**AI\_Assignment 04 Report**

**1분반 20191244 김현승**

1. **Joint distribution**

joint\_distribution\_of\_word\_counts() 함수는 여러 텍스트로 이루어진 리스트 texts를 입력으로 받아 word0와 word1을 카운트해 joint distribution을 구하는 함수이다. Pjoint는 Numpy 이차원 배열로, Pjoint[m, n]은 word0이 m번, word1이 n번 나올 확률을 나타낸다. Pjoint 배열을 초기화하기 위해 먼저 배열의 크기를 구해야한다. 따라서 counts\_word0, 1 리스트와, max\_count\_word0, 1 변수에 각 text에 포함되어 있는 word0, 1의 개수와 그 중 가장 큰 개수를 저장한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

counts\_word0, 1에는 첫 번째 텍스트부터 마지막 텍스트까지 각 텍스트에 포함되어 있는 word0, word1의 개수가 각각 차례대로 저장된다. 반복문을 돌며 최대 카운트 개수를 업데이트 해주고, 반복문이 끝나면 max\_count\_word0+1 개의 행, max\_count\_word1+1 개의 열을 갖고, 각 원소의 값은 0이 되도록 Pjoint 배열을 초기화해준다. 여기서 1을 더해주는 이유는 word0 또는 word1이 하나도 나오지 않는 경우를 처리해주기 위해 0의 자리를 포함시켜야하기 때문이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이후 (counts\_word0, counts\_word1)의 원소에 대해 반복문을 돌며 Pjoint 값을 하나씩 더해준다. 이로써 N[X0=x0, X1=x1]을 모두 구했다. 구해야 하는 것은 확률이므로 전체 경우의 수로 배열을 나눠주어 각 원소가 확률이 되도록 해준다.

1. **Marginal distribution**

marginal\_distribution\_of\_word\_counts() 함수는 joint distribution 정보를 입력받고 index에 따라 marginal distribution을 구하는 함수이다. index가 0인 경우 X0의 marginal distribution을, 1인 경우 X1의 marginal distribution을 구해야한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

만약 X0의 marginal distribution을 구해야 한다면, 각 행의 모든 원소를 더한 값을 원소로 하는 일차원 배열을 구하면 될 것이다. X1도 비슷하게 행이 아닌 열을 기준으로 배열을 구하면 된다. numpy의 sum() 함수에서 ‘axis’ 인자를 사용하면 각 행이나 열의 합을 쉽게 계산할 수 있다.

1. **Conditional distribution**

conditional\_distribution\_of\_word\_counts() 함수는 joint distribution과 X0의 marginal distribution을 받아 conditional distribution을 구하는 함수이다. conditional distribution을 저장할 Pcond 이차원 배열은 입력 받은 joint distribution과 같은 크기를 가져야한다. 따라서 numpy의 zeros\_like() 함수를 이용해 Pcond를 Pjoint와 같은 형태로 초기화한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

각 원소에 대해 나눗셈을 하기 위해 m은 0부터 max\_count\_word0까지, n은 0부터 max\_count\_word1까지 반복하도록 2중 for문을 구축하고, 분모에 해당하는 Pmarginal[m], 즉 P[X0=m] 이 0이 아닐 때, Pjoint[m, n]을 Pmarginal[n]으로 나누어 Pcond[m, n]에 저장한다. 만약 분모가 0인 경우 계산할 수 없으므로 NaN 값을 저장한다.

1. **Mean, Variance, Covariance**

* Mean

mean\_from\_distribution() 함수는 marginal distribution P를 입력으로 받아 해당 random variable의 평균을 구하는 함수이다.

폰트, 스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

numpy의 arange() 함수를 사용해 0부터 P의 원소 개수만큼 연속된 정수가 저장된 배열 n을 초기화한다. 이 n은 확률 변수에 해당한다. 이후 n \* P를 계산하면, 각 원소 별(element-wise)로 곱해진 값을 원소로 갖는 배열이 결과가 되고, 이를 sum() 함수로 모든 원소를 더해주어 평균을 구할 수 있다.

* Variance

variance\_from\_distribution() 함수는 marginal distribution P를 입력으로 받아 해당 random variable의 분산을 구하는 함수이다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

mean\_from\_distribution() 함수에서 평균을 구했던 방식과 똑같이 확률 변수 n에 대한 평균을 구해준 후 n의 제곱에 대한 평균을 구하고, n의 평균의 제곱을 빼주어 분산을 구할 수 있다.

* Covariance

covariace\_from\_distribution() 함수는 joint distribution P를 입력으로 받아 P의 두 확률 변수의 공분산 cov(X, Y)를 구하는 함수이다.

폰트, 스크린샷, 텍스트, 번호이(가) 표시된 사진

자동 생성된 설명

numpy의 indices() 함수를 사용해 P와 동일한 형태를 갖는 두 이차원 배열 m, n을 구한다. 이 떄, m의 원소 값은 각 원소 별로 행의 인덱스 값을 갖고 n의 원소 값은 각 원소 별로 열의 인덱스 값을 갖는다. 이를 통해 확률 변수 m, n을 이차원 배열과 곱할 수 있도록 한다. numpy에서 \* 연산자를 사용해 형태(행, 열의 크기)가 같은 두 이차원 배열을 곱하는 경우, 행렬곱이 아닌 원소 별 곱셈이 이루어진다. 따라서 np.sum(m \* P)의 결과는 결국 m의 평균과 같아지게 된다. 이를 통해 확률 변수 m과 n의 평균을 구해 공분산 계산에 사용할 수 있다.



위의 m과 n의 평균을 구하는 것과 같은 방식으로 배열의 원소 별 곱셈과 모든 원소들의 합으로 (m-mu\_m)\*(n-mu\_n)의 평균, 즉 m과 n의 공분산을 구할 수 있다.

1. **Expected value of a function**

expectation\_of\_a\_function() 함수는 joint distribution과 two variable function을 입력으로 받아 확률 변수 m, n에 대해 f(m, n)의 평균을 구하는 함수이다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

indices() 함수를 이용해 확률 변수 m, n을 이차원 인덱스 배열로 표현한 후, f(m, n)의 값이 저장된 이차원 배열을 P와 원소 별 곱한 후 모든 원소를 더함으로써 f(m, n)에 대한 평균을 구할 수 있다.

1. **실행 결과**

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명