머신러닝

1차 과제

NAIVE BAYESIAN CLASSIFIER DESIGN

담당 교수님: 박철수

강의 시간: 월3, 수4

학 과: 컴퓨터정보공학부

학 번: 2020202055

성 명: 최소윤

1. 과제 목적

Bayesian classifier를 이용하여 꽃을 분류하는 것이 목표이다. 주어진 데이터는 꽃잎 너비, 꽃잎 길이, 꽃받침 너비, 꽃받침 길이로 4가지의 feature가 있으며 꽃의 종류는 3가지이다. 이를 구현하기 위해서는 6개의 함수를 채워 구현해야 한다. 먼저 feature들을 normalization을 하고 test data와 train data로 나누고 가우시안 함수와 각 label과 feature에 따른 평균과 표준편차를 이용하여 likelihood값을 구하고 prior와 posterior를 구하여 어떤 꽃인지 분류하는 과제이다. 이 과제를 통해 Bayesian theorem에 대해 이해하고 수식을 코드로 작성해본다.

2. 소스 코드

```
6   def feature_normalization(data):  # 10 points
7       # parameter
8       feature_num = data.shape[1]
9       data_point = data.shape[0]
10
11      # you should get this parameter correctly
12      normal_feature = np.zeros([data_point, feature_num])
13      mu = np.zeros([feature_num])
14      std = np.zeros([feature_num])
15
16      # your code here
17      # calculate the mean and standard deviation
18      mu = np.mean(data, 0)# axis = 0
19      std = np.std(data, 0)
20      # calculate normal_feature ((x - m)/sigma)
21      normal_feature = (data - mu) / std
22      # end
23
24      return normal_feature
```

Feature_normalization함수는 data에 대한 표준화를 하여 반환해주는 함수이다. Data에 대한 평균과 표준편차를 구하기 위해 np.mean()과 np.std()함수를 사용하였다. 두번째 인자는 axis는 0으로 설정하여 row를 기준으로 연산한다. Normal_feature은 data값에 평균을 빼고 그 값을 표준편차로 나눈다.

```python
def get_normal_parameter(data, label, label_num):  # 20 points
    # parameter
    feature_num = data.shape[1]# 4

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

    # your code here
    # Store index values of data by label in a tmp array
    tmp = np.zeros((label_num,), dtype=np.ndarray)
    for i in range(label_num):
        cond = np.where(label == i)# Find a location index that satisfies that label
        tmp[i] = np.zeros((cond[0].shape[0], feature_num)) # Create as many arrays as the number of data that have that label
```

Get_normal_parameter함수는 각 label과 feature에 따른 평균과 표준편차를 구하는 함수로 평균과 표준편차는 2차원 배열이다. tmp 배열에는 각 label별로 데이터를 저장하도록 한다. for문을 사용하여 label별로 데이터의 인덱스를 구해 cond변수에 저장하고 tmp에는 label별로 데이터의 개수만큼 2차원 배열을 생성한다. label별로 데이터의 개수가 다를 수 있기 때문에 이러한 작업을 수행한다. 평균과 표준편차를 구하는 함수를 사용할 때 배열의 행 개수가 영향을 미치기 때문이다.

```python
    for tag in range(label_num):
        idx = 0
        cond = np.where(label == tag)# Find the data that fits the label
        if len(cond[0]) > 0:# Number of data indexes by label
            for i in cond[0]:
                tmp[tag][idx] = data[i]# Store data in a tmp array for each label
                idx += 1
            # Calculate the mean and standard deviation of data for each level
            mu[tag] = np.mean(tmp[tag], 0)
            sigma[tag] = np.std(tmp[tag], 0)
        else:# If no such label data exists
            # Save nan Value
            mu[tag] = np.nan
            sigma[tag] = np.nan
    # end

    return mu, sigma
```

for문을 사용하여 각 label별 데이터의 인덱스를 가져와 만약에 데이터가 없는 것이 아니라면 tmp에 해당 인덱스의 feature data를 저장한다. 해당 label에 대한 모든 데이터가 저장이 되고나서 np.mean(), np.std()함수를 이용하여 평균과 표준편차를 저장한다. 만약 해당 label에 대한 데이터가 하나도 없다면 평균과 표준편차를 np.nan값을 저장한다.

```python
def get_prior_probability(label, label_num):  # 10 points
    # parameter
    data_point = label.shape[0] # 100

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

    label = list(label)
    # your code here
    # Obtain the priority value
    # the number of data belonging to the label / the number of total data
    for i in range(label_num):
        prior[i] = list(label).count(i)/data_point
    # end
    return prior
```

Get_prior_probability 함수는 prior을 구하는 함수로 주어진 train data가 100개이므로 label의 개수도 100개이다. 각 label에 따라 데이터의 개수를 count함수를 이용해서 구하고 이를 전체 train data의 수로 나눈다.

```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
    # Gaussian distribution expression
    pdf = (1/np.sqrt(2*np.pi * sigma**2)) * np.exp(-(x-mu)**2 / (2*sigma**2))
    # end

    return pdf
```

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Gaussian_PDF는 위 가우시안 분포식을 코드화하여 수식을 그대로 표현한 것이다.

```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
    # Log Gaussian Distribution Expression using Gaussian_PDF
    log_pdf = np.log(Gaussian_PDF(x, mu, sigma))
    # end

    return log_pdf
```

Gaussian_Log_PDF는 가우시안 식에 자연로그를 씌워 구현하였다.

```python
def Gaussian_NB(mu, sigma, prior, data):  # 40 points
    # parameter
    data_point = data.shape[0]  # 50 (test_data)
    label_num = mu.shape[0]  # 3

    # you should get this parameter correctly
    likelihood = np.ones([data_point, label_num])  # matrix
    posterior = np.zeros([data_point, label_num])  # matrix
    ## 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
```

Gaussian_NB는 likelihood값을 구하여 posterior값을 구하는 함수이다. 인자로 들어온 데이터는 50개의 test data이고 likelihood와 posterior 배열을 선언한다.

```python
    for i in range(data_point):  # test data row
        for j in range(label_num):  # class (label)
            for k in range(data.shape[1]):  # feature num
                # calculate the likelihood of data values according to each label
                likelihood[i][j] *= Gaussian_PDF(data[i][k], mu[j][k], sigma[j][k])
                if np.isnan(likelihood[i][j]):  # If the data is nan, store the largest negative value
                    posterior[i][j] = -np.inf
                else:  # posterior = log(likelihood * prior)
                    posterior[i][j] = np.log1p(likelihood[i][j]*prior[j])
    # end
    return posterior
```

중첩 for문을 사용하여 likelihood값을 계산한다. likelihood값은 가우시안 함수를 사용하여 구하며 변수 likelihood의 행은 data, 열은 class이며, 변수 data의 행은 data, 열은 feature이며, 변수 mu, sigma의 행은 class, 열은 feature다. 각 feature는 독립이라고 생각하여 likelihood는 각 feature의 곱으로 구한다. 만약 likelihood값이 nan이라면 posterior에 큰 음수 값을 저장한다.

nan값을 -inf로 저장한 이유는 main_app.py에서 data shuffle을 하지 않는다면 train data에는 첫번째 label과 두번째 label의 데이터가 포함되고 세번째 label의 데이터는 포함되지 않는다. Test data에는 세번째 label의 데이터 값만 들어가서 test data를 돌렸을 때 정확도가 0%가 나와야 맞다. Train data에는 세번째 label의 데이터가 없기 때문이다. 하지만 나중에 classification을 진행했을 때 nan값이 argmax함수의 반환 값으로 나오는데 그 이유는 다른 원소 값이 음수여서 nan값이 크다고 반환되기 때문이다. 그러므로 nan값이 likelihood라면 posterior 값은 -inf로 저장하여 이 값을 선택하지 않도록 해야 한다.

만약 likelihood값이 nan이 아니라면 likelihood와 prior를 곱한 후 log를 씌운 값을 더한다. 여기서 log1p()함수를 사용한 이유는 아래와 같은 오류를 방지하기 위함이다.

```
C:\Users\user\PycharmProjects\ML\SYC.py:129: RuntimeWarning: divide by zero encountered in log
  posterior[i][j] = np.log(likelihood[i][j]*prior[j])
```

이 오류는 log함수에 0이 있으면 위와 같은 오류가 생긴다. 그러므로 log1p 함수를 사용하여 1을 더한 후 계산을 해 경고문이 나오지 않도록 방지한다.

3. 실행 결과

```
Mean: [5.84333333 3.05733333 3.758      1.19933333]
normalled_Mean: [-4.50010399e-16 -8.05281767e-16 -2.60532336e-16  2.60532336e-16]
accuracy is 98.0% !
the number of correct prediction is 49 of 50 !
```

코드 실행 결과는 위와 같이 나왔으며 mean은 각 feature에 대한 평균값이며 normalled mean은 각 feature별로 data에 평균을 빼고 표준편차로 나눈 값을 평균 낸 값이다. Test data를 사용해서 실행했을 때 정확도는 98%로 나왔고 50개의 data중 49개를 맞추었다. Main.py에서 data를 섞는 부분이 있어서 실행을 할 때마다 정확도가 다르게 나오지만 대부분 90%이상의 정확도를 가지는 것을 확인할 수 있었다.

4. 고찰

Get_normal_parameter 함수를 구현할 때 np.mean(), np.std() 함수를 사용하려면 해당 데이터의 개수와 배열의 행의 개수가 같아야 정확한 결과값이 나온다는 것을 알게 되었다. 그래서 np.where함수를 사용해서 3차원 배열에 각 label별로 데이터를 모아 저장하고 함수를 이용해서 평균과 표준편차를 구하면 된다. Gaussian_NB 함수에서 likelihood값, posterior값을 구하는 방법과 과정에 대해 자세히 알게 되었다. 또한 이번 과제를 통해 베이시안 정리와 chain rule이 어떻게 적용되는지 확인할 수 있었다.

5. 참고문헌

정규분포의 공식 /
https://angeloyeo.github.io/2020/09/14/normal_distribution_derivation.html
최대우도법(MLE) / https://angeloyeo.github.io/2020/07/17/MLE.html