서울대학교 IAB 실습자료
2021 가을학기

Sentiment Analytics

Text & SNS Analysis Practice

정교민 교수님
조교 : 김용일

Preliminary

What we will do today

- Sentiment Analysis

    - Lexicon based techniques
        - Dictionary-based
        - Corpus-based
        - Manual construction

    - Machine learning(ML) based techniques
        - Naïve Bayes
        - Maximum Entropy
        - Support Vector Machine

Lexicon-based SA

- Basic procedure
  - Tokenize text (tokenizing)
  - If the token is in dictionary:
    s ← s + w for positive token
    s ← s - w for negative token (weighting)
  - Evaluate total score:
    if s > threshold, classify as positive
    if s < threshold, classify as negative (evaluation)

Lexicon-based SA

- **Example**
  - **Given a text which express a sentiment:**

    I actually found the staff to be very friendly and helpful.

  - **Tokenize text** **(tokenizing)**

    "I" "actually" "found" "the" "staff" "to" "be" "very" "friendly" "and" "helpful" "."

  - **If the token is in dictionary:** **(weighting)**

    "I" "actually" "found" "the" "staff" "to" "be" "very" "friendly" "and" "helpful" "."

    <0> **<boost><3+1=4>** **<3>**

  - **Evaluate total score:** **(evaluation)**

    0+4+3 = 7 > 0 --> "Positive"

ML-based SA

- Mostly based on supervised classification
- Two sets of documents are needed:

Training Datasets

Labelled training corpus

Feature Selection

Bag-of-words
N-gram
POS tags
Lexicon-based score

Classifier

Naïve Bayes
Maximum Entropy
Decision Tree
SVMs

Test Data

Learn Classification functions!

Hybrid : Lexicon + ML

- Pros/cons for both methods:
  - ML algorithms vary in their ability to generalize over large sets
  - Lexicons are also detrimental to some datasets
- Hybrid approach uses lexicon for initial sentiment detection
- Then use sentiment words as features in ML method
- Training data for the classifier is the result of lexicon-based method
- No manual labeling

Session 1 :
- Dictionary-based SA : SentiStrength -

Session 1. Dictionary-based SA

- 목표: **Sentistrength**에서 제공하는 **strength data**를 **dictionary**로 이용하여 **text**의 **sentimental analysis**

- **Procedure**

  - **Tokenizing/Parsing**
  - **Weighting**
  - **Adding rules**
  - **Evaluation**
  - **Test: Web**에서 추출한 **text data**에 적용.

How SentiStrength works

- 문장에 포함된 감정적 단어들의 **positive strength**와 **negative strength**의 비교
- 단어들의 **strength**는 **supervised machine learning**을 통해 미리 학습된 **-5~+5** 사이의 정수 값을 사용
- 학습된 단어들의 **strength**들을 이용하여 **dictionary-based sentimental analysis**

Learned Data for SentiStrength

- SentiStrength에서 사용하는 기 학습 데이터
  - Dictionary of SentiStrength is a mix of supervised and unsupervised classification
  - Intuitive strength detection algorithm built with rules that are made from common sense

- Notable files
  - BoosterWordList.txt : Boosters (ex. very)
  - EmoticonLookupTable.txt : Emoticons (ex. ☺)
  - IdiomLookupTable.txt : Idiom emotions
  - NegatingWordList.txt : Negations (ex. not)
  - SentimentLookupTable.txt : Emotion values

- Source: http://sentistrength.wlv.ac.uk/SentStrength_Data/

Data

1) Rule-related files used for SentiStrength
   : stored in *SentiStrength_Data* folder

2) Text data for testing
   : stored in *6humanCodedDataSets* folder

- Twitter, MySpace, YouTube, BBC forum, Runners world, Digg
- Human evaluation 포함
- Format
  - Mean positive value [1, 5]
  - Mean negative value [1, 5]
  - Sentences
- Source: http://sentistrength.wlv.ac.uk/documentation/

Environment

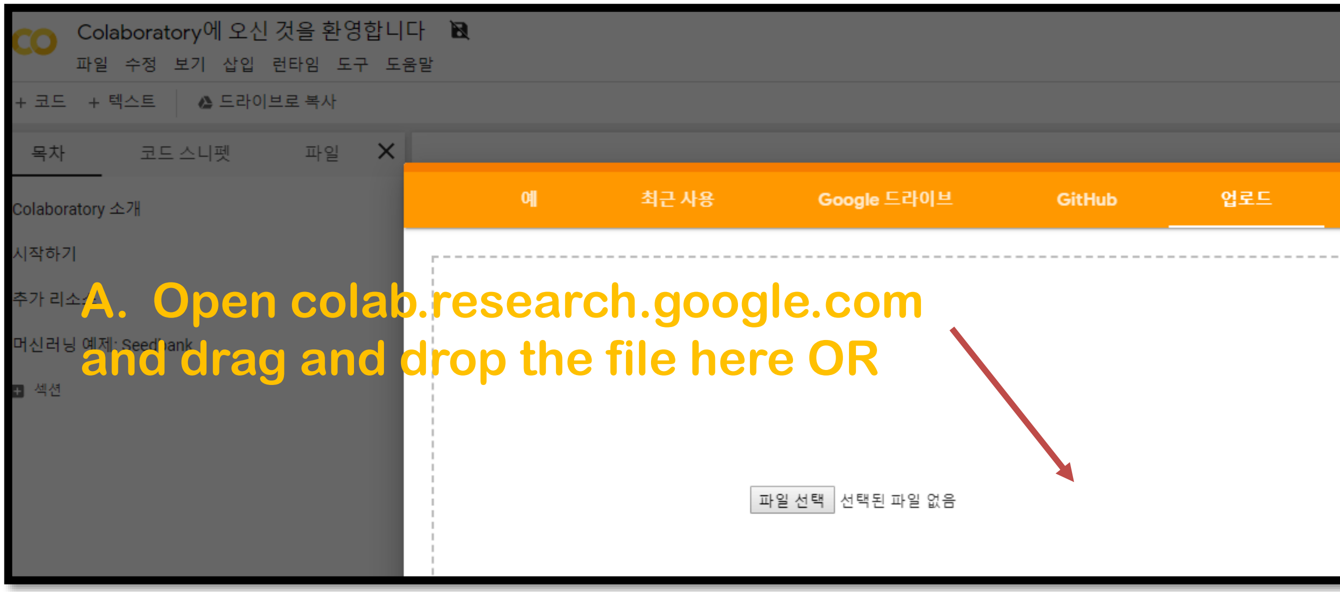
- Download the Jupyter Notebook file from eTL
  - `session1_Dictionary_SA.ipynb`
  - Full code will be uploaded shortly

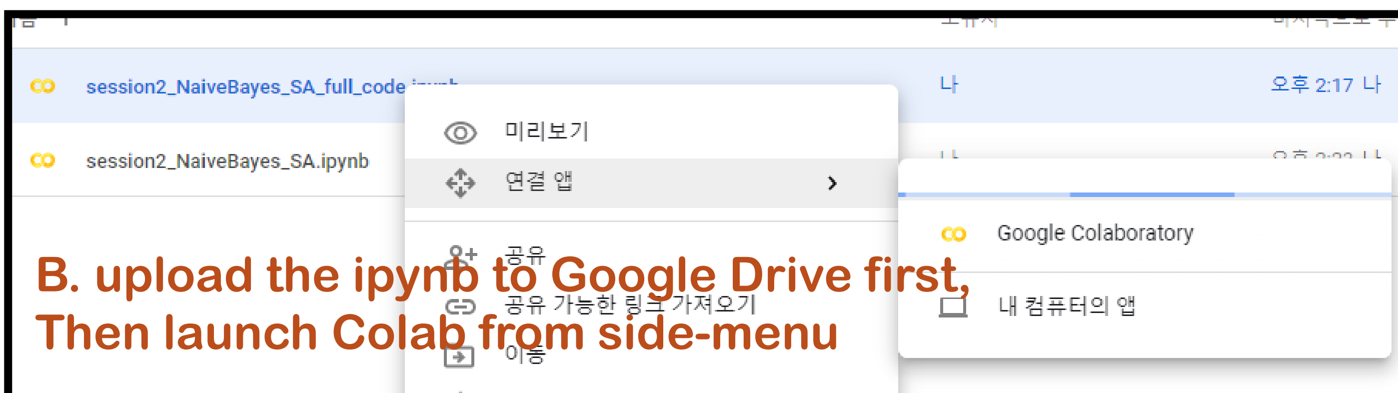
- Go to the Google Colab
  - https://colab.research.google.com/

- Sign in using your Google account
  - Colab is freely available for Google Users

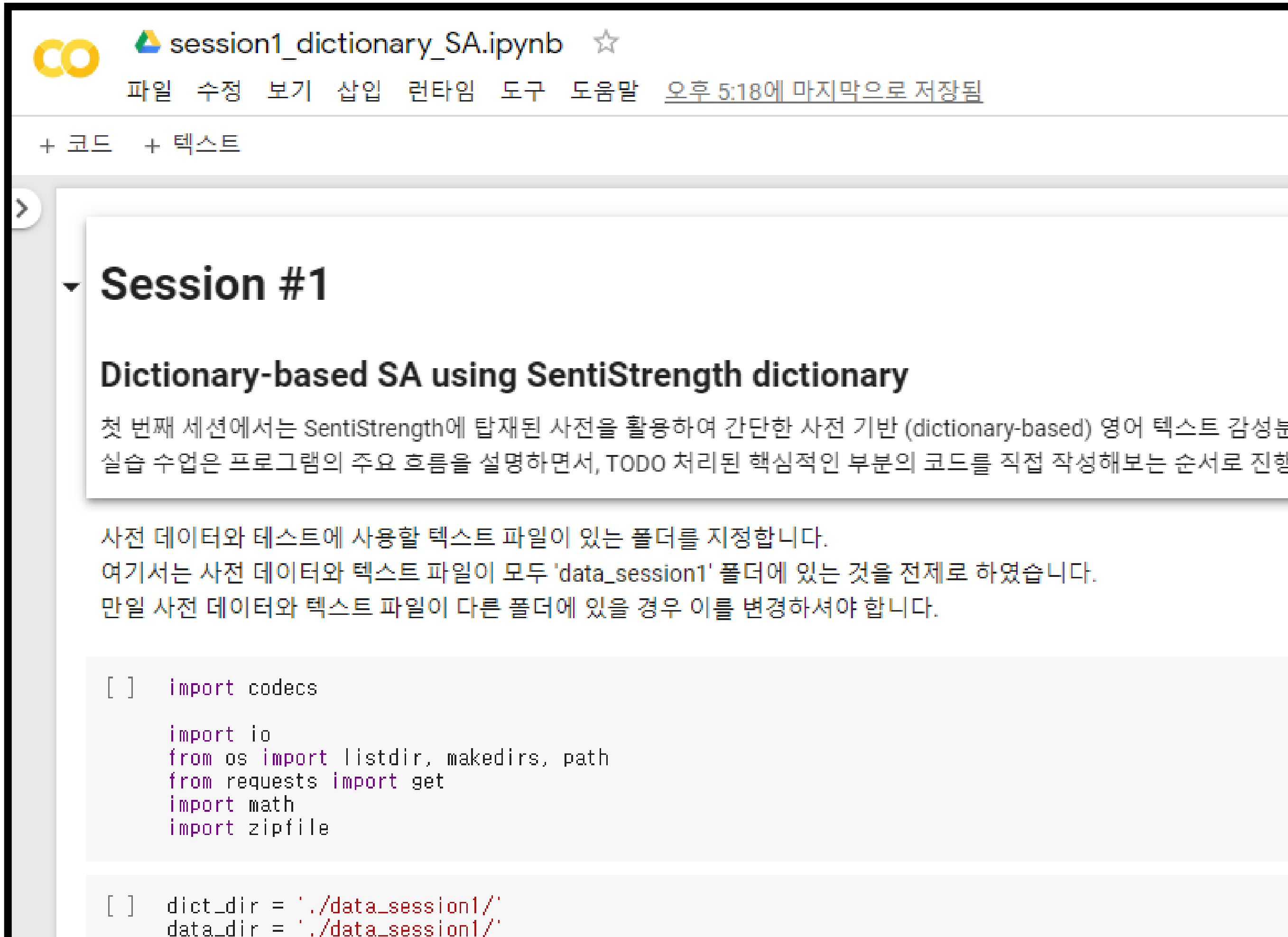
- Start a new session using the file
  - From Google Colab site
  - Or upload it to Google Drive, then launch

Launching Google Colab





Launching Google Colab



session1_dictionary_SA.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 오후 5:18에 마지막으로 저장됨

+ 코드 + 텍스트

Session #1

Dictionary-based SA using SentiStrength dictionary

첫 번째 세션에서는 SentiStrength에 탑재된 사전을 활용하여 간단한 사전 기반 (dictionary-based) 영어 텍스트 감성분
실습 수업은 프로그램의 주요 흐름을 설명하면서, TODO 처리된 핵심적인 부분의 코드를 직접 작성해보는 순서로 진행

사전 데이터와 테스트에 사용할 텍스트 파일이 있는 폴더를 지정합니다.
여기서는 사전 데이터와 텍스트 파일이 모두 'data_session1' 폴더에 있는 것을 전제로 하였습니다.
만일 사전 데이터와 텍스트 파일이 다른 폴더에 있을 경우 이를 변경하셔야 합니다.

```
[ ]  import codecs

     import io
     from os import listdir, makedirs, path
     from requests import get
     import math
     import zipfile
```

```
[ ]  dict_dir = './data_session1/'
     data_dir = './data_session1/'
```

1. Tokenizing/Parsing

- Tokenizing is a preprocessing task, which cuts a certain sentence down to a series of several terms
- Tokening strategy may differ and can be advanced to match the objective
- Punctuation, normalizing, capital letters

“split” function

split()

```
>>> a=“life is too short”
>>> a.split(“ ”)
[‘life’, ‘is’, ‘too’, ‘short’]
```

주요 함수들

- def parse_input(str)

Parse a given sentence to a list
input: sentence string
output: list of words

<할일>
1) 문장부호 없애기
2) 소문자로 변환
3) .split() 사용하여 list로 저장하기

- def parse_weight(filename)

Parse rule-related files to a dictionary
input: file name
output: dictionary of weights

<할일>
1) 파일 읽어들이기
2) 파일의 각 라인의 단어를 key로, weight값을 value로 하는 딕셔너리 생성하기

주요 함수들

- def parse_negate(filename)

\# Parse rule-related files to a dictionary
\# input: file name
\# output: dictionary of weights

<할일>
1) 파일 읽어들이기
2) 파일의 각 라인의 단어를 **key**로, **0**을 **value**로 하는 딕셔너리 생성하기

기타 고려사항

- 문장 기호 **(?!.,)** 구분

- **Emoticons**

  ex)   :) :o) ^.^ D: D-: :(:| =|

  >> EmoticonLookupTable.txt

- 여러 문장 기호**,** 알파벳**,** 숫자들의 조합으로 이루어진 것들을 따로 찾아서 구분

2. Weighting

- 미리 학습된 **dictionary**에는 단어별로 **positive/negative strength (weight)**가 부여되어 있음.
- **Parsing**한 단어들의 **weight**를 주어진 모델의 **dictionary**에서 찾는 과정.
- **I, word**와 같은 감정이 없는 일반적 단어는 **weight = 1**

```
ache*    -2
achen*   1    kev
acher*   1    kev
acheson  1    kev
acheta   1    kev
aching   -2   removed active* 2
acrimon*      -2   General Inquirer Feb 2010
addict*  -2   General Inquirer Feb 2010
admir*   3
admiral*      1    kev
admoni*  -2   General Inquirer Feb 2010
ador*    4    advantag*    2
adorn*   1    kev
adroit   2    Hannes GI add
adulterat*    -2   General Inquirer Feb 2010
adulteror     -3   kev add
adultery      -3   General Inquirer Feb 2010
adventur*     1
advers*  -2
affab*   3    Hannes GI add
affectation*       -2   General Inquirer Feb 2010
affection*    3
afflict*      -3   General Inquirer Feb 2010
afraid   -4
```

EmotionLookupTable.txt
※ ador* 4 : every string starting with ador has 4
ex) adorable, adoring

주요 함수들

def weight_default(list_words, dic_weights):

Put weights on words
Input: list of words, dictionary of weights
Output: list of (word, weight) pair

<할일>
1) 데이터 속 단어가 딕셔너리에 있을 경우
2) 딕셔너리에 없을 경우, 접두어+ '*' 의 구성에 해당되는지 확인.
3) 그마저도 없을 경우, weight를 0으로 주기.
4) 최종적으로 word와 weight의 쌍으로 이루어진 리스트 생성하기.

3. Evaluation

- **max. positive strength vs. max. negative strength**

- 문장의 최대 **positive strength**와 최대 **negative strength**를 비교.

- 비교 결과에 따라 **positive/neutral/negative**를 평가.

주요 함수들

```
def extract_max(list_pairs):

    ...

        return (pos_max, neg_max)
```

<할일>
1) 주어진 쌍들 중 최대 **pos**값 추출.
2) 주어진 쌍들 중 최대 **neg**값 추출.

 # Extract maximum weights for given (word, weight) pairs
 # Input: list of (word, weight)
 # Output: (positive, negative)

* Test(I) *

Input

>> I am so bored and tired.

Result

>> I am so bored[-2] and tired[-2].

[sentence: 1,-2, negative]

Input

>> I really love you but dislike your cold sister.

Result

I really love[3] you but dislike[-3] your cold[-2] sister.

[sentence: 3,-3, neutral]

4. Adding Rules

<u>Boosting words</u>

- 뒷 단어의 weighting 강화/약화

ex) good : +2
really good : +3
somtimes good : +1
hate : -4
really hate : -5

>>BoosterWordList.txt

ex)

some	-1	
sometimes	-1	9 June 2010
sum	-1	
total	1	9 June 2010
totally	1	9 June 2010
very	1	
would	-1	9 June 2010

4. Adding Rules

Negating words

- 뒤에 오는 **sentiment word**의 **strength**를 반전시키는 단어들
- **aren't, not, never** 등등

Positive → ×(-0.5)

Negative → ×0

- ex) don't like : +2 → -1
- isn't harm : -3 → 0

>> NegatingWordList.txt

주요 함수들

```
def weight_boost(list_pairs, dic_boost):
# Give boosting to another word
# Input: list of (word, weight)
# Output: list of (word, boosted weight)
```

<할일>
1) 입력된 단어가 **boost** 딕셔너리에 있는지 확인
2) 만약 있다면 **boost** 값을 저장하고 없다면 **boost**를 **0**으로 줌.
3) **Boost**값이 **0**이 아닌 경우,뒤에 오는 단어가 양의 **weight** 값을 가지면 **boost** 값을 더해주고,   음의 **weight** 값을 가지면 **boost** 값을 빼주기.

주요 함수들

def weight_negate(list_pairs, dic_negate):

Negate the weight for a word

Input: list of (word, weight)

Output: list of (word, negated weight)

<할일>

1) 입력된 단어가 **negate** 딕셔너리에 있는지 확인
2) 만약 있다면 **negate**값을 **1**로 저장.
3) 뒤에 오는 단어의 **weight**가 양의 값일때는 ***(-1/2)**, 음의 값일때는 **0**으로 바꿔 주기.
4) **negate**값을 다시 **0**으로 저장.

* Testing(II) *

Input

>> I really love you but dislike your cold sister.

Result

>> I really love[3] [+1 booster word] you but dislike[-3] your cold[-2] siste
r .[sentence: 4,-3, positive]

Input

>> I do not hate him.

Result

>> I do not hate[-4] [=0 negation] him. [sentence: 1,-1, neutral]

Input

>> but I dont love the spring in Macau.

Result

>> but I dont love[3] [*-0.5 approx. negated multiplier] the spring in Maca
u. [sentence: 1,-2, negative]

* Testing(II) *

사람이 매긴 **positive/negative** 점수와의 비교

1. 각각의 **text**에 대하여 사람의 평가와 프로그램 결과 비교

2. **Twitter**와 **BBC forum**의 전체적인 감성분석 결과 비교

Session 2 :
- ML based SA : Using Naïve Bayes -

Session 2. ML-based SA

- 목표: **Naïve Bayes Text Classifier** 를 구현하여 **text** 의 **sentimental analysis**

- **Procedure**
  1. **Compute log priors P(c) for each class (positive / negative)**
  2. **Compute log likelihood P(w|c) for each morpheme w given class c**
     - **Tokenizing = using "word_tokenize" in NLTK library**
     - **Apply add-one smoothing**
  3. **Evaluate movie review**

Naïve Bayes Classifier

- **Class prediction**

  Probability that given document d is in class c

  - $\hat{c} = arg\max_{c \in C} \boxed{P(c|d)} = arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$,
    where C is class and d is document

- **Each document can be represented as a bag-of-words**

  Probability that a bag of words $(w_1, w_2, \ldots, w_N)$ is observed in documents of class c

  - $\boxed{\mathrm{P(d|c)} = P(w_1, w_2, \ldots, w_N|c),}$
    where w_i *is a morpheme (or token) in the document*

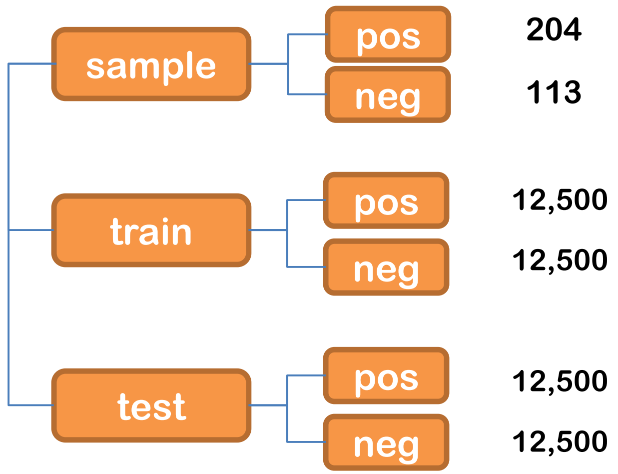
- **Naïve Bayes assumption (independence of features)**

  - $\mathrm{P(d|c)} = P(w_1, w_2, \ldots, w_N|c) = \mathrm{P}(w_1|c) \cdot \mathrm{P}(w_2|c) \cdot \cdots \cdot \mathrm{P}(w_N|c)$

    "Each words are independent on each other"

Data

- Large Movie Review dataset
  - http://ai.stanford.edu/~amaas/data/sentiment/
- 25,000 highly polar movie reviews for training, and 25,000 for testing
  - positive 50%, negative 50%
  - each file contains a one review



Environment

- Download the Jupyter Notebook file from eTL
  - `session2_NaiveBayes_SA.ipynb`
  - Full code will be uploaded shortly
- Go to the Google Colab
  - https://colab.research.google.com/
- Sign in using your Google account
  - Colab is freely available for Google Users
- Start a new session using the file
  - From Google Colab site
  - Or upload it to Google Drive, then launch

Compute prior

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- Count # of document for each class (pos, neg)
  - Count files on each class (each document is stored in a file)
- Apply logarithm
  - Ease of computation - multiplication / division on probabilities now become addition / negation
  - Improves numerical stability (prob.s are very small!)

code

$$\hat{c} = \underset{c \in C}{arg\max} P(c|d) = \underset{c \in C}{arg\max} \frac{P(d|c)P(c)}{P(d)}$$

1. compute prior
```
log_prior_pos = math.log(count_file(base_dir_for_train + '\\pos'))
log_prior_neg = math.log(count_file(base_dir_for_train + '\\neg'))
```

```
# dir_path 에 있는 파일의 개수를 count

def count_file(dir_path):

   --- To Do ---
   list_files = …
   ----------------

    return len(list_files )
```

- 폴더 내의 모든 파일을 파악해야 함

Compute likelihoods P(v|c)

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- $\mathrm{P}(\mathrm{d}|\mathrm{c}) = P(w_1, w_2, \ldots, w_N|c) = \mathrm{P}(w_1|c) \cdot \mathrm{P}(w_2|c) \cdot \cdots \cdot \mathrm{P}(w_N|c)$

- Create dictionary
  - process whole document
  - apply tokenizer
  - put new token into dictionary

- Compute likelihood for each token in class (pos/neg)
  - $\mathrm{P}(v_i|c) = \dfrac{\#\, of\ v_i\ in\ class\ c}{\sum_i \#\, of\ v_i\ in\ class\ c}$

- Apply log

code

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

```
# 2-1 전체 vocabulary dictionary 생성
build_dic(base_dir_for_train + '\\pos\\', voca_dic)
build_dic(base_dir_for_train + '\\neg\\', voca_dic)


# dir_path 에 있는 파일들을 읽어서 tokenize 후, vocabulary dictionary 생성

def build_dic(dir_path, dic):


   --- To Do ---

   dir_path 내에 있는 파일들을 loop

      하나의 파일에 있는 내용을 읽은 후 tokenize
      ex) tokens = word_tokenize(line.strip().lower())

      tokens 에 담겨 있는 token 들을 dictionary 에 추가
```

- 폴더 내의 모든 파일을 하나 씩 읽음
- 파일의 텍스트를 단어 단위로 분할 (tokenize)
- 단어를 dictionary에 추가

code

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

```
# 2-2 각 class 의 token 을 count 한 dictionary 생성
pos_dic = create_class_dic(base_dir_for_train + '\\pos\\', voca_dic)
neg_dic = create_class_dic(base_dir_for_train + '\\neg\\', voca_dic)

def create_class_dic(dir_path, base_dic):

    # base_dic 을 복사하여 새로운 likelihood_table 생성  (모든 count 는 1로 initialize)
    likelihood_table = {}
    likelihood_table = dict( (nkey, 1) for nkey in [key for key in base_dic.keys()])

    list_files = [f for f in listdir(dir_path) if isfile(join(dir_path, f))]

    for file in list_files:

        try:

            f = open(dir_path + file, 'r')
            line = f.readline()
            tokens = word_tokenize(line.strip().lower())

            for token in tokens:

                # To Do
                # ……

            f.close()

        except:
            pass

    return likelihood_table
```

- 2-1과 동일하나, 단어의 유무 및 단어가 나오는 횟수까지 기록해야 함

code

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

```
# 2-3 table 값을 log probability 값으로 변환
log_likelihood_pos = compute_log_likelihood_table(pos_dic)
log_likelihood_neg = compute_log_likelihood_table(neg_dic)
```

```
def compute_log_likelihood_table(class_dic):

    new_table = {}
    word_sum = sum(class_dic.values())

    -- ToDO ----
    new_table =
    ----------------

    return new_table
```

- $\mathrm{P}(d|c) = \mathrm{P}(v_i|c) = \frac{\#\ of\ v_i\ in\ class\ c}{\sum_i \#\ of\ v_i\ in\ class\ c}$
- $\mathrm{P}(v_i|c)$ 의 분자 부분은 **class**에서 각 단어가 나타난 횟수
- 분모 부분은 **class** 내에서 사용된 모든 단어에 대한 총합
- 분모 부분은 **1**번만 계산 가능, 분자는 각 단어 별로 계산하게 반복문 작성 필요

Evaluation

$$\hat{c} = arg \max_{c \in C} P(c|d) = arg \max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- $\mathrm{P}(\mathrm{d}|\mathrm{c}) = P(v_1, v_2, \ldots, v_N|c) = \mathrm{P}(v_1|c) \cdot \mathrm{P}(v_2|c) \cdot \cdots \cdot \mathrm{P}(v_N|c)$
  - $\log(\mathrm{P}(\mathrm{d}|\mathrm{c})) = \sum_N \log(\mathrm{P}(v_i|c))$

- **With the log likelihood table and log prior information, let's classify a document**

code

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

```
# 3. 특정 문장 classify 수행
document = 'this is  my favorite. sooo exciting'
ret = classify_doc(document, log_prior_pos, log_prior_neg, log_likelihood_pos, log_likelihood_neg)
print ret


def classify_doc(document, log_prior_pos, log_prior_neg, log_likelihood_pos, log_likelihood_neg):

   pos_prob = 0
   neg_prob = 0

   tokens = word_tokenize(document.strip().lower())

   for token in tokens:

      -- ToDO -----------------------
      pos_prob 에 주어진 토큰의 해당 클래스에 따른 확률 값을 누적
      neg_prob 에 주어진 토큰의 해당 클래스에 따른 확률 값을 누적
     -----------------------------------

   pos_prob = pos_prob + log_prior_pos
   neg_prob = neg_prob + log_prior_neg

   if pos_prob > neg_prob:
      return 'positive'
   else:
      return 'negative'
```

- 텍스트가 **document** 로 들어오면 이것을 먼저 단어 단위로 분할
- 분할된 단어를 가지고 모든 **class** 에 대해 $P(d|c)P(c)$ 를 구함
- $P(d|c)P(c)$ 가 높은 쪽으로 이 **document** 의 **class** 를 예측

summary

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

```
copy_dir = 'data'
base_dir_for_train = copy_dir + '\\sample'

# 1. compute prior
log_prior_pos = math.log(count_file(base_dir_for_train + '\\pos'))
log_prior_neg = math.log(count_file(base_dir_for_train + '\\neg'))

#  2. compute likelihood
#  2-1 create vocabulary dictionary

build_dic(base_dir_for_train + '\\pos\\', voca_dic)
build_dic(base_dir_for_train + '\\neg\\', voca_dic)

#  2-2 create dictionary counting class token
pos_dic = create_class_dic(base_dir_for_train + '\\pos\\', voca_dic)
neg_dic = create_class_dic(base_dir_for_train + '\\neg\\', voca_dic)

#  2-3 apply log to table value
log_likelihood_pos = compute_log_likelihood_table(pos_dic)
log_likelihood_neg = compute_log_likelihood_table(neg_dic)

# 3. run classification
document = 'this is  my favorite. sooo exciting'
ret = classify_doc(document, log_prior_pos, log_prior_neg, log_likelihood_pos, log_likelihood_neg)
```

```
# Variables

# token dictionary
voca_dic = {}

# prior probability for positive class
log_prior_pos = 1

# prior probability for negative class
log_prior_neg = 1

log_likelihood_pos = {}
log_likelihood_neg = {}
```

Accuracy calculation

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- $\mathrm{P}(\mathrm{d}|\mathrm{c}) = P(v_1, v_2, \ldots, v_N|c) = \mathrm{P}(v_1|c) \cdot \mathrm{P}(v_2|c) \cdot \cdots \cdot \mathrm{P}(v_N|c)$

- **positive/negative data** 들이 들어 있는 폴더를 모두 **classify** 한 후 **accuracy** 측정

```
evaluate_all(copy_dir + '\\test\\neg\\')
evaluate_all(copy_dir + '\\test\\pos\\')
```

In-sample error / out-of-sample error

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- $\mathrm{P(d|c)} = P(v_1, v_2, \dots, v_N|c) = \mathrm{P}(v_1|c) \cdot \mathrm{P}(v_2|c) \cdot \cdots \cdot \mathrm{P}(v_N|c)$

- In-sample error
  - Sample/training data 로 training
    - Sample/training data evaluation

- Out-of-sample error
  - Training data 로 training
    - Test data 로 evaluation