기 계 학 습 실 습

3 주 차

3주차 학습 목표

1. 2주차 실습 내용인 numpy와 pandas를 익숙하게 사용할 수 있습니다
2. Scikit-learn library를 사용하여 확률적 예측을 할 수 있습니다
3. Plotly를 사용하여 데이터와 결과에 대한 시각화를 할 수 있습니다

1. Load datasets

Dataset을 불러오고, 가공하기 위해 이하 library를 사용합니다

1. numpy - 불러온 data를 가공하기 위한 library입니다
2. pandas - dataset을 불러오고 저장하기 위한 library입니다
3. os - 경로설정을 위한 library입니다

```
import numpy as np
import pandas as pds
import os
```

1. Load datasets

실습에 사용될 dataset은 자료실에 올려두었습니다

내려 받은 후, 아래 경로를 데이터를 받은 경로로 입력해주세요

OS에 의존하지 않는 경로설정을 위해 os library를 사용합니다

os library의 document는 아래 주소입니다

https://docs.python.org/ko/3/library/os.html

```
path=os.path.join('/','tensor2','dataset','toy','toy_3','data.csv')
print(path)
```

```
/tensor2/dataset/toy/toy_3/data.csv
```

1. Load datasets

Dataset을 불러온 후, 변수와 target으로 dataset을 분리합니다

```
data=pds.read_csv(path)
data.head()
```

	x1	x2	y
0	3.278761	-0.505664	1.0
1	3.321877	-0.122872	1.0
2	3.040468	0.024588	1.0
3	4.496212	0.686581	1.0
4	4.715463	0.024845	1.0

```
target=data.iloc[:,-1]
data=data.iloc[:,:-1]

data.shape,target.shape
```

```
((18000, 2), (18000,))
```

2. Plot.ly

Plotly는 최근 개발된 python을 위한 시각화 툴입니다

Plotly를 활용하여 data의 시각화를 시도하겠습니다

Plotly의 documents는 아래 주소입니다

https://plotly.com/python/

Data는 2차원이기 때문에 x1, x2 평면상에 점으로 표현하겠습니다

2. Plot.ly

Plotly의 plot하는 과정은 크게 두 단계로 나뉩니다

1. 그림을 그릴 데이터들을 그릴 방식과 함께 입력합니다(graph_objs)
2. 그림의 여백을 생성하고 표현합니다(figure, subplots)

시각화를 위해 dataset을 변수 별로 분리합니다

```
from plotly import graph_objs as go

x1=data.transpose().iloc[0]
x2=data.transpose().iloc[1]
```

2. Plot.ly

```
ax=go.Scatter(x=x1,y=x2,mode='markers')

fig=go.Figure()
fig.add_trace(ax)

fig_options={
    'layout':dict(template='simple_white',
                  width=800,height=700,
                  title='Plotting toy example'),
}

fig.update(fig_options)
fig.show()
```

ax라는 변수에 graph_objs를 저장합니다

저장된 object는 평면 위의 점들을 plot한 data이며 각각 x1, x2입니다

2. Plot.ly

```
ax=go.Scatter(x=x1,y=x2,mode='markers')

fig=go.Figure()
fig.add_trace(ax)

fig_options={
    'layout':dict(template='simple_white',
                  width=800,height=700,
                  title='Plotting toy example'),
}

fig.update(fig_options)
fig.show()
```

변수 fig에 여백 figure를 저장한 후,

add_trace로 직전에 만들어둔 plot object를 전달합니다

2. Plot.ly

```
ax=go.Scatter(x=x1,y=x2,mode='markers')

fig=go.Figure()
fig.add_trace(ax)

fig_options={
    'layout':dict(template='simple_white',
                  width=800,height=700,
                  title='Plotting toy example'),
}

fig.update(fig_options)
fig.show()
```

figure를 show로 나타내기 전에

User의 기호에 따라 figure를 설정합니다

figure_option을 update하고, 시각화합니다

2. Plot.ly

```
ax1=go.Scatter(x=x1.loc[target==0],
               y=x2.loc[target==0],mode='markers',
               marker=dict(color='rgba(200, 200, 40, 0.6)')
               ,name='Target 0')
ax2=go.Scatter(x=x1.loc[target==1],
               y=x2.loc[target==1],mode='markers',
               marker=dict(color='rgba(50, 171, 50 , 0.6)')
               ,name='Target 1')
```

Target별 data들을 시각화하기 위한 object를 생성합니다

```
fig=go.Figure()
fig.add_trace(ax1)
fig.add_trace(ax2)
```

Figure를 생성하고 object들을 update합니다

```
fig_options={'layout':dict(template='simple_white',
                   width=800,height=700,
                   title='Plotting toy example'),}

fig.update(fig_options)
fig.show()
```

Figure option을 설정하고 update 후 figure를 시각화합니다

3. Naïve bayes classifier

Scikit-learn의 Gaussian naïve bayes를 사용하겠습니다

각 class별 prior는 9:1로 설정합니다

```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB(priors=[0.9,0.1])
```

3. Naïve bayes classifier

model.fit을 사용하여 모델의 parameter를 학습시키거나,

준비상태로 만들 수 있습니다

fit 함수의 인자로 data와 target을 전달해야합니다

```
model.fit(data,target)
```

```
GaussianNB(priors=[0.9, 0.1])
```

3. Naïve bayes classifier

model.predict를 사용하여 학습된 모델로 예측할 수 있습니다

predict하는 과정에서는 data만 인자로 입력되어야 합니다

학습된 Naïve bayes classifier로 예측한 결과와

실제 target을 비교하면 아래와 같습니다

```
print(model.predict(data))
print(target.values)
```

```
[0. 0. 0. ... 0. 0. 0.]
[1. 1. 1. ... 0. 0. 0.]
```

3. Naïve bayes classifier

결과의 정확도와 예측한 결과의 확률 값들은 다음과 같습니다

```
(model.predict(data)==target.values).mean()*100
```

```
89.83333333333333
```

```
print(model.predict_proba(data)
print(trad_probs.values)
```

```
[[0.90041311 0.09958689]
 [0.90028257 0.09971743]
 [0.91038271 0.08961729]
 ...
 [0.86779785 0.13220215]
 [0.89535336 0.10464664]
 [0.91714716 0.08285284]]
```

3. Naïve bayes classifier

결과의 정확도와 예측한 결과의 확률 값들은 다음과 같습니다

```
(model.predict(data)==target.values).mean()*100
```

```
89.83333333333333
```

```
print(model.predict_proba(data)
print(trad_probs.values)
```

```
[[0.90041311 0.09958689]
 [0.90028257 0.09971743]
 [0.91038271 0.08961729]
 ...
 [0.86779785 0.13220215]
 [0.89535336 0.10464664]
 [0.91714716 0.08285284]]
```

3. Naïve bayes classifier

Prior가 1:1인 경우 다음과 같이 model을 생성하게 됩니다

그리고 결과는 아래와 같습니다

```
equal_model=GaussianNB(priors=[0.5,0.5])
equal_model.fit(data,target)
print('prior를 주어진 target으로 계산한 경우 : ',model.predict(data))
print('prior를 특정한 경우 : ',equal_model.predict(data))
print('실제 target의 경우 : ',target.values)
```

```
prior를 주어진 target으로 계산한 경우 :  [0. 0. 0. ... 0. 0. 0.]
prior를 특정한 경우 :  [0. 0. 0. ... 1. 1. 0.]
실제 target의 경우 :  [1. 1. 1. ... 0. 0. 0.]
```

3. Naïve bayes classifier

Prior가 1:1인 경우 다음과 같이 model을 생성하게 됩니다

그리고 예측 결과는 아래와 같습니다

```
equal_model=GaussianNB(priors=[0.5,0.5])
equal_model.fit(data,target)
print('prior를 주어진 target으로 계산한 경우 : ',model.predict(data))
print('prior를 특정한 경우 : ',equal_model.predict(data))
print('실제 target의 경우 : ',target.values)
```

```
prior를 주어진 target으로 계산한 경우 :  [0. 0. 0. ... 0. 0. 0.]
prior를 특정한 경우 :  [0. 0. 0. ... 1. 1. 0.]
실제 target의 경우 :  [1. 1. 1. ... 0. 0. 0.]
```

3. Naïve bayes classifier

Prior가 target의 분포에 기반을 둘 때와
user가 입력한 prior에 기반을 둘 때의 결과비교입니다

```
print(f'prior를 입력하지 않은 경우의 정확도 : {
(model.predict(data)==target.values).mean()*100:.4}%')
print(f'prior를 입력하지 않은 경우 1이라고 예측한 횟수 : {
(model.predict(data)).sum()}₩n')
print(f'prior를 입력한 경우의 정확도 : {
(equal_model.predict(data)==target.values).mean()*100:.4}%')
print(f'prior를 입력한 경우 1이라고 예측한 횟수 : {
(equal_model.predict(data)).sum()}')
```

```
prior를 입력하지 않은 경우의 정확도 : 89.83%
prior를 입력하지 않은 경우 1이라고 예측한 횟수 : 0.0

prior를 입력한 경우의 정확도 : 72.18%
prior를 입력한 경우 1이라고 예측한 횟수 : 6432.0
```

3. Naïve bayes classifier

Prior가 target의 분포에 기반을 둘 때와
user가 입력한 prior에 기반을 둘 때의 결과비교입니다

```
print(f'prior를 입력하지 않은 경우의 정확도 : {
(model.predict(data)==target.values).mean()*100:.4}%')
print(f'prior를 입력하지 않은 경우 1이라고 예측한 횟수 : {
(model.predict(data)).sum()}₩n')
print(f'prior를 입력한 경우의 정확도 : {
(equal_model.predict(data)==target.values).mean()*100:.4}%')
print(f'prior를 입력한 경우 1이라고 예측한 횟수 : {
(equal_model.predict(data)).sum()}')
```

```
prior를 입력하지 않은 경우의 정확도 : 89.83%
prior를 입력하지 않은 경우 1이라고 예측한 횟수 : 0.0

prior를 입력한 경우의 정확도 : 72.18%
prior를 입력한 경우 1이라고 예측한 횟수 : 6432.0
```

4. Conclusion

prior별 예측 결과와 실제 target을 한번에 plot하기 위한 code입니다

다양한 object를 하나의 여백에 그려야 하기 때문에,

여백 안쪽에 그림을 그릴 공간 3개를 만들어줍니다

```
from plotly.subplots import make_subplots

fig = make_subplots(rows=1, cols=3,
                    subplot_titles=("Real data",
                                    "Ratio of priors = 9:1",
                                    "Ratio of priors = 1:1")
                   )
```

4. Conclusion

첫 번째 공간에 들어갈 그림입니다

실제 target의 정보를 담은 code이고,

trace update할 때 공간을 row =1, col = 1로 명시해줍니다

```
ax1=go.Scatter(x=x1.loc[target==0],
               y=x2.loc[target==0],mode='markers',
               marker=dict(color='rgba(200, 200, 40, 0.6)'),
               showlegend=False)
ax2=go.Scatter(x=x1.loc[target==1],
               y=x2.loc[target==1],mode='markers',
               marker=dict(color='rgba(50, 171, 50 , 0.6)'),
               showlegend=False)


fig.add_trace(ax1,row=1,col=1)
fig.add_trace(ax2,row=1,col=1)
```

4. Conclusion

두 번째 공간에 들어갈 그림입니다

prior가 9:1인 model의 예측 결과 정보를 담은 code이고,

trace updat할 때 공간을 row =1, col = 2로 명시해줍니다

```
ax1=go.Scatter(x=x1.loc[model.predict(data)==0],
               y=x2.loc[model.predict(data)==0],mode='markers',
               marker=dict(color='rgba(200, 200, 40, 0.6)'),
               showlegend=False)
ax2=go.Scatter(x=x1.loc[model.predict(data)==1],
               y=x2.loc[model.predict(data)==1],mode='markers',
               marker=dict(color='rgba(50, 171, 50 , 0.6)'),
               showlegend=False)


fig.add_trace(ax1,row=1,col=2)
fig.add_trace(ax2,row=1,col=2)
```

4. Conclusion

세 번째 공간에 들어갈 그림입니다

prior가 1:1인 model의 예측 결과 정보를 담은 code이고,

trace updat할 때 공간을 row =1, col = 3로 명시해줍니다

```
ax1=go.Scatter(x=x1.loc[equal_model.predict(data)==0],
               y=x2.loc[equal_model.predict(data)==0],mode='markers',
               marker=dict(color='rgba(200, 200, 40, 0.6)'),name='Target 0')
ax2=go.Scatter(x=x1.loc[equal_model.predict(data)==1],
               y=x2.loc[equal_model.predict(data)==1],mode='markers',
               marker=dict(color='rgba(50, 171, 50 , 0.6)'),name='Target 1')


fig.add_trace(ax1,row=1,col=3)
fig.add_trace(ax2,row=1,col=3)
```

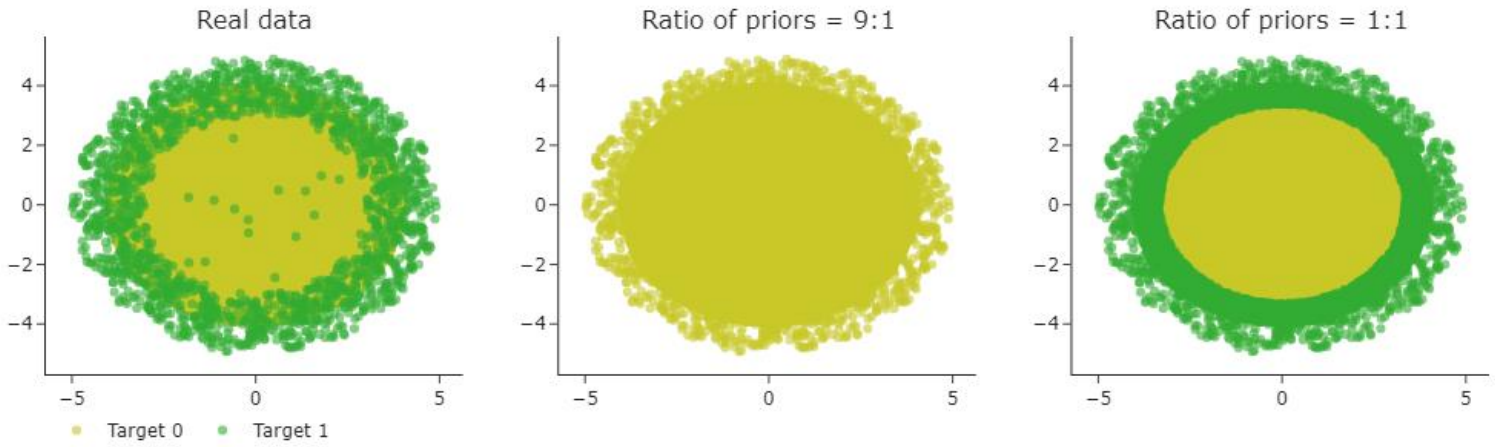
4. Conclusion

figure를 update하고
시각화합니다

그 결과는 아래와 같습니다

```
fig_options={'layout':dict(template='simple_white',
                           width=1100,height=400,
                           legend=dict(orientation="h",
                                       yanchor="top"))}

fig.update(fig_options)
fig.show()
```



4. Conclusion

Naïve bayes classifier의 결론입니다

1. Naïve bayes에 기반하여 분류를 수행하기 위해,
   중요한 두 가지 가정이 존재했습니다

   1) 각 변수들은 독립적입니다

   2) 데이터의 변수 별 분포는 모두 정규분포입니다

2. 예측 결과는 사전 확률인 prior에 의존적이고,
   prior에 따라 모델의 예측 성능이 조절 될 수 있습니다

4. Conclusion

과제 공지

메일로 질문을 받으면, 답변은 질문과 함께 e-learning의

질문답변 게시판을 통해 드리도록 하겠습니다

과제와 관련된 질문은 수요일 자정 이전까지 해주셔야 합니다

4. Conclusion

Reference.

1. Maximum likelihood estimation

https://en.wikipedia.org/wiki/Maximum_likelihood_estimation
https://ratsgo.github.io/statistics/2017/09/23/MLE/

2. Navie bayes classifier

https://en.wikipedia.org/wiki/Naive_Bayes_classifier
https://datascienceschool.net/view-notebook/c19b48e3c7b048668f2bb0a113bd25f7/