로 학습하여 유추해낸 결과라고 볼 수 있다. 하지만 paid, guarantor, rent 조건은 주어진 dataset에 이미 들어있고, 그때의 결과label은 true로 되어있다. 하지만 이 모델에서 예측한 결과값은 false이다. 이유를 생각해보니 이 모델은 Naïve bayes 방법을 사용하여 각각의 확률을 기반으로 예측하기 때문에 주어진 dataset의 결과를 따르지 않을 수도 있다는 생각이 들었다. 주어진 dataset에서는 true라고 얘기하고 있지만 이 모델이 학습을 통해 예측을 해보니 확률상 false인 경우가 더 많다고 결과가 나왔기 때문에 예측값이 false인 것 같다.

문제2. Social_Network_Ads.csv 데이터를활용하여, 다음을수행

2-1. Hold-out 방법을 이용한 학습과 테스트를 수행하고, 테스트 결과예측정확도를 제시
2-2. 10-fold validation을 수행하여 예측정확도평균과 분산을제시(Hold-out과 차이점 확인)
2-3. Hold-out 방법을 통해 Precision, Recall, Sensitivity, Specificity, AUC 스코어를 확인 후 ROC 커브를 제시

2번문제 접근

2번 문제를 풀이하려면 Hold-out 방법과 K-fold validation에 대해 알아야 한다.
Hold-out 방법이란 dataset을 training set과 test set으로 분리한다. 사람마다 나누는 비율은 다르겠지만 예를들자면 7:3의 비율로 나눈다면 dataset의 70%를 training set으로 삼아 모델을 훈련시키는데 사용하고, 30%를 test set으로 이용해서 그 모델의 성능을 평가하는 방법이다.
이때 training set과 test set으로만 data를 나눠서 모델의 성능을 평가하게 되면 test set에만 국한되는 overfitting이 발생하게 될 수 있다. 때문에 dataset을 2개로만 나누지 않고 validation set으로도 분할하여 training set을 통해 모델을 훈련시키고, validation set으로 모델의 최적 파라미터를 찾은후에 (반복될 수 있음) 마지막에 test set으로 모델의 성능을 평가한다.

K-fold 방법은 dataset을 K개로 분할한다. 예를들어 문제에서처럼 10-fold validation의 경우 10개로 분할한다. 10개의 분할 set중에서 단 1개만 test set으로 활용하고, 나머지 9개(K-1개)는 training set으로 활용한다. 첫번째 분할set만을 test set으로 , 두번째 분할set만을 test set으로 … 이러한 방식으로 총 K번 측정한 모델의 성능을 평균을 통해 최종적으로 평가하는 방식이다.

2번문제 구현

구현 방식에서 직접 for문을 통해 분할을 하다보니 코드가 굉장히 길어 sklearn에서 제공하는 여러 메소드들을 가져다 사용했다. 메소드 하나하나 사용법과 예시, 파라미터와 return값을 확인하는 과정에서 굉장히 새로운 경험을 하게 되었고 python의 확장성에 또한번 감탄하게되었다.

```
# Import library of Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def replace(df):
    df = df.replace(['Male', 'Female'], [1, 0])
    return df

def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='red', label='ROC')
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend()
    plt.show()

df = pd.read_csv('./Social_Network_Ads.csv')
res = replace(df)

#print(res)
gender = np.array(res.Gender)
age = np.array(res.Age)
estimatedsalary = np.array(res.EstimatedSalary)
purchased=np.array(res.Purchased)

X = np.array([[gender[i],age[i],estimatedsalary[i]] for i in range(400)])
Y = np.array(purchased)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2,random_state=1)
#X_train, X_valid, Y_train, Y_valid = train_test_split(X_train,Y_train, test_size=0.2,random_state=2)

kfold = KFold(n_splits=10, shuffle=True)
dt = KFold()
score = cross_val_score(dt, X, Y, cv=kfold, scoring="accuracy")

model = GaussianNB()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
#pred_prob = model.predict_proba(X_test)
#print(predicted)
#print(pred_prob)

#precision : TP/(TP+FP)
#recall : TP/(TP+FN)
#sensitivity : TP/(TP+FN)
#specificity : TN/(FP+TN)
TN, FP, FN, TP = confusion_matrix(Y_test, predicted).ravel()
print('Hold-Out accuracy_score :',accuracy_score(Y_test, predicted))
print('Hold-Out precision_score :',TP/(TP+FP))
print('Hold-Out recall_score :',TP/(TP+FN))
print('Hold-Out sensitivity_score :',TP/(TP+FN))
print('Hold-Out specificity_score :',TN/(FP+TN))
FPR, TPR, thresholds = roc_curve(Y_test, predicted, pos_label=1)
print('Hold-Out auc score :',auc(FPR, TPR))
print(' 10-fold ave:',score.mean())
print(' 10-fold var:',score.var())
print(FPR, TPR, thresholds)
plot roc curve(FPR, TPR)
```

(주석처리한 코드들은 배제후 작성하였음)

```
def replace(df): //1번문제와 마찬가지로 주어진 csv파일에서 string 형식을 int로 전환
    df = df.replace(['Male', 'Female'], [1, 0])
    return df

def plot_roc_curve(fpr, tpr): //ROC커브를 작성하는 함수 FPR,TPR 입력시 ROC커브가 그려짐
    plt.plot(fpr, tpr, color='red', label='ROC')
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend()
    plt.show()

gender = np.array(res.Gender) //gender에 res.Gender을 list 형식으로 입력
age = np.array(res.Age) //age에 res.Age를 list 형식으로 입력
estimatedsalary = np.array(res.EstimatedSalary) //estimatedsalary에 res.EstimatedSalary를 list 형식으로 입력
purchased=np.array(res.Purchased) //purchased에 res.Purchased를 list 형식으로 입력

X = np.array([[gender[i],age[i],estimatedsalary[i]] for i in range(400)]) //X에 gender, age, estimatedsalary를 차례로 입력
Y = np.array(purchased) // Y에 purchased를 차례로 입력

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2,random_state=1) //Hold-out방법으로 X,Y dataset을 분할하여 각각의 _train,_test set에 할당. train과 test의 비율은 8:2 (validation set을 만들려면 X_train, Y_train을 가지고 한번더 Hold-out을 진행하면됨)

kfold = KFold(n_splits=10, shuffle=True) //Kflod는 주어진 data를 10개로 쪼개는 역할
dt = DecisionTreeClassifier() //kfold를 위한 모델을 의사결정트리로 생성
score = cross_val_score(dt, X, Y, cv=kfold, scoring="accuracy") //cross_val_score함수로 Kfold방법의 estimate score를 return

model = GaussianNB()
model.fit(X_train, Y_train) //Hold-out방식에서 trainining set으로 모델훈련
predicted = model.predict(X_test) //Hold-out방식에서 test set으로 모델의 성능평가
```

```
TN, FP, FN, TP = confusion_matrix(Y_test, predicted).ravel() //confusion_matrix를 통해 실제 값 ==Y_test, 예측한 값 == predicted()를 입력하여 TN,FP,FN,TP를 return

#문제에서 구현해야하는 Precision, Recall, Sensitivity, Specificity는 다음과 같은 방법으로 계산
#precision : TP/(TP+FP)
#recall : TP/(TP+FN)
#sensitivity : TP/(TP+FN)
#specificity : TN/(FP+TN)

print('Hold-Out accuracy_score :',accuracy_score(Y_test, predicted)) //Hold-out방법의 정확도
print('Hold-Out precision_score :',TP/(TP+FP))
print('Hold-Out recall_score :',TP/(TP+FN))
print('Hold-Out sensitivity_score :',TP/(TP+FN))
print('Hold-Out specificity_score :',TN/(FP+TN))

FPR, TPR, thresholds = roc_curve(Y_test, predicted, pos_label=1) //ROC커브를 그리기위해 FPR과 TPR, thresholds(한계점) 설정
print('Hold-Out auc score :',auc(FPR, TPR)) //AUC score

print(' 10-fold ave:',score.mean()) //K-fold방법으로 구한 모델의 성능의 평균값
print(' 10-fold var:',score.var()) //K-fold방법으로 구한 모델의 성능의 분산값
print(FPR, TPR, thresholds)
plot_roc_curve(FPR, TPR) //ROC커브 출력
```

2번문제 실험(데이터)

X	Array of int64	(400, 3)	[[1 19 19000] [1 35 20000]
X_test	Array of int64	(80, 3)	[[1 36 33000] [0 39 61000]
X_train	Array of int64	(320, 3)	[[0 29 28000] [0 45 22000]
Y	Array of int64	(400,)	[0 0 0 ... 1 0 1]
Y_test	Array of int64	(80,)	[0 0 1 ... 0 0 0]
Y_train	Array of int64	(320,)	[0 1 1 ... 1 1 0]

Help Variable explorer Plots Files

X에는 csv파일에서 3개의 feature가 400개씩 들어있고, Y에는 1개의 label이 400개 들어있다. X_test와 X_train을 보면 80개와 320개로 2:8의 비율로 분할 되어있는 것을 확인할 수 있다.
Y_test와 Y_train 또한 80개와 320개로 2:8의 비율로 분할 되어있는 것을 확인할 수 있다.

2번문제 실험(실험 내용)

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2,random_state=1)
#X_train, X_valid, Y_train, Y_valid = train_test_split(X_train,Y_train, test_size=0.2,random_state=2)

kfold = KFold(n_splits=10, shuffle=True)
dt = DecisionTreeClassifier()
score = cross_val_score(dt, X, Y, cv=kfold, scoring="accuracy")

model = GaussianNB()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
pred_prob = model.predict_proba(X_test)
#print(predicted)
#print(pred_prob)

#precision : TP/(TP+FP)
#recall : TP/(TP+FN)
#sensitivity : TP/(TP+FN)
#specificity : TN/(FP+TN)
TN, FP, FN, TP = confusion_matrix(Y_test, predicted).ravel()
print('Hold-Out accuracy_score :',accuracy_score(Y_test, predicted))
print('Hold-Out precision_score :',TP/(TP+FP))
print('Hold-Out recall_score :',TP/(TP+FN))
print('Hold-Out sensitivity_score :',TP/(TP+FN))
print('Hold-Out specificity_score :',TN/(FP+TN))
FPR, TPR, thresholds = roc_curve(Y_test, predicted, pos_label=1)
print('Hold-Out auc score :',auc(FPR, TPR))
print(' 10-fold ave:',score.mean())
print(' 10-fold var:',score.var())
print(FPR, TPR, thresholds)
plot_roc_curve(FPR, TPR)
```

Hold-out방법과 K-fold방법을 통하여 각각 모델을 구현하여 Hold-out의 경우 Precision, Recall, Sensitivity, Specificity, AUC score를 확인하였고, K-fold의 경우 평균과 분산을 확인했다. 또한 Hod-out방법으로 AUC score를 확인한 후에 ROC커브를 구현했다.

2번문제 결과

Hold-out 방법의 정확도 : 86.25%

Precision score : 0.8

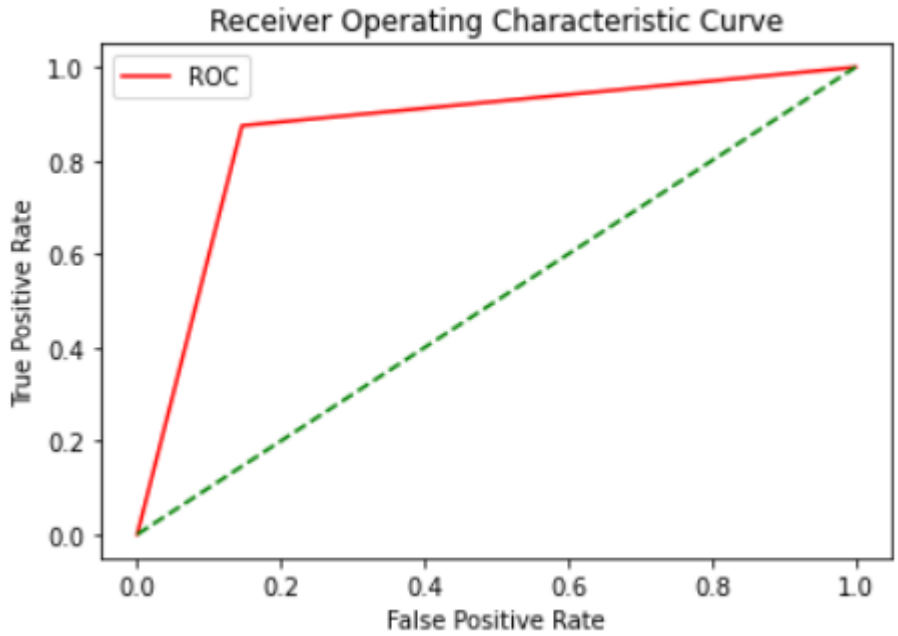
Recall score : 0.875

Sensitivity score : 0.875

Specificity score : 0.8541666…

AUC score : 0.8645833333333334

K-fold 방법의 모델의 성능 평균 : 0.8550000000000001 (실험을 반복할때마다 값의 차이가 발생, 하지만 대체로 0.85를 유지)
K-fold 방법의 모델의 성능 분산 : 0.0026000000000000025 (실험을 반복할때마다 값의 차이가 발생, 하지만 대체로 0.0012~0.0031을 유지)
K-fold 방법의 모델의 성능 평균과 Hold-out 방법의 정확도를 비교해보면 Hold-out 방법의 정확도가 미세하게 더 높은 것을 볼 수 있다. 하지만 그 차이가 크지 않다. Hold-out 방법에서 training set의 data들과 test set의 data들이 유사도가 높다면 hold-out 방법에서의 정확도가 충분히 높게 나올 수 있다. 혹은 그 반대의 경우도 가능하다.



ROC커브가 좌상단에 근접해 있다고 볼 수 있기 때문에(AUC score : 0.864…) 이 모델은 구현이 잘 된 모델이라고 할 수 있다.

Hold-out 방법으로 모델을 구현한다면 쉽고 빠르게 구현이 가능하지만 전체데이터에서 test set은 trainining에 사용할 수 없게 되므로 data를 모두 사용할 수 없다는 단점이 존재한다
K-fold 방법 같은 경우 모든 data를 training과 test에 사용할 수 있고 각각 분할했을 때 새로운 dataset이 되기 때문에 정확도가 조금 더 올라간다는 장점이 있다. 하지만 K값이 증가할수록 수행시간과 계산량이 많아지기 때문에 K값 설정에 유의해야한다.

고찰 및 결론

1번 문제에서 Naïve bayes를 이용하여 각각의 feature들에 따른 결과값을 예측하면서 예측 값이 주어진 dataset에 항상 적합하는 결과가 나오지 않는다는 것을 알게 되었다. 이는 naïve bayes를 이용한 분류모델이 확률을 기반으로 결과를 예측하기 때문에 모델이 학습한 확률의 결과에 따라 예측값이 충분히 달라질 수 있기 때문이기 때문이다. 2번 문제에서는 Social_Network_Ads.csv파일을 Hold-out 방법과 K-fold방법을 통해 두 방법 간의 정확도를 비교하고, ROC커브를 그리면서 ROC커브가 의미하는 바가 무엇인지 생각하게 되었다. ROC커브는 분류모델의 성능을 표현하는 커브로써 FPR과 TPR의 비율을 표현한다. 최종적으로는 커브가 좌상단에 가까울수록 더 좋은 모델임을 알 수 있다.

2번 문제에서 실험을 진행하기 전에 공부한 내용을 바탕으로 결과를 예상했을 때 Hold-out방법보다 K-fold방법의 정확도가 더 높을 것으로 예상했다. 왜냐하면 K-fold방법은 K번 분할할 때마다 각각의 validation set이 매번 새로워서 overfitting의 가능성이 낮아 좀더 좋은 training을 할 것이라고 생각했기 때문이다. 하지만 Hold-out의 정확도는 86.25 (86.25%), K-fold 방법의 정확도의 평균은 0.85 (85%)를 유지했다. 예상과 틀린 이유를 생각해보니 Hold-out 방법에서 training set에서 학습한 data와 test set에서 성능을 평가한 data가 유사한다면 Hold-out의 정확도가 더 높을 수 도 있다는 결론이 나왔다. 하지만 K-fold 방법이 K번에 걸쳐 training, validation, test하기 때문에 K-fold방법의 정확도에 신뢰가 간다.

이번 중간고사 대체과제를 진행하면서 Naïve bayes 기반 분류에서 Hold-out방법과 K-fold방법에 대해 좀더 자세히 공부하고, 직접 실험할 수 있어 유익한 시간이었던 것 같다. Naïve bayes는 각각의 feature들을 모두 독립으로 보는데 실제사례에서는 그 feature들 간의 연관성을 무시할 수 없다는 것 또한 알게 되어 feature들 간의 관계를 충분히 고려할 만한 분류기법이 있는지 궁금해졌다. 또한 주어진 문제를 해결하기 위해 sklearn의 여러 함수들과 라이브러리들의 구현방식, 파라미터와 return값들에 대해 좀더 자세히 알아보는 기회가 되었다.