기초 인공지능(CSE4185) Assignment #4

2분반 20200183 박정원

1. 각 알고리즘마다 구현한 방법에 대한 설명

(1) Joint distribution 구하기

```python
def joint_distribution_of_word_counts(texts, word0, word1):
    """
    Parameters:
    texts (list of lists) - a list of texts; each text is a list of words
    word0 (str) - the first word to count
    word1 (str) - the second word to count

    Output:
    Pjoint (numpy array) - Pjoint[m,n] = P(X0=m,X1=n), where
      X0 is the number of times that word0 occurs in a given text,
      X1 is the number of times that word1 occurs in the same text.
    """
    word0_max = 0 # 확률 변수 X0이 가질 수 있는 최대 값
    word1_max = 0 # 확률 변수 X1이 가질 수 있는 최대 값

    cnt_tuple = [] # 각 텍스트에 대해서 (X0=x0, X1=x1) 튜플 형태로 저장

    for i in range(len(texts)): # iterate each text which is a list of words
      cnt0 = texts[i].count(word0)
      cnt1 = texts[i].count(word1)
      word0_max = max(word0_max, cnt0) # 확률 변수 X0이 가질 수 있는 최대 값, 즉 the maximum number of times that word0 occurs in a given text
      word1_max = max(word1_max, cnt1) # 확률 변수 X1이 가질 수 있는 최대 값, 즉 the maximum number of times that word1 occurs in a given text
      cnt_tuple.append((cnt0,cnt1)) # 각 텍스트에 대해서 (X0=x0, X1=x1) 튜플 형태로 저장


    denom = len(texts) # 분모로 들어갈 모든 경우의 수는 텍스트의 개수와 같다.
    Pjoint = np.zeros((word0_max+1, word1_max+1)) # (word0_max+1, word1_max+1) shape np array

    for i in range(word0_max+1):
      for j in range(word1_max+1):
        Pjoint[i,j] = cnt_tuple.count((i,j)) / denom # calculate the joint distribution

    # raise RuntimeError("You need to write this part!")
    return Pjoint
```

먼저, joint distribution을 구하는 수식을 살펴보자.

수식)

Joint distribution: $$P(X_0 = x_0, X_1 = x_1) = \frac{N(X_0 = x_0, X_1 = x_1)}{\sum_{x_0}\sum_{x_1} N(X_0 = x_0, X_1 = x_1)}$$

이 수식을 해석하자면, X0 = x0, X1 = x1 일 때의 joint distribution은 모든 텍스트에 대해서 X0 = x0, X1 = x1 을 가진 텍스트의 개수 / 모든 경우의 수 (X0 = 0 ~ 최댓값, X1 = 0 ~ 최댓값) 이다.

따라서, 확률변수 X0이 가질 수 있는 최댓값과 X1이 가질 수 있는 최댓값을 구해야 한다.

우선, texts 리스트를 돌면서 각각의 text에 대해 word0 과 word1의 개수를 세어주고, 각각 word0_max 와 word1_max에 max 연산을 취해주면서 최댓값을 구한다.

또한, 이 때 cnt_tuple 이라는 리스트에 해당 텍스트의 word0 개수와 word1 개수를 튜플 형태로 넣어준다. 나중에 해당 튜플 쌍이 전체 텍스트에서 몇 개인지 카운트할 때 쓰기 위해서이다.

위 수식에서 분모로 오는 값은 전체 텍스트의 개수이므로 len(texts) 값을 취해주면 된다.
넘파이 함수 zeros를 통해서 (word0_max+1, word1_max+1) 형태의 2차원 넘파이 배열을 만들어준다. 그 이유는 확률 변수는 0부터 max까지의 값을 가지기 때문에 +1 한 형태로 만들어주어야 한다.

그 후에는 0 ~ word0_max+1, 0 ~ word1_max+1 2차원 루프를 돌면서, 각각의 i, j 인덱스에 대해서 joint distribution 을 수식에 대입해서 구해주면 된다.

(2) Marginal distribution 구하기

```
def marginal_distribution_of_word_counts(Pjoint, index):
    """
    Parameters:
    Pjoint (numpy array) - Pjoint[m,n] = P(X0=m,X1=n), where
      X0 is the number of times that word0 occurs in a given text,
      X1 is the number of times that word1 occurs in the same text.
    index (0 or 1) - which variable to retain (marginalize the other)

    Output:
    Pmarginal (numpy array) - Pmarginal[x] = P(X=x), where
      if index==0, then X is X0
      if index==1, then X is X1
    """
    rows, cols = Pjoint.shape
    if index == 0:
      Pmarginal = np.sum(Pjoint, axis=1).tolist() # calculate the marginal distribution of X0, which means retaining X0 and marginalize X1
    else:
      Pmarginal = np.sum(Pjoint, axis=0).tolist() # calculate the marginal distribution of X1, which means retaining X0 and marginalize X0
    # raise RuntimeError("You need to write this part!")
    return Pmarginal
```

그 다음은 marginal distribution을 구해야 한다. 먼저 수식을 살펴보자.

수식)

Marginal distribution: $P(X_0 = x_0) = \sum_{x_1} P(X_0 = x_0, X_1 = x_1)$,

$$P(X_1 = x_1) = \sum_{x_0} P(X_0 = x_0, X_1 = x_1)$$

함수의 인자로 index가 들어오는데, index가 0 이라면 X0 확률 변수를 retain 하고, X0 = x0 에 대해서 각각 marginal distribution을 구해주어야 한다. 반환형은 1차원 배열이 될 것이다.

반대로 index가 1 이라면 X1 확률 변수를 retain 하고, X1 = x1 에 대해서 각각 marginal distribution을 구해 주어야 한다. 반환형은 똑같이 1차원 배열이 될 것이다.

넘파이의 sum 함수를 사용하면 쉽게 marginal distribution을 구할 수 있다. axis = 1 로 하면, 각 행 별로 합(X0)을 넘파이 배열 형태로 반환하므로, tolist()를 통하여 리스트 형태로 바꾸어준다. axis = 0 로 하면, 각 열 별로 합(X1)을 넘파이 배열 형태로 반환하므로, tolist()를 통하여 리스트 형태로 바꾸어준다.

(3) Conditional distribution 구하기

```python
def conditional_distribution_of_word_counts(Pjoint, Pmarginal):
    """
    Parameters:
    Pjoint (numpy array) - Pjoint[m,n] = P(X0=m,X1=n), where
      X0 is the number of times that word0 occurs in a given text,
      X1 is the number of times that word1 occurs in the same text.
    Pmarginal (numpy array) - Pmarginal[m] = P(X0=m)

    Outputs:
    Pcond (numpy array) - Pcond[m,n] = P(X1=n|X0=m)
    """
    rows, cols = Pjoint.shape
    Pcond = np.zeros_like(Pjoint) # make the same shape 2d array like Pjoint

    for i in range(rows):
      for j in range(cols):
        if Pmarginal[i] == 0:
          Pcond[i, j] = np.nan # if divide by zero, make it nan
        else:
          Pcond[i,j] = Pjoint[i, j] / Pmarginal[i] # calculate the conditional distribution using the formula

    # raise RuntimeError("You need to write this part!")
    return Pcond
```

다음은 conditional distribution 이다. 먼저 수식을 살펴보자.

수식)

Conditional distribution: $P(X_1 = x_1 | X_0 = x_0) = \frac{P(X_0 = x_0, X_1 = x_1)}{P(X_0 = x_0)}$

Pcond의 모양은 Pjoint와 동일한 모양이므로, 넘파이의 zeros_like를 활용해 만들어주었다. Conditional distribution은 앞서 구해둔 Pjoint와 Pmarginal을 인자로 받는데, 이를 그대로 활용해 수식에 대입해서 값을 구하면 끝이다. 주의할 점은 Pmarginal 이 0 인 경우에는 nan 값을 넣어주도록 예외처리를 해주면 경고없이 정상적으로 출력이 되는 모습을 볼 수 있다.

(4) Mean, Variance, Covariance 구하기

```python
def mean_from_distribution(P):
    """
    Parameters:
    P (numpy array) - P[n] = P(X=n)

    Outputs:
    mu (float) - the mean of X
    """
    mu = 0.0
    for i in range(len(P)):
      mu += (i * P[i]) # calculate the mean from distribution using the formula

    mu = round(mu, 3) # round to the third
    # raise RuntimeError("You need to write this part!")
    return mu
```

다음은 mean from distribution을 구하는 함수이다.

$$\text{mean: } \mu = \sum_x x \cdot P(X = x)$$

reader.py 의 Pathe의 X1(the)에 대한 marginal distribution인 Pthe 1차원 배열 P가 인자로 들어오게 되고, 이를 수식에 그대로 적용하여 구하면 distribution의 평균을 구할 수 있다.

```python
def variance_from_distribution(P):
    """
    Parameters:
    P (numpy array) - P[n] = P(X=n)

    Outputs:
    var (float) - the variance of X
    """
    mu = mean_from_distribution(P) # calculate the mean first
    var = 0.0
    for i in range(len(P)):
      var += (((i - mu) ** 2) * P[i]) # using the mean, calculate the variance using the formula

    var = round(var, 3) # round to the third
    # raise RuntimeError("You need to write this part!")
    return var
```

$$\text{variance: } \sigma^2 = \sum_x (x - \mu)^2 \cdot P(X = x)$$

다음은 분산이다.
분산에서는 mean 값을 활용하므로 앞서 구현해둔 mean_from_distribution 함수를 써서 Pthe 의 평균을 구한 뒤에 이를 그대로 사용해 분산을 구하는 수식에 대입해서 구하면 분산도 쉽게 구할 수 있다.

```python
def covariance_from_distribution(P):
    """
    Parameters:
    P (numpy array) - P[m,n] = P(X0=m,X1=n)

    Outputs:
    covar (float) - the covariance of X0 and X1
    """
    rows, cols = P.shape
    X0 = marginal_distribution_of_word_counts(P, 0) # get the marginal distribution of X0 of P
    X1 = marginal_distribution_of_word_counts(P, 1) # get the marginal distribution of X1 of P
    mu_X0 = mean_from_distribution(X0) # get the mean of X0
    mu_X1 = mean_from_distribution(X1) # get the mean of X1

    covar = 0.0
    for i in range(rows):
      for j in range(cols):
        covar += ((i - mu_X0) * (j - mu_X1) * P[i, j]) # calculate the covariance using the formula

    covar = round(covar, 3) # round to the third

    # raise RuntimeError("You need to write this part!")
    return covar
```

Covariance: $Cov(X_0, X_1) = \sum_{x_0, x_1} (x_0 - \mu_{X_0})(x_1 - \mu_{X_1}) \cdot P(X_0 = x_0, X_1 = x_1)$

공분산을 구할 때도 평균값을 쓰기 때문에 X0에 대한 marginal distribution과 X1에 대한 marginal distribution을 각각 구하고, 이에 대한 평균값을 각각 구한다.

그 후에 이 값들을 활용하여 수식에 그대로 대입해서 구하면 공분산이 나오게 된다.

평균, 분산, 공분산 모두 소수점 아래 셋째 자리까지 반올림하여 준다. (round 함수 활용)

(5) Expected value of a Function 구하기

```python
def expectation_of_a_function(P, f):
    """
    Parameters:
    P (numpy array) - joint distribution, P[m,n] = P(X0=m,X1=n)
    f (function) - f should be a function that takes two
        real-valued inputs, x0 and x1.  The output, z=f(x0,x1),
        must be a real number for all values of (x0,x1)
        such that P(X0=x0,X1=x1) is nonzero.

    Output:
    expected (float) - the expected value, E[f(X0,X1)]
    """
    rows, cols = P.shape
    expected = 0.0
    for i in range(rows):
      for j in range(cols):
        expected += (f(i, j) * P[i, j]) # calculate the expected value of a function using the formula

    expected = round(expected, 3) # round to the third
    # raise RuntimeError("You need to write this part!")
    return expected
```

수식)

If $f(x_0, x_1)$ is some real-valued function of variables x_0 and x_1 then its expected value is: $E[f(X_0, X_1)] = \sum_{x_0, x_1} f(x_0, x_1) P(X_0 = x_0, X_1 = x_1)$

reader.py 에는 변수 x0과 x1에 대한 real-valued function인 f가 정의되어 있고, 이 함수가 joint distribution과 함께 인자로 넘어오게 된다.

이렇게 받아온 함수와 joint distribution인 P를 그대로 위 수식에 대입하여 더해주면, expected value of a function을 구할 수 있다. (x0과 x1이 가질 수 있는 모든 값에 대해서 계산하여 더해준다.)

이 역시 실수값이므로, 소수점 아래 셋째 자리까지 반올림하여 반환한다.

2. 실행 결과 캡처 화면

```
~/Desktop/AI4 python reader.py
100%|
문제1. Joint distribution:
[[0.964 0.024 0.002 0.    0.002]
 [0.006 0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.   ]
 [0.002 0.    0.    0.    0.   ]]
------------------------------------------
문제2. Marginal distribution:
P0: [0.992, 0.006, 0.0, 0.0, 0.002]
P1: [0.972, 0.024, 0.002, 0.0, 0.002]
------------------------------------------
문제3. Conditional distribution:
[[0.97177419 0.02419355 0.00201613 0.         0.00201613]
 [1.         0.         0.         0.         0.        ]
 [       nan        nan        nan        nan        nan]
 [       nan        nan        nan        nan        nan]
 [1.         0.         0.         0.         0.        ]]
------------------------------------------
문제4-1. Mean from distribution:
4.432
문제4-2. Variance from distribution:
41.601
문제4-3. Covariance from distribution:
9.245
------------------------------------------
문제5. Expectation of a function:
1.772
```