머신러닝

HW01

NAÏVE BAYESIAN CLASSIFIER DESIGN

2019202050 이강현

Introduction

꽃 사진 데이터를 이용해서 Bayesian classifier를 디자인해본다. 꽃 이미지 데이터를 기반으로 하여 bayesian theorem의 prior, likelihood, posterior를 구하기 위한 각각의 함수들을 구현해보면서 bayesian theorem의 실제 적용법에 대해 이해할 수 있다. 또한 이를 기반으로 뽑아낸 확률분포를 평가해보면서 이미지 분류를 해볼 수 있다.

Source Code

```
def split_data(data, label, split_factor):
    return  data[:split_factor], data[split_factor:], label[:split_factor], label[split_factor:] # split data based on split_factor
```

```
  split_factor: 100
∨ test_data: array([[-5.37177559e-01,  1.47939788e+00, -1.28338910e+00,
  > special variables
  > [0:50] : [array([-0.53717756,  1.47939788, -1.2833891 , -1.3154443 ]), array([-1.5…
  > dtype: dtype('float64')
    max: 2.6303817157821316
    min: -1.7433568431321493
  > shape: (50, 4)
    size: 200
> test_label: array([0, 0, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 2, 1, 2, 2, 0, 1, 2, 1, 0,
∨ training_data: array([[-1.02184904e+00, -1.31979479e-01, -1.22655167e+00,
  > special variables
  > [0:100] : [array([-1.02184904, -0.13197948, -1.22655167, -1.3154443 ]), array([ 1.…
  > dtype: dtype('float64')
    max: 3.090775248299425
    min: -2.433947141908088
  > shape: (100, 4)
```

split_data 함수 실행 후 split_factor 값 100을 기준으로 training data, test data로 각각 100장, 50장으로 나뉜 것을 확인할 수 있다.

```python
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])

    # Calculate average and standard deviation value based on feature
    mu = np.mean(data,axis = 0)
    std = np.std(data,axis = 0)

    #normalization
    normal_feature = (data - mu)/std

    return normal_feature
```

데이터에서 평균을 빼고 표준편차로 나누어주어 normalization을 진행하였다.

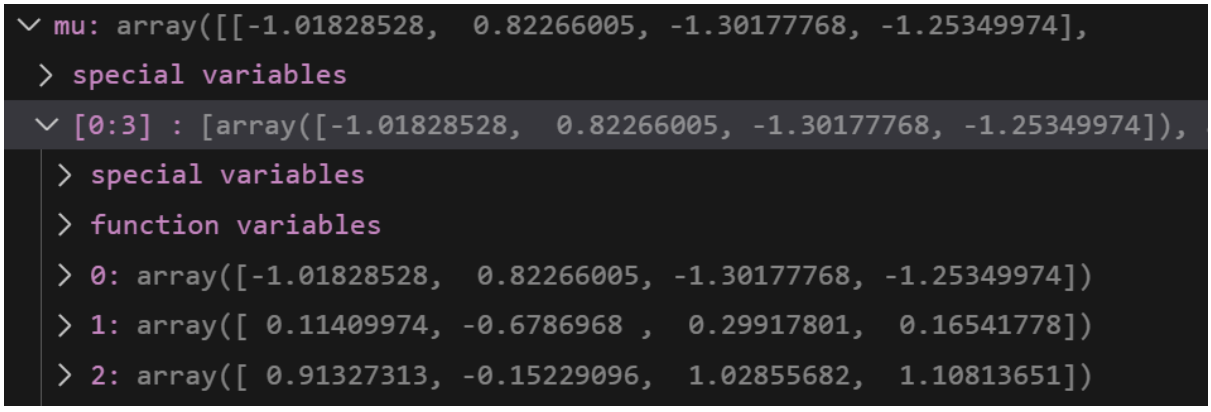
```
> 000: array([5.7, 2.5, 5. , 2. ])
> 001: array([6.7, 3.3, 5.7, 2.1])
> 002: array([6.7, 3.3, 5.7, 2.5])
> 003: array([5.5, 4.2, 1.4, 0.2])
> 004: array([5. , 2. , 3.5, 1. ])
> 005: array([6.6, 2.9, 4.6, 1.3])
> 006: array([6. , 3.4, 4.5, 1.6])
> 007: array([6.7, 3. , 5. , 1.7])
> 008: array([6.2, 2.9, 4.3, 1.3])
> 009: array([5.7, 3.8, 1.7, 0.3])
> 010: array([5.5, 2.4, 3.7, 1. ])
> 011: array([6. , 2.7, 5.1, 1.6])
> 012: array([5.5, 3.5, 1.3, 0.2])
> 013: array([6.1, 2.6, 5.6, 1.4])
> 014: array([6.4, 3.2, 4.5, 1.5])
```

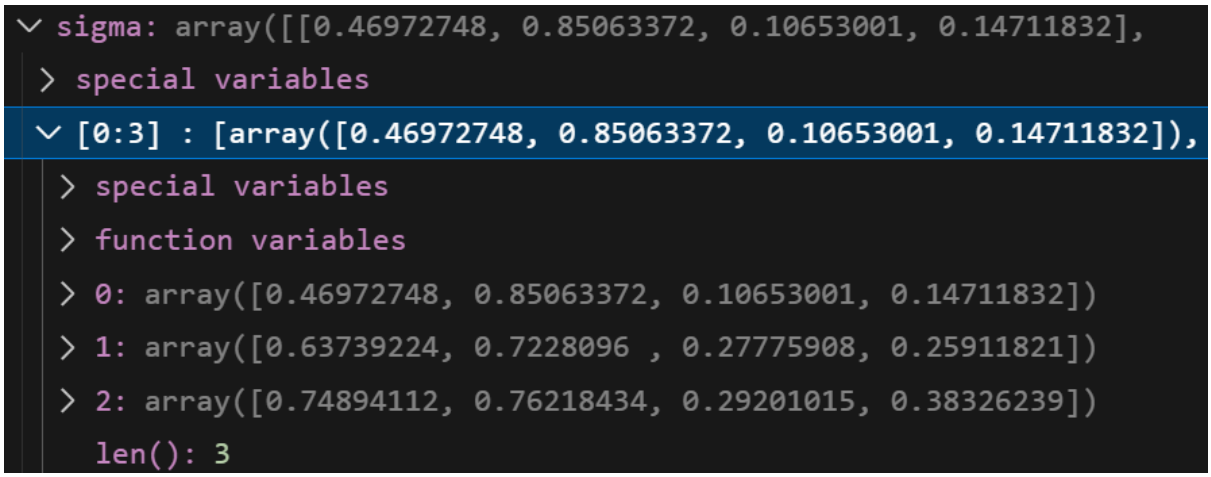
```
> 000: array([-0.17367395, -1.28296331,  0.70592084,  1.05393502])
> 001: array([1.03800476, 0.55861082, 1.10378283, 1.18556721])
> 002: array([1.03800476, 0.55861082, 1.10378283, 1.71209594])
> 003: array([-0.41600969,  2.63038172, -1.34022653, -1.3154443 ])
> 004: array([-1.02184904, -2.43394714, -0.14664056, -0.26238682])
> 005: array([ 0.91683689, -0.36217625,  0.47857113,  0.13250973])
> 006: array([0.18982966, 0.78880759, 0.42173371, 0.52740629])
> 007: array([ 1.03800476, -0.13197948,  0.70592084,  0.65903847])
> 008: array([ 0.4321654 , -0.36217625,  0.30805885,  0.13250973])
> 009: array([-0.17367395,  1.70959465, -1.16971425, -1.18381211])
> 010: array([-0.41600969, -1.51316008, -0.03296571, -0.26238682])
> 011: array([ 0.18982966, -0.82256978,  0.76275827,  0.52740629])
> 012: array([-0.41600969,  1.01900435, -1.39706395, -1.3154443 ])
> 013: array([ 0.31099753, -1.05276654,  1.0469454 ,  0.26414192])
> 014: array([0.67450115, 0.32841405, 0.42173371, 0.3957741 ])
```

위와 같이 기존 feature들이 normalization되어 바뀐 것을 확인할 수 있다.

```python
def get_normal_parameter(data, label, label_num): # 20 p
    # 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])

    for i in range(label_num):
        data_i = data[np.where(label == i)] # Extracting
        mu[i] = np.mean(data_i,axis=0) # Calculating fea
        sigma[i] = np.std(data_i,axis=0)# Calculating fe

    return mu, sigma
```

training data를 기반으로 평균과 표준편차를 뽑아내는 함수이다. 각 라벨당 그리고 각 피쳐당 평균 및 표준편차가 계산된다. 둘 다 shape는 3x4(class 3개 x feature 4개)이다.





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

    # you should get this parameter correctly
    prior = np.zeros([label_num])
    for i in range(label_num):
        prior[i] = np.size(np.where(label == i)) # Calculatin

    return prior/data_point # Convert to percentage
```

training data를 기반으로 클래스당 prior를 계산하는 함수이다. 클래스 별로 training data속 사진 수를 이용하여 계산한다.

```
prior: array([0.34, 0.32, 0.34])
```

위의 결과는 0,1,2 클래스가 각각 34장, 32장, 34장 존재한다는 의미이다.

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

    # Calculate pdf based on Gaussian function
    sigma_squared = sigma ** 2
    coefficient = 1 / np.sqrt(2 * np.pi * sigma_squared)
    exponent = -((x - mu) ** 2) / (2 * sigma_squared)
    pdf = coefficient * np.exp(exponent)

    return pdf
```

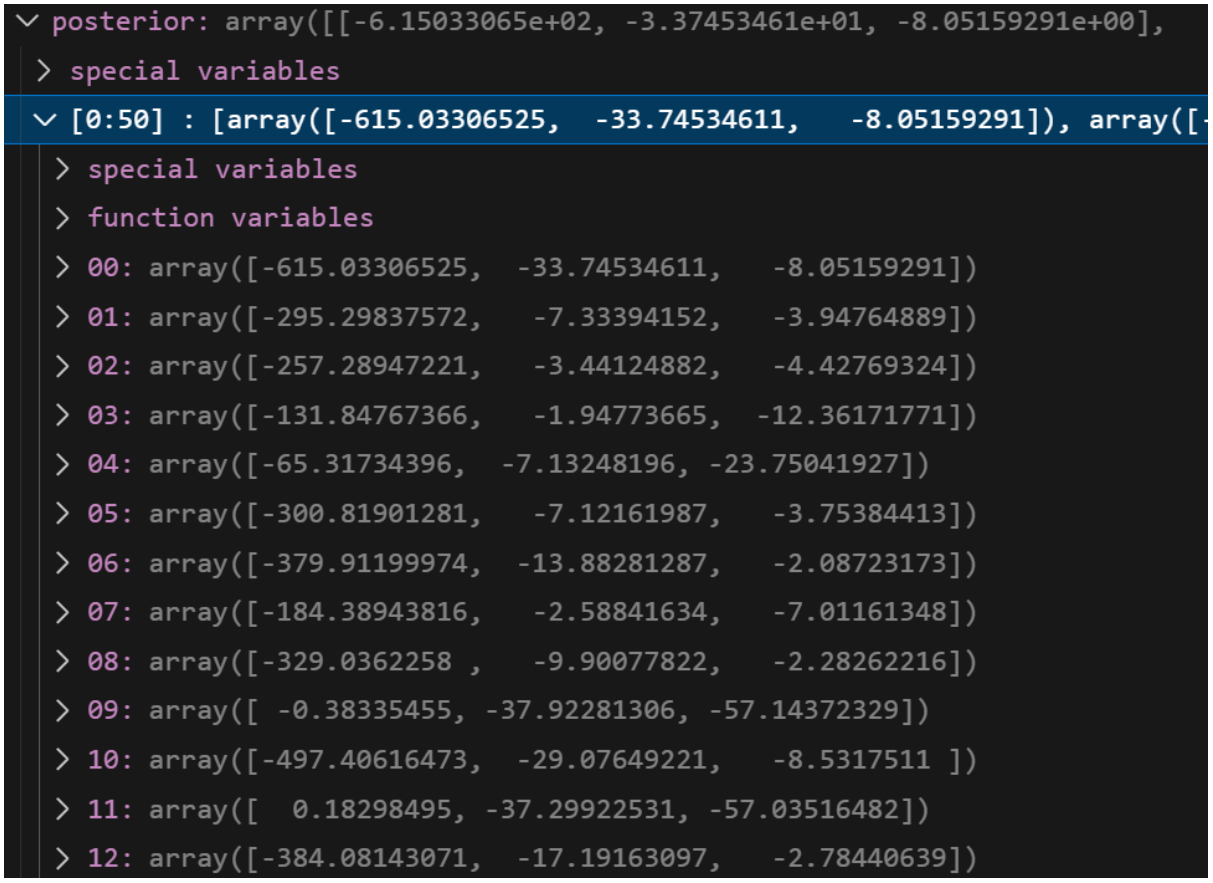
가우시안 함수를 그대로 구현한 코드이다. 가우시안 함수의 핵심 파라미터인 mu,sigma 그리고 데이터를 받아 해당 데이터의 likelihood를 구하는데 사용된다.

```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

    # Implementation of log of Gaussian function
    sigma_squared = sigma ** 2
    log_term = 1 / np.sqrt(2 * np.pi * sigma_squared)
    exponent_term = -((x - mu) ** 2) / (2 * sigma_squared)
    log_pdf = np.log(log_term) + exponent_term

    return log_pdf
```

가우시안 함수에 log를 취한 형태를 코드로 구현한 함수이다. Gaussian_PDF함수와 다른 점은 log가 취해져 자연상수의 차수부분이 pdf를 구하는데 그대로 더해지는 계산과정상의 차이가 있다.

```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

    for i in range(data_point):
        for j in range(label_num):
            likelihood[i][j] = np.sum(Gaussian_Log_PDF(data[i],mu[j],sigma[j])) # A
            posterior[i][j] = likelihood[i][j] + np.log(prior[j]) # Add the log of

    return posterior
```

training data를 이용해 구한 mu, sigma 그리고 데이터의 feature값들을 이용해서 test data당 클래스별 likelihood를 구하고 이를 prior와 곱해 posterior를 구한다. 위 사진은 log를 취하였기에 np.sum 함수를 이용해 더한 결과이다.



test data당 posterior가 저장된 결과이다.

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

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

    return prediction
```

위의 posterior값들을 이용해서 label을 예측하는 코드이다.

```
> [0:50] : [2, 2, 1, 1, 1,
```

```
> 00: array([-615.03306525,  -33.74534611,   -8.05159291])
> 01: array([-295.29837572,   -7.33394152,   -3.94764889])
> 02: array([-257.28947221,   -3.44124882,   -4.42769324])
> 03: array([-131.84767366,   -1.94773665,  -12.36171771])
> 04: array([-65.31734396,  -7.13248196, -23.75041927])
> 05: array([-300.81901281,   -7.12161987,   -3.75384413])
```

상위 5개만 뽑아서 확인해보면 가장 큰 posterior에 해당하는 label을 prediction에 저장해둔 것을 확인할 수 있다.

```python
def accuracy(pred, gnd):
    data_point = len(gnd)

    hit_num = np.sum(pred == gnd) # Calculating num

    return (hit_num / data_point) * 100, hit_num

    ## total 100 point you can get
```

마지막으로 accuracy 함수를 이용해서 예측값들을 실제 label과 비교하여 맞춘 횟수를 전체 데이터 개수로 나누어 정확도를 뽑아낸다.

result

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.02640884e-16 -8.10462808e-16 -2.45729363e-16  2.48689958e-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: [-4.33727128e-16 -7.93439388e-16 -2.69414121e-16  0.00000000e+00]
accuracy is 94.0% !
the number of correct prediction is 47 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.32246831e-16 -8.24525633e-16 -2.78295905e-16  2.78295905e-16]
accuracy is 92.0% !
the number of correct prediction is 46 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.54451291e-16 -8.17124146e-16 -2.69414121e-16  2.81256500e-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 -2.66453526e-16 -3.61192557e-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: [-4.44089210e-16 -8.05281767e-16 -2.60532336e-16 -5.32907052e-17]
accuracy is 94.0% !
the number of correct prediction is 47 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.26325641e-16 -8.05281767e-16 -2.66453526e-16 -2.96059473e-17]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.26325641e-16 -8.11202957e-16 -2.57571742e-16 -4.58892184e-17]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.14483263e-16 -7.99360578e-16 -2.72374715e-16 -2.36847579e-17]
accuracy is 92.0% !
the number of correct prediction is 46 of 50 !
```

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [ 6.21724894e-16 -8.05281767e-16  2.54611147e-16 -1.48029737e-18]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

실행	정확도
1	100%
2	94%
3	92%
4	94%
5	94%
6	94%
7	98%
8	98%
9	92%
10	98%

평균: 95.4%

결과는 최대 100%부터 최소 92%까지 나오는 것을 확인했다 이는 데이터가 shuffle되면서 training data가 계속 바뀌기 때문에 달라질 수 있다고 생각한다. 10회 실행 후 평균을 계산해보니 95.4%의 정확도를 얻었다.

reference

머신러닝 수업 pdf와 과제 제안서만을 참고하였습니다.