[과제 2] 한글 문장의 유사도 계산 (2)

<과제 목적> ngram 모델 이해, 형태소 분석 및 토크나이징 이해

- 입력: 한글 문장 2개 (매우 유사하거나 약간 유사한 문장)
- 출력: 유사도 (%)
- 방법: 1) 음절 ngram 방식, 2) 형태소 분석기(또는 WPM model) 두 가지 방식으로 토큰 추출 후에 유사도 계산

  <참고1> 유사도 계산식은 위 1번 또는 다른 유사도 계산식을 사용해도 됨.

  <참고2> 음절 ngram 모델은 bigram, trigram, bi+tri 3가지로 각각 구현

입력문장에서 ngram 토큰으로 분할(또는 추출)하는 함수의 인자값이 2는 bigram, 3은 trigram, 5는 bi tri 둘다 추출하는 방식으로 구현!

<참고3> 형태소 분석기는 KoNLPy 등에 공개된 것 중에서 사용, 또는

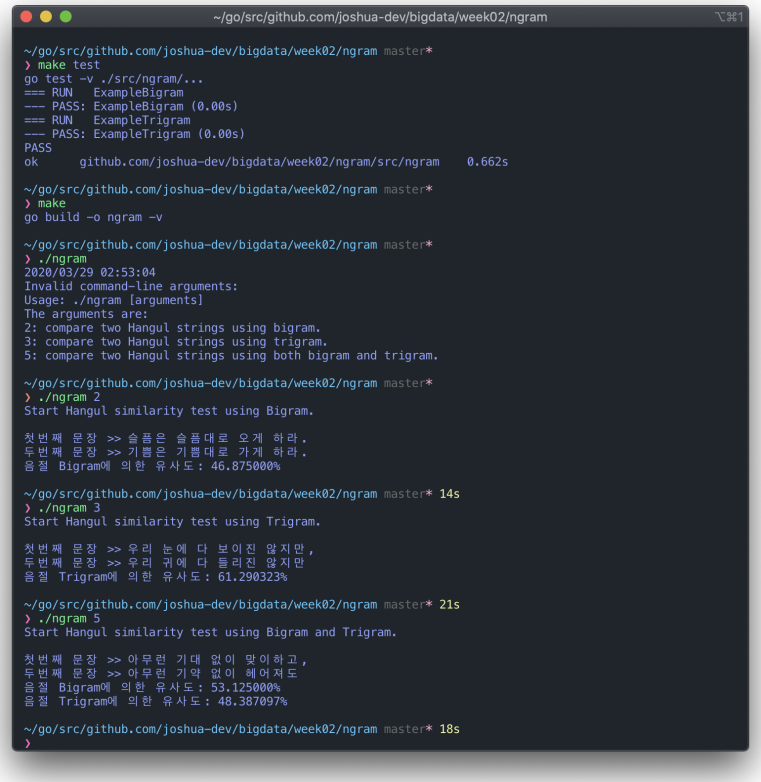
형태소 분석기 대신에 WPM(또는 SPM) tokenizer를 사용해도 됨.

- 제출하는 모든 실습 및 과제의 소스 코드는 아래 GitHub 경로에 있습니다.

```
https://github.com/joshua-dev/bigdata
```

1. 음절 ngram 방식

실행 결과



Installation

```
go get -v github.com/joshua-dev/bigdata/week02/ngram/src/ngram
```

Run

- Bigram

```
make build
./ngram 2
```

- Trigram

```
make build
./ngram 3
```

- Bigram + Trigram

```
make build
./ngram 5
```

Test

```
make test
```

구현 방법

- 아래와 같이 ngram 모델을 이용한 한글 유사도 계산 알고리즘을 구현했다.

  1. ngram이라는 type을 정의한다.

  2. 문장 1개와 n이 주어지면 해당 문장을 ngram으로 변환한다. 이 때, 해당 문장의 공백과 문장 부호를 모두 제거한다.

  3. **2** 에서 얻은 ngram의 각 token들의 출현 횟수를 계산하고 map 구조에 저장한다.

  4. 두 문장을 받으면 각 문장을 1 ~ 3 의 과정을 거쳐 각 token들의 출현 횟수를 구하고, 출현 횟수 정보가 담긴 두 map을 비교하여 공통된 token의 총 갯수를 구한다.

  5. 공통된 token의 총 갯수를 길이가 짧은 문장의 toekn 갯수로 나누고 100을 곱하여 유사도를 구한다. (% 단위이므로)

- 1 을 수행하는 코드

```
// Ngram is n-gram type.
type Ngram []string
```

- 2 를 구현하는 함수 New

```
// New returns a new n-gram with the given n and string.
func New(n int, s string) Ngram
```

이때, 문장 부호와 공백을 제거하는 함수 cleanse를 호출한다.

```
// cleanse returns a string with punctuation removed.
func cleanse(s string) string
```

- 3 을 구현하는 함수 count

```
// count counts the frequency of tokens.
func count(tokens Ngram) (map[string]int, int)
```

- 4 를 구현하는 함수 Compare

```
// Compare compares two hangul string with ngram.
func Compare(first, second string, n int) float64
```

- 5 를 계산하는 코드 (Compare 함수에 포함)

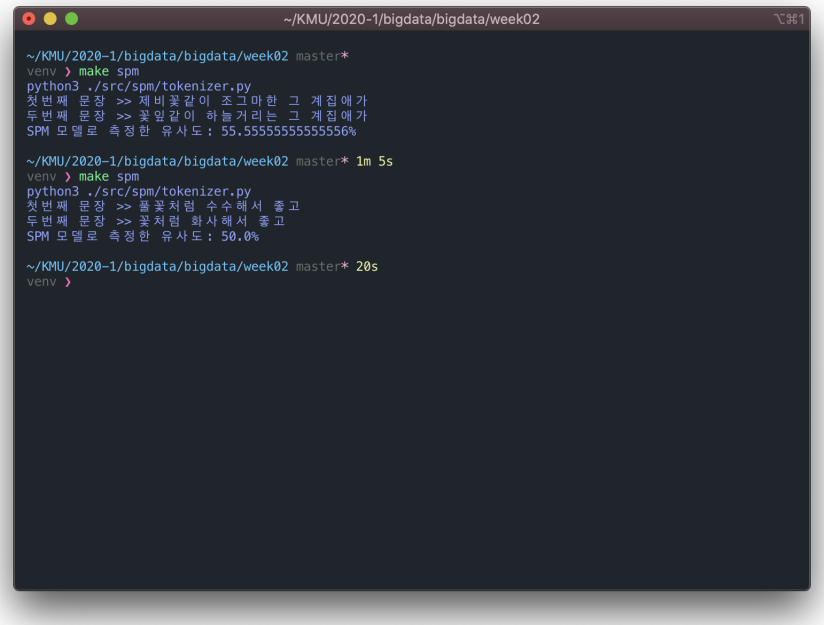
```
var common float64

for token, cnt1 := range firstCount {
  if cnt2, exists := secondCount[token]; exists {
    common += math.Min(float64(cnt1), float64(cnt2))
  }
}

return 100 * common / float64(firstLen)
```

2. SPM model

실행 결과



Installation

```
pip install -r requirements.txt
```

Run

```
make spm
```

구현 방법

Google에서 제공하는 SPM model의 tokenizer API (sentencepiece) 를 사용했다.

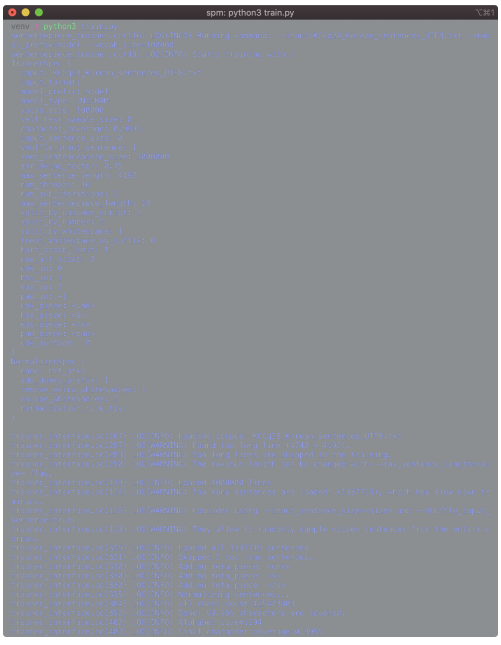
유사도 계산 알고리즘은 다음과 같이 구현했다.

1. KCC 원시 말뭉치를 이용하여 spm model을 생성한다.
2. processor를 생성하고 1 을 통해 얻은 model을 로딩하여 tokenization 준비를 마친다.
3. 두 문장이 주어지면 processor를 통해 tokenization을 각각 수행한다.
4. 3 을 통해 얻은 token 배열을 비교하여 유사도를 계산한다.

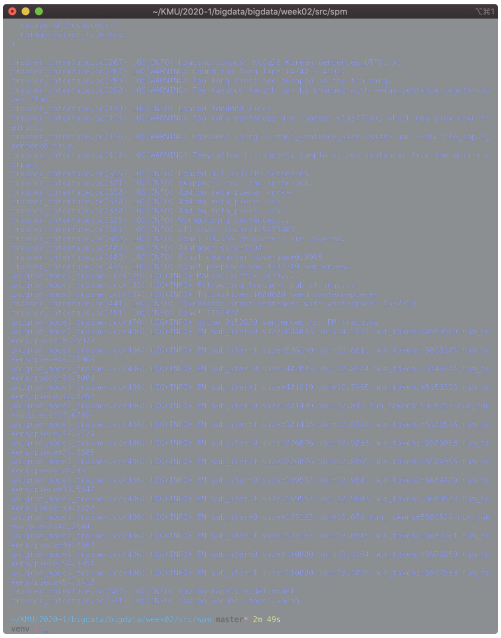
- 1 을 구현한 train.py

```
spm.SentencePieceTrainer.Train('--input=KCCq28_Korean_sentences_UTF8.txt --model_prefix=model --vocab_size=100000')
```

훈련 시작 모습



훈련 종료 모습



훈련 결과 model.model과 model.vocab 파일을 얻게 된다.

- 2, 3 을 구현한 함수 tokenize

```
def tokenize(first: str, second: str):

  '''
  tokenize tokenizes two given strings using trained model obtained by train.py

  >>> @param

    first: a Hangul sentence.
    second: a Hangul sentence.

  >>> @return

    two token list of first and second.

  '''
```

- 4 를 구현한 함수 compare

```
def compare(first: str, second: str) -> float:

  '''
  compare compares two string using spm model.

  >>> @param

    first: a Hangul sentence.
    second: a Hangul sentence.

  >>> @return

    similarity between two sentences using spm model.

  '''
```