

# **Sentiment analysis on product reviews based on weighted word embeddings and deep neural networks**

2021144276 장은준



# Introduct

## 감성 분석

1. **Lexicon** 기반 방식
    - 긍정단어와 부정단어와 관련된 단어 사전을 사전에 만든다.
    - 새로 문서가 들어오면 그 문서를 **token** 화 해서 문서에 나온 단어들의 긍정, 부정을 파악하여 감정을 분석.
  2. 머신러닝 기반 방식
    - 정답을 가지고 있는 데이터를 사용함.
    - 이 데이터를 사용해서 지도기반 머신러닝 분류 모델을 만들 수 있음. (나이브 베이즈, **SVM**, **K**최근접 이웃 알고리즘, 랜덤포레스트 등등)
- 
- 최근 NLP 작업에서 **Neural network** 를 사용하여 텍스트 처리를 하면 예측 성능을 향상시킬 수 있다는 연구가 많음.
  - 그 중에 딥러닝 **RNN**, **CNN**, **GRU**, **LSTM** 등이 사용될 수 있다.



# Introduct

각 기술마다의 예측성능을 실험을 통해 비교.

- embedding skill : (word2vec, fastText, Glove, LDA2vec, and DOC2vec) 사용
- weight function : (TF-IDF, SIF, IDF) 사용
- architecture : CNN, LSTM, RNN, CNN - LSTM


---

# Methodology




# Methodology - Data

1. Twitter product reviews corpus
  - 93,000 개 영어 트윗.
  - 상품과 관련된 것.
  - 각각의 트윗에 감정 상태 labeling.
  - 4명의 전문가가 직접 기록
    - Fleiss's kappa (k) metric을 계산했더니 0.81 k, 괜찮다고 판단.
  - 최종적으로 긍정 데이터 43,000개, 부정 데이터 43,000개를 얻었다.
2. Target dependent twitter corpus
  - 훈련 데이터 6,248개, 테스트 데이터 692개



# Methodology - Embedding

1. word2vec
  - input layer, hidden layer, output layer
  - CBOW : 맥락 단어들을 Input으로 넣어서 target 단어들을 예측.
  - SG(SKIP-GRAM) : 반대로 target 단어로부터 맥락 단어를 예측.
  - 추론을 잘하는 신경망을 학습을 통해서 만들고, 입력 가중치 매개변수의 행을 각 단어의 분산 표현으로 지정하고 벡터로 선정한다.
2. GloVe
  - 동시 발생 행렬과, 동시 발생 확률을 구함.
  - 임베딩된 중심 단어와 주변 단어의 내적이 두 단어의 동시 단어 발생 확률이 되게끔 만든다.
  - 동시 발생 행렬 희소 문제를 해결하기 위해서 단어의 빈도수에 가중치 부여
    1. 문서 내에서 가깝게 출현한 단어들이나 동시 발생 확률이 높은 단어들은 더 중요한 단어로 간주.
3. fastText
  - word2vec 응용.
  - 비슷한 구조를 가지고 있는 단어들에 대해서 각각 다른 벡터를 출력하던 타 모델과 달리 비슷한 벡터를 출력.
  - 철자 단위 정보 n-gram 활용
    1. n-gram을 length(n) 을 각기 달리하여 각각의 character n-gram 을 만든다 => "<", ">" 사용.
    2. n-gram 내부에 단어들이 존재할텐데, 이 단어들을 word2vec 으로 변환하고 이 벡터들의 총합을 구한다.
    3. 그 벡터 값이 바로 단어의 벡터 표현.



# Methodology - Embedding

1. LDA2vec
  - word2vec 과 LDA 기반의 임베딩 기술.
  - 텍스트들의 집합에 대해서 주제를 밝히고 word vector 들은 주제에 따라서 조절.
2. Doc2vec
  - word2vec 응용
  - 문장이나 문서 전체에 대한 vector 를 얻어보자는 아이디어에서 나온 기술.
  - word2vec 의 원리와 비슷하지만 word2vec 에 사용되는 입력값에 문장ID를 입력값으로 추가한다.
    1. 모든 문장에 ID를 부여해서, 문장ID를 얻는다.
    2. 이를 똑같이 벡터화 해서 input 값으로 넣는다.
    3. 문장의 모든 맥락 단어들이 학습에 전부 사용될 때까지 똑같은 문장ID 벡터를 입력값으로 넣어주면서, 문장ID 벡터를 수정한다.



# Methodology - Weighted function

1. IDF
  - 전체 문서의 수 /  $i$ 단어를 포함하는 문서의 수이다.
  - 해당 단어가 문서에 드물게 나올수록 값이 커진다.
  - IDF 값이 커질수록 해당 단어가 중요한 단어임을 의미한다.
2. TF
  - 이 문서에 해당 단어가 등장한 횟수 / 문서내의 전체 단어 수
  - 단어가 얼마나 자주 등장하는지를 드러내는 수치.
3. TF - IDF
  - 문서에 많이 나온 단어 + 전체 문서에서 얼마나 희귀하게 나왔는지 같이 고려하는 방법
  - $tf * idf$
4. SIF
  - $a / a + tf\_ic$
  - $a = 10^{-4}$  로 나타냄.
  - $tf\_ic$  커지면 가중치 값은 낮아지고 낮으면 가중치 값은 높아진다.






## Methodology - Vector aggregation function

1. The weighted sum
  - 기본적인 방식.
2. center-based aggregation
  - 텍스트 문서의 중심 포인트가 계산
  - 다른 문서들에도 적용
3. Delta rule
  - 드물게 출현하는 단어는 제거

$$c = \sum_i w_i \text{tf}_i v_i.$$


$$c = \sum_{i \in V_d} w_i \text{tf}_i v_i - \sum_{i \in V} w_i \text{tf}_{ic} v_i.$$

$$c = \sum_{i \in V_d} w_i \text{tf}_i v_i - \sum_{i \in V_d} w_i \text{tf}_{ic} v_i.$$



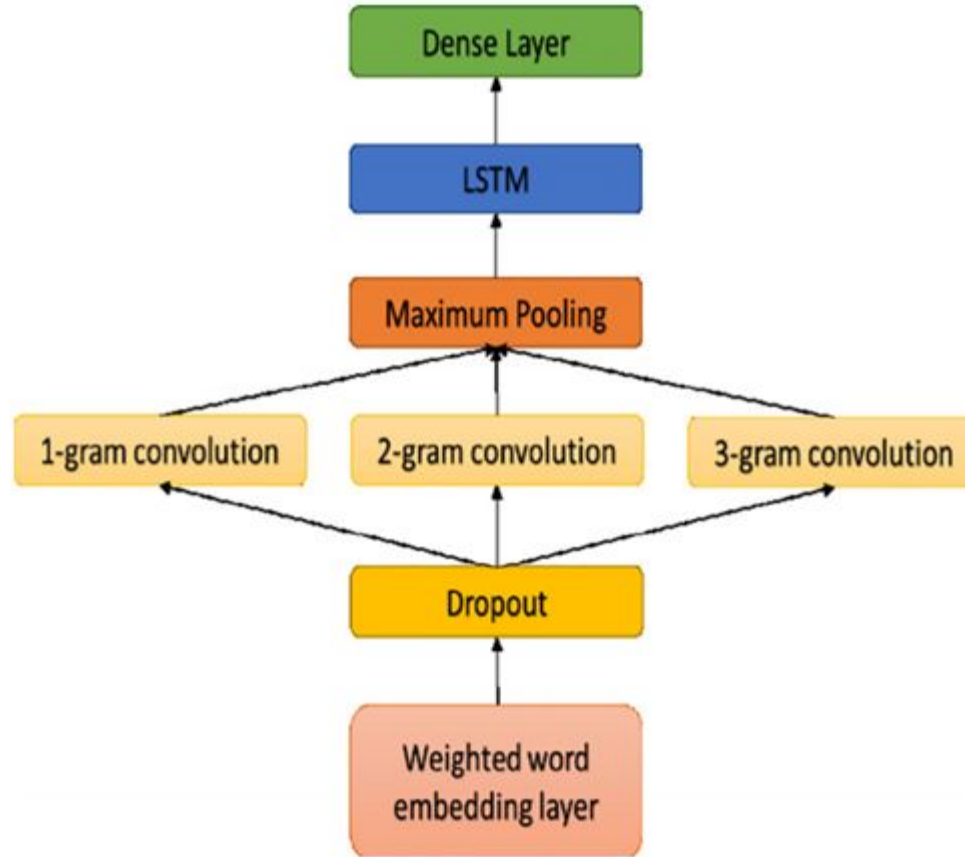
# Methodology - Architecture

1. CNN:
  - Convolution 사용
    - input, output, hidden layer가 있다.
  - hidden layer 구성
    - convolution layer : 출력 feature map 은 입력 feature map 에 대한 합성곱 연산에 의해서 얻어진다.
    - activation function : ReLU 활성화 함수 사용.
    - pooling layer : 맥스 풀링, 평균 풀링 등등이 존재, 데이터의 차원을 줄여준다. 그 중에 맥스 풀링 사용.
    - fully connected layer : 최종적인 결과를 얻기 위한 Affine 층.
2. RNN:
  - 시퀀스 데이터에 사용하기 위해 만들어진 모델
  - RNN 계층은 그 계층으로부터의 입력( $X_t$ )과 그 전 시각으로부터의 출력( $H_{t-1}$ )을 받는다. 그리고 이 두 정보를 바탕으로 현 시각의 출력을( $H_t$ ) 계산한다. 출력값( $H_t$ )은 위로 다음 계층으로도 나아가지만 시계열 방향으로도 나아간다.
  - 장기 의존 관계를 잘 학습할 수 없다.
    - 기울기 소실 문제, 기울기 폭발 문제



# Methodology - Architecture

1. LSTM:
  - RNN의 기울기 소실 문제를 보완하기 위해 만들어짐.
  - RNN에 게이트를 추가
    - 정보를 잊을지, 추가할지 조절
    - input, output, forget
2. GRU:
  - RNN 기반
  - LSTM과 달리 두 개의 게이트 존재.



Proposed Architecture



# Methodology

## 1. CNN-LSTM

### ○ 구조(초반 3개)

1. weighted embedding layer
  - TF-IDF, GloVe, center-based aggregation 이 성능 제일 좋게 나와서 weighted embedding layer 에 이 방법 사용.
  - 미리 훈련한 word vector 사용
  - vector 의 길이는 300.
2. dropout
  - overfitting 방지하기 위해서 사용
3. convolution layer
  - 3개 넣음
  - 필터 개수 80개



# Methodology

$$o_i = M[i : i + h - 1] \otimes w_i,$$

$$c_i = f(o_i + b),$$

1. text matrix
  - 모든 input text 들은 word vector 들의 결합이 된다.
  - input text 는 vector 들의 시퀀스가 된다.  $V = [v_1, v_2, v_3, \dots v_n]$
  - vector sequence는 matrix 가 된다.
    - matrix의 행과 열은 word embedding 차원의 수와 input text의 길이와 관련있다.
2. extract feature
  - 특징을 추출하기 위해서 합성곱 연산을 진행한다.
  - convolution layer 의 필터가 matrix 의 수직방향으로 진행된다.
  - 필터의 넓이는 word embedding width 와 같다.
  - convolution layer out put 도출
    - 1번째 식은 내가 이해하기로는 필터가 아래로 이동하면서 계산하며 나오는  $1 \times 1$  벡터 하나를 의미.
    - 2번째 식은 bias 를 그  $1 \times 1$  벡터에 더해준 추출된 특징,그니까 모든 연산을 다 하면 수직으로 벡터 하나가 다 도출, 그게  $C_i$ , 이  $C_i$  가 여러 필터 종류 중 하나의 결과값.
    - ReLU 활성화 함수 사용



# Methodology

1. MAX POOLING
  - 3개의 Convolution 에서 도출된 값들을 max pooling
2. dense
  - feed forward 신경망
  - L2 정규화 방식
  - drop out 사용해서 overfitting 방지
  - binary cross entropy 손실 함수 사용
  - optimizer adam 사용



# Experiments and results

1. 평가 척도로 분류에서 사용되는 정확도 척도를 사용
2. 각각 다른 **word embedding** 방식과 **weighted function** 방법을 전통적인 딥러닝 방식에 사용해서 평가
3. 최적의 하이퍼 파라미터를 찾을 때 베이지안 최적화 사용.
4. **word2vec**, **fastText embedding** 방식으로 **CBOW**, **SG** 사용, **SG** 방식이 더 좋은 성능을 보임.(벡터 사이즈 300)





# Experiments and results

## 1. table1 result

- 전통 모델에서는 LSTM, 근데 CNN + LSTM 이 성능 더 좋음 (unweighted 중)
- unweighted embedding 기술 중에선 GloVe가 성능이 제일 좋았음.
- sentence padding 이 들어간게 unweighted embedding 보다 더 성능이 좋았다. 근데 weighted word embedding 과 비교하면 안좋다.

Twitter product reviews corpus					
Word embedding scheme	CNN	RNN	LSTM	GRU	CNN-LSTM
word2vec	68,62	72,27	73,79	72,25	77,61
GloVe	72,34	76,01	77,03	75,93	80,92
fastText	72,26	74,53	76,78	75,79	79,67
LDA2vec	70,16	73,78	76,71	75,57	79,03
Doc2vec	69,94	72,64	76,61	74,90	79,00
word2vec + sentence padding	75,28	78,72	79,89	79,72	83,52
GloVe + sentence padding	79,55	83,08	84,08	83,71	87,65
fastText + sentence padding	79,46	81,95	83,86	83,71	86,68
LDA2vec + sentence padding	78,55	81,37	83,84	82,09	86,23
Doc2vec + sentence padding	78,07	80,61	83,46	81,75	86,18
Weighted word embedding	87,12	87,60	91,11	89,00	93,85



# Experiments and results

2. table2 result
  - weight function, vector aggregation function, architecture 고려됐다.
  - 결과를 요약한 그래프가 있다.
    - tf-idf 가 제일 우수.
    - center based aggregation 가 제일 우수.
    - GloVe 가 제일 우수.
3. table3 result
  - target dependent Twitter corpus.
  - table2 와 똑같은 패턴의 결과.

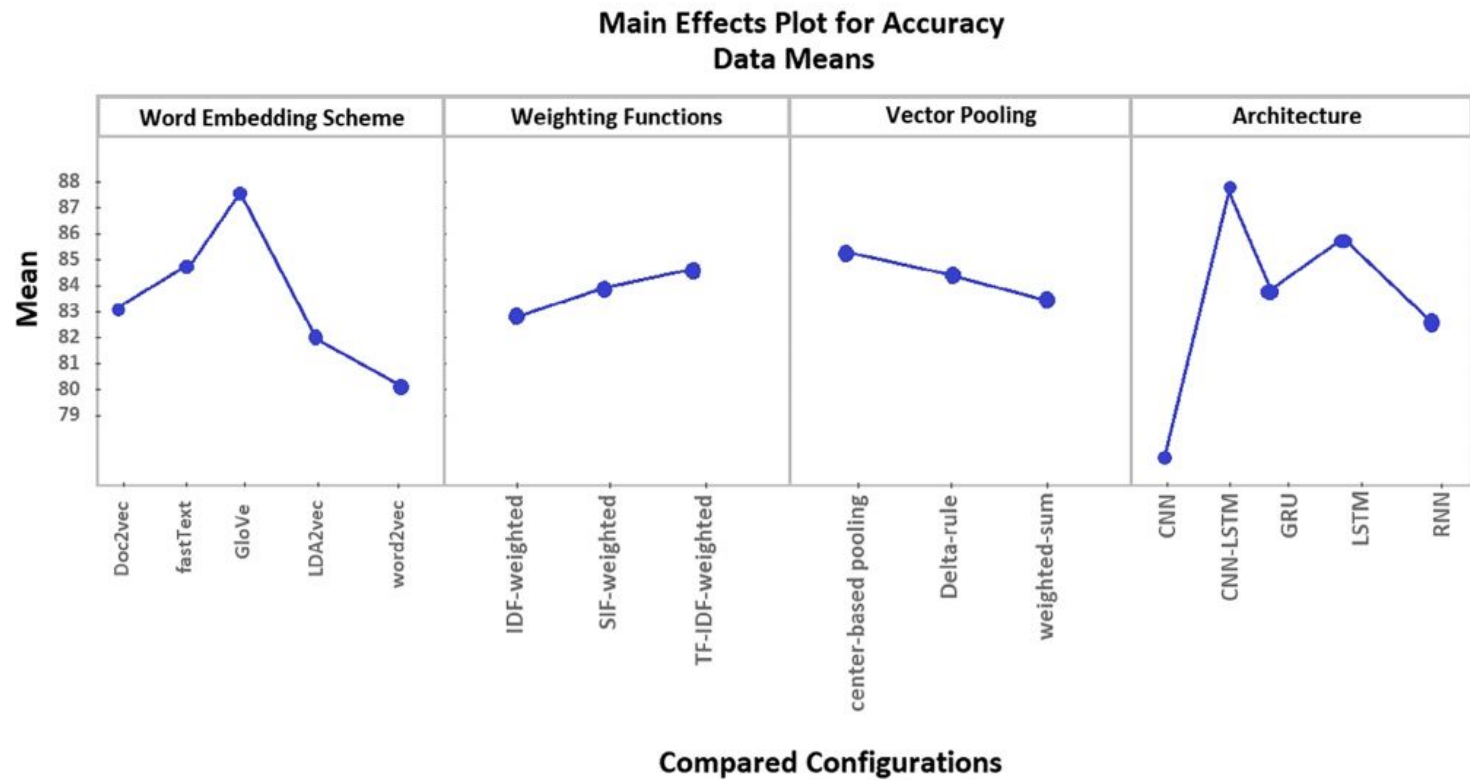


Table 2 graph



# Experiments and results

최종 인사이트

1. word embedding 에 weighted 한게 unweighted embedding 보다 더 성능이 좋다.
2. GloVe가 성능이 제일 좋은데, 이는 통계적인 정보도 GloVe이 담고 있기 때문이다.
3. TF-IDF 가 Weighted function 중에 제일 좋다.
4. center-based aggregation 이 제일 성능이 좋은 집계 함수다.
5. padding 을 한 word embedding 이 unweighthed embedding 보다 더 좋다. 하지만 weighted word embedding 이 제일 좋다.
6. 아키텍처 중에서 전통적인 모델보다 CNN + LSTM 모델이 제일 좋다.
7. 드롭아웃사용
8. 두 개의 데이터를 제안한 모델의 성능평가를 하기 위해서 사용했는데, 데이터가 다르면 어떻게 될지 모른다. 따라서 가장 최적의 결과만을 내놓는 하나의 접근법만 있는 게 아니다.



## Conclusion

감성 분석에서, unweighted word embedding 보다 weighted embedding 이 성능이 제일 좋았고,

전통적인 방식보다 CNN + LSTM 방식이 더 성능이 좋았으니

weighted word embedding CNN LSTM 방식은 좋은 성능을 보장한다. (분류 정확도 성능이 93.95% 까지 나왔다.)