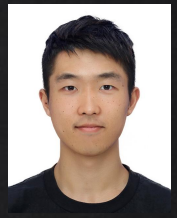
DL Chatbot seminar
Day 01

Introduction to NLP
Chatbot Components
Pipelining
Serving

HELLO!

I am Jaemin Cho



- Vision & Learning Lab @ SNU
- NLP / ML / Generative Model
- Looking for Ph.D. / Research programs

You can find me at:

- heythisischo@gmail.com
- j-min
- J-min Cho
- Jaemin Cho

Today We will cover

- ✘ Introduction to NLP
- ✘ Chatbot Components
- ✘ Pipelining
- ✘ Serving

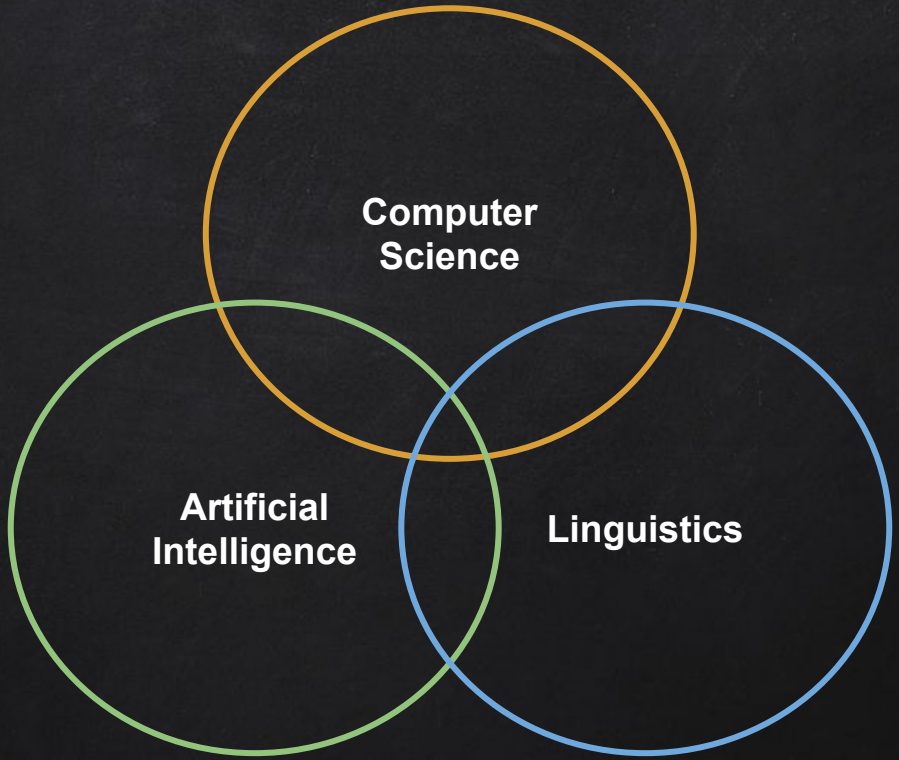
Why?

- ✘ Introduction to NLP
  - 텍스트 데이터의 특징은?!
- ✘ Chatbot Components
  - 챗봇의 각 구성요소 구현
- ✘ Pipelining / Tuning
  - 각 구성요소를 하나로 연결하기 (sklearn.pipeline)
  - 성능을 올리기 위해서 이것지것 시도해보기!
    - Vectorizer, Tokenizer, Classifier 등 세부모델 교체
    - Hyperparameter Search
- ✘ Serving
  - 실제 챗봇에도 적용해 봐야죠! 간단한 카톡봇을 만들어 봅시다.

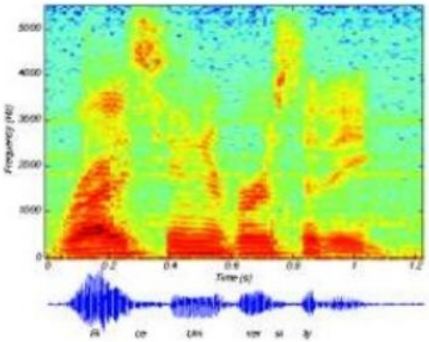
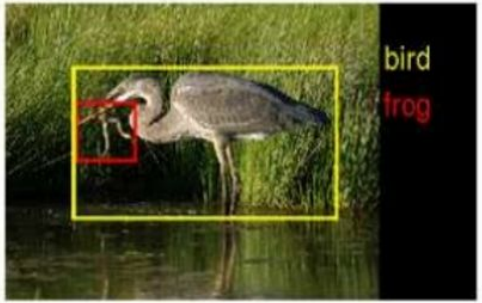
Introduction to NLP

NLP Applications
Libraries
Corpus

Natural Language Processing (NLP)



Natural Language Data

Audio	Image	Text
		
연속	연속	불연속

"tensorflow.org" (2015)

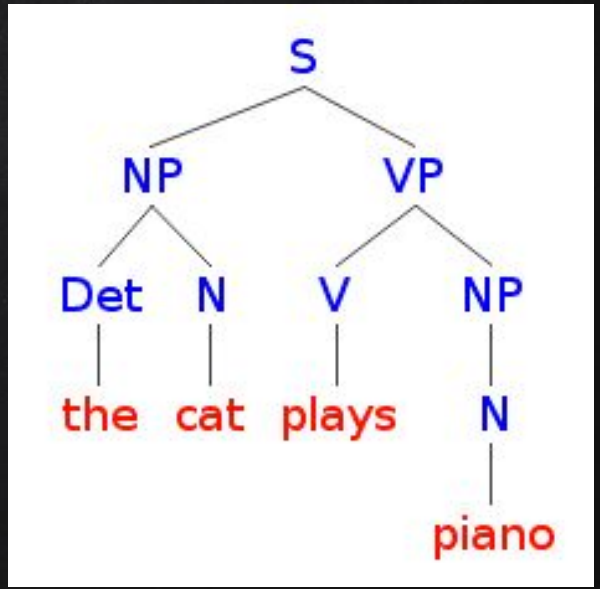
Natural Language Data

- ✘ Sequence of symbols
  - 글자, 단어, 숫자, 특수문자 등의 나열
  - 아무렇게나 나열하면 안 됨
  - 자보화가러영 (?)

    => 영화보러가자 (Language Modeling)

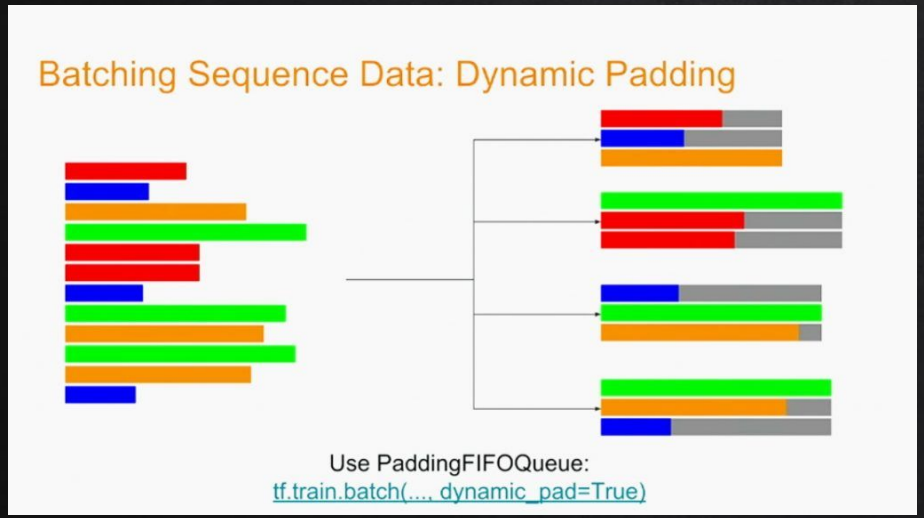
- ✘ 계층 구조
  - 이러한 기호들은 모두 동등하지 않음
  - 몇 개씩 모여서 새로운 의미를 만듦
  - thecatplayspiano

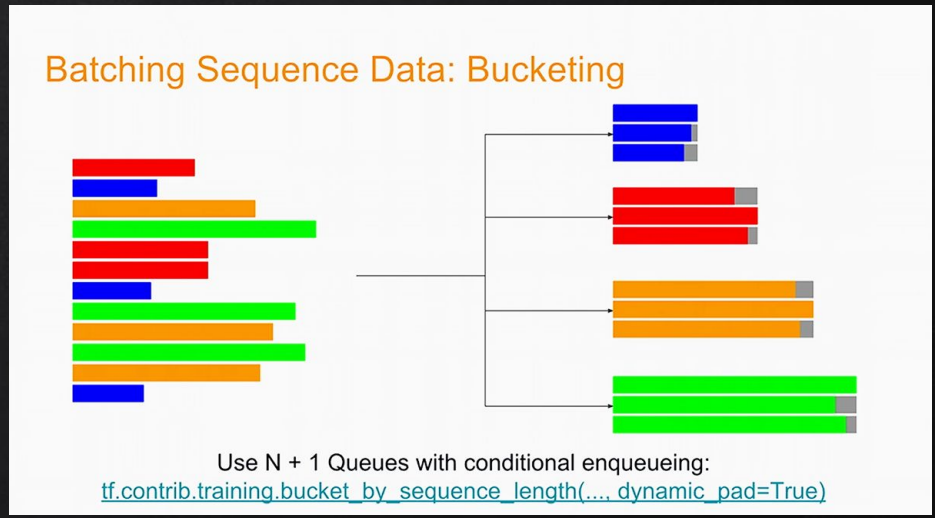
    => The cat plays piano (Spacing)
  - 이런 규칙을 학습하는 것이 목표



Natural Language Data

- ✘ 모든 데이터의 길이가 다름
  - 그런데 병렬처리를 위해서는 네모난 **Tensor** 형태로 만들어야 함
    - 제일 긴 데이터 길이에 맞게 **Padding** 해 주어야 함... ㅠㅠ
    - **Bucketing**





"Sequence Models and the RNN API" (TensorFlow Dev Summit 2017)

NLP Models

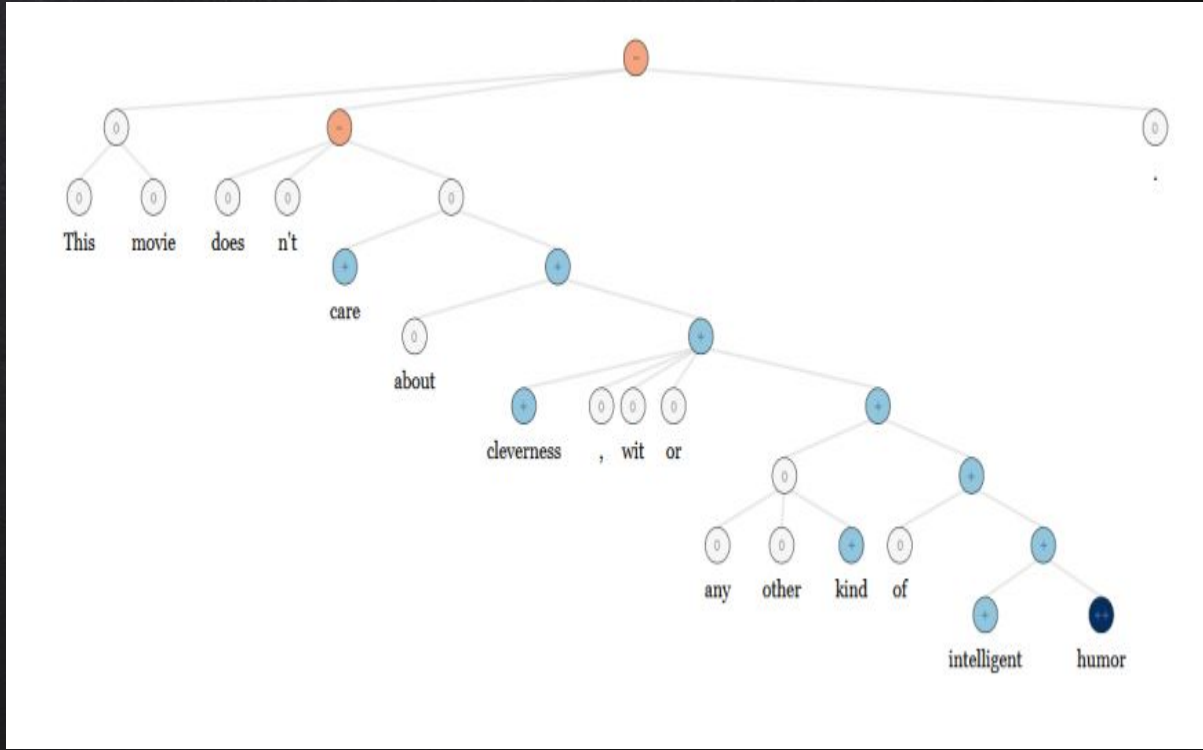
- ✘ Many-to-One
  - 여러 단어로 이루어진 텍스트 => 하나의 값
  - 감정 분석, 주제 찾기, 시험 채점

- ✘ Many-to-Many
  - 여러 단어로 이루어진 텍스트 => 여러 값 / 여러 단어
  - 형태소 분석, 구문 분석, 번역, 요약, 질의 응답

NLP Application – Sentiment Analysis

- ✘ 감정/주제 분석

- ✘ Naive Bayes
- ✘ SVM
- ✘ Random Forest
- ✘ CNN
- ✘ Recurrent NN
- ✘ Recursive NN



"Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank" (2013)

NLP Application – Question Answering

- ✘ 질의응답
- ✘ Retrieval Model
  - ○ Random Forest
  - ○ SVM
  - ○ CNN
- ✘ Generative Model
  - ○ RNN
  - ○ Transformer

closed-domain – 정해진 분야의 질문에 응답

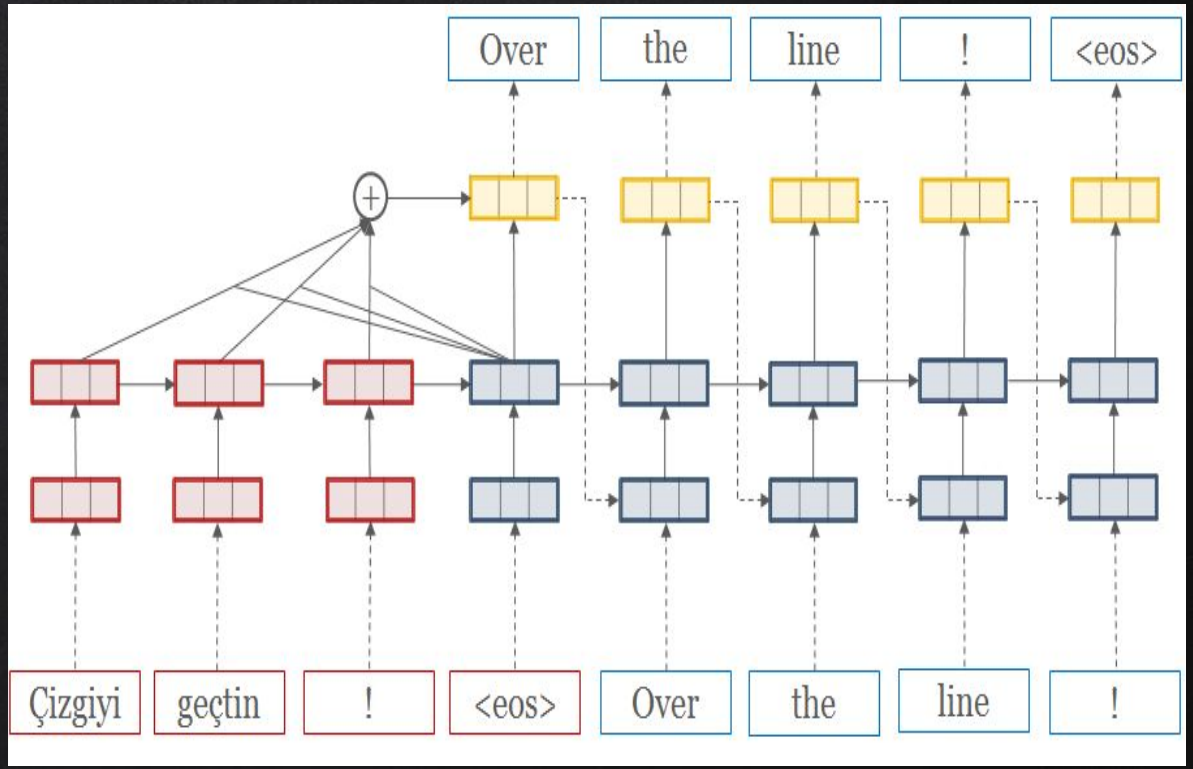
「라마는 무슨 과 ? 」 → 「낙타과」

open-domain – 어떠한 질문에도 응답

「왜 나는 결혼을 못하나 ?」 → 「...」

NLP Application – Machine Translation

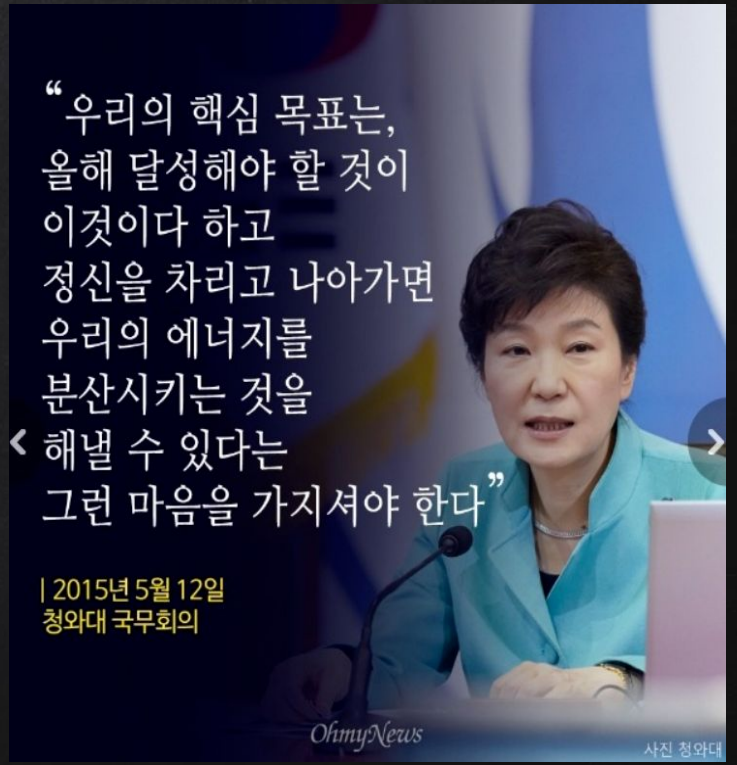
- ✘ 기계 번역
- ✘ Seq2Seq + Attention
  - RNN-RNN
  - CNN-RNN
  - CNN-CNN
  - Transformer
    - self-attention



"openmt.net" (2016)

Why is NLP hard?

- ✘ 언어는 실존하는 개념을 상징하는 함수
  - 글자를 읽을 줄 안다고 뜻을 이해하는 것은 아님
  - 우리 영어 다 잘 읽잖아요…
- ✘ 같은 개념이라도 다른 단어로 표현될 수 있고,
  같은 단어라도 다른 개념을 뜻할 수 있음
- ✘ 특히 한글은 문법, 어순이 중요하지 않음
  - 단어의 출현/순서만으로는 문장의 뜻을 이해하기 어려움
- ✘ 세대, 직종에 따라 사용하는 '언어' 자체에 큰 차이가 있음
- ✘ 판사님 이 자료는 고양이가 만들었습니다.



Korean Text Data

- ✘ 말뭉치 (코퍼스)
  - 세종 코퍼스 / KAIST 코퍼스
    - 세종계획 부실의혹
  - 위키피디아 / 나무위키 덤프
  - Naver sentiment movie corpus
  - https://github.com/datanada/Awesome-Korean-NLP
  - 대부분 옛 서적들 => 신조어 대응 불가

- ✘ 웹 스크래핑
  - 구어체, 신조어 많음
  - SNS (Facebook, Twitter, Instagram)
  - 블로그, 포럼, 카페

- ✘ 공개되어있는/쓸 만한 한글 대화 데이터는 없습니다. (사실상 MNIST조차 없다고 보아야...)
  - 챗봇 구축에 필요한 데이터를 충분히 확보한 후 시작하세요

English QA Data

- ✘ bAbI
  - 페이스북
  - 20개 task
  - 단답형 주관식

Task 1: Single Supporting Fact

Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 2: Two Supporting Facts

John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 19: Path Finding

The kitchen is north of the hallway.
The bathroom is west of the bedroom.
The den is east of the hallway.
The office is south of the bedroom.
How do you go from den to kitchen? A: west, north
How do you go from office to bathroom? A: north, west

Task 20: Agent's Motivations

John is hungry.
John goes to the kitchen.
John grabbed the apple there.
Daniel is hungry.
Where does Daniel go? A:kitchen
Why did John go to the kitchen? A:hungry

- ✘ SQUAD
  - 스탠퍼드 대학교
  - 수능 형태
  - 지문 / 문제 => 답
  - 지문 속에 답이 있음
  - "밑줄 치기" 문제

Airport

The Stanford Question Answering Dataset

An airport is an aerodrome with facilities for flights to take off and land. Airports often have facilities to store and maintain aircraft, and a control tower. An airport consists of a landing area, which comprises an aerially accessible open space including at least one operationally active surface such as a runway for a plane to take off or a helipad, and often includes adjacent utility buildings such as control towers, hangars and terminals. Larger airports may have fixed base operator services, airport aprons, air traffic control centres, passenger facilities such as restaurants and lounges, and emergency services.

What is an aerodome with facilities for flights to take off and land?
airport

What is an aerially accessible open space that includes at least one active surface such as a runway or a helipad?
landing area

What is an airport?
aerodrome with facilities for flights to take off and land

English QA Data

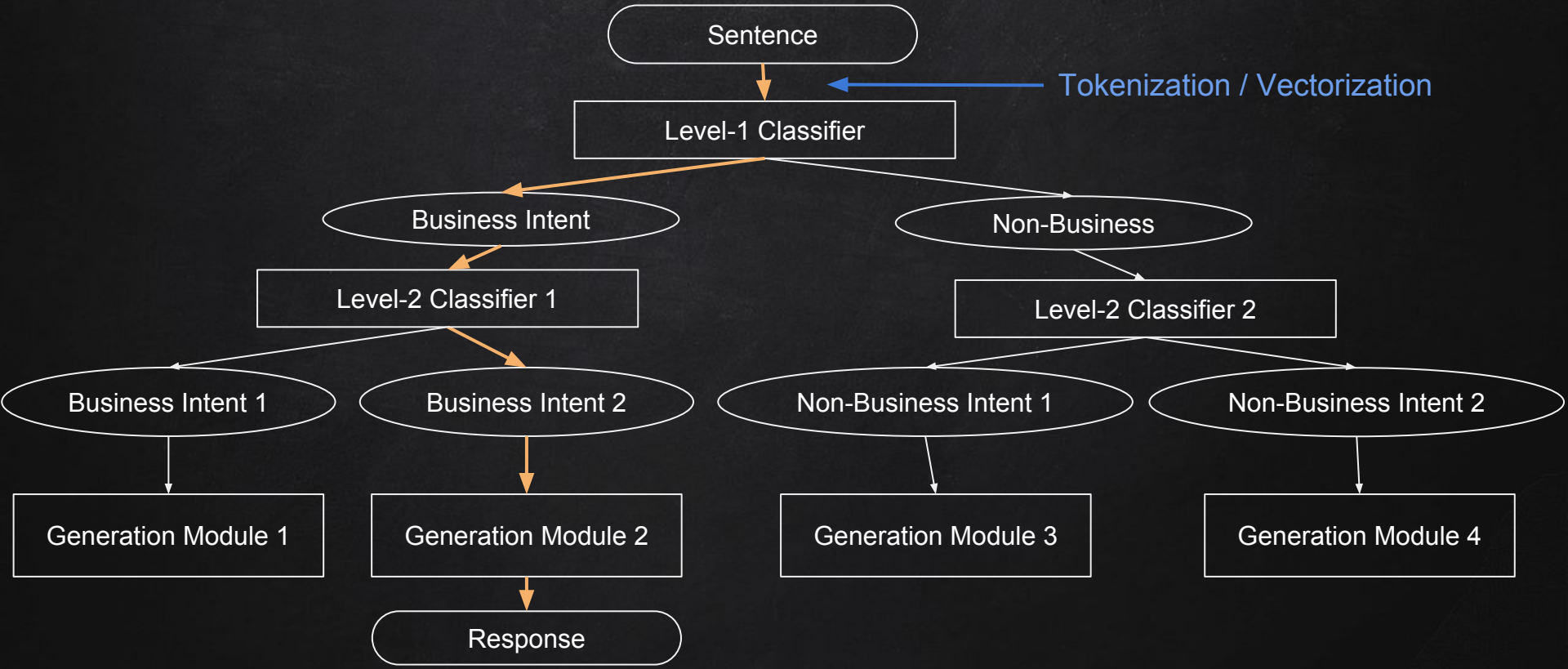
- ✘ ParlAI
  - https://github.com/facebookresearch/ParlAI
  - 페이스북
  - 대화 모델 학습을 위한 framework
  - SQuAD, bAbI tasks, MS MARCO, MCTest, WikiQA, WebQuestions, SimpleQuestions, WikiMovies, QACNN & QADailyMail, CBT, BookTest, bAbI Dialog tasks, Ubuntu Dialog, OpenSubtitles, Cornell Movie, VQA-COCO2014, VisDial and CLEVR

- ✘ Malluba
  - https://datasets.maluuba.com/
  - 마이크로소프트
  - NewsQA
    - 뉴스 읽고 질문에 답하기
  - Frames
    - 세계 여러 도시 정보를 가지고 최단 거리 예약하기 (Goal-oriented)

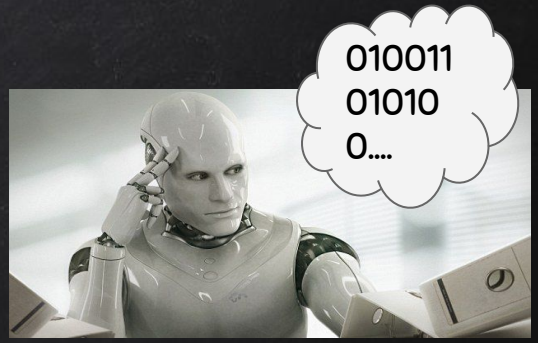
Chatbot Implementation

Tokenizer
Feature extractor
Classification
Answer

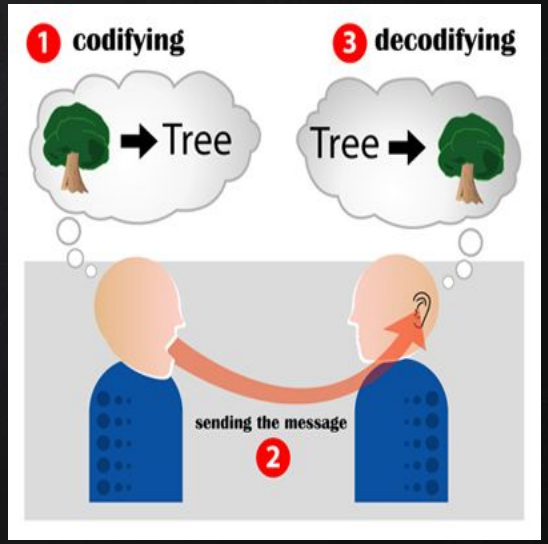
Schema


Sentence
Tokenization / Vectorization
Level-1 Classifier
Business Intent
Non-Business
Level-2 Classifier 1
Level-2 Classifier 2
Business Intent 1
Business Intent 2
Non-Business Intent 1
Non-Business Intent 2
Generation Module 1
Generation Module 2
Generation Module 3
Generation Module 4
Response

Vectorization



- 컴퓨터 프로그램은 전기적 신호 (= 숫자) 만을 이해할 수 있음
- 정보를 컴퓨터가 이해할 수 있는 형태 (= 벡터) 로 바꾸어 주어야 하고,
  컴퓨터의 출력을 우리가 이해할 수 있는 형태로 다시 바꾸어야 함.
- 비슷한 의미로 사용되는 개념들
  - Feature engineering
  - Encoding
  - Embedding
  - Knowledge representation
  - Latent (hidden) Variable
  - Machine Understanding



Tokenization

- ✘ 문장을 어떻게 숫자로 나타내는 것이 효율적일까?
- ✘ 총 한글 단어가 100개이고, 문장이 최대 10단어로 이루어져 있다면,
  - 10^100 (= GooGol)
  - = 메모리 사망...
- ✘ 문장은 단어들의 집합이니까, 단어에 각각 고유번호를 부여하자!
- ✘ 예) 나는 사과를 먹는다
  - 나는 = 1 / 사과를 = 2 / 먹는다 =3
  - 나는 사과를 먹는다 => [1, 2, 3]

Tokenization

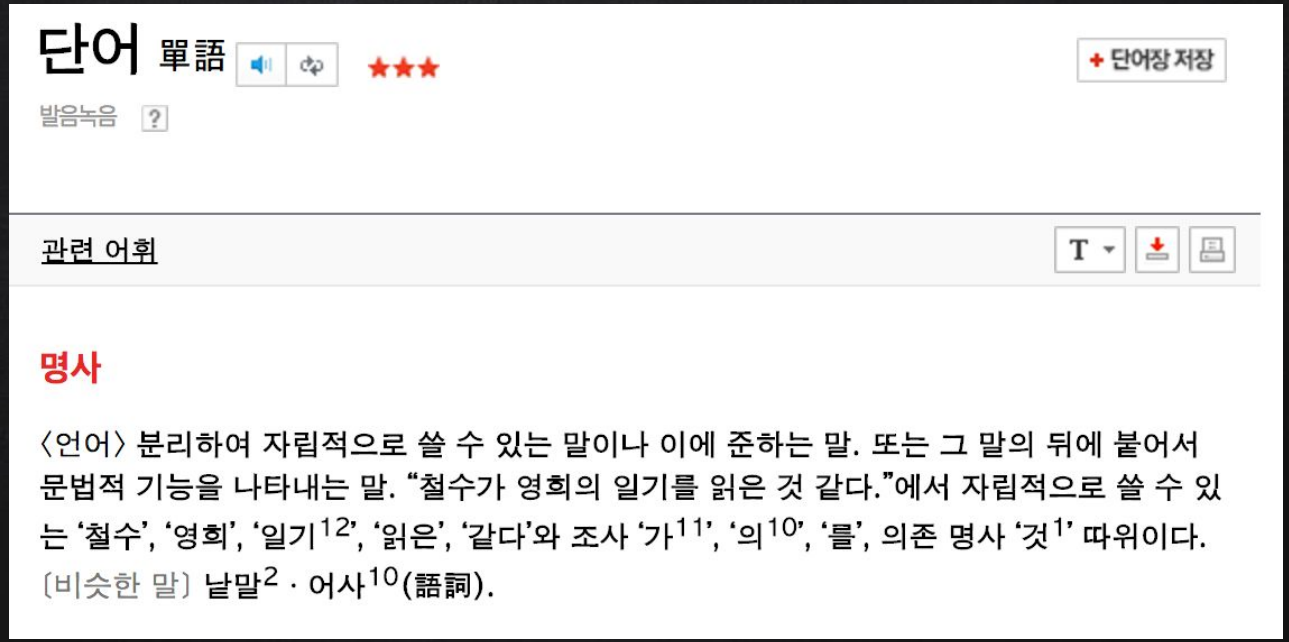
- ✘ 그런데, 모든 문장은 길이가 다 다르다!
- ✘ 병렬처리를 위해서 텐서를 사용해야 함
- ✘ 최대 문장 길이를 정하고, 길이에 미치지 못하는 문장들은 **padding**을 해 주어야 함.
- ✘ 예) 나는 사과를 먹는다 / 나는 복숭아를 아주 좋아한다
  - 나는 = 1 / 사과를 = 2 / 먹는다 = 3 / 복숭아를 = 4 / 아주 = 5 / 좋아한다 = 6
  - Batch_size = 2
  - Vocab_size = 6
  - [나는 사과를 먹는다,
    나는 복숭아를 아주 좋아한다]
    => [[1, 2, 3],
    [1, 4, 5, 6]]

Tokenization

- ✘ 그런데, 모든 문장은 길이가 다 다르다!
- ✘ 병렬처리를 위해서 텐서를 사용해야 함
- ✘ 최대 문장 길이를 정하고, 길이에 미치지 못하는 문장들은 padding을 해 주어야 함.
- ✘ 예) 나는 사과를 먹는다 / 나는 복숭아를 아주 좋아한다
  - <pad> = 0 / 나는 = 1 / 사과를 = 2 / 먹는다 = 3 / 복숭아를 = 4 / 아주 = 5 / 좋아한다 = 6
  - Batch_size = 2
  - Vocab_size = 7
  - [나는 사과를 먹는다,
    나는 복숭아를 아주 좋아한다]
    => [[1, 2, 3, 0],
    [1, 4, 5, 6]]

Tokenization

- ✘ 그런데, 한글 문장도 단어의 리스트일까?



Tokenization

- ✘ 영어와는 다르다! 영어와는…
- ✘ 영어에는 없고 한글에는 있는 '조사'!
  - 그런데 딱히 중요한 의미를 담고 있지는 않음
  - 잘 떼어내어야 문장에서 의미있는 부분만 골라낼 수 있음
- ✘ 띄어쓰기가 잘 되어있다는 가정 하에,
  - 영어는 str.split() 만으로 단어 분리가 가능
  - 한글은…
    - 정규표현식 (Regular Expression)
      - import re / from nltk.tokenize import regexp
    - 형태소분석기
      - from konlpy.tag import Mecab, Twitter
    - KoNLPy 는 현업에서 쓰일 정도로 다양한 단어 및 신조어를 커버하지 못합니다.
      - 지속적인 사용자 사전 추가를 통한 tokenizer업그레이드 필요
      - https://github.com/lovit/soynlp

	격의 종류	형 태
격조사	주 격	-이/-가, -께서, -에서
	서술격	-이다
	목적격	-을/-를
	보 격	-이/-가
	관형격	-의
	부사격	-에, -에서, -에게, -에게서, -한테, -(으)로, -와/-과…
	호 격	-아, -야

Tokenization

정규표현식

```
import re
```

executed in 21ms, finished 09:44:09 2017-08-19

```
def korean_filter(text):
    text = re.sub(r'[^가-힣 +]', # 이 패턴에 매칭되는 입력은
                  '', # 이걸로 대체하겠다. (지워버리겠다)
                  text)
    return text
```

executed in 24ms, finished 09:44:09 2017-08-19

```
korean_filter('This is English 그리고 이건 한글!').split()
```

executed in 58ms, finished 09:44:09 2017-08-19

```
['그리고', '이건', '한글']
```

KoNLPy

```
from konlpy.tag import Twitter
```

executed in 101ms, finished 09:42:23 2017-08-19

```
twitter = Twitter()
```

executed in 284ms, finished 09:42:23 2017-08-19

```
twitter.pos('This is English 그리고 이건 한글!')
```

executed in 1.35s, finished 09:42:25 2017-08-19

```
[('This', 'Alpha'),
 ('is', 'Alpha'),
 ('English', 'Alpha'),
 ('그리고', 'Conjunction'),
 ('이건', 'Noun'),
 ('한글', 'Noun'),
 ('!', 'Punctuation')]
```

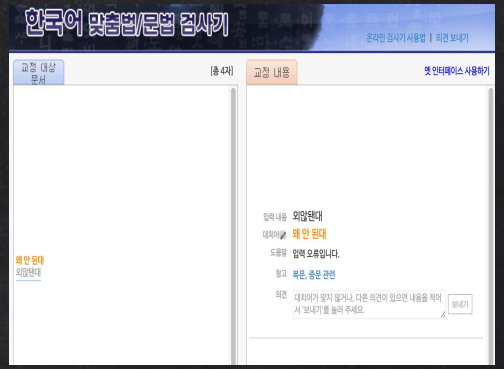
Tokenization

- 물론 이건 띄어쓰기/맞춤법을 잘 지켰다는 가정 하에서..

통계청, http://hikostat.kr/2282 (2014)

Tokenization

- ✘ 띄어쓰기 / 맞춤법 교정
  - API
    - 부산대 맞춤법 검사기
    - Daum



- ✘ 널리 쓰이는 알고리즘
  - CRF
    - Pycrfsuite
      - python-crfsuite를 사용해서 한국어 자동 띄어쓰기를 학습해보자.
  - LSTM
    - https://github.com/dsindex/segm-lstm (TensorFlow)
  - LSTM-CRF
    - 논문: Bidirectional LSTM-CRF Models for Sequence Tagging
    - Pytorch Official tutorial
  - ex) 이문장띄어쓸래

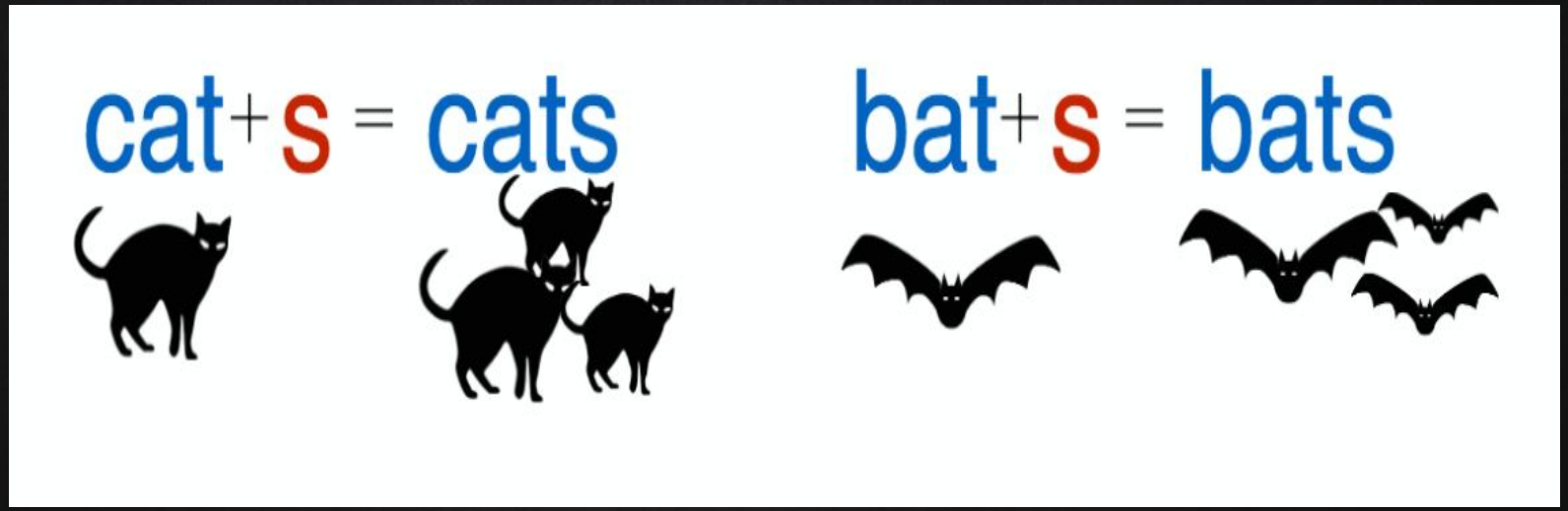
    => 이/B 문/B 장/I 띄/B 어/I 쓸/I 래/I => 이 문장 띄어 쓸래

Tokenization

- ✘ 일반적으로는 단어 단위의 Vocabulary가 널리 사용되지만, 더 작은 단위도 사용됩니다.
  - OOV (out-of-vocabulary; 고객이 학습하지 않은 단어를 사용했을 때) 문제 해결에 큰 도움
  - 삼촌 / 외삼촌 / 숙모 / 외숙모 / 할아버지 / 외할아버지 / 아버지
- ✘ 음절 단위 (Character-level)
  - 삼 / 촌 / 숙 / 모 / 할 / 아 / 버 / 지 / 외
  - 한글은 글자도 이미 조합된 것이기 때문에 자음/모음 단위 (ㅅ + ㅏ + ㅁ = 삼) 도 사용
- ✘ 글자의 일부분 혹은 전체를 모델링 (Subword)
  - Byte-pair encoding (BPE)
    - 삼촌 / 숙모 / 아버지 / 할 / 외
- ✘ Hybrid
  - Word vector는 Char vector들을 합친 것
    - 평균 / NN, CNN, LSTM
  - 외할아버지 = 외 + 할 + 아버지 = 외 + 할 + (아+버+지)

Tokenization

- ✘ **Tokenization + Vectorization**이 **Continuous Space**에서 잘 이루어지면!
  - 한 번도 보지 못한 단어라도 **Token**들을 조합해서 뜻을 유추할 수 있습니다.



Tokenization

- ✘ 참고하세요!

- ✘ 형태소 (Morpheme)
  - 뜻을 가진 가장 작은 말의 단위
  - ex) 풋사과 => 풋 / 사과

- ✘ 단어 (Word)
  - 자립할 수 있는 말, 자립할 수 있는 형태소에 붙어서 쉽게 분리할 수 있는 말
  - ex) 푸른 하늘을 향해 물줄기가 치솟고 있는 것을 바라보았다.
    - 푸른 / 하늘 / 을 / 향해 / 물줄기 / 가 / 치솟고 / 있는 / 것 / 을 / 바라보았다

- ✘ 품사 (Part-of-Speech)
  - 단어를 문법적인 성질의 공통성에 따라 나눈 부류
  - 형식 분류
    - 불변어: 체언, 수식언, 독립언, 관계언 (서술격조사 제외)
    - 가변어: 용언, 서술격 조사
  - 기능 분류
    - 체언 / 용언 / 수식언 / 관계언 / 독립언
  - 의미 분류
    - 명사 / 대명사 / 수사 / 동사 / 형용사 / 관형사 / 부사 / 관형사 / 조사

Vectorization

- ✘ 뭐 들어온 문장을 단어/형태소/음절 단위로 잘 분리했다고 치고...
- ✘ 그럼 문장을 어떻게 벡터로 만들까?

Vectorization Techniques

- ✘ Count-based representation (출현 빈도 중요시)
  - TF-IDF => "같은 단어가 많으면 비슷한 문서 + 너무 자주 나오는 단어는 무시하기"
    - Scikit-Learn
    - (from sklearn.feature_extraction.text import TfidfVectorizer)

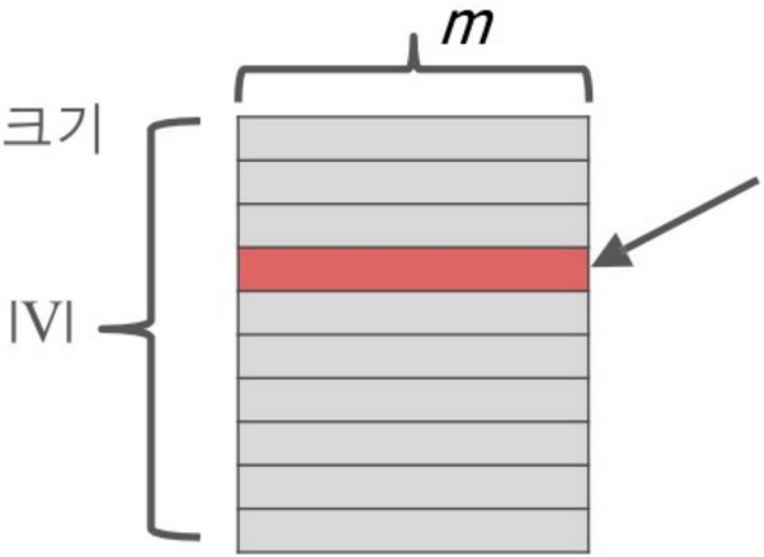
- ✘ Distributed representation (문맥 중요시)
  - Neural Probabilistic Language Model => "딥러닝이 알아서 해주겠지"
    - PyTorch (nn.Embedding)
    - TensorFlow (tf.nn.embedding_lookup)
  - Skip-gram / CBoW / FastText => "단어의 의미는 문맥에 따라서 결정된다"
    - Gensim (from gensim.models import word2vec)
    - FastText

- ✘ Combined
  - GloVe => "둘 다 중요하니까 둘 다 쓰자"
    - SpaCy / glove-python

Embedding Matrix

단어벡터(임베드 벡터)의 집합

m 은 임베드벡터의 크기
(임의값)

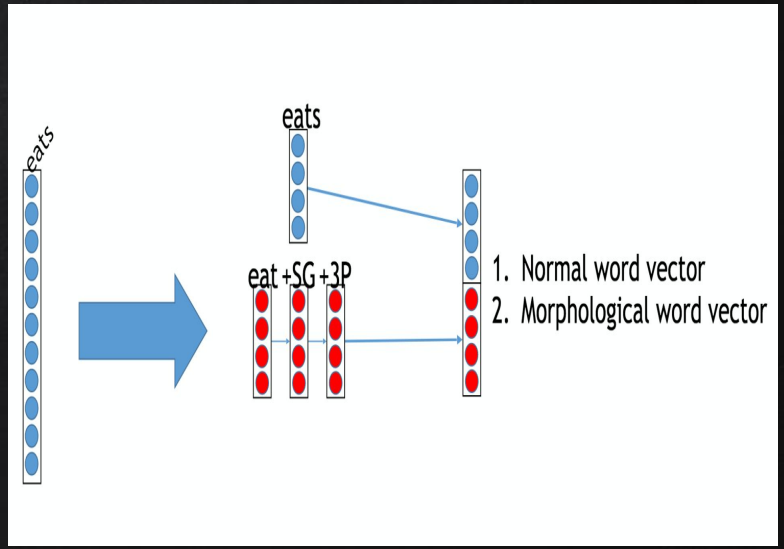


i 번째 행
= i 번째 단어를 나타내는 벡터

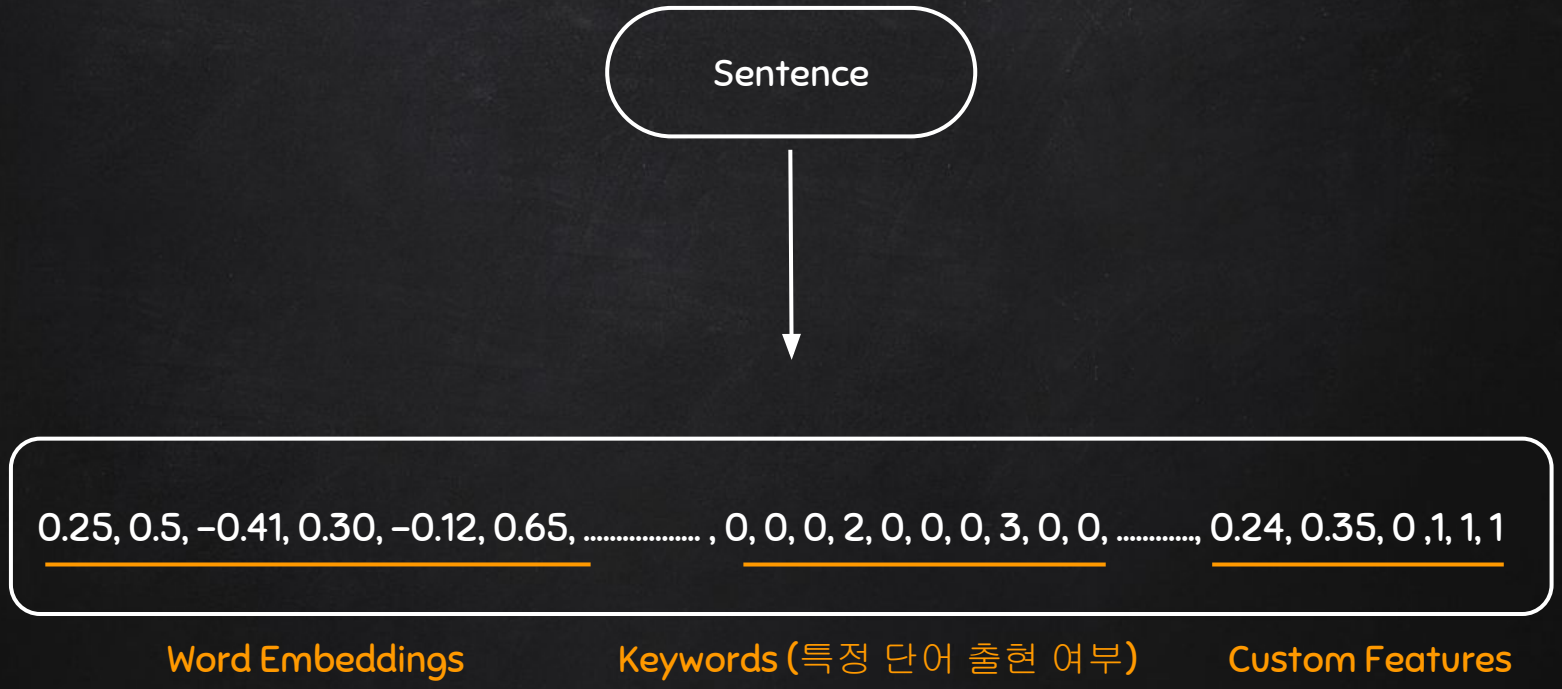
10000+ 단어 x 1000+ 차원
=> 학습하는 데 오래 걸림
=> pretrained 많이 이용
https://github.com/Kyubyong/wordvectors

Vectorization

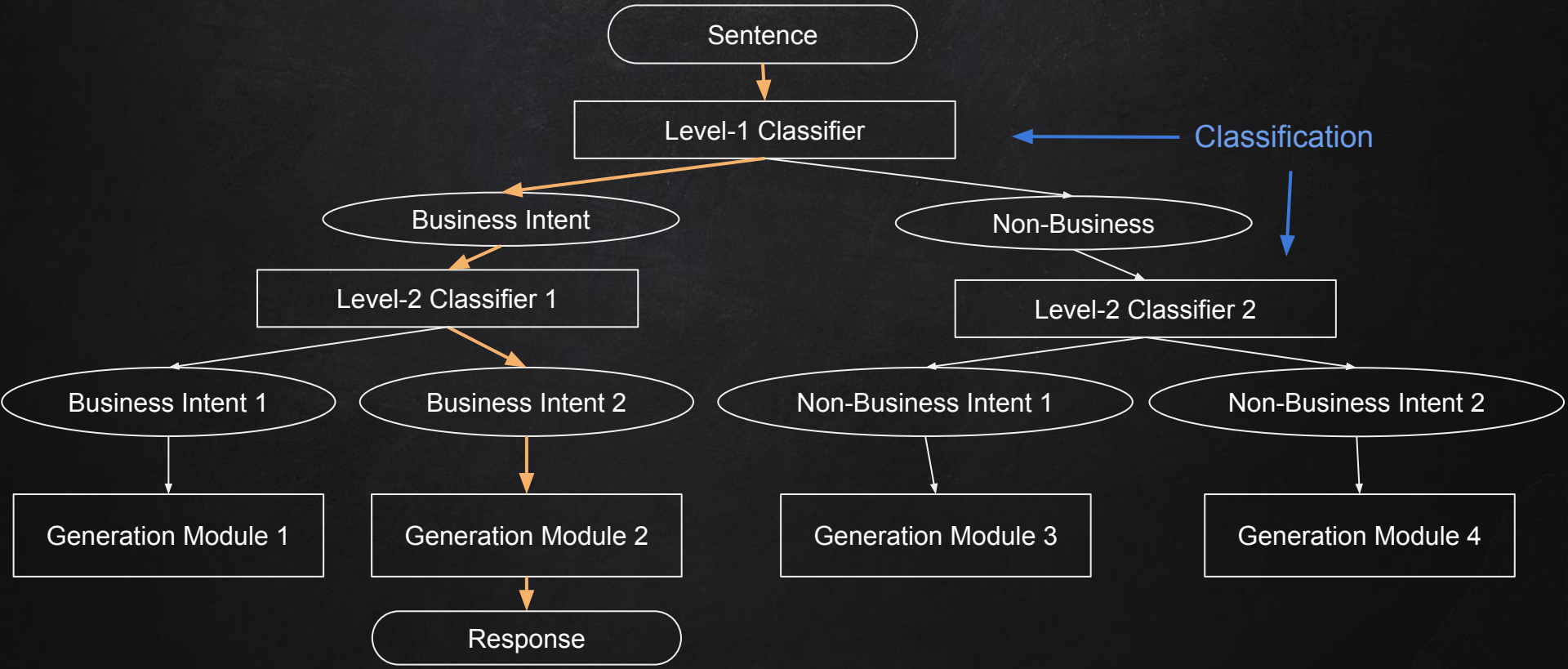
- ✘ Word-embedding은 그 단어가 어떤 단어 주위에 많이 분포하는지에 따라 결정됩니다.
- ✘ 따라서 그 단어가 무슨 뜻인지는 전혀 신경쓰지 않습니다.
- ✘ 딥러닝으로 의미도 학습할 줄 알았지만…
- ✘ “대출하고싶어요" = “투자하고싶어요"
- ✘ 대출 = 투자 …?
- ✘ 그래서 추가적인 정보를 만들어서 붙여주기도 합니다.



Vectorization



Schema


Sentence
Level-1 Classifier
Classification
Business Intent
Non-Business
Level-2 Classifier 1
Level-2 Classifier 2
Business Intent 1
Business Intent 2
Non-Business Intent 1
Non-Business Intent 2
Generation Module 1
Generation Module 2
Generation Module 3
Generation Module 4
Response

Classification

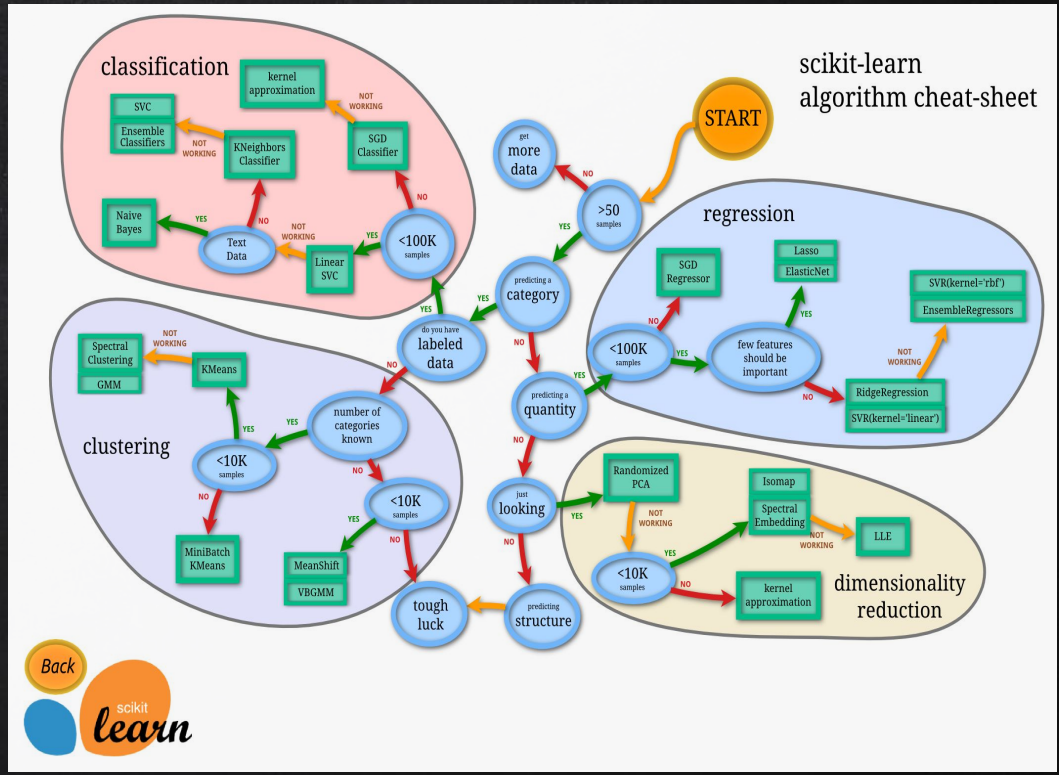
- ✘ 텍스트를 벡터로 잘 변환했다고 치고..
- ✘ 이제 이 벡터를 가지고 좋은 대답을 잘 생성해야 할 텐데,
- ✘ 아쉽게도 **RNN** 기반 **Generative model** 은 현업에서 사용하기에는 부족한 점이 많습니다.
  - 앞 문맥을 자꾸 까먹는다.
  - 자아가 없다.
  - 문법에 맞게 말을 만들어내지 못한다.
  - 디버깅이 어렵다.
- ✘ 실제로 현업에서 쓰이는 챗봇은 아래와 같이 대답을 생성합니다.
  - 고객의 의도는 유한 개가 있다고 가정
  - 고객의 질문 및 문맥을 보고 유한 개의 주제/의도 (Intent) 중 하나를 선택 (분류 문제)
  - 각 **Intent** 의 **answer module**을 이용해서 대답 생성

Classification

- 여러 가지 텍스트 분류 모델들
  - Tree-based models
    - Decision Tree / Random Forest / Gradient Boosting
    - Scikit-learn / Xgboost
  - Neural Networks
    - Fully Connected NN / CNN / RNN
    - PyTorch / TensorFlow
  - TF-IDF
  - Support Vector Machines (SVM)
  - Naive Bayes
  - Latent Dirichlet Allocation (LDA)

Classification

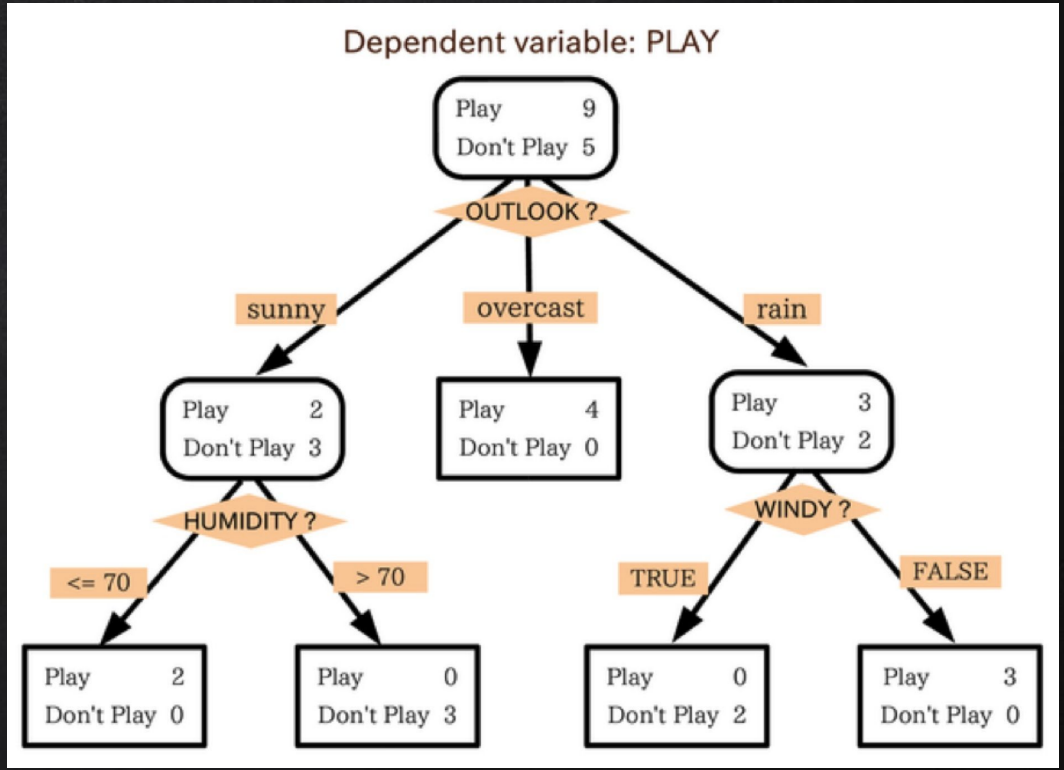
- ✘ 모델이 너무 많아요 ㅠㅠㅠㅠ
- ✘ 어떤 모델을 써야 할까요?
  - 풀려고 하는 문제가 무엇인가요?
  - 데이터를 봅시다!
- ✘ Scikit-Learn cheet-sheet



http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

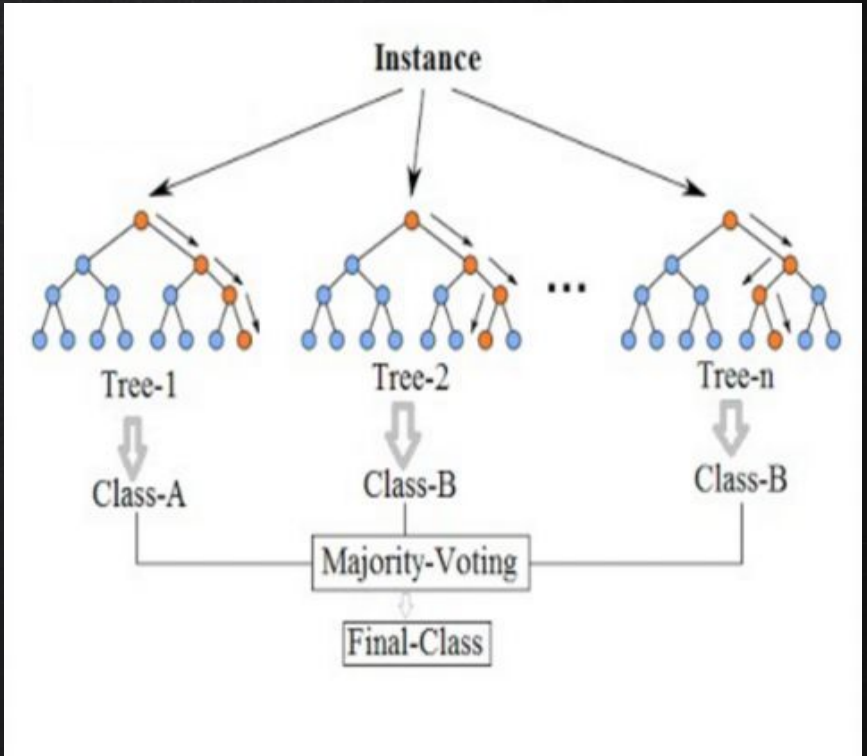
Classification

- ✘ Decision Tree
  - 오늘 놀러 나갈까?
  - 고려할 요소들
    - 날씨
    - 습도
    - 바람이 부는지
- ✘ Custom feature 무한정 생성 가능
  - 데이터가 적으면 Feature Engineering 해야죠...
  - ex)
    - 문장 길이
    - 특정 단어 나왔나?



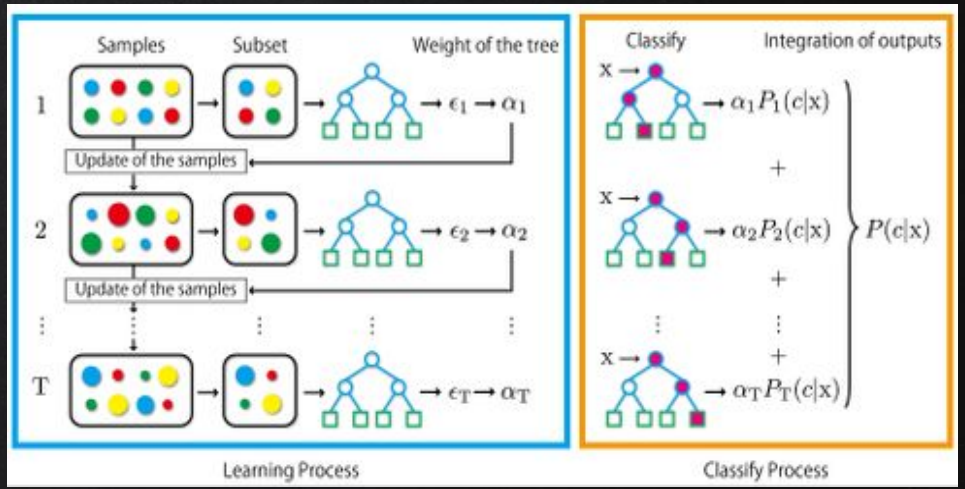
Classification

- ✘ Random Forest
  - 깊은 Decision Tree => Overfitting
  - n개의 Decision Tree 생성
  - 각 Tree 는 Training data의 일부분으로 학습
    - 무작위, 중복 허용
  - 민주적(?) 인 분류 방식
    - Tree의 다수결 선택 (majority-voting)
  - Kaggle에서 널리 쓰이는 모델



Classification

- ✘ Gradient Boosting
  - 오답노트를 작성하는 Random Forest
  - n개의 Decision Tree 를 순차적으로 생성
  - 매 Tree 는 앞 단계에서 분류에 실패한 샘플들에 가중치를 크게 두어 학습
  - 병렬 학습 불가 => 느림
  - Kaggle에서 널리 쓰이는 모델
- ✘ XgBoost 라는 전용 라이브러리가 있음



TF-IDF

- ✘ Similarity score: Term-Frequency * Inverse Document Frequency

- ✘ 전제조건
  - 지금까지 들어온 고객 질문들을 DB에 전부 저장해두고 Intent labeling을 해 놓음
  - ex)
    - 과거 질문 1: "저 투자하고싶은데 어떻게 해야 하나요?" => 'Intent 01: 투자상담'
    - 과거 질문 2: "저 대출받고싶어요ㅠㅠ" => 'Intent 02: 대출상담'
    - 과거 질문 3: "자장면 배달좀요" => 'Intent 03: 장난'

- ✘ Idea
  - 고객에게 들어오는 질문들은 다 비슷하다
  - 새 질문이 들어오면 과거 들어왔던 질문 중 가장 비슷한 질문 선택 (Intent 01 ~ Intent 03)
  - 준비해 놓은 답변을 그대로 대답하기
  - ex)
    - 현재 질문: "저 여기 투자하고 싶습니다."

TF-IDF

- ✘ Term-Frequency / Inverse Document Frequency

- ✘ 전제조건
  - 지금까지 들어온 고객 질문들을 DB에 전부 저장해두고 Intent labeling이 되어있음
  - ex)
    - 과거 질문 1: "저 투자하고싶은데 어떻게 해야 하나요?" => 'Intent 01: 투자상담'
    - 과거 질문 2: "저 대출받고싶어요ㅠㅠ" => 'Intent 02: 대출상담'
    - 과거 질문 3: "자장면 배달좀요" => 'Intent 03: 장난'

- ✘ Idea
  - 고객에게 들어오는 질문들은 다 비슷하다
  - 새 질문이 들어오면 과거 들어왔던 질문 중 가장 비슷한 질문 선택 (Intent 01 ~ Intent 03)
  - 준비해 놓은 답변을 그대로 대답하기
  - ex)
    - 현재 질문: "저 여기 투자하고 싶습니다." => 'Intent 01: 투자상담'

TF-IDF

- ✘ 어떻게 비슷한 질문을 고를까?
  - 과거 질문들과 현재 들어온 질문들을 벡터로 만들어서 Cosine similarity 계산
  - 지금 들어온 질문과 가장 비슷한 질문 = 가장 Cosine similarity가 높은 질문
  - Tip: 벡터의 크기를 1로 만들어 두면 (unit vector)
    - Dot product = Cosine similarity

- ✘ 어떻게 질문을 벡터로 만들까?
  - 벡터의 차원 = Vocabulary size
  - 벡터의 k-번째 원소 = k-번째 단어의 정보량 = 현재 질문에서의 출현 횟수 / 전체 출현 빈도
  - 출현 횟수 = term-frequency (TF)
    - 단어가 현재 질문에 많이 나타나면 커짐
    - 그런데 "ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ"는 "ㅋ"보다 10배 기분 좋은 것은 아님
      - Log-frequency weighting
  - 전체 출현 빈도 = document-frequency (IDF)
    - 아무 질문에나 나타나는 흔한 단어일수록 중요하지 않음 => 나눠줌
    - 역시 log를 취함

TF-IDF

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

$$\mathrm{tf}(\text{"this"}, d_1) = \frac{1}{5} = 0.2$$
$$\mathrm{tf}(\text{"this"}, d_2) = \frac{1}{7} \approx 0.14$$

$$\mathrm{idf}(\text{"this"}, D) = \log\left(\frac{2}{2}\right) = 0$$

$$\mathrm{tfidf}(\text{"this"}, d_1) = 0.2 \times 0 = 0$$
$$\mathrm{tfidf}(\text{"this"}, d_2) = 0.14 \times 0 = 0$$

https://en.wikipedia.org/wiki/Tf-idf

TF-IDF

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

$$\mathrm{tf}(\text{"example"}, d_1) = \frac{0}{5} = 0$$
$$\mathrm{tf}(\text{"example"}, d_2) = \frac{3}{7} \approx 0.429$$

$$\mathrm{idf}(\text{"example"}, D) = \log\left(\frac{2}{1}\right) = 0.301$$

$$\mathrm{tfidf}(\text{"example"}, d_1) = \mathrm{tf}(\text{"example"}, d_1) \times \mathrm{idf}(\text{"example"}, D) = 0 \times 0.301 = 0$$
$$\mathrm{tfidf}(\text{"example"}, d_2) = \mathrm{tf}(\text{"example"}, d_2) \times \mathrm{idf}(\text{"example"}, D) = 0.429 \times 0.301 \approx 0.13$$

https://en.wikipedia.org/wiki/Tf-idf

- Problem

query (유저 입력) 과

가장 가까운 document 찾기

(intent)

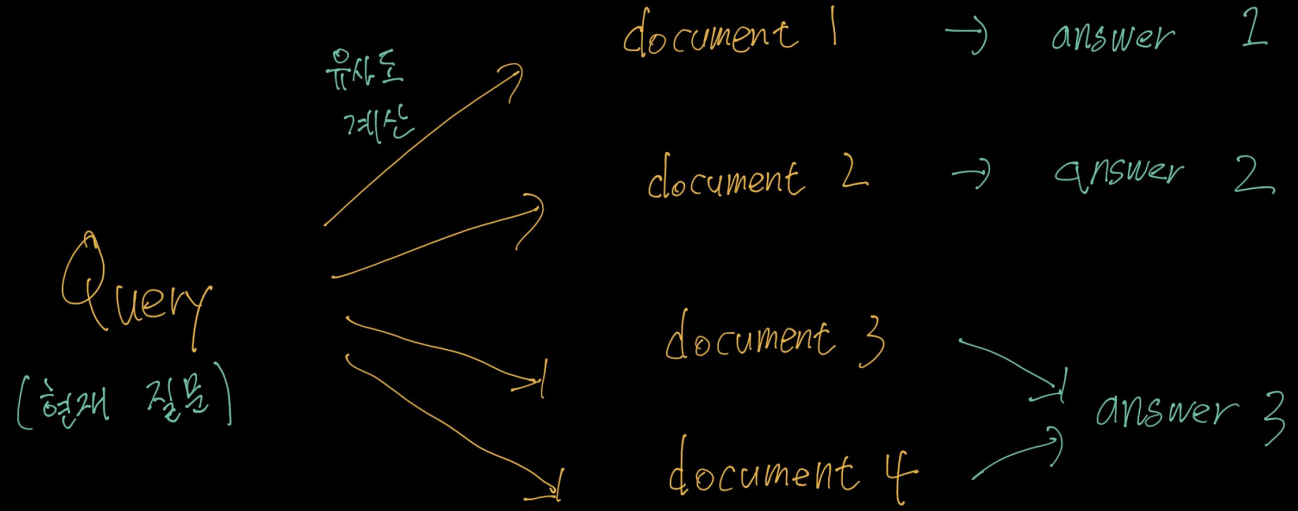
↓

cosine similarity

↓

(query vector
  document vector) 잘 만들기

Document (DB = 과거 질문)



"가장 비슷한 과거 질문을 찾아,
해당 질문에 대한 정답을 → 미리 label 되어야 할
출력하라 !!"

	document$_1$	document$_2$	document$_3$	document$_4$
word$_1$	0	1	1	0
word$_2$	1	1	1	1
word$_3$	1	0	1	1
word$_4$	0	0	1	0
word$_5$	1	0	1	1

vocabulary size $|V| = 5$ $\Rightarrow$ vector dimension $= |V| = 5$

Binary

	$document_1$	$document_2$	$document_3$	$document_4$
$word_1$	0	2	1	0
$word_2$	3	3	3	3
$word_3$	1	0	3	1
$word_4$	0	0	1	0
$word_5$	1	0	1	3

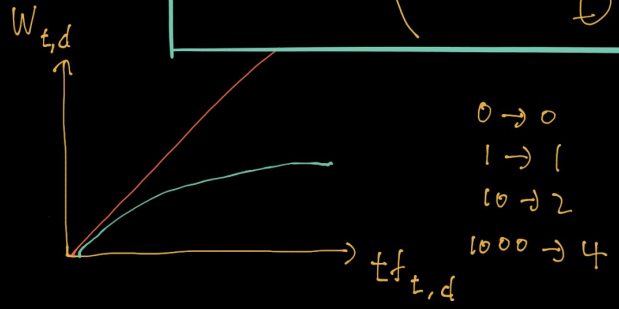
vocabulary size $|V| = 5$ $\Rightarrow$ vector dimension $= |V| = 5$

Count

$tf_{t,d}$: # of term t in document d

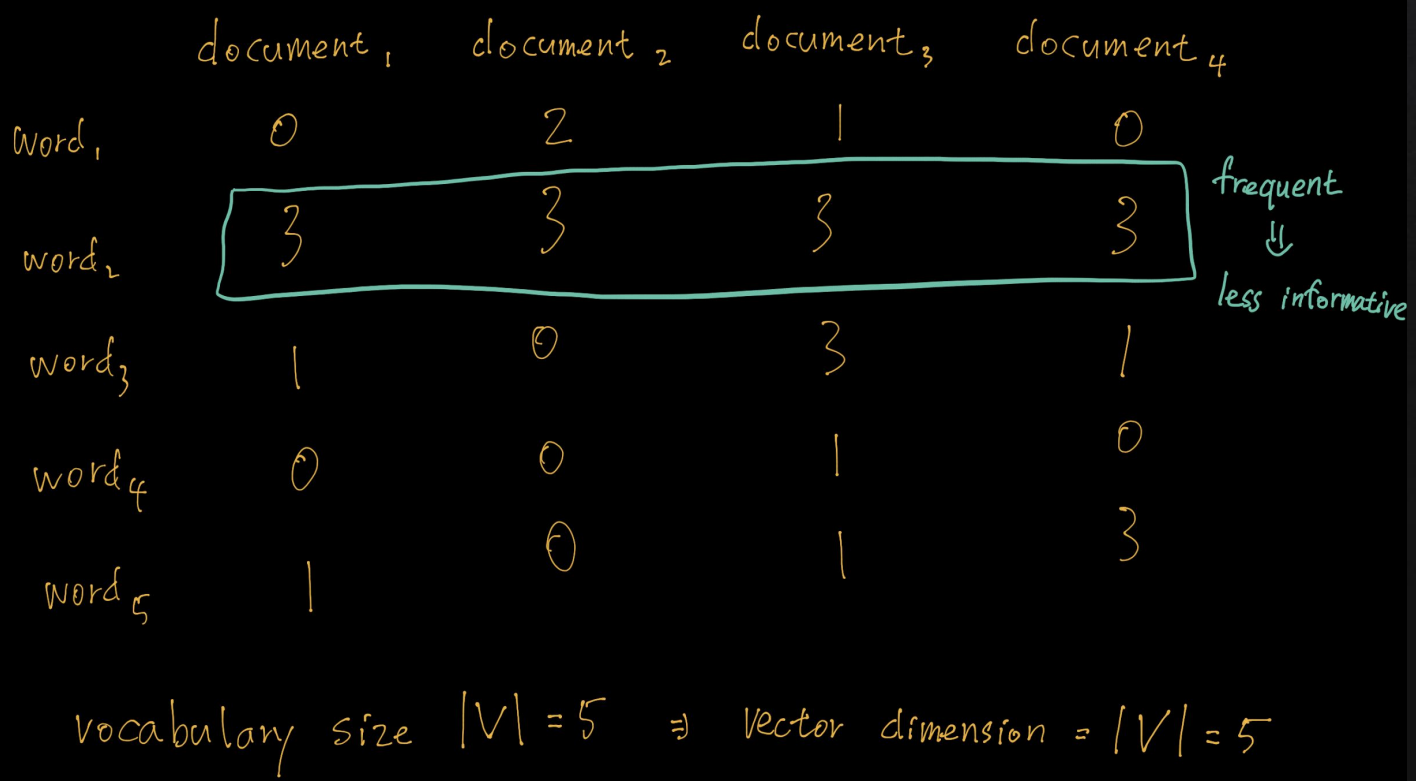
"log-frequency weighting"

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$



$0 \to 0$
$1 \to 1$
$10 \to 2$
$1000 \to 4$

$$score(q,d) = \sum_{t \in q \cap d} (1 + \log_{10} tf_{t,d})$$


document 1
document 2
document 3
document 4
word 1
0
2
1
0
word 2
3
3
3
3
word 3
1
0
3
1
word 4
0
0
1
0
word 5
1
0
1
3
frequent
less informative
vocabulary size |V| = 5 ⇒ vector dimension = |V| = 5

• Information

Rare term >>> frequent term

ex) high, increase

보험, 상품, 투자

ex) and, or, which

(stop words)

그리고, 이것, 저것

주세요, 입니다

df_t : # of documents that contain term t

idf (inverse document frequency)

$$idf_t = \log_{10}(N / df_t)$$

N = # of total documents

$N = 1\,M$

term	df_t	idf_t
보험	1	6
연도	10	5
하다	100	4
이것	1,000	3
그리고	1,000,000	0

$$w_{t,d} = (1 + \log tf_{t,d}) \times \log_{10}(N / df_t)$$

tf: term-frequency

df: document-frequency

$$Score(q,d) = \sum_{t \in q \cap d} tf.idf_{q,d}$$

TF-IDF

Example) 자동차 보험

Document : 자동차 보험 차량 보험

Query : 최고의 자동차 보험

Term	Query						Document				
	tf count	tf log	df	idf	wt	unit	tf count	tf log	×	wt	unit
차량	0	0	5000	2.3	0	0	1	1		1	0.52
최고의	1	1	50000	1.3	1.3	0.34	0	0		0	0
자동차	1	1	10000	2.0	2.0	0.52	1	1		1	0.52
보험	1	1	1000	3.0	3.0	0.78	2	1.3		1.3	0.68

tf * idf = weight　Unit vector　tf = weight　Unit vector

$(0, 0.34, 0.52, 0.78)^T (0.52, 0, 0.52, 0.68) = 0.8$

- 유사도: Query vector와 document vector의 내적
- 벡터 크기 = 1
  => 내적 = cosine similarity

TF-IDF vs CNN

- ✘ CNN을 29층이나 쌓았지만..
- ✘ (딥러닝이 아닌) n-gram TF-IDF모델이 작은 데이터에서는 성능이 더 좋습니다.
- ✘ 아래 표를 보고, 내가 이것보다는 데이터가 많은 것 같으면 딥러닝을 써보세요

Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

에러율 (낮을수록 좋음)

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

Method	n-TFIDF	n-TFIDF	n-TFIDF	ngrams	Conv	Conv+RNN	Conv	Conv
Author	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Xiao]	[Zhang]	[Zhang]
Error	7.64	2.81	1.31	4.36	37.95*	28.26	40.43*	4.93*

Answer

- ✘ 이제 고객의 의도까지 파악했으면 대답을 잘 생성해야죠.

- ✘ 역시 제대로 된 대답을 생성 하기에는 아직 **RNN**이 많이 부족하니, 아래 **2**가지 방법이 널리 사용됩니다.
  - Predefined Answer
    - 미리 저장된 대답 그대로 출력
    - **Tip:** 한 **Intent**에 **10**가지 대답을 만들어놓고 매번 랜덤하게 출력
      - 보다 다양한 대답 생성
  - Slot Filling
    - '대답 템플릿'을 미리 만들어두고 질문 및 주어진 정보로 빈칸을 채워나가기
    - **RNN Seq2Seq**으로 어느정도 풀 수 있습니다.
    - **ML +** 알맞은 **DB** 연동
    - **ex)** 음식점 추천 봇
      - 강남 유명한 스시집 추천해줘
      - '음식점 추천' **Intent**

        => Answer template : "[장소] [상호] 레스토랑을 추천드립니다."

Pipelining / Tuning

Scikit-Learn 기반의 분류 모델
Tokenizer: 정규표현식 / 형태소 분석기
Feature Extraction: TF-IDF
Classification: Logistic Regression / SVM

Time to get your hands dirty!

- ✘ 실습시간입니다!
- ✘ https://github.com/j-min/fastcampus_dl_nlp_chatbot/tree/master/Day_01

Serving

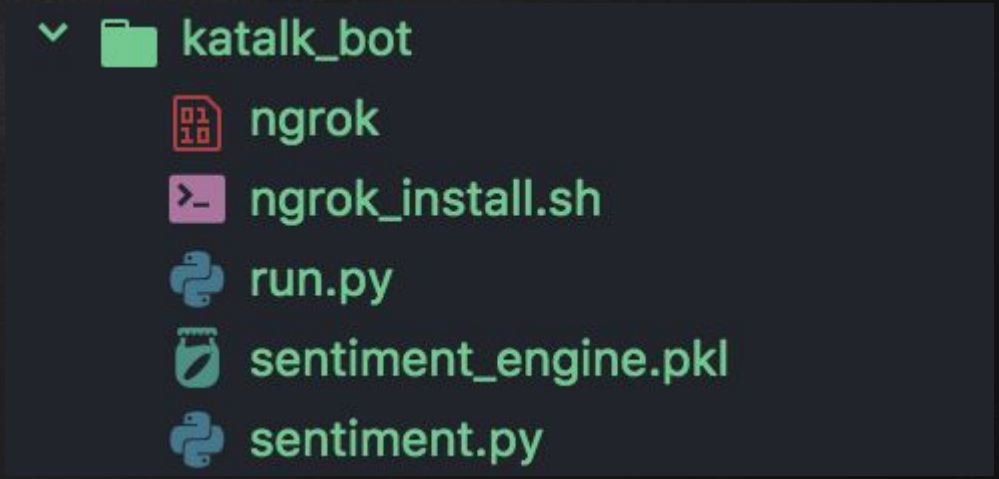
Flask 기반의 카카오톡 봇
ngrok 을 통한 http 우회

KaKaoTalk Bot

- ✘ 이제 학습시킨 모델로 긍정/부정을 판단해서 대답하는 카톡봇을 만들어 봅시다!

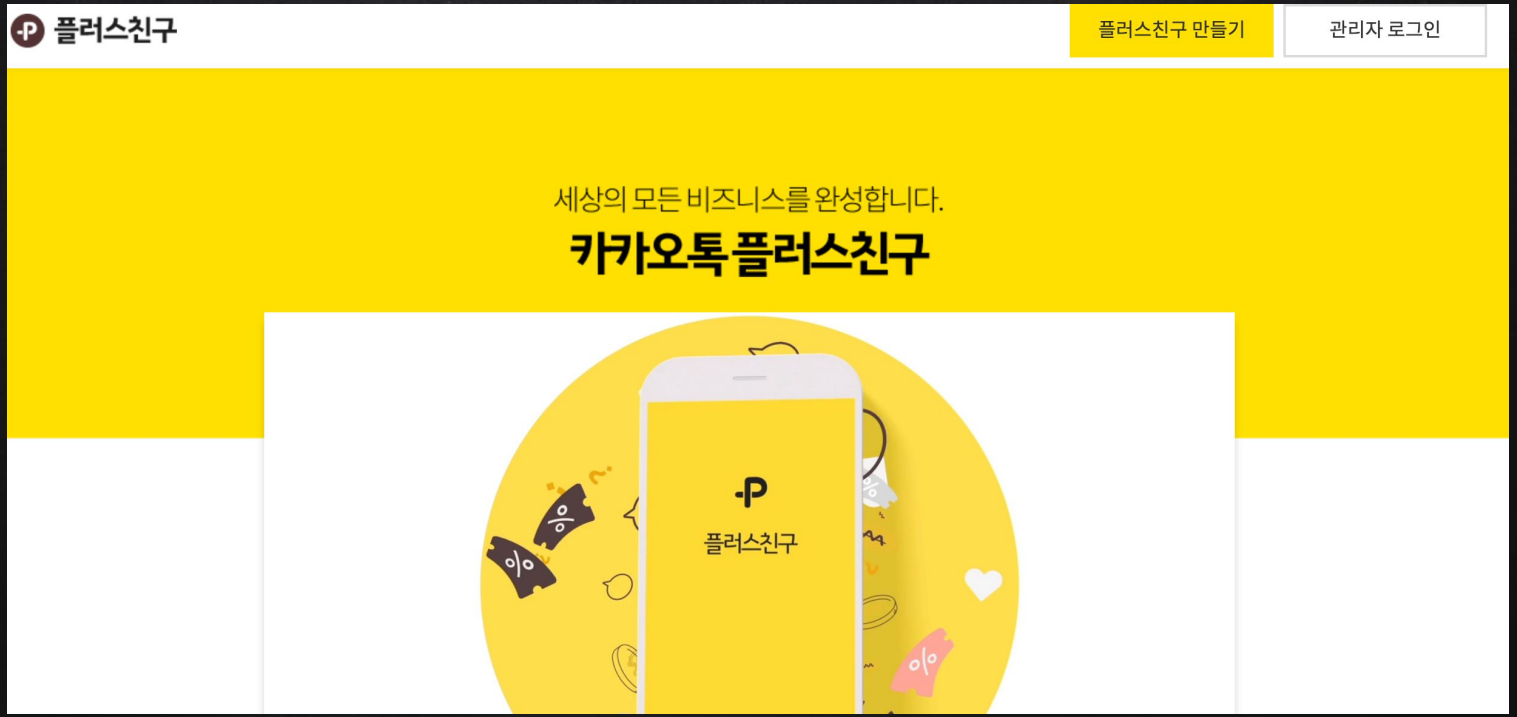
- ✘ Flask 기반

- ✘ Ngrok
  - 로컬 웹서버의 URL 생성기
- ✘ Ngrok_install.sh
  - ngrok 설치 스크립트
- ✘ Run.py
  - 카톡 봇 서버 실행 스크립트
- ✘ Sentiment_engine.pkl
  - 저장된 긍정/부정 판단 모델
- ✘ Sentiment.py
  - 긍정/부정 판단 모델 클래스



KaKaoTalk Bot

- 플러스친구 관리자 로그인
- https://center-pf.kakao.com/login



KaKaoTalk Bot

✘ 플러스친구 만들기

내 플러스친구 1

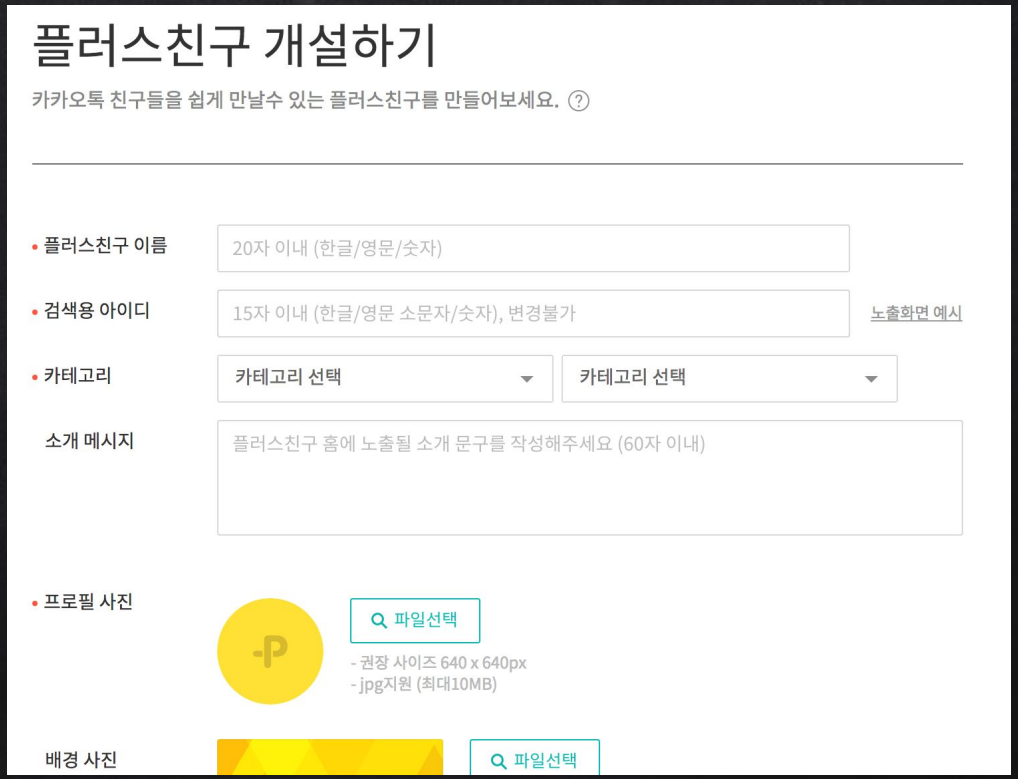
+ 새 플러스친구 만들기

flask_demo • 마스터

@flask_demo

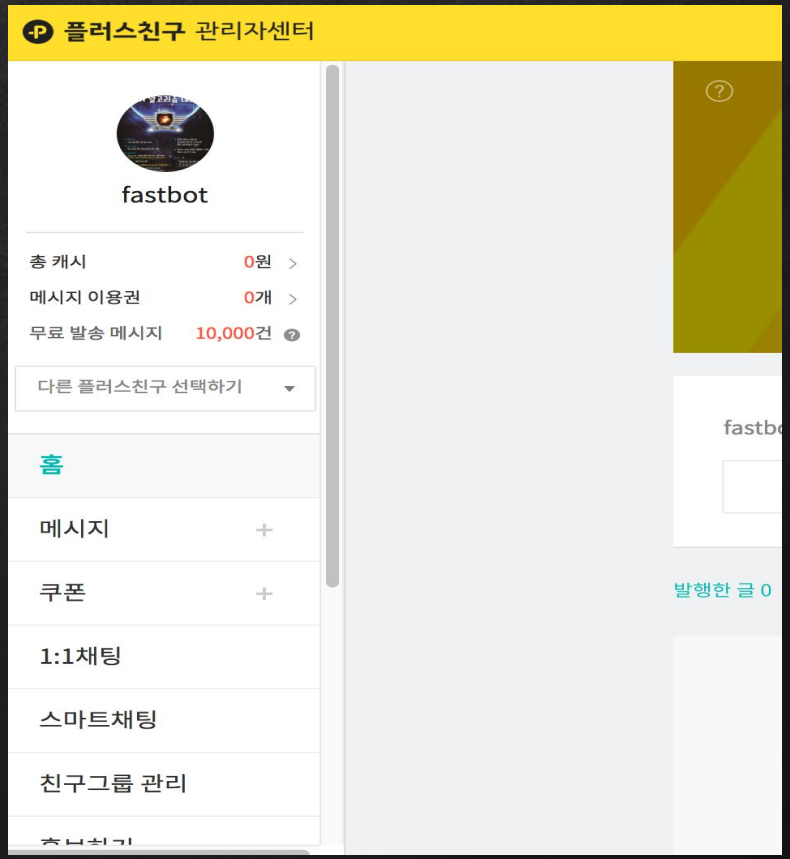
KaKaoTalk Bot

✘ 플러스친구 만들기



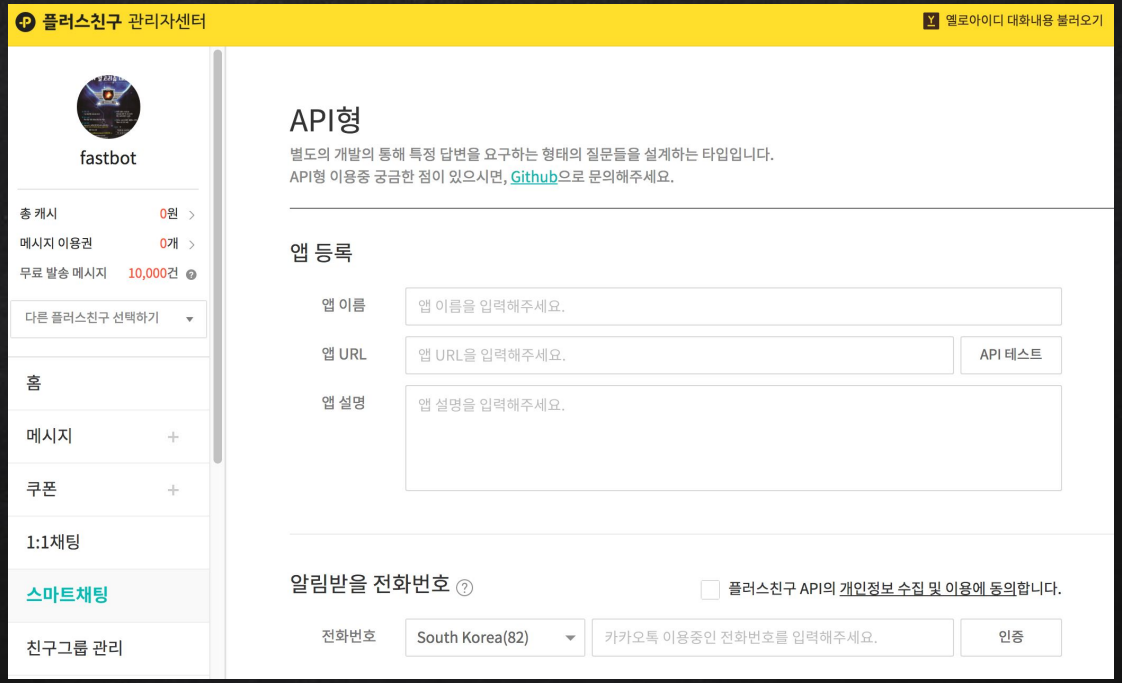
KaKaoTalk Bot

- 관리자센터-스마트채팅



KaKaoTalk Bot

- 관리자센터 => 스마트채팅 => API형



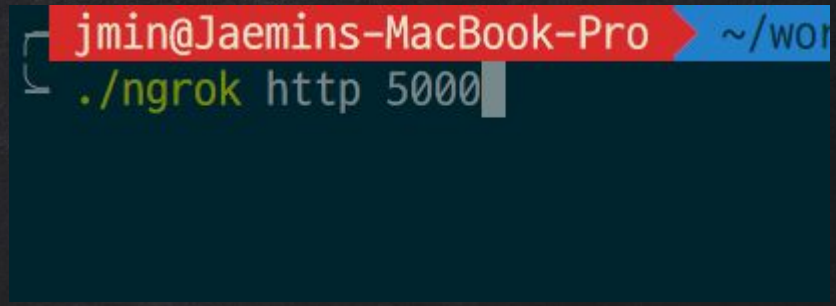
KaKaoTalk Bot

- ✘ 서버 실행
- ✘ python run.py

```
jmin@Jaemins-MacBook-Pro ~/workspace/fastcampus_chatbot/Day_01/katalk_bot
└ python run.py
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [13/Sep/2017 18:28:15] "DELETE /friend/test HTTP/1.1" 200 -
127.0.0.1 - - [13/Sep/2017 18:28:16] "POST /friend HTTP/1.1" 200 -
127.0.0.1 - - [13/Sep/2017 18:28:17] "DELETE /chat_room/test HTTP/1.1" 200 -
127.0.0.1 - - [13/Sep/2017 18:28:17] "GET /keyboard HTTP/1.1" 200 -
```

KaKaoTalk Bot

- ✘ ngrok 실행
- ✘ http://xxxx.ngrok.io 복사



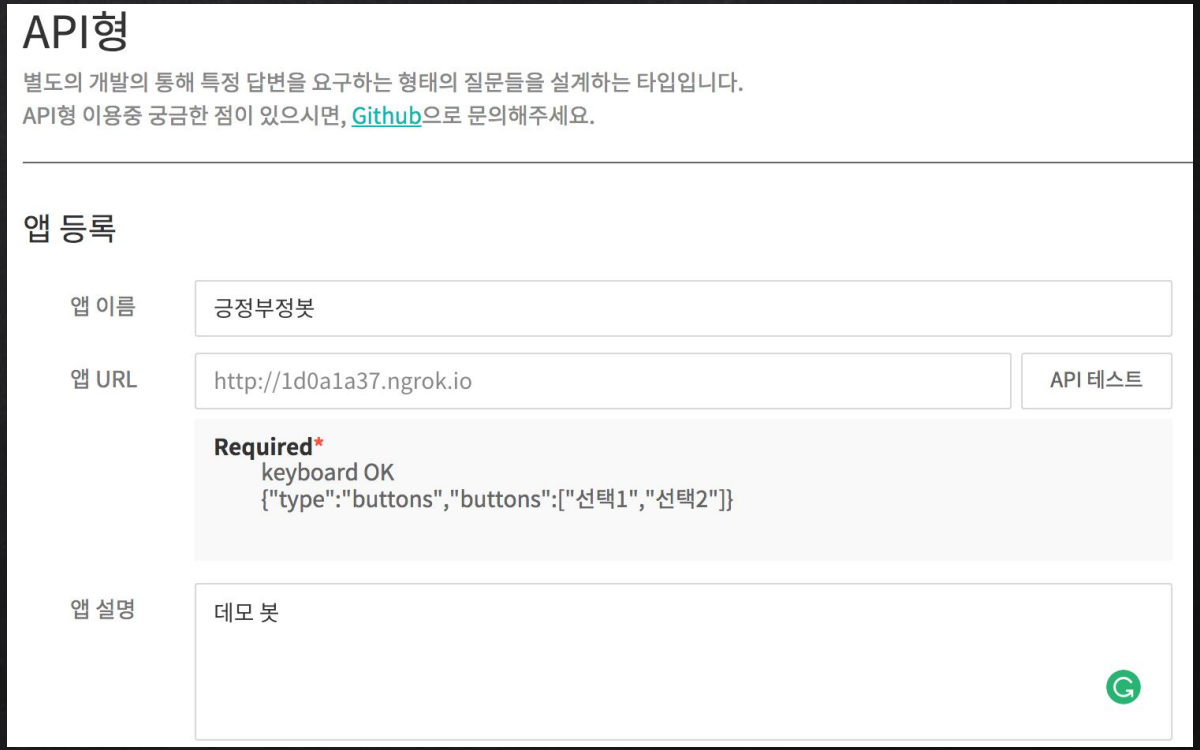

```
ngrok by @inconshreveable

Session Status                online
Version                       2.2.8
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://1d0a1a37.ngrok.io -> localhost:5000
Forwarding                    https://1d0a1a37.ngrok.io -> localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

KaKaoTalk Bot

- ✘ ngrok의 우회 url 을 앱 url로 입력하고 나머지 정보 입력
- ✘ API 테스트
- ✘ 전화번호 인증

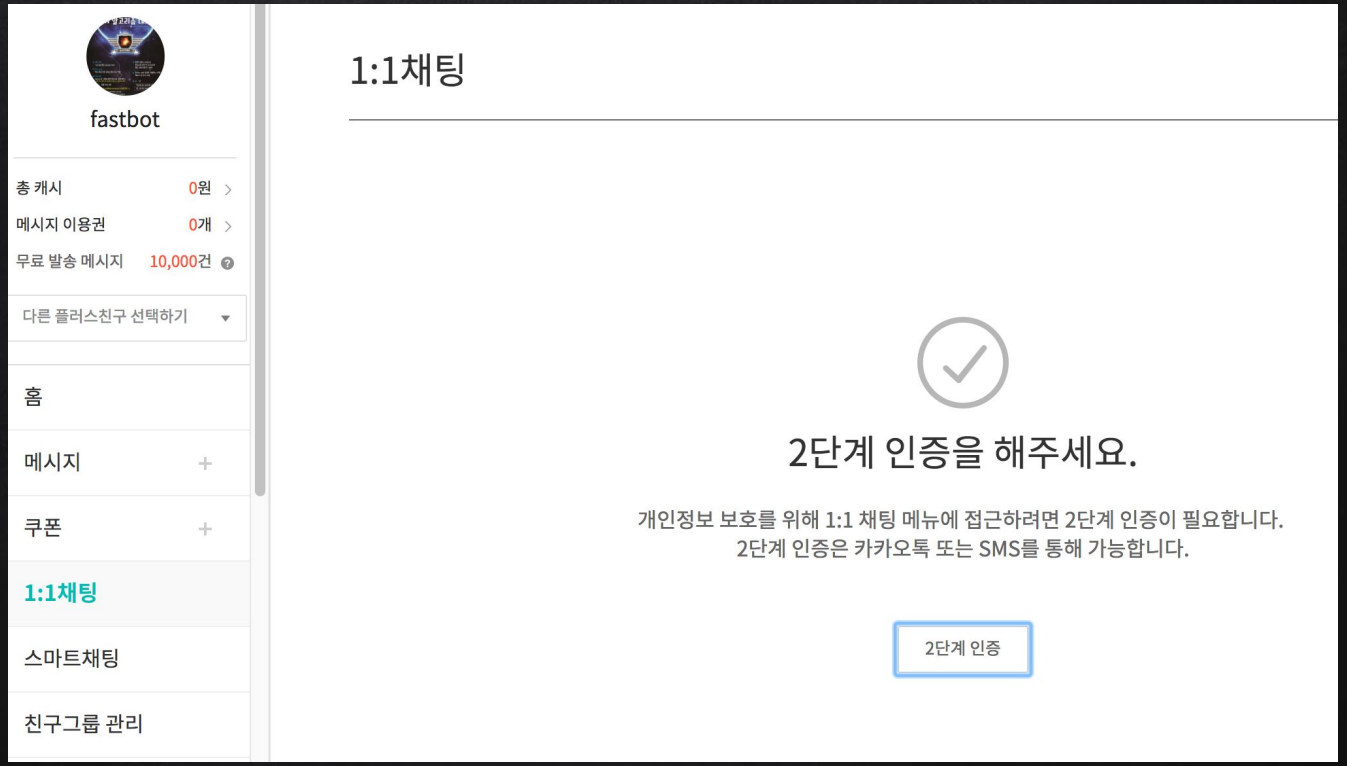


KaKaoTalk Bot

- ✘ 스마트채팅 – API형 – 시작하기

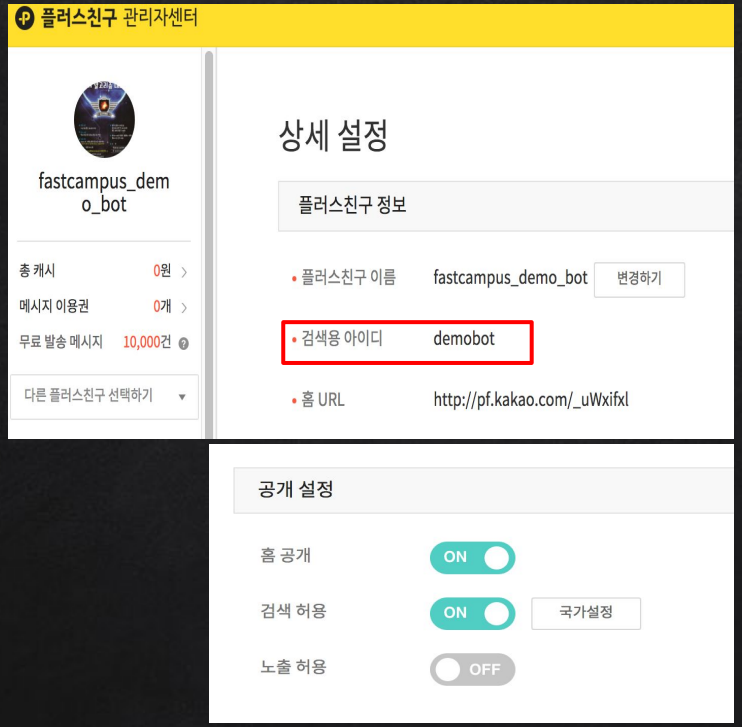
KaKaoTalk Bot

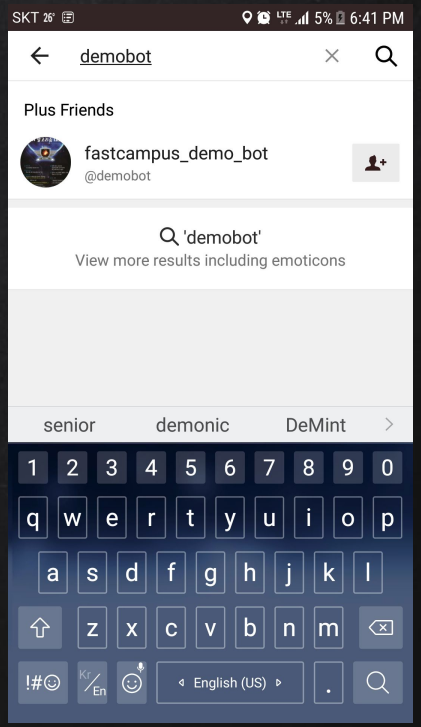
- 2단계 인증



KaKaoTalk Bot

- 플러스친구 검색 및 등록





Review

- ✘ Dataset
  - Korean: Sejong / Wiki / Namu / Naver movie sentiment
- ✘ Tokenization
  - Whitespace
  - Regular expression
  - POS-tagger => Noun / Verb only
- ✘ Vectorization
  - N-gram
  - TF-IDF
  - CBOW/Skip-gram
  - Word2Vec / Glove
  - Character embedding
  - Byte-pair encoding
- ✘ Classification
  - Tree-based models
  - Neural Networks

THANKS!

Any questions?