한양대학교

SW프로젝트 요약서

프로젝트 기간	**2022.02.08. - 2022.11.19. (총11개월)**
프로젝트 팀원	**김성애(컴퓨터소프트웨어학부, 4학년)** **** 김남혁(컴퓨터소프트웨어학부, 4학년) – 결과물 개별 제출**
지도교수	**서지원**
프로젝트 명	**자연어 처리(NLP) 모델 Adversarial Attack 분석,개선**
프로젝트 내용	자연어 처리 모델은 Text Classification, Natural Language Inference, Sentence Paraphrase Task 처럼 다양한 분야에서 활용되고 있고, 사람이 보기에는 자연스럽지만, 모델의 입력에 약간의 변화를 주어 모델이 오동작하는 입력을 생성하는 Adversarial Attack 분야도 활발히 연구되고 있다. 최근 자연어 처리 모델 중 문장 분류기 모델을 공격하는 방법으로 CLARE가 있다. CLARE는 문장의 문맥을 고려하면서 Mask-and-Infill 알고리즘으로 Token을 Replace, Insert, Merge 기법을 적용시켜 공격하는 방법이다. Mask-and-Infill 방법은 입력의 특정 Token에 Mask 를 씌우고, 해당 부분을 다른 대안의 Token 으로 채워 공격을 수행한다. 우리의 프로젝트는 관련된 논문과 소스코드를 분석해서 자연어 처리 모델에 대한 Adversarial Attack 을 개선하는 것을 목적으로 한다. 기존의 Mask-and-Infill 에 새로운 Operation 추가한다. - Delete : 주어진 문장에서 문맥적 혹은 문법적으로도 이상이 없는 일부 Token을 지워서 공격을 수행한다. Delete 추가를 통해 기존의 방식인 Replace, Merge, Insert 와 비교하여 Attack rate 가 어떻게 변화하는지 살펴보고 Attack success rate 를 높인다.
기대효과 및 개선방향	자연어 처리 모델과 Adversarial Attack 에 대해 이해한다. Textattack 오픈소스를 분석하고 Word Delete 기능을 추가하여 Attack 성공률을 높일 수 있는 방법을 찾는다.

한양대학교

SW프로젝트 결과보고서

프로젝트명	**자연어 처리(NLP) 모델 Adversarial Attack 분석,개선**
프로젝트 요약	CLARE 모델을 이용하여 Adversarial attack 을 실행하는데, 방법은 Replace, Insert, Merge 로 3가지이다. 여기서 단어를 지우는 Delete 기능을 추가하여 CLARE 의 Adversarial attack 방법을 늘리고 이에 따라 attack success rate 의 변화를 알아보는 것을 목표로 한다. Delete 를 구현할 때 랜덤하게 지우는 것보다 문법 체크 기능과 품사 파악 후 영향을 미칠 수 있는 단어를 지우도록 구현함으로써 성능을 향상시켰다.
프로젝트 기간	**2022.02.08. - 2022.11.19. (총11개월)**
산출물	졸업 작품 (O), 졸업 논문 ( ) (해당 내용은 동그라미 표시)
성과물	특허 ( ), 논문( ), SW(O) (해당 내용은 동그라미 표시)

학과	학번	학년	이름	연락처
컴퓨터소프트웨어	2018007983	4	김성애	abc4571998@naver.com 010-3941-9805
컴퓨터소프트웨어	2015004366	4	김남혁	rlaska951@naver.com 010-2494-1093

한양대학교

목 차

자연어 처리 모델은 다양한 분야에서 활용되고 있고, 사람이 보기에는 자연스럽지만, 모델의 입력에 약간의 변화를 주어 모델이 잘못된 결과를 생성하는 Adversarial Attack 분야도 활발히 연구되고 있다. 최근 자연어 처리 모델 중 문장 분류기 모델을 공격하는 방법으로 CLARE가 있다. 관련된 논문과 Textattack 소스코드를 분석해서 자연어 처리 모델에 대한 Adversarial Attack 을 개선하는 것을 목적으로 한다.

기존의 Mask-and-Infill 에 새로운 Operation 추가하는 것을 목표로 한다.
새로운 Operation 으로 Delete 를 추가하는데, 주어진 문장에서 문맥적 혹은 문법적으로도 이상이 없는 일부 Token을 지워서 공격을 수행하도록 구현한다.
위 방법을 기존 CLARE 에 추가하여 어떠한 변화가 있는지 알아보고 Adversarial attack 을 발전시킨다.

<Contextualized Perturbation for Textual Adversarial Attack> 논문을 이해

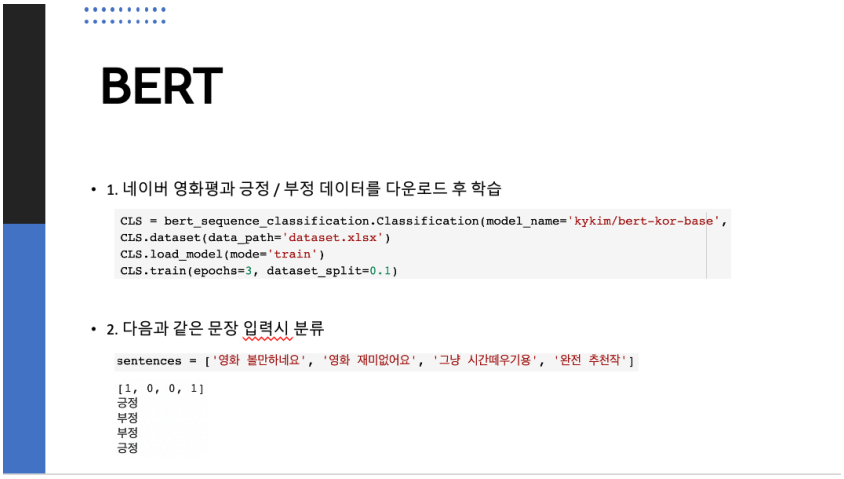
- Adversarial Example 을 생성하는 것은 input text에 perturbation 을 주는 것인데, 출력은 원본과 비슷하게 유지하면서 오류를 발생시킨다.
- CLARE 모델은 3 contextualized perturbation 을 주는데, Insert, Replace, Merge 가 있고, Mask-then-infill 방식을 통해 문법에 맞는 출력을 생성한다.
- CLARE 모델이 주는 perturbation 에서 추가로 만들 수 있는 방법을 고려해보는 것이 다음 목표

2) Transformer & BERT 이해 및 발표

- 트랜스포머의 인코더-디코더 구조와 attention 에 대해 이해한다.
- pre-trained word embedding 을 이해한다.
- BERT 모델에 대해 알아보고 Masked Language Model 의 동작을 파악한다.

3) Image 를 이용한 attack 실습 및 발표

- BERT 모델을 이용해 NLP 를 이해하고, 문장의 긍정과 부정을 분류하는 코드를 실행한다.



- Gradient 를 이용해 적대적 예시를 생성하는 기법인 FGSM 에 대해 공부하고 코드를 통해 확인한다.

1. MNIST dataset 을 이용해 epsilon 값을 변경하여 결과를 비교한다.

```
epsilons = [0, .05, .1, .15, .2, .25, .3]
pretrained_model = "data/lenet_mnist_model.pth"
use_cuda=True
```

2. Gradient 의 부호를 구하고 epsilon 을 곱해 왜곡된 이미지를 생성한다.

```
def fgsm_attack(image, epsilon, data_grad):

    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon*sign_data_grad
    # 값 범위를 [0,1]로 유지하기 위해 자르기(clipping)를 추가
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image
```

- Epsilon 이 커질수록 정확도는 낮아짐을 확인하였다.

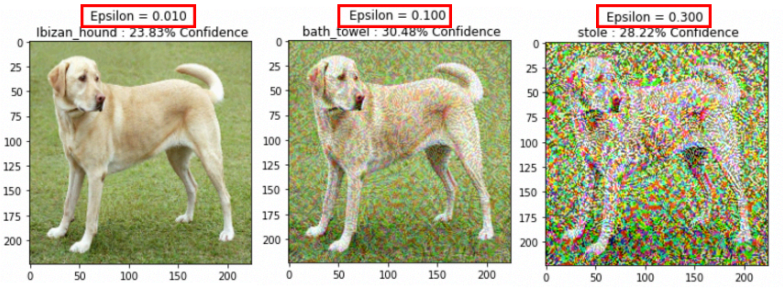
```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.05   Test Accuracy = 9426 / 10000 = 0.9426
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.25   Test Accuracy = 2082 / 10000 = 0.2082
Epsilon: 0.3    Test Accuracy = 869 / 10000 = 0.0869
```

- Sample image 를 이용해 adversarial example 을 생성하는데, epsilon 을 다르게 주어 이미지를 왜곡한다.

```
epsilons = [0, 0.01, 0.05, 0.1, 0.3, 0.5]
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]

for i, eps in enumerate(epsilons):
  adv_x = image + eps*perturbations
  adv_x = tf.clip_by_value(adv_x, 0, 1)
  display_images(adv_x, descriptions[i])
```

- Epsilon 을 적게 주었을 때 왜곡이 눈에 보이지 않았고, 크게 줄수록 왜곡된 이미지가 잘 보인다.



- 현재 CLARE 모델 코드 실행을 시도하고 있으나, 코랩 환경에서의 버전 문제로 인한 실행 오류가 발생하였다. **-> 버전 문제를 수정했지만 일정시간 지나면 코드를 처음부터 재실행해야 한다는 문제와 속도가 지나치게 느린 문제가 발생하여 코드를 테스트하기에는 어렵다고 판단하였다. -> 8/25 연구실 서버를 이용하는 방법으로 해결하였다.**

3. 프로젝트의 기술적 내용

*** 4) 부터는 프로젝트를 본격적으로 프로젝트 코드 구현이 이루어진 것이므로 기술적 내용에 서술한다.**

4) Textattack 오픈소스 분석 및 실행 발표

- Textattack 오픈소스를 git clone 을 통해 내려받고 연구실 서버에서 테스트를 진행하였다.
- Textattack 내의 코드를 수정하고 실행했을 때, 실행값을 변경되지 않는 문제가 발생하였다.
  -> textattack 이 실행되는 path 를 변경해주어 해결하였다.
- Textattack 에서 제공하는 cmd line 명령어를 파악한다. (augment, peek-dataset, train, eval, attack 등이 존재한다.)
- 오픈소스를 이용하다 보니 코드 동작에 대한 이해가 부족했고, 오류를 수정하는 데 많은 시간이 걸려서 성과가 부족했다. 오류를 수정한 후에 잘 동작하는 것을 확인하였고, 다음 발표까지 코드를 구현하는 것을 목표로 하였다.

5) Textattack 에 word deletion 추가 및 구현 방법 발표 (1차)

- 문장의 단어를 랜덤하게 삭제하는 방법에서 시작한다. 이때, 문법을 신경쓰지 않고 랜덤하게 단어를 삭제하기 때문에 문법에 맞지 않는 문장들이 나온다.
- 이를 해결하기 위해 문법 체크 기능을 추가해야 한다고 생각하였고, **Gramformer** 를 이용하였다.
- Gramformer 는 correct 함수를 통해서 파라미터로 들어온 text 를 문법에 맞게 수정하는데, 이때 주어와 동사가 없는 문장에는 주어와 동사를 추가하여 올바른 문장으로 만들어 준다.
- Gramformer 사용 전과 후를 비교하면 "We want to go school." 을 예로 들면 기본적인 랜덤하게 삭제하는 방법의 결과는 "We to go school." 로 올바르지 않은 문장이 나온다. 반면에, 랜덤하게 삭제된 문장에 gramformer의 correct 를 이용하면 주어를 삭제해 "We go to school." 이라는 문법에 맞는 문장으로 수정한다.

```
for item in example:
    print(gf.correct(item, max_candidates=1))
```

- Gramformer 를 활용하는 방법들을 생각해 보았다.
  *** 단어를 삭제한 문장에서 문맥에 맞게 필요한 단어를 추가하기 (Delete + Insert 와 유사하게 동작) -> 앞에서 설명한 부분**

*** 문장에서 단어를 하나씩 삭제해 보면서 새로운 단어를 추가하지 않아도 문법에 맞는 문장들을 가져오기 (Only Delete) – "textattack/transformatios/word_deletion_sengae.py"**

-> 단어가 삭제되더라도 주어+동사를 가지고 있도록 구현한다.

Nltk 를 이용해 tokenize 한 후에 품사를 태그로 만들어 준다.

```
import functools

@functools.lru_cache(maxsize=2**14)
def get_entities(sentence):
    tokens = nltk.word_tokenize(sentence)
    tagged = nltk.pos_tag(tokens)
    entities = nltk.chunk.ne_chunk(tagged, binary
    return entities.leaves()
```

```
sentence = "Because the weather is so cold, I can't go picnic"
get_entities(sentence)
```

```
[('Because', 'IN'),
('the', 'DT'),
('weather', 'NN'),
```

주어 동사를 확인하고 주어 다음에 동사가 오는 순서인지도 확인한다.

```
for index, item in enumerate(grammar_texts):
    entities = get_entities(item)
    check_subject, check_verb = False, False
    for idx, entity in enumerate(entities):
        if entity[1] in ['NNP', 'NE', 'PRP', 'NNS', 'NN', 'NP
            check_subject = True
        ...
        if check_subject == True and entity[1] in ['VBP', 'VB
            check_verb = True
    ........
```

**** 결과는 10번 중에 1번 attack 에 성공하는데, input 문장이 주어+동사 순으로 되어있지 않는 경우가 많아서 기본적으로 문장이 주어+동사로 구성되어 있을 것이라는 예측이 틀렸다. Attack 성공률을 높이기 위해 delete 개선이 필요하고, 다른 방법을 구상해 볼 예정이다.**

```
+--------------------------------+--------+
| Attack Results                 |        |
+--------------------------------+--------+
| Number of successful attacks:  | 1      |
| Number of failed attacks:      | 6      |
| Number of skipped attacks:     | 3      |
| Original accuracy:             | 70.0%  |
| Accuracy under attack:         | 60.0%  |
| Attack success rate:           | 14.29% |
| Average perturbed word %:      | 38.46% |
| Average num. words per input:  | 15.4   |
| Avg num queries:               | 5.0    |
+--------------------------------+--------+
```

6) Word Delete 개선 및 결과 발표 (2차)

- 기존에는 grammar_texts 에 sentence 를 모두 append 하는 방식으로 구현했다. -> 이미 sentence 가 존재하는지 확인 후에 append 해줌으로써 다음 for문에서 grammar_texts 를 확인할 때 중복된 작업을 줄인다. **("word_delete.py" 35 line)**

```
if grammar_sentence not in grammar_texts:
    grammar_texts.append(grammar_sentence.lower())
```

- 이때, 문장에서 한 단어씩 빼고 gramformer 를 통해 문법을 체크하는 방법은 동일하고 nltk로 tokenize 하는 과정은 코드상의 변화가 있지만 기능은 이전과 동일하게 품사 태그를 갖도록 한다. **("word_delete.py" 33-41 line)**
- Tokenize 한 후에 단어 중에 not 이 있을 경우 없애는 처리를 먼저 진행한다. -> not 으로 인해 문장의 긍정/부정이 바뀔 가능성이 높기 때문이다. **("word_delete.py" 43- 50line)**

```
for token in pos:
    if token[0] == "n't" or token[0] == "not":
        del tokenized_sentence[tokenized_sentence.index(token[0])]
        continue
```

- 부사, 형용사, 관사, 접속사도 지워준다. 부사와 형용사는 지웠을 때 의미가 달라질 수 있다고 판단했고, 관사와 접속사는 gramformer 에서 지워주지 않기 때문에 삭제해주는 작업을 추가했다.

```
elif token[1] == "JJ" or token[1] == "RB" or token[1] == "DT" or token[1]
    del tokenized_sentence[tokenized_sentence.index(token[0])]
    continue
```

- 마지막에 문법체크를 한번 더 해주도록 코드를 추가하고, 리턴은 string 에서 input 문장과 같은 타입인 AttackedText 으로 변경한다.
- 공격 성공률은 증가하였지만, 여전히 분류하기 애매한 문장들도 존재하는 문제가 있다.

```
+--------------------------------+--------+
| Attack Results                 |        |
+--------------------------------+--------+
| Number of successful attacks:  | 2      |
| Number of failed attacks:      | 1      |
| Number of skipped attacks:     | 2      |
| Original accuracy:             | 60.0%  |
| Accuracy under attack:         | 20.0%  |
| Attack success rate:           | 66.67% |
| Average perturbed word %:      | 54.17% |
| Average num. words per input:  | 13.4   |
| Avg num queries:               | 24.0   |
+--------------------------------+--------+
```

7) 2차 발표 후 수정 및 개선

- Dataset 변경 : 기존에는 dataset-from-huggingface rotten_tomatoes 를 사용했는데, 문장이 문법이 완벽하지 않은 경우가 많아서 textattack 을 시도하기에 적절하지 못한 상황이 발생하였다. **-> 직접 데이터 파일을 넣어주는 방법을 시도**

- My_dataset.py 를 Textattack 폴더 안에 만들어주고, 다음과 같이 영화 리뷰 문장들을 넣어주었다. **-> 만들어진 dataset 은 문장이 필요 이상으로 길고 문법적으로 맞지 않는 문제가 있었는데, 이 방법으로 적당한 길이의 문장을 선별해서 넣을 수 있기 때문에 잘 동작되는지 결과를 한눈에 보기 좋았다.**

```
import textattack
dataset = [("I enjoyed the movie a lot!", 1),
```

- 기존에는 positive -> negative (1 -> 0) attack 은 가능하지만 negative -> positive (0 -> 1) 는 attack 에 성공하지 못하는 문제점이 있었다. 이전 dataset 에서 positive 와 negative 를 구분하기 애매한 문장들이 존재했기 때문이고, 구분이 더 명확한 dataset 을 넣어주었을 때 해결되었다. 다음은 CLARE 의 replace 가 적용된 결과이고 단어를 바꾸어 positive 와 negative 가 적절하게 변함을 알 수 있었다.

```
1 (100%) --> 0 (100%)

I enjoyed the movie a lot!

I hated the movie a lot!


[Succeeded / Failed / Skippe
 loaded..
<AttackedText "Absolutely ho
[Succeeded / Failed / Skippe
lt 2 ----------------------
0 (100%) --> 1 (100%)

Absolutely horrible film.

Absolutely fine film.
```

- **다음을 delete 를 수행했을 때 결과로, waste 단어를 삭제하고 gramformer 의 correct 를 통해 문법에 맞게 문장을 고쳤을 때 spend 가 오면서 attack 이 성공한 결과이다.**

```
0 (100%) --> 1 (99%)

there are better ways to waste two hours of your life .

there are better ways to spend two hours of your life.
```

- **Word_deletion 에서 only deletion 만 수행된 경우로, 다음과 같이 공격에 성공하는 예시들을 많이 볼 수 있다.**

```
0 (100%) --> 1 (100%)

Never go see the terrible movie .

go see.
```

```
0 (100%) --> 1 (100%)

I do not watch this movie again.

i will watch.
```

- 다음과 같이 attack 에 실패한 경우도 존재한다.

```
0 (100%) --> [FAILED]

Absolutely horrible film.
```

- 다음 문장은 positive 한 문장인데 처음부터 잘못 분류되었기 때문에 Skipped 처리되었다.

```
0 (100%) --> [SKIPPED]

every now and again it's fun to watch a really bad mov
ie .
```

- **2차 발표와 비교해 공격 성공률이 크게 좋아진 것을 알 수 있다.**

```
+--------------------------------+--------+
| Attack Results                 |        |
+--------------------------------+--------+
| Number of successful attacks:  | 2      |
| Number of failed attacks:      | 1      |
| Number of skipped attacks:     | 2      |
| Original accuracy:             | 60.0%  |
| Accuracy under attack:         | 20.0%  |
| Attack success rate:           | 66.67% |
| Average perturbed word %:      | 54.17% |
| Average num. words per input:  | 13.4   |
| Avg num queries:               | 24.0   |
+--------------------------------+--------+
```

```
+--------------------------------+--------+
| Attack Results                 |        |
+--------------------------------+--------+
| Number of successful attacks:  | 7      |
| Number of failed attacks:      | 1      |
| Number of skipped attacks:     | 2      |
| Original accuracy:             | 80.0%  |
| Accuracy under attack:         | 10.0%  |
| Attack success rate:           | 87.5%  |
| Average perturbed word %:      | 54.71% |
| Average num. words per input:  | 8.4    |
| Avg num queries:               | 9.12   |
+--------------------------------+--------+
```

4. 프로젝트의 역할 분담

4.1 개별 임무 분담

자연어 처리에 대한 기본 학습부터 논문 분석, 모델 이해까지 공통 작업으로 함께 진행하였고, textattack 오픈 소스를 분석하고 연구실 서버에서 코드를 구현하면서부터 역할을 분담했다. Delete Operation 을 추가하는 공동 목표를 가지고 각자의 방법으로 코드를 구현하였고, 이때 공통적으로 Gramformer 를 이용하였다. 개별 결과물 제출을 위해 각자의 폴더에서 다른 방식으로 구현하였다.

4.2 개발 일정

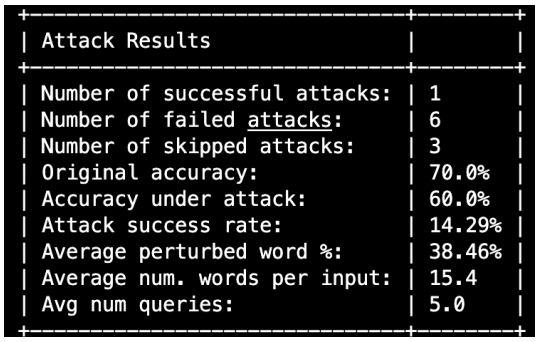
- 2022.02. : pytorch 학습, 논문 이해, 자연어 처리 모델 학습
- 2022.02.25 : 논문 분석 발표
- 2022.03 : transformer & bert , clare 모델 이해
- 2022.04 : Adversarial attack 이해 및 코드 실습
- 2022.04.04 : transformer & bert 발표 및 향후 방향 논의
- 2022.04.29 : BERT 를 이용한 NLP 이해와 긍정과 부정 문정을 구분하는 코드 실습 (네이버 영화평을 이용), MNIST dataset 을 이용한 이미지 Adversarial attack (epsilon 값을 조절)
- 2022.05 : Textattack 오픈소스 코드 분석 및 이해, command line 명령어를 통한 train, attackm augment 등을 숙지
- 2022.06.03 : 진행상황 발표, 오류 파악 및 해결하여 개발환경 세팅
- 2022.06 – 2022.08 : Textattack word delete 파일을 추가하고 실행 결과 분석 및 개선 연구
- 2022.08.16 : 진행상황 발표 및 어려움 논의 (버전 오류 및 colab 환경 문제)
- 2022.08 – 2022.09 : 연구실 서버 사용으로 개발 환경 세팅 및 팀내 코드 분리 및 역할 분담
- 2022.09.20 : 진행 상황 발표 및 코드 실행의 문제 해결
- 2022.10 : Delete 코드 구현 및 문법 체크 기능
- 2022.10.25 : Delete 코드 구현 발표 > 기존 delete 방식에서 Gramformer 를 추가해 문법을 체크할 수 있도록 변경 (랜덤하게 단어를 삭제하는 기존 방식에서 삭제하더라도 문장이 이루어지도록 개선)
- 2022.11 : Delete 문제점 파악, CLARE 모델의 기본 attack 방법 (Replace, Insert, Merge) 와 비교하였을 때 성능을 향상시킬 수 있도록 코드 수정
- 2022.11.19 : 변경된 delete 코드 발표 및 결과 발표 (결론에 대한 추가적인 코드 수정 및 데이터 필요)
- 2022.11.19 ~ : 기존 dataset (huggingface rotten tomatos) 보다 긍정과 부정이 더 명확하게 구분되는 문장들로 dataset 만들어서 추가 (“textattack/my_dataset.py”) , train 횟수를 늘리고 attack 진행, CLARE attack 방법에 추가하여 공격 성공률을 비교

5. 결론 및 기대효과

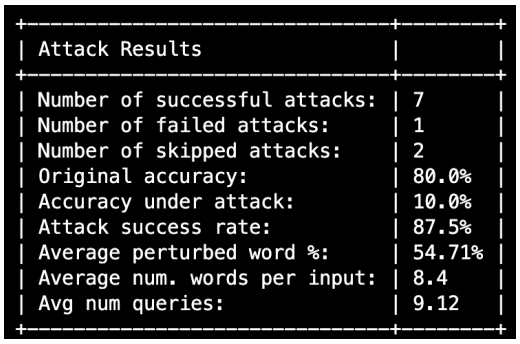
- 문장에서 랜덤하게 단어를 지우는 방식은 주어와 동사의 부재나 문법에 맞지 않는 문장으로 바뀌는 경우가 대부분이다. 이때, Gramformer 을 이용해 문법을 체크할 수 있고, 문법에 맞는

문장이 되도록 단어를 추가하거나 삭제한다. 이렇게 만들어진 문장에서 nltk 를 이용해 토큰화하고 각 단어들의 품사를 알 수 있는데, 사라졌을 때 의미가 바뀔 수 있는 품사들을 삭제했다. 특히, not 의 경우에는 긍정과 부정을 바꿀 수 있는 강력한 단어 후보라고 생각해 지우도록 코드를 구현했다.

처음 랜덤하게 단어를 지우는 방식은 다음 왼쪽 사진과 같이 attack 실패가 많았는데, 마지막 수정 후에는 오른쪽과 같이 실패는 줄이고 공격 성공률을 높일 수 있었다.

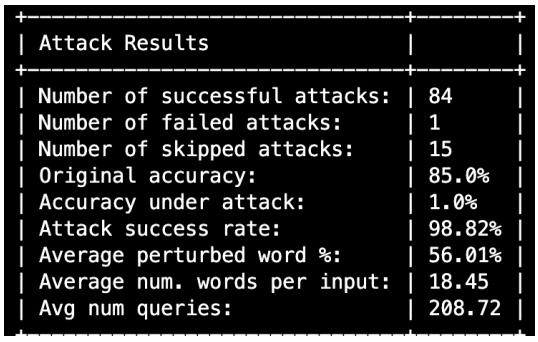


Attack Results	
Number of successful attacks:	1
Number of failed attacks:	6
Number of skipped attacks:	3
Original accuracy:	70.0%
Accuracy under attack:	60.0%
Attack success rate:	14.29%
Average perturbed word %:	38.46%
Average num. words per input:	15.4
Avg num queries:	5.0

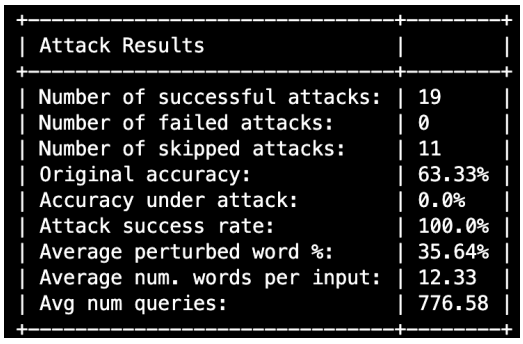


Attack Results	
Number of successful attacks:	7
Number of failed attacks:	1
Number of skipped attacks:	2
Original accuracy:	80.0%
Accuracy under attack:	10.0%
Attack success rate:	87.5%
Average perturbed word %:	54.71%
Average num. words per input:	8.4
Avg num queries:	9.12

- 다음의 왼쪽은 기본 CLARE 모델의 결과이고, 오른쪽은 Delete 기능을 추가한 CLARE 모델이다. 기본 모델과 비교하여 올바르게 분류된 문장인데도 처음부터 잘못 분류되었다고 판단하는 skipped 의 문제가 더욱 많은 dataset 으로 train 을 시키고 attack 을 한다면, 개선될 것으로 기대한다.
  또한, delete 가 단어들을 지워가면서 후보 문장을 만들다 보니, 문장이 길어질수록 처리하는 속도가 크게 증가한다. Avg num queries 에서 차이가 나는데 이는 Delete 에 추가적인 제한을 둔다면 속도를 향상시킬 수 있을 것으로 예상한다.



Attack Results	
Number of successful attacks:	84
Number of failed attacks:	1
Number of skipped attacks:	15
Original accuracy:	85.0%
Accuracy under attack:	1.0%
Attack success rate:	98.82%
Average perturbed word %:	56.01%
Average num. words per input:	18.45
Avg num queries:	208.72



Attack Results	
Number of successful attacks:	19
Number of failed attacks:	0
Number of skipped attacks:	11
Original accuracy:	63.33%
Accuracy under attack:	0.0%
Attack success rate:	100.0%
Average perturbed word %:	35.64%
Average num. words per input:	12.33
Avg num queries:	776.58