인공지능 2차 과제 보고서

2013011640 정대한

1. **코드 설명** 
   1. **함수설명**

**Def manipulate()** -> 데이터를 가공시키는 함수입니다. ratings\_train.txt 혹은 ratings\_valid.txt 를 파일 입출력을 통해 입력 받은 후, Okt 형태소 분석 함수(Okt().pos)를 사용하여 각 리뷰들의 단어를 분류합니다. 그 후, 이 단어가 명사, 형용사, 동사, 숫자 중 하나로 분류된 단어일 경우, 이 단어 자체를 토큰으로 봅니다. 그리고 그 라인에 있는 리뷰의 라벨이 0 일 경우, 토큰들을 neg 배열에 입력해 둡니다. 또한, 라벨이 1일 경우 pos 배열에 입력해 둡니다. 이런 식으로 라벨에 따라서 주요 토큰 단어들이 긍정 혹은 부정으로 분류되어 두 개의 배열에 각각 저장됩니다.

**Def naïve\_bayes\_classifier(pos,pos\_set,neg,neg\_set,all\_cnt,pos\_cnt,neg\_cnt)** -> 나이브 베이즈 분류기 입니다. Manipulate 함수에서 train 시켜서 만든 pos 와 neg를 또 가공하여 토큰과 그것의 확률자체를 튜플로 묶어서 배열화 시킨 pos\_set, 그리고 neg\_set을 토대로 ratings\_test.txt에 있는 리뷰들을 한 줄 한 줄 naïve bayes classification으로 분류하여 긍정 리뷰인지 부정 리뷰인지를 판단한 후, ratings\_result.txt에 문장과 함께 라벨을 저장합니다. Pos 변수는 긍정 토큰들을 저장한 배열 변수입니다. Neg 변수는 부정 토큰들을 저장한 배열 변수입니다. All\_cnt 변수는 pos와 neg 안 토큰들의 총 개수(중복일 경우 1개로 계산)입니다. pos\_cnt 변수는 pos 안 토큰들의 총 개수이고, neg\_cnt 변수는 neg 안 토큰들의 총 개수입니다.

* 1. **실행 과정**

먼저 manipulate 함수를 통하여 ratings\_test.txt 데이터를 가공하여 pos와 neg를 만듭니다. 그 후 pos.txt 안의 모든 토큰 개수와 neg.txt안의 모든 토큰 개수를 각각 pos\_cnt, neg\_cnt로 저장하고, 두 텍스트 파일 안의 모든 토큰들의 개수를 all\_cnt 로 저장합니다(토큰이 중복될 경우 1개로 계산). 그 후 pos\_set과 neg\_set 튜플 배열 두 개를 만들어서 이 단어가 나올 확률인 조건부확률을 구한다음 이를 그 토큰과 함께 묶어 튜플로 저장합니다. 그러나 이 조건부확률이 0이 되면 이 문장의 다른 단어들의 확률이 고려되지 못하므로 smoothing을 진행합니다. 제가 진행한 smoothing은 분자에 무조건 1을 더하고 분모에 all\_cnt을 더한 것입니다. 또한 여기에서 각 확률마다 ln을 취하여 확률이 너무 작게 나오는 것을 방지할 것입니다. 그 후 나이브 베이즈 분류기로 들어갑니다. test리뷰를 한 줄씩 받아 Okt 형태소 분석합니다. 그 후 분석된 단어가 명사, 형용사, 동사, 숫자 중 하나로 분류됐을 경우, 긍정일 때 토큰에 해당하는 튜플로 가서 그 확률을 찾아 prob\_pos 에 더해줍니다. 결과적으로, ln(p(단어1/긍정)\*p(단어2/긍정)….p(단어n/긍정)\*(1/2)) 가 prob\_pos 에 저장됩니다.(여기서 맨 뒤 1/2는 긍정과 부정 중 하나일 확률로써, 둘 중 하나가 될 확률은 50%입니다.) 그 후, math.exp(prob\_pos) 를 통해서 ln 로그를 지워준 다음 나오는 값이 바로 이 문장이 긍정문일 확률(prob\_pos)입니다. 부정문일 확률(prob\_neg)을 구하는 과정도 동일하므로 생략하겠습니다. 그 후 이 두 확률을 비교하여, 긍정일 가능성이 높다면 라벨을 1로, 부정일 가능성이 높다면 라벨을 0으로 하여 ratings\_result.txt에 id, document와 함께 저장합니다. 이 과정을 계속 거쳐 나가면서, test 파일에 있는 리뷰들에 대한 긍정,부정 분류를 마무리해 나가는 것이 코드 실행 과정입니다.

1. **실험 결과**

처음에는 Kkma 형태소 분석기를 사용했습니다. 다른 형태소 분석기보다 좀더 세밀하게 한글 형태소를 분석하기 때문에, 긍정 리뷰인지 부정 리뷰인지를 판단하는 데에 있어서 높은 정확도를 보일 것으로 예측했기 때문입니다. 하지만 train 텍스트 파일을 분석하면서 heap 메모리에 대한 오류가 났습니다. Valid 텍스트 파일을 분석하는 데에는 문제가 없었지만, train 텍스트 파일을 분석하는 데에는 문제가 있는 것으로 보았을 때, 분석해야 하는 양이 너무 많아지면 메모리가 Kkma 형태소 분석을 감당하기 힘들게 되는 것으로 판단하였습니다. 따라서 조금의 정확도를 포기하더라도 보다 간단하게 형태소를 분석하는 Okt 형태소 분석기를 사용했습니다. 이는 형태소들을 Kkma 보다 좀 더 간단하게 분석합니다. 예를 들어 태그가 조사인 경우, Okt 분석기는 그냥 조사 라고만 분석하는 반면, Kkma는 조사를 보격 조사,관형격 조사, 호격 조사 등으로 분류하여 더욱 세밀하게 분석합니다. 그러나, 실행오류로 인해 어쩔 수 없이 Okt 형태소를 이용해 과제를 수행하였고, 그 결과 프로그램이 마무리 되는 데에는 별다른 문제가 발생하지 않았습니다. 하지만 총 실행시간이 1시간 정도 걸리기 때문에, 이 시간을 어떻게 하면 단축시킬 수 있을지를 고민해보았습니다. 시간이 많이 걸리는 구간은 총 두 군데라고 생각했습니다. 첫 번째는 train 텍스트 데이터를 형태소 분석하는 시간입니다. 이 시간은 앞서 설명드렸듯 형태소 분석기 마다 다양한 속도 차이를 보이고, 그렇다고 이 토큰화 진행을 안 할 수는 없기 때문에 형태소 분석기 중 가장 실행시간이 덜 걸리는 Okt 로 선택했던 것입니다. 두 번째가 중요한데, 토큰화 시켜서 pos 배열 혹은 neg 배열에 넣어주고, 튜플을 만드는 과정( ‘토큰’, ‘확률’ 쌍)에서 확률을 계산하는 시간이 많이 걸리는 것으로 생각됩니다. 중복된 토큰 수가 몇 개인 지를 체크하는 과정에서의 시간, 즉 count 함수를 쓸 때, 배열을 전부 들여다 봐야 하므로 이 시간이 오래 걸리는 것 같습니다. 처음에는 이 과정을 튜플을 만들지 않고, 바로 나이브 베이즈 분류 함수로 들어가서 test 텍스트 파일을 한 줄 씩 받을 때마다 형태소 분석을 진행하고, pos 배열과 neg 배열을 뒤져 같은 토큰들을 모두 count 하여 확률을 계산하였습니다. 하지만 이렇게 하면, 나중에 같은 단어가 나오더라도 전에 했던 똑같은 계산을 계속 해주어야 한다는 부담이 있고, 이렇게 짠 코드는 실행시간이 3시간이 넘어갔습니다. 따라서 두 번째 방법으로 튜플 배열을 미리 만들어서 토큰마다의 확률을 계산해 놓고, 나이브 베이즈 분류 함수에서 토큰이 나올 때마다(물론 튜플 배열에 없는 토큰이라면 분자에는 1만 들어갑니다.) 그 토큰의 확률을 꺼내서 쓰는 방식으로 진행하였습니다. 실행시간은 많이 단축되어 1시간 정도가 걸리지만, 이 역시 부담되는 시간입니다. 세 번째 방법으로 manipulate 함수에서 형태소를 분류하여 긍정과 부정을 분류할 때 긍정 배열 혹은 부정 배열에 이미 있는 토큰이 나올 경우, 카운트를 1씩 더하여 쌍(튜플)으로 저장해 놓는 방법을 생각해보았습니다. 그러나 이 방법도 배열에 이미 토큰이 있다는 것을 확인하기 위해 배열 안을 찾아보는 시간은 존재합니다. 마지막 방법으로 key 값과 value 값이 매칭되는 해시 개념의 딕셔너리를 사용하여 문제를 해결해보려 하였습니다.