1 프로젝트 개요

가. 프로그램명

: 의도적인 표기 변형이 이루어진 문장을 표준어로 교정하는 모델 구축

나. 목적

· 주제에 따른 개발 목표 및 설계 목표 기술

일상적인 채팅에서 우리는 구어성이 두드러진 언어 사용을 흔히 볼 수 있다. 예로, 밥은 먹었어?라는 문장을 채팅에서 밥은 머거써? 또는 밥은 먹어써? 등으로 표기하는 것을 들 수 있다. 이러한 언어 습관은 한국어 채팅 데이터 분석을 어렵게 한다. 다양한 자동 문법 교정 프로그램에서도 이는 완벽히 교정되지 않고 있다. 실제로 Google 대체 단어 suggestion 에서 지금 모해?는 지금 뭐해?로 제안하지만 밥은 머금?은 대체어를 제안하지 못하는 것을 볼 수 있다. 오탈자가 아니라 사용자가 의도적으로 표기 오류를 낸 것으로, OCR 연구와 misspell checking 과는 결이 다르므로 연구가 필요하다.

본 모델은 의도적인 표기 변형이 이루어진 문장을 표준어로 교정하는 모델이다. 채팅체로 쓰여진 문장을 문법에 맞는 문장으로 교정한다. 채팅 데이터 분석 전처리 과정에 활용되길 기대한다. 그리고 후속 연구로는 모델을 반대의 flow 로 배치해 사람처럼 말하는 챗봇을 만들어 보고 싶다.

다. 주요 내용 (그림 또는 표를 포함 자유롭게 기술)

· 프로젝트 관련 자료 및 분석

연구를 위해 직접 카카오톡 데이터를 수집했다. 우선 채팅체에 최적화된 띄어쓰기 framework 인 'chatspace'로 문장을 단어로 분리했다. AI Hub 에 요청해서 받음 대화체 텍스트 (13.7 만개)와 대조해 표기 오류가 있는 단어와 표기 오류가 없는 단어를 분류했다. 결과적으로 아래와 같은 표기 오류 단어 2432 개를 찾았다. 그리고 모델 학습을 위해 교정을 진행해 라벨링했다.

text	label
먹찌	먹지
왜왱	왜왜
자기얌	자기야
모해써	뭐했어
저기욤	저기요
꼬래	그래
대써	됐어
대지	돼지
에바여	에바야
샀냐구	샀냐고
귀욤	귀엽
되어용	되어요
잤넹	잤네
라구	라고
머것냠	먹었냐
신기햄	신기해

· 개발 환경(개발 시스템, 사용언어 등) 및 사용 기술 (알고리즘, 오픈소스 등)

- Python, Pandas
- Pytorch
- Edit distance, Seq2Seq with Attention
- 채팅체에 최적화된 띄어쓰기 API ‘chatspace’

· 프로젝트 주요 기능

다음 1), 2), 3), 4),5)의 관행적으로 쓰이는 표기 변형 습관이 반영된 채팅체 문장을 표기에 맞는 문장으로 바꾸어 준다.

1) 자음의 변형 & 변형 + 이동
    - 할 수 있다 → 할 수 이따
    - 할 수 있다 → 할 수 잇다 (* 소리나는 대로 적기 보다 타이핑 편의를 위해)
2) 자음의 생략 & 생략 + 이동
    - 먹었어 → 머거써
    - 싫어 → 시러
3) 모음의 변형
    - 뭐해? → 모해?
    - 먹어요 → 먹어여
    - 해봐 → 해바
4) 종성에 자음이 추가되는 경우
    - -하세용, -하세욧, -했당, -했답, 넵, 넹
5) 기타 발견하지 못한 case

교정을 위해서 다음 두 가지 방법을 사용했다.

1. Edit Distance

텍스트 유사도를 구하기 위한 척도로 자모 분리 후의 Edit Distance 를 채택했다. 이는 오탈자 교정에 자주 사용되는 기법이다. 표기 오류가 있는 단어와 표기 오류가 없는 단어간의 텍스트 유사도를 구해 가장 유사한 표준어를 찾아 교정한다. 표기 오류가 있는 단어 전체에 대해, 구축한 구어체 데이터셋에서 수정 거리가 가까운 단어들을 리스트업했다.

하지만, 다음과 같은 한계점을 발견했다. [모해]가 input 일 때, [모래, 못해, 묘해, 오해, 토해, 뭐해] 는 모두 [모해]와 동일한 0.3333… 의 수정 거리를 갖는다. 모두 자음 또는 모음 하나만 바뀌기 때문이다. 따라서 output 중 어느 것을 최종 output 으로 선정할 지 우선순위를 부여할 수 없다. 이를 보완할, 표기 변형 패턴을 학습하는 딥러닝 모델이 필요하다고 느꼈다.

2. Seq2Seq with Attention

모델 학습에는 Seq2Seq with Attention 모델을 구현해 사용했다. 더 나은 모델 학습을 위해 단어의 초,중,종성을 분리했다. 종성이 없는 경우 padding 을 넣었다. 비교적 적은 데이터를 보완하고, 일반화하기 위해 n-gram 기법을 적용해 10159 개로 학습 데이터를 늘렸다.

ㅁㅗpㅎㅐp	ㅁㅝpㅎㅐㅆ
ㅗpㅎㅐpㅆ	ㅝpㅎㅐㅆㅇ
pㅎㅐpㅆㅓ	pㅎㅐㅆㅇㅓ
ㅎㅐpㅆㅓp	ㅎㅐㅆㅇㅓp

2 시스템 구성

가. 전체 시스템 구성

· 전체 시스템에 대한 기능 및 동작 설명(그림을 포함하여 자유롭게 기술)

[Workflow]

1. 입력으로 들어온 문장은 띄어쓰기 framework (chatspace)에 의해 단어로 나뉜다.
2. 각 단어들을 표기 오류가 없는 구어체 데이터셋과 대조 (약 13.7 만개, '대화체' 데이터 직접 수집)
3. 데이터셋에 없는 단어를 교정이 필요한 단어로 분류한다.
4. 단어에 포함된 특수문자는 전처리를 위해 제거되지만, 단어 끝의 {,.!?} 문장 부호는 제거되지 않는다.
5. 교정이 필요한 단어에 대해 custom model 및 edit distance 에 의해 교정된다.
6. 교정이 완료된 단어와 교정이 필요 없는 단어들은 다시 문장으로 조합되어 반환된다.
7. 이로써 의도적인 표기 오류에 대한 교정이 완료된 문장을 반환한다.
8. 입력으로 교정이 필요한 단어가 들어오는 경우에 대해서도 함수를 만들어 사용할 수 있도록 했다.

[Github Repository] https://github.com/seoyoungh/ko-chat-checker

나. 세부 기능(또는 모듈)별 동작 원리

세부 기능	주요 내용
모듈 명	• 주요 기능 및 동작
model_only(input)	표기 오류 단어가 포함된 문장을 인풋으로 받고, 모델 예측 결과로만 문장을 교정합니다.
both(input)	표기 오류 단어가 포함된 문장을 인풋으로 받고, 자체 rule 에 따라 모델, Edit Distance 를 모두 활용해 문장을 교정합니다. Edit Distance 의 경우 동일한 거리를 갖는 모든 단어를 리스트로 return 합니다.
model_only_word(input)	표기 오류가 발생한 단어를 인풋으로 받고, 모델 예측 결과로 단어를 교정합니다.
edit_only_word(input)	표기 오류가 발생한 단어를 인풋으로 받고, Edit Distance 로 단어를 교정합니다.
both_word(input)	표기 오류가 발생한 단어를 인풋으로 받고, 두 경우 모두의 output 을 return 합니다.

· 각 세부 기능(또는 모듈)에 대한 기능 및 동작 설명 (그림을 포함하여 자유롭게 기술)

Installation

```
pip install chatchecker
```

```
from chatchecker import ChatChecker
```

Using Example

1. 표기 오류가 있는 단어가 input일 때

```
test_word = "모행"
model_word = ChatChecker.model_only_word(test_word) # 모델만 사용해 교정
edit_word = ChatChecker.edit_only_word(test_word) # Edit Distance만 사용해 교정
model_word, edit_word = ChatChecker.both_word(test_word) # 두 결과 모두 return
```

output

```
model_word: 뭐해
edit_word: 모형
```

2. 표기 오류가 있는 단어와 없는 단어가 섞인 문장이 input일 때

```
test = "이짜나, 배고픈뎅 머행 밥머것어! 아까 밥먹었즤 월욜에보까? 조아요 사랑행 😌!"
model_sentence = ChatChecker.model_only(test) # 모델만 사용해 교정
both_sentence = ChatChecker.both(test) # 자체 rule에 따라 두 방법을 조합해 교정
```

output

```
있잖아, 배고픈데 뭐해 밥 먹었어! 아까 밥 먹었지 월요일에 볼까? 좋아요 사랑해 😌!
```

3 프로젝트 구현 내용

가. 주요 소스 코드 포함 설명(그림을 포함하여 자유롭게 기술)

코드는 모듈화했다. 사용자는 메인 모듈인 ChatChecker.py 의 함수만 불러와 사용하고, 패키지 안에 자모 분리 및 병합 모듈 Jamo.py, Seq2Seq 모델 구현 모듈 CustomModel.py, edit distance 관련 등 기타 필요한 함수 모음 모듈 Utils.py 이 존재한다.

메인 모듈 ChatChecker.py

```
from . import CustomModel # seq2seq module
from . import Utils # other functions

from .CustomModel import *
from .Utils import *

def model_only(sentence):
  text = spacer(sentence)
  text_list = tokenizer(text)
  id_list, wrong_list = check_error(text_list)
  final_wrong, pceq = clean_w_pceq(id_list, text_list)
  corrected = seq2seq(final_wrong)
  output = correct(text_list, id_list, corrected, pceq)
  output = sum(output)
  return output

def both(sentence):
  text = spacer(sentence)
  text_list = tokenizer(text)
  id_list, wrong_list = check_error(text_list)
  final_wrong, pceq = clean_w_pceq(id_list, text_list)
  corrected = seq2seq(final_wrong)
  corrected2 = edit_distance_04(final_wrong)
  result = compare(final_wrong, corrected, corrected2)
  output = correct(text_list, id_list, result, pceq)
  output = sum(output)
  return output

def model_only_word(word):
  result = seq2seq([word])
  suggestion = result[0]
  # print("Suggestion: ", suggestion)
  return suggestion

def edit_only_word(word):
  result = edit_distance_04([word])
  suggestions = ""

  for i in range(len(result[0])):
    if i == len(result[0]) - 1:
      suggestions += result[0][i][0]
    else:
      suggestions += result[0][i][0] + ", "

  # print("Suggestions: ", suggestions)
  return suggestions

def both_word(word):
  model_prediction = model_only_word(word)
  edit_prediction = edit_only_word(word)
  # print("Model suggestion: ", model_prediction)
  # print("Edit Distance suggestion: ", edit_prediction)
  return model_prediction, edit_prediction
```

모델 구현 모듈 CustomModel.py 일부

```
class Seq2Seq(nn.Module):
    def __init__(self,
                 encoder,
                 decoder,
                 src_pad_idx,
                 trg_pad_idx,
                 device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
        self.src_pad_idx = src_pad_idx
        self.trg_pad_idx = trg_pad_idx
        self.device = device

    def make_src_mask(self, src):

        #src = [batch size, src len]

        src_mask = (src != self.src_pad_idx).unsqueeze(1).unsqueeze(2)

        #src_mask = [batch size, 1, 1, src len]

        return src_mask

    def make_trg_mask(self, trg):

        #trg = [batch size, trg len]

        trg_pad_mask = (trg != self.trg_pad_idx).unsqueeze(1).unsqueeze(2)

        #trg_pad_mask = [batch size, 1, 1, trg len]

        trg_len = trg.shape[1]

        trg_sub_mask = torch.tril(torch.ones((trg_len, trg_len), device = self.device)).bool()

        #trg_sub_mask = [trg len, trg len]

        trg_mask = trg_pad_mask & trg_sub_mask

        #trg_mask = [batch size, 1, trg len, trg len]

        return trg_mask

    def forward(self, src, trg):

        #src = [batch size, src len]
        #trg = [batch size, trg len]

        src_mask = self.make_src_mask(src)
        trg_mask = self.make_trg_mask(trg)

        #src_mask = [batch size, 1, 1, src len]
        #trg_mask = [batch size, 1, trg len, trg len]

        enc_src = self.encoder(src, src_mask)

        #enc_src = [batch size, src len, hid dim]

        output, attention = self.decoder(trg, enc_src, trg_mask, src_mask)

        #output = [batch size, trg len, output dim]
        #attention = [batch size, n heads, trg len, src len]

        return output, attention
```

```
device = "cuda:0" if torch.cuda.is_available() else "cpu"

fn = pathlib.Path(__file__).parent / 'vocab.pt'

src_field_stoi, trg_field_stoi, src_field_itos, trg_field_itos = torch.load(fn)

INPUT_DIM = 50 # len(data.TEXT.vocab)
OUTPUT_DIM = 51 # len(data.LABEL.vocab)
HID_DIM = 256
ENC_LAYERS = 3
DEC_LAYERS = 3
ENC_HEADS = 8
DEC_HEADS = 8
ENC_PF_DIM = 512
DEC_PF_DIM = 512
ENC_DROPOUT = 0.1
DEC_DROPOUT = 0.1

enc = Encoder(INPUT_DIM,
              HID_DIM,
              ENC_LAYERS,
              ENC_HEADS,
              ENC_PF_DIM,
              ENC_DROPOUT,
              device)

dec = Decoder(OUTPUT_DIM,
              HID_DIM,
              DEC_LAYERS,
              DEC_HEADS,
              DEC_PF_DIM,
              DEC_DROPOUT,
              device)

SRC_PAD_IDX = 1
TRG_PAD_IDX = 1

model = Seq2Seq(enc, dec, SRC_PAD_IDX, TRG_PAD_IDX, device).to(device)

fn = pathlib.Path(__file__).parent / 'final-model.pt'
model.load_state_dict(torch.load(fn))
```

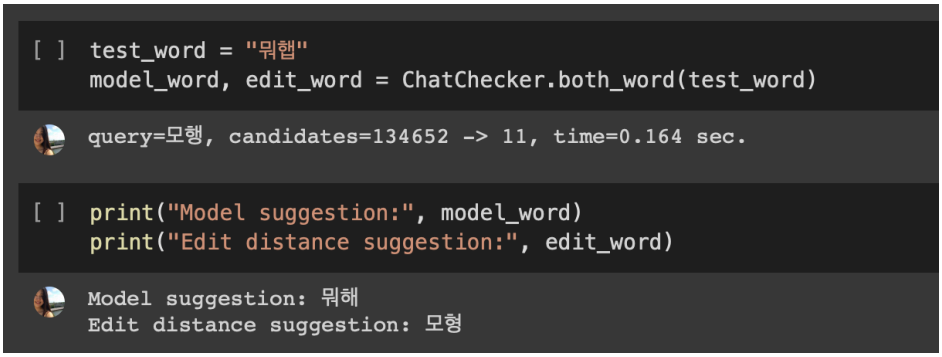
4 프로젝트 동작 및 실험 결과

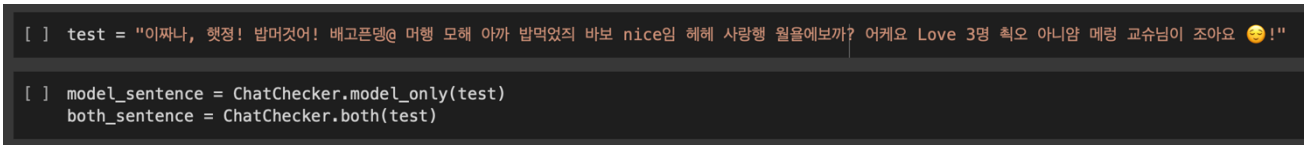
가. 프로젝트 구현에 따른 동작 결과 및 실험 결과 (그림을 포함하여 자유롭게 기술)

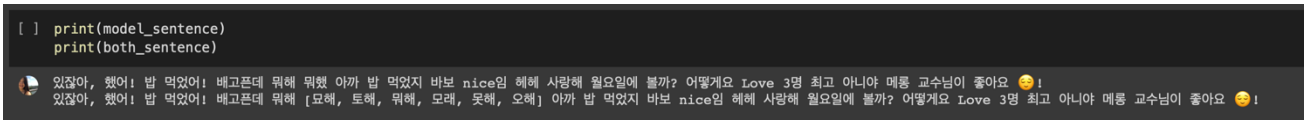
결과적으로, 아래와 같은 성능을 내는 모델을 구축했다.

- test accuracy: 82.3% (400/486)
- train accuracy: 94.3% (8753/9284)
- n-gram 제외 전체 데이터: 89.3% (2171/2432)
- n-gram 포함 전체 데이터: 93.3% (9476/10159)

프로그램 실행 예시







5 프로젝트 수행 결과 분석

가. 기초지식의 활용 내용

자료구조 수업에서 배운 'encapsulation'이 패키지화 할 때 매우 도움이 많이 되었다. 그리고 소프트웨어개발방법및도구 수업에서 배운 Agile 기법, 데이터센터프로그래밍 수업에서 배운 Cloud computing 등을 활용해볼 수 있었다. 데이터마이닝 수업에서 기른 데이터 분석 능력도 도움이 많이 되었다.

나. 수행과정에서의 설계 능력 향상 내용

어떻게 검색 시간을 줄일 수 있을지, 어떻게 모듈화를 해야할지, 코드를 어떻게 짧게 효율적으로 짤 수 있을지, 함수명을 어떻게 통일시켜야 할지 등 실전 코딩에 있어서 설계 능력을 향상시킬 수 있었다. 무엇보다 다른 유저들도 사용할 수 있도록 pip 패키지로 배포하면서 기존에 경험하지 못했던 배포 작업을 해볼 수 있어서 재밌었다. 배포할 때는 package 를 어떻게 import 하도록 코딩해야 하는지 알게 되었다. 그리고 기존에는 TensorFlow 만 사용했었는데, 이번에 PyTorch 를 사용하며 배울 수 있었는데, 후자가 훨씬 직관적이라고 느꼈다. 다양하게 프로그래밍 역량을 키울 수 있었다.

다. 수행과정에서의 문제 해결 내용

기존 OCR, Missspell Checking 에서 사용되던 Edit distance 가 이번 문제에 100% 적합하지 않았다. 따라서 새로운 방법을 찾기 위해 다양한 딥러닝 모델 리서치를 진행했고, 이번 task 에 적합한 Seq2Seq 모델을 찾을 수 있었다. 그리고 저작권 문제로 현재 가용한 맞춤법 API 가 존재하지 않는 상황에서 표기 오류를 찾아내기 위해 대화체 데이터 말뭉치와 대조하는 방안을 생각해냈다. AI Hub 에서 데이터를 요청해, 대규모의 구어체 데이터를 수집할 수 있었다. 하지만 아무래도 표기 오류를 찾기 위해서는 13.7 만개의 데이터와 모두 대조해야해서 시간이 오래 걸렸는데, 이때 dictionary 를 구축해 대조 과정에서의 검색 시간을 줄일 수 있었다.

6 프로젝트 수행 후기

기존에 있는 dataset 으로 해결 방법이 알려진 task 를 수행하는 것이 아니라, dataset 수집부터 모델 선정, 구현까지 직접 해야하는 작업이라 막막하고 힘들었던건 사실이다. 하지만, 프로젝트를 진행하면서 관심 있는 분야인 자연어처리 공부를 마음껏 할 수 있어서 좋았다. 그 전에는 대략적으로 해당 분야를 이해하고 있었다면, 이번 프로젝트를 수행하면서 자연어 처리 task 해결에 적합한 모델을 찾고, 직접 구현까지 할 수 있게 되었다. 딥러닝 프로젝트는 처음이었는데, 정말 재밌었고 배울 수 있는게 더 무궁무진하다고 느껴 앞으로도 관련 프로젝트를 이어 나가려고 한다. 그리고, 패키지 형태 오픈소스로 공개한 것도 처음이었는데, 나름대로 오픈소스 생태계에 기여한 것 같아 기쁘다. 한글 자연어처리 분야는 아직 연구가 많이 진행되지 않은게 사실이다. 특히 주로 채팅으로 대화하는 현 시대 흐름과는 달리, 채팅 데이터에 대한 관심은 적은 편이다. 관련 연구도 없고 레퍼런스도 거의 없었는데, 해당 분야에 대한 토대를 나름 마련한 것 같아 기쁘고 뿌듯하다.