머신러닝

[HW01- NAÏVE BAYESIAN CLASSIFIER DESIGN]

담당교수 : 박철수 교수님

학　　번 : 2021202058

성　　명 : 송채영

[Introduction]

이 과제의 제목은 NAÏVE BAYESIAN CLASSIFIER DESIGN이며, 과제의 목적은 Naïve Bayesian classifier를 구현하고, setosa, versicolor, virginica, 이 세가지 꽃 이미지에 따라 꽃의 종류를 얼마나 정확하게 분류하는지 test하는 것이다. 각 함수를 구현한 한 후, 그것을 바탕으로 prior, likelihood, posterior을 구해본다. 이 과제를 통해 전반적인 Bayesian theorem에 대해 이해해보고 실제로 적용해본다.

[Assignment]

다음은 소스코드에 관한 설명이다.

```
def feature_normalization(data): # 10 points
    # parameter
    feature_num = data.shape[1]
    data_point = data.shape[0]

    # you should get this parameter correctly
    normal_feature = np.zeros([data_point, feature_num])
    mu = np.zeros([feature_num])
    std = np.zeros([feature_num])

    # your code here
    # calculate mean and standard deviation
    mu = np.mean(data, axis = 0)
    std = np.std(data, axis = 0)

    # normalize data
    normal_feature = (data - mu) / std
    # end

    return normal_feature
```

위 코드는 주어진 데이터를 feature normalization하는 함수이다. normal_feature는 정규화된 데이터를 저장할 배열이며, mu는 각 feature의 평균을 저장하는 배열, std는 각 feature의 표준 편차를 저장하는 배열이다. np.mean()과 np.std() 함수를 사용하여 data의 각 feature에 대한 평균(mu)과 표준 편차(std)를 계산하였다. $\bar{x} = \frac{x-mu}{\sigma}$ 공식(데이터에서 평균을 빼고 표준편차로 나눔)을 사용하여 normalization을 진행하였다.

```
# split the data and labels into training and testing sets based on the split factor
def split_data(data, label, split_factor):
    return  data[:split_factor], data[split_factor:], label[:split_factor], label[split_factor
```

위 코드는 data와 label을 트레이닝 세트와 테스트 세트로 분할하는 함수이다. main 함수에서 이 함수를 호출하면 split_factor 값을 기준으로 트레이닝 세트와 테스트 세트로 나누어진다. main 함

수에서 해당 값이 100으로 설정되어 있어, 100을 기준으로 100장, 50장으로 나뉜다.

```
def get_normal_parameter(data, label, label_num): # 20 points
    # parameter
    feature_num = data.shape[1]

    # you should get this parameter correctly
    mu = np.zeros([label_num,feature_num])
    sigma = np.zeros([label_num,feature_num])

    # your code here
    # calculate mean and standard deviation for each feature for data with label i
    for i in range(label_num):
        mu[i] = np.mean(data[label == i], axis=0)
        sigma[i] = np.std(data[label == i], axis=0)
    # end

    return mu, sigma
```

위 코드는 트레이닝 데이터에 대해 평균과 표준 편차를 계산하는 함수이다. 반복문을 사용하여 label의 수만큼 반복하며 각 label에 대한 데이터의 평균과 표준 편차를 계산한다. 이때 data[label == i]는 label이 i인 데이터만을 선택하여 해당 label에 대한 데이터를 추출하는 것이다.

```
def get_prior_probability(label, label_num): # 10 points
    # parameter
    data_point = label.shape[0]

    # you should get this parameter correctly
    prior = np.zeros([label_num])

    # your code here
    # calculate prior probability for each label
    for i in range(label_num):
        prior[i] = np.sum(label == i) / data_point
    # end
    return prior
```

위 코드는 트레이닝 데이터에 대한 prior를 계산하는 함수이다. 반복문을 사용하여 label의 수만큼 반복하며 각 label에 대한 prior를 계산한다. 이때 np.sum(label == i)를 사용하여 해당 label(i)가 발생한 횟수를 모두 더해주어 데이터 포인트의 수로 나누어주어 구하였다.

```python
def Gaussian_PDF(x, mu, sigma): # 10 points
    # calculate a probability (PDF) using given parameters
    # you should get this parameter correctly
    pdf = 0

    # your code here
    # calculate the exponent part
    exponent = -(x - mu) ** 2 / (2 * sigma ** 2)
    # calculate the pdf using the Gaussian formula
    pdf = (1 / (np.sqrt(2 * np.pi * sigma ** 2))) * np.exp(exponent)
    # end

    return pdf
```

위 코드는 주어진 입력값 x에 대한 가우시안 확률 밀도 함수(Gaussian Probability Density Function, PDF)를 계산하는 함수이다. 과제 제안서 그대로 구현하였으며 수식은 아래와 같다.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-u)^2}{2\sigma^2}}$$

```python
def Gaussian_Log_PDF(x, mu, sigma): # 10 points
    # calculate a probability (PDF) using given parameters
    # you should get this parameter correctly
    log_pdf = 0

    # your code here
    # calculate the exponent part
    exponent = -(x - mu) ** 2 / (2 * sigma ** 2)
    # calculate the log pdf using the Gaussian formula
    log_pdf = -0.5 * np.log(2 * np.pi * sigma ** 2) + exponent
    # end

    return log_pdf
```

위 코드는 주어진 입력값 x에 대한 가우시안 확률 밀도 함수 (Gaussian Probability Density Function, PDF)의 log를 계산하는 함수이다. 과제 제안서 그대로 구현하였으며 수식은 아래와 같다.

$$\log(f(x)) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x-u)^2}{2\sigma^2} = (-\frac{1}{2})(\frac{(x-u)^2}{2\sigma^2} + \log(2\pi\sigma^2))$$

```python
def Gaussian_NB(mu, sigma, prior, data): # 40 points
    # parameter
    data_point = data.shape[0]
    label_num = mu.shape[0]

    # you should get this parameter correctly
    likelihood = np.zeros([data_point, label_num])
    posterior = np.zeros([data_point, label_num])
    ## evidence can be ommitted because it is a constant

    # your code here
        ## Function Gaussian_PDF or Gaussian_Log_PDF should be used in this section
    # loop through each data point
    for i in range(data_point):
        # Loop through each class
        for j in range(label_num):
            # calculate the log likelihood
            likelihood[i, j] = np.sum(Gaussian_Log_PDF(data[i], mu[j], sigma[j]))
            # calculate the posterior probability
            posterior[i, j] = likelihood[i, j] + np.log(prior[j])

            #likelihood[i, j] = np.prod(Gaussian_PDF(data[i], mu[j], sigma[j]))
            #posterior[i, j] = np.log(likelihood[i, j]) + np.log(prior[j])
    # end
    return posterior
```

위 코드는 가우시안 Naïve Bayes 분류기를 구현한 함수로 입력된 데이터를 기반으로 각 label에 속할 확률을 구한다. data point를 반복하며 각 data point 마다 모든 클래스에 대해 likelihood 및 posterior을 계산한다. likelihood는 Gaussian_Log_PDF 함수를 사용하여 계산하였고, posterior은 계산된 likelihood에 prior를 더해(log로 인해 덧셈이 됨) 계산한다. 추가적으로 주석으로 Gaussian_PDF 함수를 사용한 버전도 구현하였는데, log를 사용한 이유는 수치적인 관점에서 더 좋다고 생각했기 때문이다.

```python
# predicts class labels based on posterior probabilities
def classifier(posterior):
    data_point = posterior.shape[0]
    prediction = np.zeros([data_point])

    prediction = np.argmax(posterior, axis=1)

    return prediction

# calculates classification accuracy
def accuracy(pred, gnd):
    data_point = len(gnd)

    hit_num = np.sum(pred == gnd)

    return (hit_num / data_point) * 100, hit_num
```

위 코드는 classifier, 분류기와 accuracy, 분류 정확도를 계산하는 함수이다. 우선 classifier 함수는

posterior probabilities를 기반으로 class label을 예측하며, accuracy 함수는 예측 값과 실제 값을 기반으로 분류 정확도를 계산한다.

[Result]

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.32246831e-16 -8.02321172e-16 -2.45729363e-16 -1.48029737e-17]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.44089210e-16 -8.05281767e-16  7.57912251e-16  2.66453526e-16]
accuracy is 96.0% !
the number of correct prediction is 48 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.44089210e-16 -8.05281767e-16  7.49030467e-16  2.66453526e-16]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.38168020e-16 -7.87518199e-16  2.36847579e-16  2.48689958e-16]
accuracy is 96.0% !
the number of correct prediction is 48 of 50 !
```

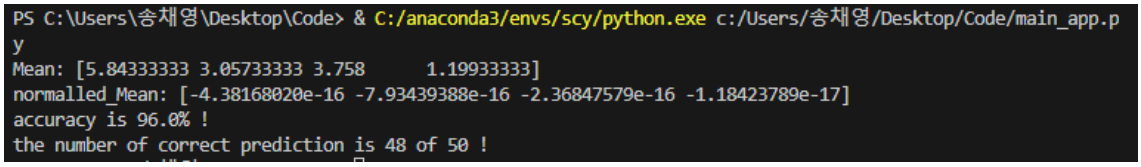
```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.44089210e-16 -8.11202957e-16 -2.44249065e-16  2.66453526e-16]
accuracy is 92.0% !
the number of correct prediction is 46 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [ 6.09882515e-16  1.22716652e-15 -2.60532336e-16 -3.73034936e-16]
accuracy is 96.0% !
the number of correct prediction is 48 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.26325641e-16  2.33886984e-16 -2.36847579e-16 -3.58231963e-16]
accuracy is 96.0% !
the number of correct prediction is 48 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.27805939e-16  1.77635684e-16 -2.66453526e-16  2.69414121e-16]
accuracy is 100.0% !
the number of correct prediction is 50 of 50 !
```

```
PS C:\Users\송채영\Desktop\Code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/Code/main_app.p
y
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.44089210e-16 -8.06762065e-16 -2.30926389e-16  2.48689958e-16]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```



결과를 10번 출력해 본 결과이다. 데이터가 shuffle 되기 때문에 매 실행마다 다른 결과가 나오는 것을 확인할 수 있었다. 10번 실행(98%, 96%, 98%, 96%, 92%, 96%, 96%, 100%, 98%, 96%)을 통해 얻은 accuracy의 평균을 구해보니 96.6%가 나왔으며, 이를 통해 모델이 안정적으로 작동함을 알 수 있다.

[Reference]

- 머신러닝 과제 제안서
- 머신러닝 ch 4 pdf
- Naive Bayes classifier - Wikipedia
- Standard score - Wikipedia
- [개념 정리] Maximum Likelihood Estimation 와 Log Likelihood (tistory.com)