1 과제 목적

3종의 꽃의 4 feature로 구성된 주어진 data를 가지고 training하여, feature를 통해 꽃의 종을 분류하는 NAIVE BAYESIAN CLASSIFIER를 디자인한다. 각 feature는 가우시안 분포를 따른다고 가정하며 bayesian theorem을 이용해서 확률을 계산한다.

2 코드 설명

2.1 feature_normalization 함수

```
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

    # 각 feature의 mean, std 얻어서
    mu = np.mean(data, 0)
    std = np.std(data, 0)
    # data가 표준정규분포 속성 가지도록 data-mu / std
    normal_feature = ((data + np.negative(mu))/std)

    # end

    return normal_feature
```

입력받은 data를 각 feature에 대해 정규화해서 반환하는 함수이다. 각 feature data 값에 평균을 빼고 표준편차로 나누어서, 해당 feature에 대한 data가 평균이 0이고 표준편차가 1인 표준정규분포를 따르도록 할 수 있다.

2.2 split_data 함수

```
def split_data(data, label, split_factor):
    return  data[:split_factor], data[split_factor:], label[:split_factor], label[split_factor:]
```

입력받은 data와 label을 split_factor를 기준으로 분할해서 반환하는 함수이다. training을 통해 classifier를 디자인한 다음에는 구현한 classifier가 얼마나 정확한지 test가 필요하므로, 미리 data를 어디에 사용할 것인지 구분해놓을 필요가 있다.

```python
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

    # 각 label에 대해서
    for i in range(label_num):
        # 특정 label의 data index
        index = np.where(label == i)
        # index와 data를 가지고 mean, std 계산
        mu[i, :] = np.mean(data[index], 0)
        sigma[i, :] = np.std(data[index], 0)

    # end

    return mu, sigma
```

2.3 get_normal_parameter 함수

data와 label을 입력받아서 각 label에 해당하는 data의 평균과 분산을 반환하는 함수이다. data를 classify하기 위해서는 각 label에 해당하는 data의 분포가 어떤 형태를 띄고 있는지 알아야 한다. data의 분포를 통해 새로운 data가 어느 label에 해당할 확률이 높은지 계산할 수 있기 때문이다.

2.4 get_prior_probability 함수

```python
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

    # 각 label에 대해서
    for i in range(label_num):
        # label의 확률 계산
        prior[i] = np.size(np.where(label == i)) / data_point

    # end

    return prior
```

label을 입력받아서 prior를 계산해서 반환하는 함수이다. prior는 각 label이 나올 확률이다.

2.5 Gaussian_PDF 함수

x, mu, sigma를 입력받아 mu, sigma를 parameter로 하는 gaussian pdf에 x를 대입했을 때의 결과 값을 반환하는 함수이다.

2.6 Gaussian_Log_PDF 함수

x, mu, sigma를 입력받아 mu, sigma를 parameter로 하는 gaussian pdf에 x를 대입했을 때의 결과 값을 log를 취해 반환하는 함수이다.

```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

    # 정규분포 pdf식에 mu, sigma, x를 대입해서 pdf값 계산
    pdf = np.exp(- (x-mu)**2 / (2*(sigma**2)) ) / (np.sqrt(2*np.pi)*sigma)

    # end

    return pdf
```

```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

    # log를 취한 정규분포 pdf식에 mu, sigma, x를 대입해서 log_pdf값 계산
    log_pdf = -np.log(np.sqrt(2*np.pi)*sigma) - (x-mu)**2 / (2*(sigma**2))

    # end

    return log_pdf
```

2.7 Gaussian_NB 함수

```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

    # 각 data에 대해
    for i in range(data_point): #100
        # 각 label에 대해
        for j in range(label_num): #3
            # 각 feature와 특정 label의 Gaussian_Log_PDF 합을 통해 likelihood 계산
            # Log_PDF이므로 sum으로 계산가능
            likelihood[i, j] = np.sum(Gaussian_Log_PDF(data[i,:], mu[j,:], sigma[j,:]))
            # likelihood에 log를 취한 prior를 더해서 posterior 계산
            # p(x)는 uniform이므로 argmax 계산에 영향을 끼치지 않음
            posterior[i, j] = likelihood[i, j] + np.log(prior[j])

    # end

    return posterior
```

입력받은 mu, sigma, prior, data를 가지고 posterior를 계산해서 반환하는 함수이다. 먼저 Gaussian_PDF 함수 또는 Gaussian_Log_PDF 함수를 이용해서 특정 data의 feature값들의 각 label에 대한 유사도 likelihood를 계산한다. 그리고 likelihood 값에 prior을 추가해서 posterior를 계산한다. BAYESIAN에서는 p(x)를 나누어주지만, p(x)가 모든 data에서 동일하여 classifier에 영향을 끼치지 않는다. Gaussian_PDF 함수를 사용하면 각 확률값을 곱하고, Gaussian_Log_PDF 함수를

사용하면 각 확률값에 log를 취해서 더해준다.

2.8 classifier 함수

```
def classifier(posterior):
    data_point = posterior.shape[0]
    prediction = np.zeros([data_point])

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

    return prediction
```

data별 posterior의 값을 비교해서 posterior가 최대가 되는 위치를 반환하는 함수이다. data의 label별로 posterior값을 계산했으므로, posterior가 가장 큰 위치의 label에 속한다고 판단할 수 있다.

2.9 accuracy 함수

```
def accuracy(pred, gnd):
    data_point = len(gnd)
    hit_num = np.sum(pred == gnd)

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

pred(예측값), gnd(실제값)을 비교해서 정확도를 측정하는 함수이다.

3 실행 결과

1만번 training 및 test를 수행해본 결과 98.75%가 90% 이상의 정확도를 보였다.

References

[1] wikipedia. Normal distribution. https://en.wikipedia.org/wiki/Normal_distribution.

```
PS D:\HD\강의자료\4_1_강의자료\머신러닝\과제\Homework_01>  d:; cd 'd:\HD\강의자료
'; & 'C:\Scoop\apps\python\current\python.exe' 'C:\Scoop\persist\vscode\data\exte
Files\lib\python\debugpy\launcher' '11318' '--' 'd:\HD\강의자료\4_1_강의자료\머신
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.44089210e-16 -8.08242362e-16 -2.60532336e-16  2.57571742e-16]
accuracy is 94.0% !
the number of correct prediction is 47 of 50 !
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.08562073e-16 -7.93439388e-16 -2.79406128e-16  2.64973229e-16]
accuracy is 100.0% !
the number of correct prediction is 50 of 50 !
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [ 6.18764299e-16 -8.05281767e-16  2.25745348e-16  2.36847579e-16]
accuracy is 94.0% !
the number of correct prediction is 47 of 50 !
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.35207426e-16 -8.05281767e-16  2.33886984e-16  2.66453526e-16]
accuracy is 96.0% !
the number of correct prediction is 48 of 50 !
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.35207426e-16 -8.05281767e-16  2.30926389e-16 -7.40148683e-18]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

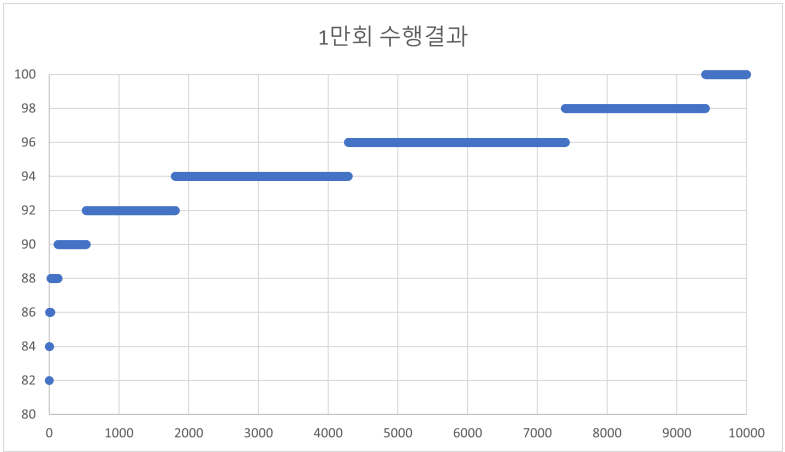

Figure 1: (a) 콘솔 출력 결과 (b) 1만회 수행 결과