데이터 청년 캠퍼스

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

캐글 실습 -2

2021년 7월

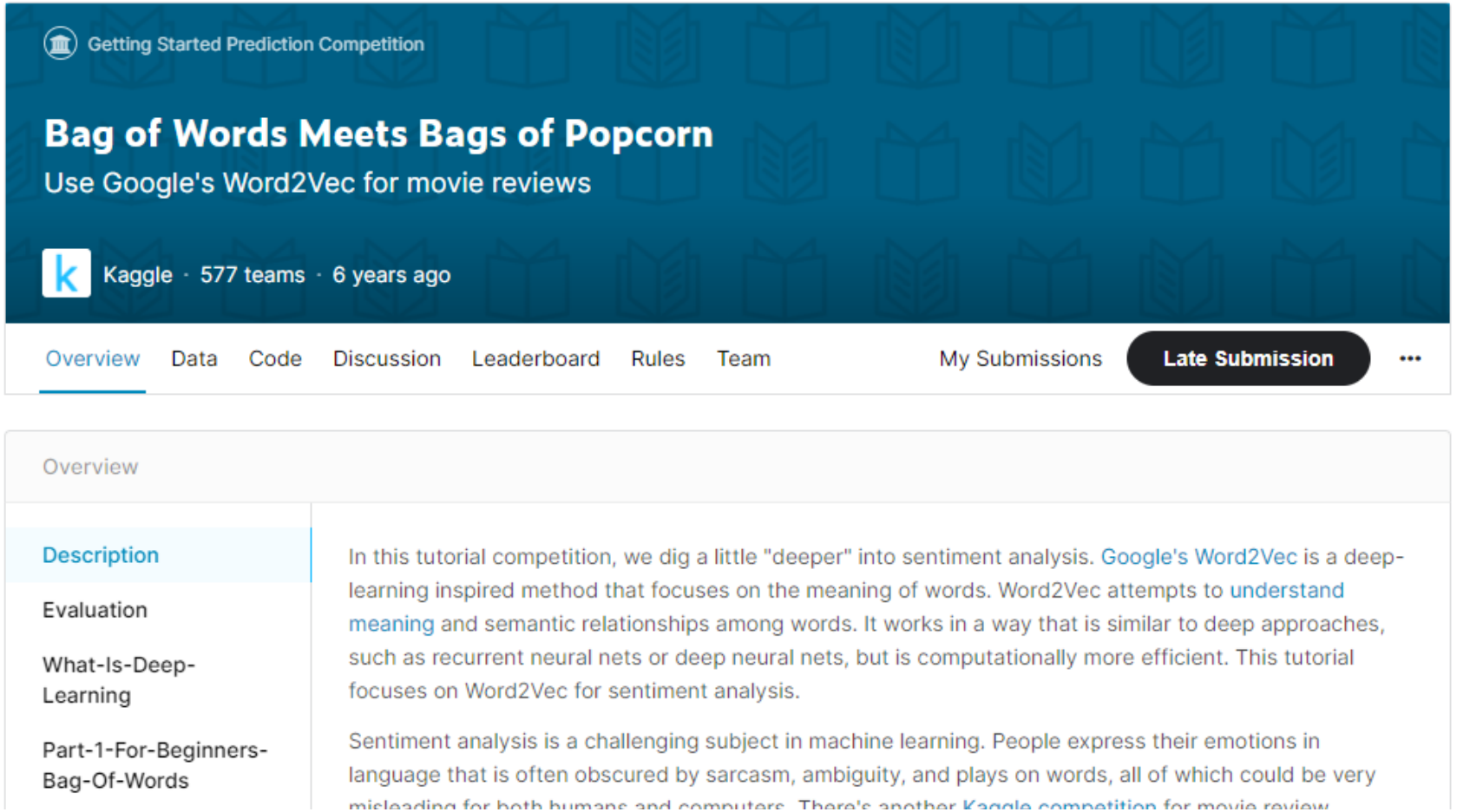
박 민 규

Today

1. Bag of Words Meets Bags of Popcorn
   - 데이터 분석 및 전처리
   - Logistic Regression Example with TF-IDF
   - Linear Regression with Word2Vec
   - Random Forest
2. 네이버 영화 리뷰 감정 분석

워드 팝콘

- https://www.kaggle.com/c/word2vec-nlp-tutorial/
  - IMDB 영화평점에 따른 감성분석 문제



평가방법

Overview

- Description
- Evaluation
- What-Is-Deep-Learning
- Part-1-For-Beginners-Bag-Of-Words
- Part-2-Word-Vectors
- Part-3-More-Fun-With-Word-Vectors
- Setting-Up-Your-System

Metric

Submissions are judged on area under the ROC curve.

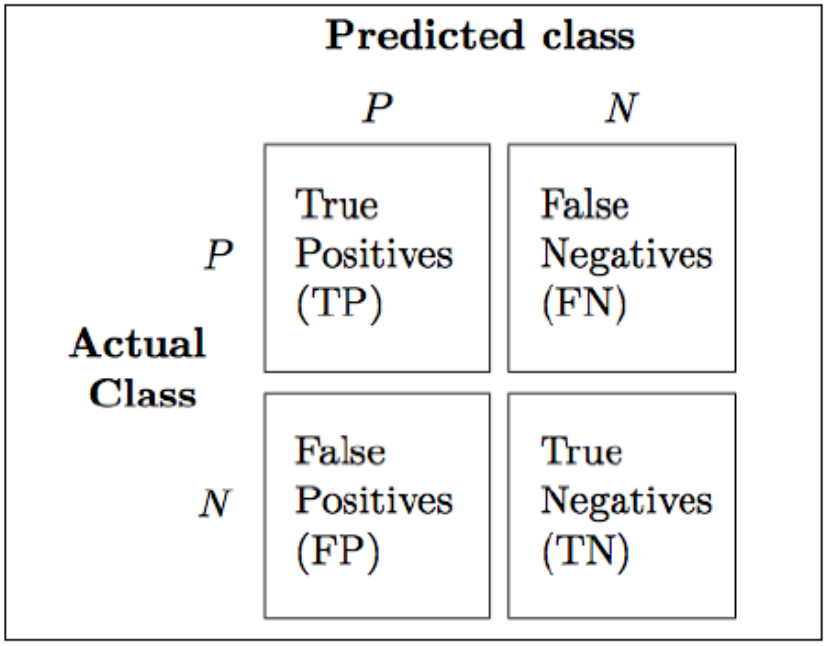
Submission Instructions

You should submit a comma-separated file with 25,000 row plus a header row. There should be 2 columns: "id" and "sentiment", which contain your binary predictions: 1 for positive reviews, 0 for negative reviews. For an example, see "sampleSubmission.csv" on the Data page.

```
id,sentiment
123_45,0
678_90,1
12_34,0
...
```

Performance Metric

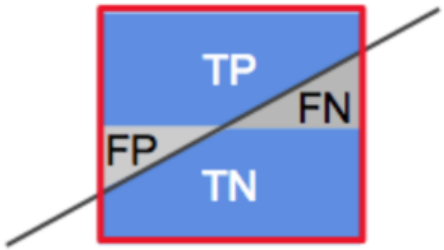
- Confusion Matrix
  - A square matrix that reports the counts of the *true positive*, *true negative*, *false positive*, and *false negative* predictions of a classifier



Performance Metric

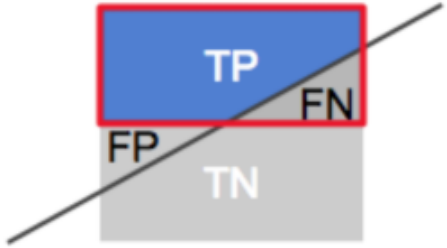
- ACC, TPR, FPR
  - Accuracy(ACC)

$$ACC = \frac{TP + TN}{FP + FN + TP + TN}$$



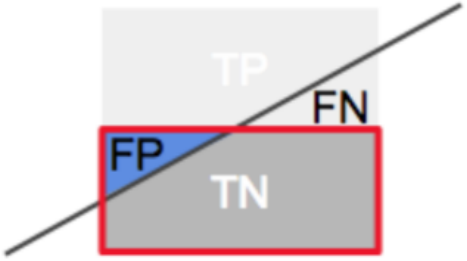
  - True Positive Rate(TPR)

$$TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$



  - False Positive Rate(FPR)

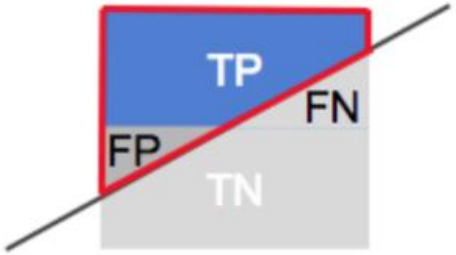
$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$



Performance Metric

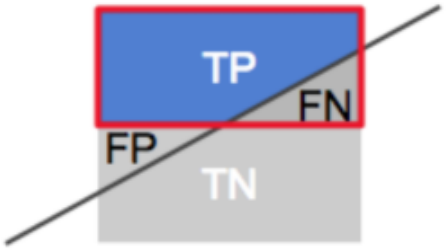
- Precision, Recall, F1
  - Precision(PRE)

$$PRE = \frac{TP}{TP + FP}$$



  - Recall(REC)

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

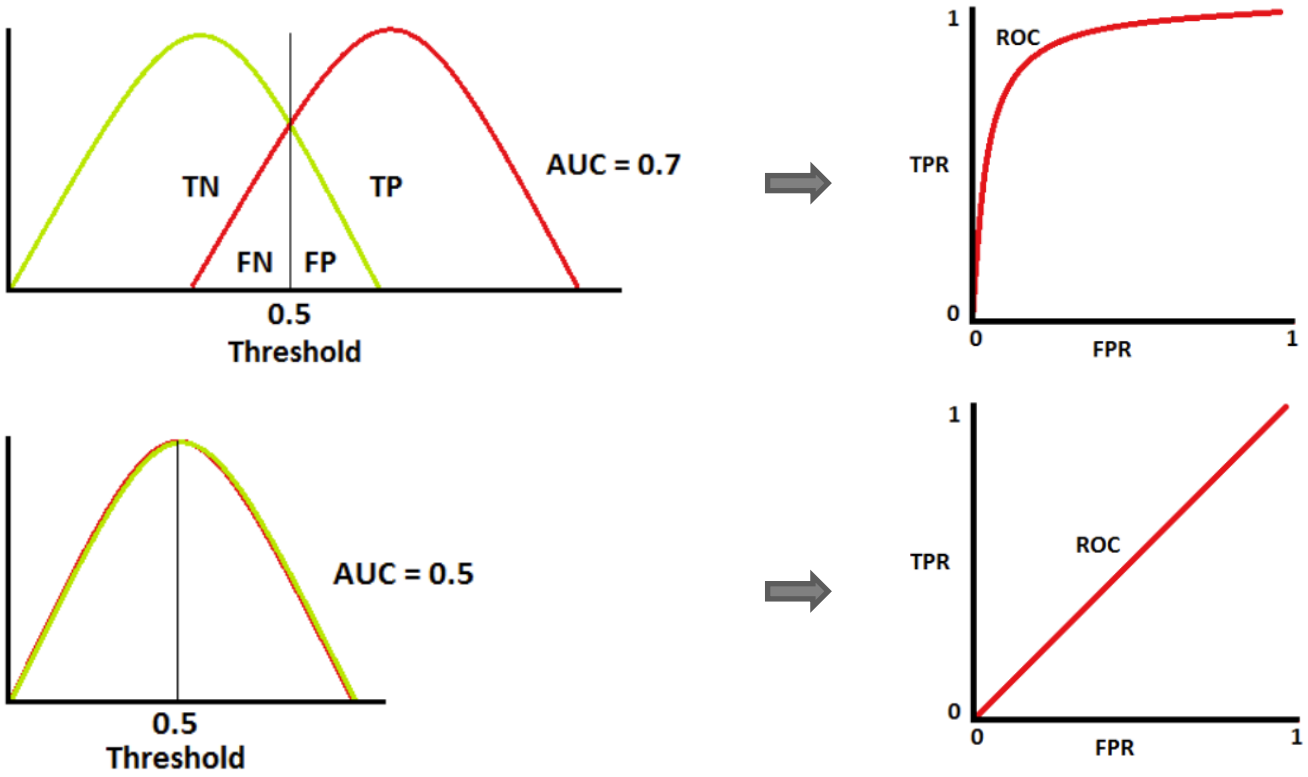


  - $F1 - score$

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

Performance Metric

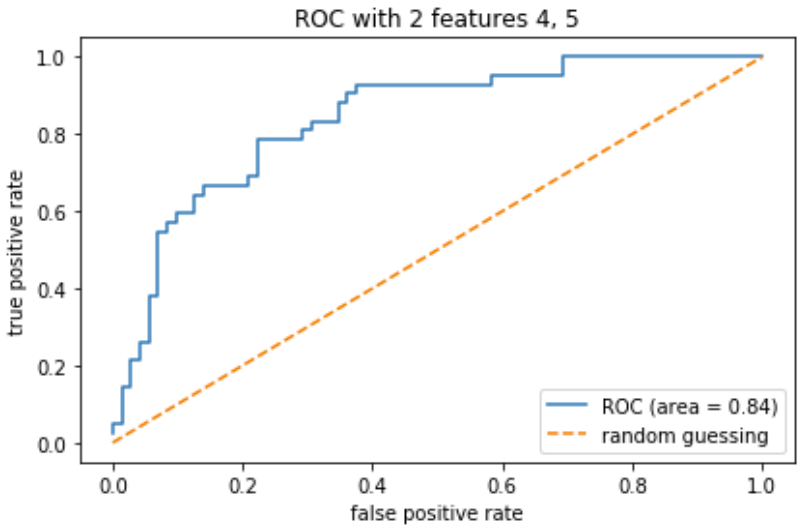
- ROC (Receiver Operating Characteristic) Curve
  - Plotting TPR vs. FPR
  - ROC tells how much a model can distinguish between classes



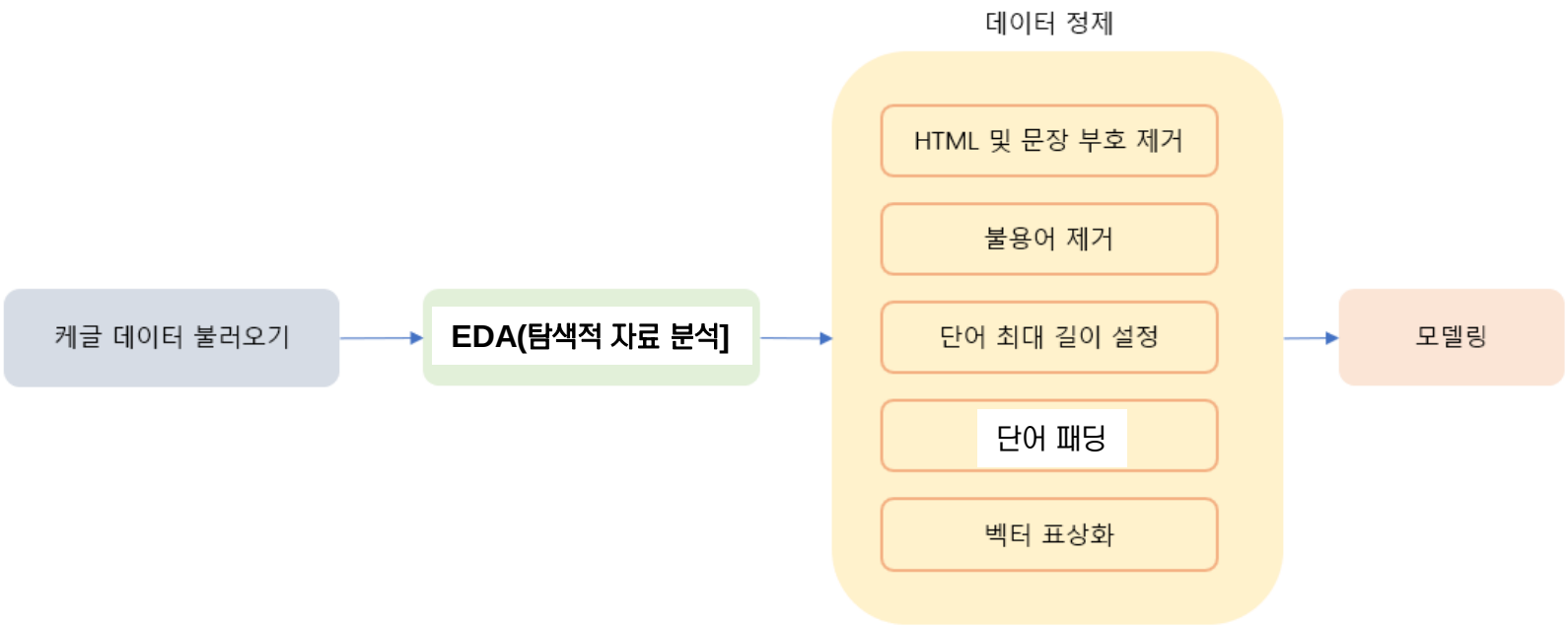
Performance Metric

- roc_curve
  - Compute FPR, TPR for different decision thresholds

```
from sklearn.metrics import roc_curve, auc

# prediction probability of class 1
pipe_lr.fit(X_train2, y_train)
probas = pipe_lr.predict_proba(X_test2)

# FPR, TPR, AUC
fpr, tpr, thresholds = roc_curve(y_test, probas[:,1], pos_label=1)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr,
         label='ROC (area = %0.2f)' % roc_auc)
```



데이터 분석 및 전처리


데이터 정제
케글 데이터 불러오기
EDA(탐색적 자료 분석]
HTML 및 문장 부호 제거
불용어 제거
단어 최대 길이 설정
단어 패딩
벡터 표상화
모델링

■ 데이터 분석은 아래 순서로 진행한다.

1. 데이터 크기
2. 데이터의 개수
3. 각 리뷰의 문자 길이 분포
4. 많이 사용된 단어
5. 긍정, 부정 데이터의 분포
6. 각 리뷰의 단어 개수 분포
7. 특수문자 및 대문자, 소문자 비율

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

데이터 크기

```
print("파일 크기 : ")
for file in os.listdir(DATA_IN_PATH):
    if 'tsv' in file and 'zip' not in file:
        print(file.ljust(30) + str(round(os.path.getsize(DATA_IN_PATH + file) / 1000000, 2)) + 'MB')
```

```
파일 크기 :
labeledTrainData.tsv          33.56MB
testData.tsv                  32.72MB
unlabeledTrainData.tsv        67.28MB
```

```
train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv', header = 0, delimiter = '\t', quoting = 3)
train_data.head()
```

	id	sentiment	review
0	"5814_8"	1	"With all this stuff going down at the moment ...
1	"2381_9"	1	"\"The Classic War of the Worlds\" by Timothy ...
2	"7759_3"	0	"The film starts with a manager (Nicholas Bell...

데이터 개수

```
print('전체 학습데이터의 개수: {}'.format(len(train_data)))
```

```
전체 학습데이터의 개수: 25000
```

각 리뷰의 문자 길이 분포

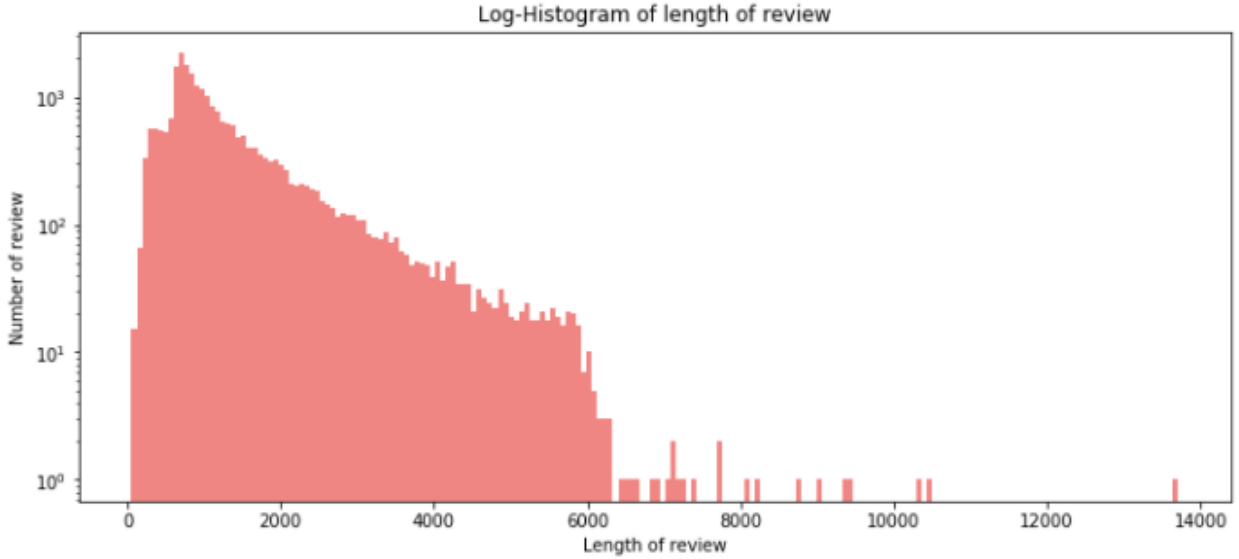
```
train_length = train_data['review'].apply(len)
```

```
train_length.head()
```

```
0    2304
1     948
2    2451
3    2247
4    2233
Name: review, dtype: int64
```

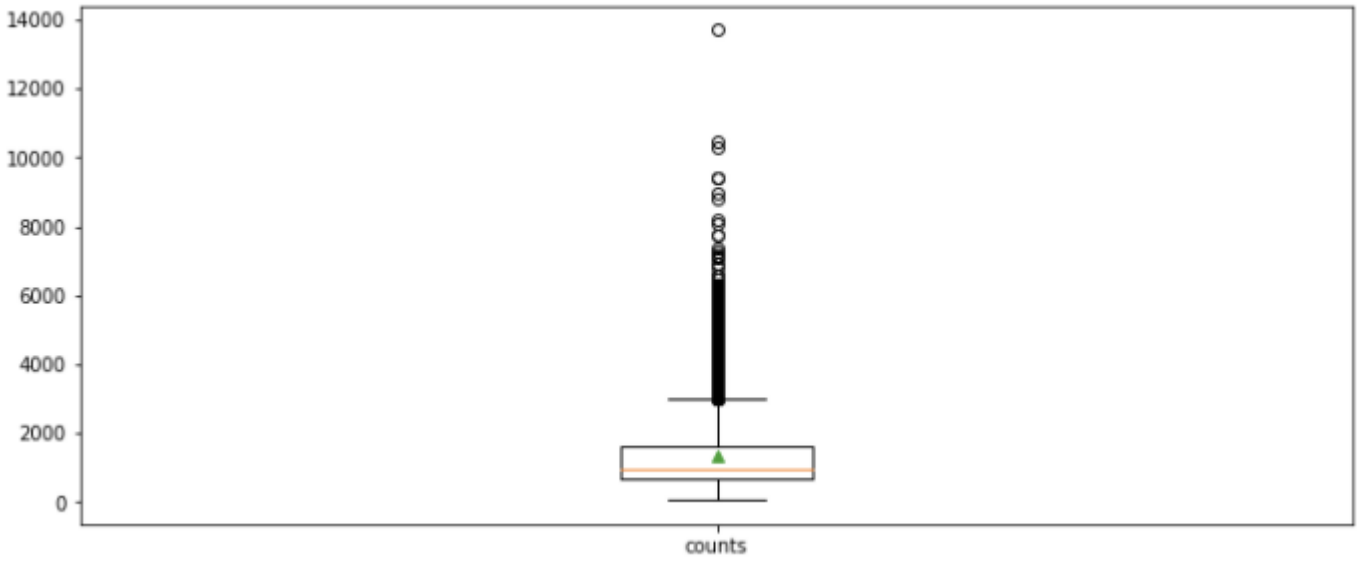
```
# 그래프에 대한 이미지 사이즈 선언
# figsize: (가로, 세로) 형태의 튜플로 입력
plt.figure(figsize=(12, 5))
# 히스토그램 선언
# bins: 히스토그램 값들에 대한 버켓 범위
# range: x축 값의 범위
# alpha: 그래프 색상 투명도
# color: 그래프 색상
# label: 그래프에 대한 라벨
plt.hist(train_length, bins=200, alpha=0.5, color= 'r', label='word')
plt.yscale('log', nonposy='clip')
# 그래프 제목
plt.title('Log-Histogram of length of review')
# 그래프 x 축 라벨
plt.xlabel('Length of review')
# 그래프 y 축 라벨
plt.ylabel('Number of review')
```

```
Text(0, 0.5, 'Number of revie
```



```
print('리뷰 길이 최대 값: {}'.format(np.max(train_length)))
print('리뷰 길이 최소 값: {}'.format(np.min(train_length)))
print('리뷰 길이 평균 값: {:.2f}'.format(np.mean(train_length)))
print('리뷰 길이 표준편차: {:.2f}'.format(np.std(train_length)))
print('리뷰 길이 중간 값: {}'.format(np.median(train_length)))
# 사분위의 대한 경우는 0~100 스케일로 되어있음
print('리뷰 길이 제 1 사분위: {}'.format(np.percentile(train_length, 25)))
print('리뷰 길이 제 3 사분위: {}'.format(np.percentile(train_length, 75)))
```

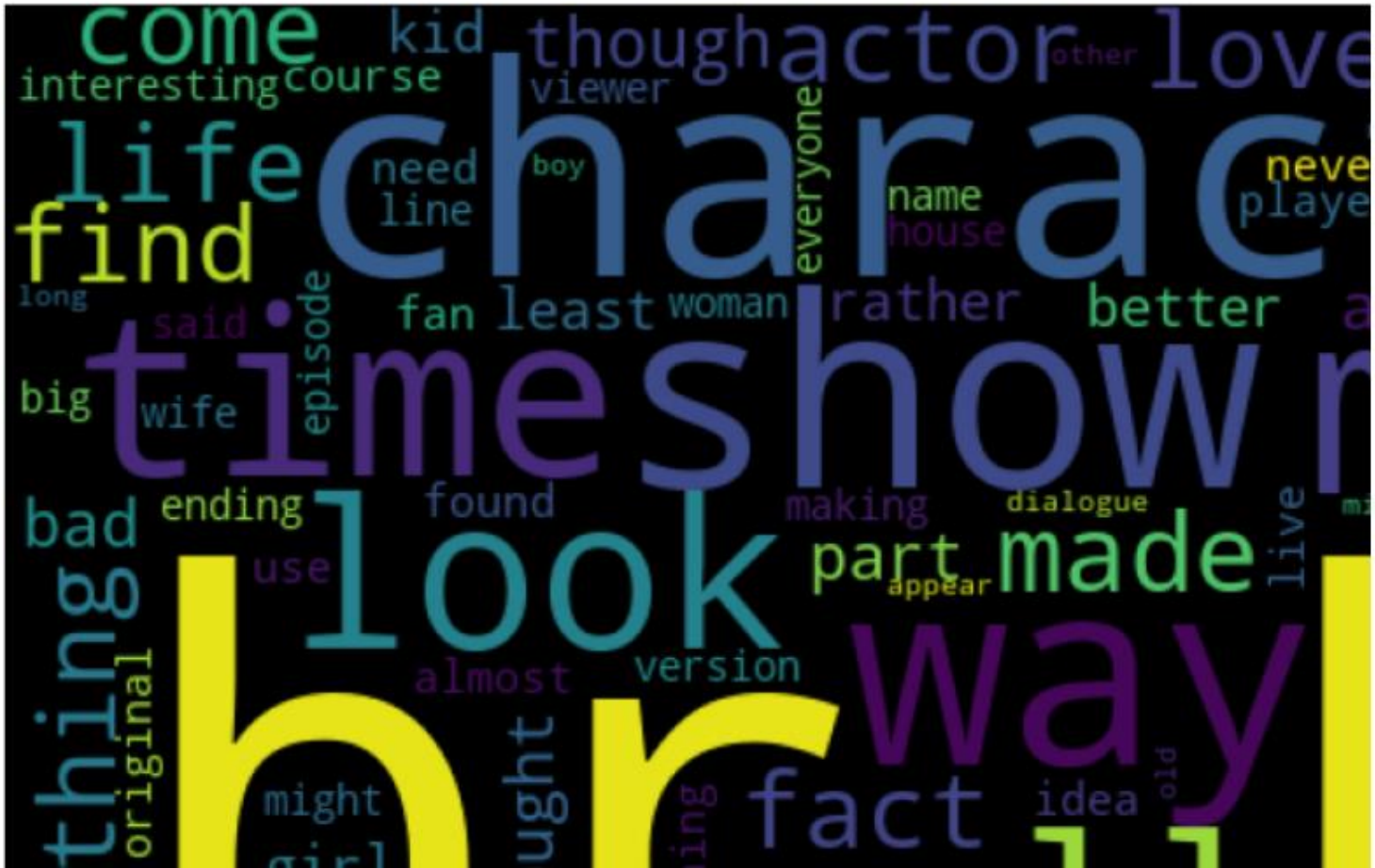
```
리뷰 길이 최대 값: 13710
리뷰 길이 최소 값: 54
리뷰 길이 평균 값: 1329.71
리뷰 길이 표준편차: 1005.22
리뷰 길이 중간 값: 983.0
리뷰 길이 제 1 사분위: 705.0
리뷰 길이 제 3 사분위: 1619.0
```



많이 사용된 단어

```
from wordcloud import WordCloud
cloud = WordCloud(width=800, height=600).generate(" ".join(train_data['review']))
plt.figure(figsize=(20, 15))
plt.imshow(cloud)
plt.axis('off')
```

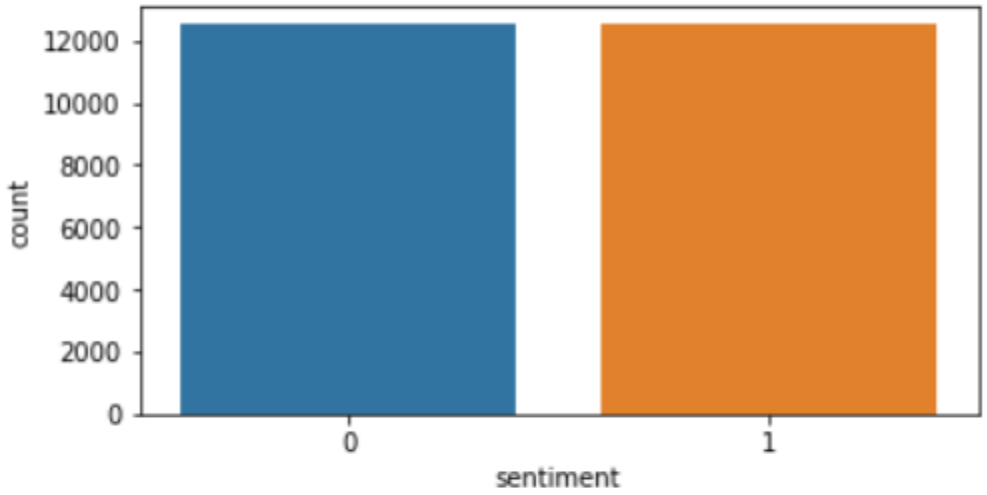
```
(-0.5, 799.5, 599.5, -0.5)
```



긍, 부정 데이터의 분포

```
fig, axe = plt.subplots(ncols=1)
fig.set_size_inches(6, 3)
sns.countplot(train_data['sentiment'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f81ef2bf3c8>
```



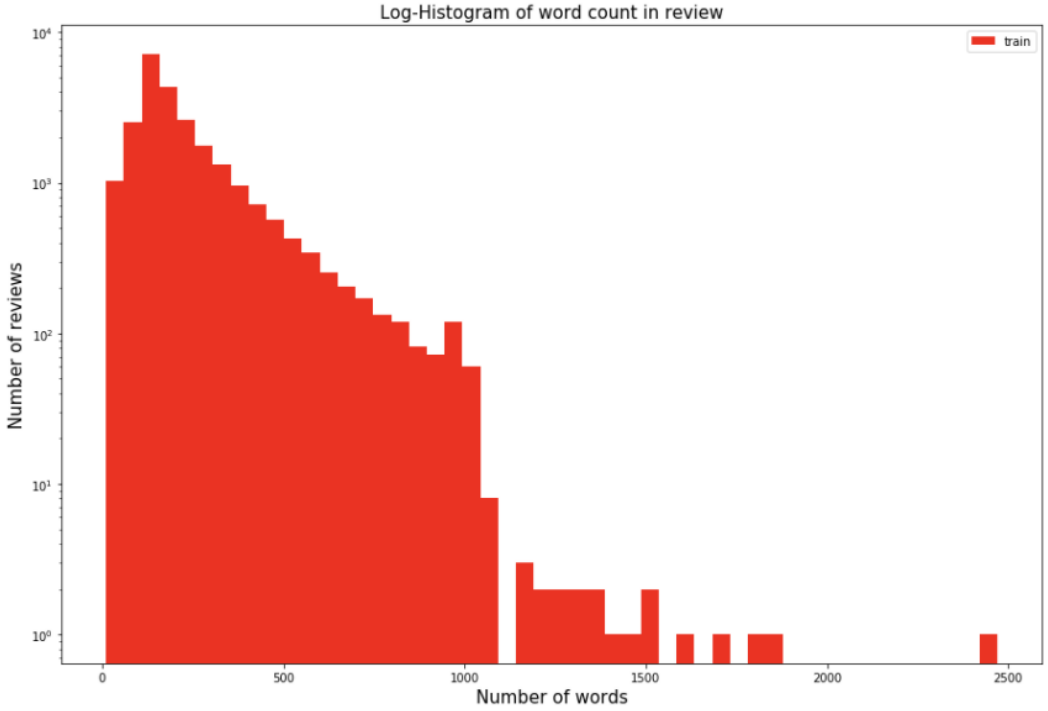

```
print("긍정 리뷰 개수: {}".format(train_data['sentiment'].value_counts()[1]))
print("부정 리뷰 개수: {}".format(train_data['sentiment'].value_counts()[0]))
```

```
긍정 리뷰 개수: 12500
부정 리뷰 개수: 12500
```

각 리뷰의 단어 개수 분포

```
train_word_counts = train_data['review'].apply(lambda x:len(x.split(' ')))
```

```
plt.figure(figsize=(15, 10))
plt.hist(train_word_counts, bins=50, facecolor='r',label='train')
plt.title('Log-Histogram of word count in review', fontsize=15)
plt.yscale('log', nonposy='clip')
plt.legend()
plt.xlabel('Number of words', fontsize=15)
plt.ylabel('Number of reviews', fontsize=15)
```



특수문자 및 대, 소문자 비율

```
qmarks = np.mean(train_data['review'].apply(lambda x: '?' in x)) # 물음표가 구두점으로 쓰
fullstop = np.mean(train_data['review'].apply(lambda x: '.' in x)) # 마침표
capital_first = np.mean(train_data['review'].apply(lambda x: x[0].isupper())) #  첫번째
capitals = np.mean(train_data['review'].apply(lambda x: max([y.isupper() for y in x])))
numbers = np.mean(train_data['review'].apply(lambda x: max([y.isdigit() for y in x]))) 

print('물음표가있는 질문: {:.2f}%'.format(qmarks * 100))
print('마침표가 있는 질문: {:.2f}%'.format(fullstop * 100))
print('첫 글자가 대문자 인 질문: {:.2f}%'.format(capital_first * 100))
print('대문자가있는 질문: {:.2f}%'.format(capitals * 100))
print('숫자가있는 질문: {:.2f}%'.format(numbers * 100))
```

```
물음표가있는 질문: 29.55%
마침표가 있는 질문: 99.69%
첫 글자가 대문자 인 질문: 0.00%
대문자가있는 질문: 99.59%
숫자가있는 질문: 56.66%
```

데이터 전처리

데이터 분석과정을 바탕으로 데이터를 모델에 적용시키기 위해 전처리 과정을 진행한다.

```
import re
import json
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.preprocessing.text import Tokenizer
```

```
DATA_IN_PATH = './data_in/'

train_data = pd.read_csv( DATA_IN_PATH + 'labeledTrainData.tsv',
                          header = 0, delimiter = '\t', quoting = 3)
print(train_data['review'][0])
```

```
"With all this stuff going down at the moment with MJ i've started listening to his mus
ic, watching the odd documentary here and there, watched The Wiz and watched Moonwalker
again. Maybe i just want to get a certain insight into this guy who i thought was reall
y cool in the eighties just to maybe make up my mind whether he is guilty or innocent.
Moonwalker is part biography, part feature film which i remember going to see at the ci
nema when it was originally released. Some of it has subtle messages about MJ's feeling
towards the press and also the obvious message of drugs are bad m'kay.<br /><br />Visua
```

```
review = train_data['review'][0] # 리뷰 중 하나를 가져온다.
review_text = BeautifulSoup(review,"html5lib").get_text() # HTML 태그 제거
review_text = re.sub("[^a-zA-Z]", " ", review_text )
# 영어 문자를 제외한 나머지는 모두 공백으로 바꾼다.
```

```
print(review_text)
```

```
 With all this stuff going down at the moment with MJ i ve started listening to his mus
ic  watching the odd documentary here and there  watched The Wiz and watched Moonwalker
again  Maybe i just want to get a certain insight into this guy who i thought was reall
y cool in the eighties just to maybe make up my mind whether he is guilty or innocent
Moonwalker is part biography  part feature film which i remember going to see at the ci
nema when it was originally released  Some of it has subtle messages about MJ s feeling
towards the press and also the obvious message of drugs are bad m kay Visually impressi
ve but of course this is all about Michael Jackson so unless you remotely like MJ in an
yway then you are going to hate this and find it boring  Some may call MJ an egotist fo
```

```
stop_words = set(stopwords.words('english')) # 영어 불용어들의 set을 만든다.

review_text = review_text.lower()
words = review_text.split() # 소문자 변환 후 단어마다 나눠서 단어 리스트로 만든다.
words = [w for w in words if not w in stop_words] # 불용어 제거한 리스트를 만든다
```

```
print(words)
```

```
['stuff', 'going', 'moment', 'mj', 'started', 'listening', 'music', 'watching', 'odd',
'documentary', 'watched', 'wiz', 'watched', 'moonwalker', 'maybe', 'want', 'get', 'cert
ain', 'insight', 'guy', 'thought', 'really', 'cool', 'eighties', 'maybe', 'make', 'min
d', 'whether', 'guilty', 'innocent', 'moonwalker', 'part', 'biography', 'part', 'featur
e', 'film', 'remember', 'going', 'see', 'cinema', 'originally', 'released', 'subtle',
'messages', 'mj', 'feeling', 'towards', 'press', 'also', 'obvious', 'message', 'drugs',
'bad', 'kay', 'visually', 'impressive', 'course', 'michael', 'jackson', 'unless', 'remo
tely', 'like', 'mj', 'anyway', 'going', 'hate', 'find', 'boring', 'may', 'call', 'mj',
'egotist', 'consenting', 'making', 'movie', 'mj', 'fans', 'would', 'say', 'made', 'fan
s', 'true', 'really', 'nice', 'actual', 'feature', 'film', 'bit', 'finally', 'starts',
'minutes', 'excluding', 'smooth', 'criminal', 'sequence', 'joe', 'pesci', 'convincing',
'psychopathic', 'powerful', 'drug', 'lord', 'wants', 'mj', 'dead', 'bad', 'beyond', 'm
j', 'overheard', 'plans', 'nah', 'joe', 'pesci', 'character', 'ranted', 'wanted', 'peop
le', 'know', 'supplying', 'drugs', 'etc', 'dunno', 'maybe', 'hates', 'mj', 'music', 'lo
```

```
clean_review = ' '.join(words) # 단어 리스트들을 다시 하나의 글로 합친다.
print(clean_review)
```

```
stuff going moment mj started listening music watching odd documentary watched wiz watc
hed moonwalker maybe want get certain insight guy thought really cool eighties maybe ma
ke mind whether guilty innocent moonwalker part biography part feature film remember go
ing see cinema originally released subtle messages mj feeling towards press also obviou
s message drugs bad kay visually impressive course michael jackson unless remotely like
mj anyway going hate find boring may call mj egotist consenting making movie mj fans wo
uld say made fans true really nice actual feature film bit finally starts minutes exclu
ding smooth criminal sequence joe pesci convincing psychopathic powerful drug lord want
s mj dead bad beyond mj overheard plans nah joe pesci character ranted wanted people kn
ow supplying drugs etc dunno maybe hates mj music lots cool things like mj turning car
robot whole speed demon sequence also director must patience saint came filming kiddy b
ad sequence usually directors hate working one kid let alone whole bunch performing com
plex dance scene bottom line movie people like mj one level another think people stay a
way try give wholesome message ironically mj bestest buddy movie girl michael jackson t
ruly one talented people ever grace planet guilty well attention gave subject hmmm well
know people different behind closed doors know fact either extremely nice stupid guy on
e sickest liars hope latter
```

```python
def preprocessing( review, remove_stopwords = False ):
    # 불용어 제거는 옵션으로 선택 가능하다.

    # 1. HTML 태그 제거
    review_text = BeautifulSoup(review, "html5lib").get_text()

    # 2. 영어가 아닌 특수문자들을 공백(" ")으로 바꾸기
    review_text = re.sub("[^a-zA-Z]", " ", review_text)

    # 3. 대문자들을 소문자로 바꾸고 공백단위로 텍스트들 나눠서 리스트로 만든다.
    words = review_text.lower().split()

    if remove_stopwords:
        # 4. 불용어들을 제거

        #영어에 관련된 불용어 불러오기
        stops = set(stopwords.words("english"))
        # 불용어가 아닌 단어들로 이루어진 새로운 리스트 생성
        words = [w for w in words if not w in stops]
        # 5. 단어 리스트를 공백을 넣어서 하나의 글로 합친다.
        clean_review = ' '.join(words)

    else: # 불용어 제거하지 않을 때
        clean_review = ' '.join(words)

    return clean_review
```

```python
clean_train_reviews = []
for review in train_data['review']:
    clean_train_reviews.append(preprocessing(review, remove_stopwords = True))

# 전처리한 데이터 출력
clean_train_reviews[0]
```

```
'stuff going moment mj started listening music watching odd documentary watched wiz wat
```

```
clean_train_df = pd.DataFrame({'review': clean_train_reviews,
                               'sentiment': train_data['sentiment']})
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(clean_train_reviews)
text_sequences = tokenizer.texts_to_sequences(clean_train_reviews)
```

```
print(text_sequences[0])
```

```
[404, 70, 419, 8815, 506, 2456, 115, 54, 873, 516, 178, 18686, 178, 11242, 165, 78, 14,
662, 2457, 117, 92, 10, 499, 4074, 165, 22, 210, 581, 2333, 1194, 11242, 71, 4826, 71,
635, 2, 253, 70, 11, 302, 1663, 486, 1144, 3265, 8815, 411, 793, 3342, 17, 441, 600, 15
00, 15, 4424, 1851, 998, 146, 342, 1442, 743, 2424, 4, 8815, 418, 70, 637, 69, 237, 94,
541, 8815, 26055, 26056, 120, 1, 8815, 323, 8, 47, 20, 323, 167, 10, 207, 633, 635, 2,
116, 291, 382, 121, 15535, 3315, 1501, 574, 734, 10013, 923, 11578, 822, 1239, 1408, 36
0, 8815, 221, 15, 576, 8815, 22224, 2274, 13426, 734, 10013, 27, 28606, 340, 16, 41, 18
687, 1500, 388, 11243, 165, 3962, 8815, 115, 627, 499, 79, 4, 8815, 1430, 380, 2163, 11
4, 1919, 2503, 574, 17, 60, 100, 4875, 5100, 260, 1268, 26057, 15, 574, 493, 744, 637,
631, 3, 394, 164, 446, 114, 615, 3266, 1160, 684, 48, 1175, 224, 1, 16, 4, 8815, 3, 50
7, 62, 25, 16, 640, 133, 231, 95, 7426, 600, 3439, 8815, 37248, 1864, 1, 128, 342, 144
2, 247, 3, 865, 16, 42, 1487, 997, 2333, 12, 549, 386, 717, 6920, 12, 41, 16, 158, 362,
4392, 3388, 41, 87, 225, 438, 207, 254, 117, 3, 18688, 18689, 316, 1356]
```

```
word_vocab = tokenizer.word_index
word_vocab["<PAD>"] = 0
```

```
print("전체 단어 개수: ", len(word_vocab))
```

```
전체 단어 개수:  74066
```

```
data_configs = {}

data_configs['vocab'] = word_vocab
data_configs['vocab_size'] = len(word_vocab)
```

```
MAX_SEQUENCE_LENGTH = 174

train_inputs = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH,
                             padding='post')

print('Shape of train data: ', train_inputs.shape)
```

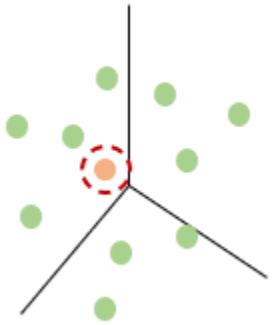
```
Shape of train data:  (25000, 174)
```

```
train_labels = np.array(train_data['sentiment'])
print('Shape of label tensor:', train_labels.shape)
```

```
Shape of label tensor: (25000,)
```

```
TRAIN_INPUT_DATA = 'train_input.npy'
TRAIN_LABEL_DATA = 'train_label.npy'
TRAIN_CLEAN_DATA = 'train_clean.csv'
DATA_CONFIGS = 'data_configs.json'

import os
# 저장하는 디렉토리가 존재하지 않으면 생성
if not os.path.exists(DATA_IN_PATH):
    os.makedirs(DATA_IN_PATH)
```

```
# 전처리 된 데이터를 넘파이 형태로 저장
np.save(open(DATA_IN_PATH + TRAIN_INPUT_DATA, 'wb'), train_inputs)
np.save(open(DATA_IN_PATH + TRAIN_LABEL_DATA, 'wb'), train_labels)

# 정제된 텍스트를 csv 형태로 저장
clean_train_df.to_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA, index = False)


# 데이터 사전을 json 형태로 저장
json.dump(data_configs, open(DATA_IN_PATH + DATA_CONFIGS, 'w'), ensure_ascii=False)
```

원본	I love this movie so much I do not know why this movie is fun
인덱스 변환	[4, 7, 9, 1, 22, 8] [4, 10, 29, 44, 12, 33, 22, 24, 44]
사이즈 패딩 Ex) 길이 10	[4, 7, 9, 1, 22, 8 , 0 , 0 , 0 , 0] [4, 10, 29, 44, 12, 33, 22, 24, 44 , 0]
벡터화	

TF-IDF

- **Step1:** Filter out the stopwords. After removing the stopwords, we have
  - d_1: "*sky blue*
  - d_2: "*sun bright today*"
  - d_3: "*sun sky bright*"
  - d_4: "*can see shining sun bright sun*"
- **Step2:** Compute TF, therefore, we find document-word matrix and then normalize the rows to sum to 1.

$$f_{t,d}$$

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

TF score computation. [Image Source]

- **Step3:** Compute IDF: Find the number of documents in which each word occurs, then compute the formula:

$f_{t,d}$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0
n_t	1	3	1	1	1	2	3	1

$N = 4$

$$\text{idf}(t, D) = \log_{10} \frac{N}{n_t}$$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$\log_{10} \frac{4}{1} = 0.602$ $\log_{10} \frac{4}{3} = 0.125$

IDF score computation. [Image Source]

- **Step4:** Compute TF-IDF: Multiply TF and IDF scores.

$\text{tf}(t, d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

x

$\text{idf}(t, D)$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
  - Most important word for each document is highlighted

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

	blue	bright	can	see	shining	sky	sun	today
1	**0.301**	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	**0.201**
3	0	0.0417	0	0	0	**0.100**	0.0417	0
4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

TF-IDF score computation. [Image Source]

What is Scikit-learn?

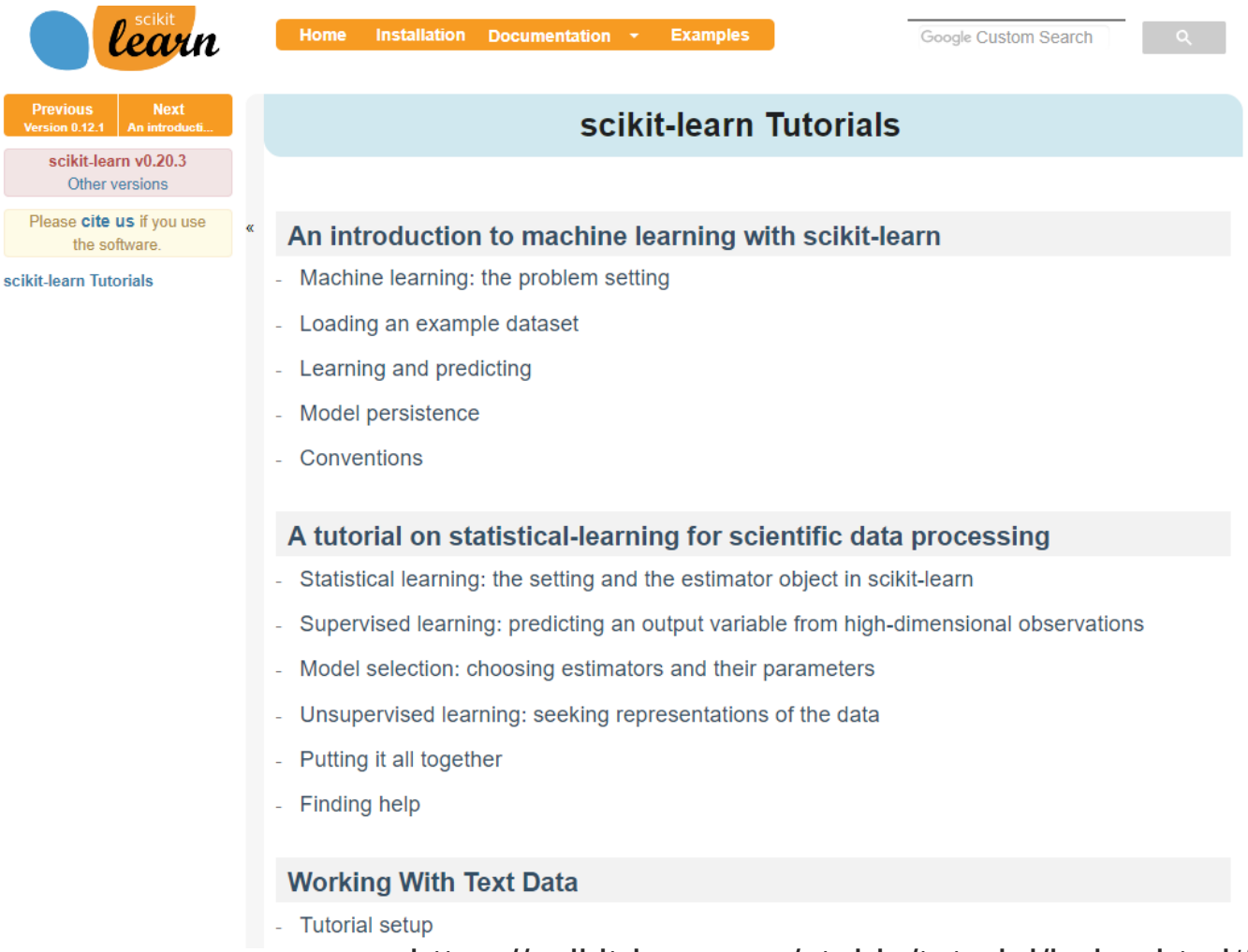
- Scikit-learn
  - Open source machine learning library for the Python that supports supervised and unsupervised learning
  - It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries
  - It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

https://scikit-learn.org/stable/

scikit-learn.org

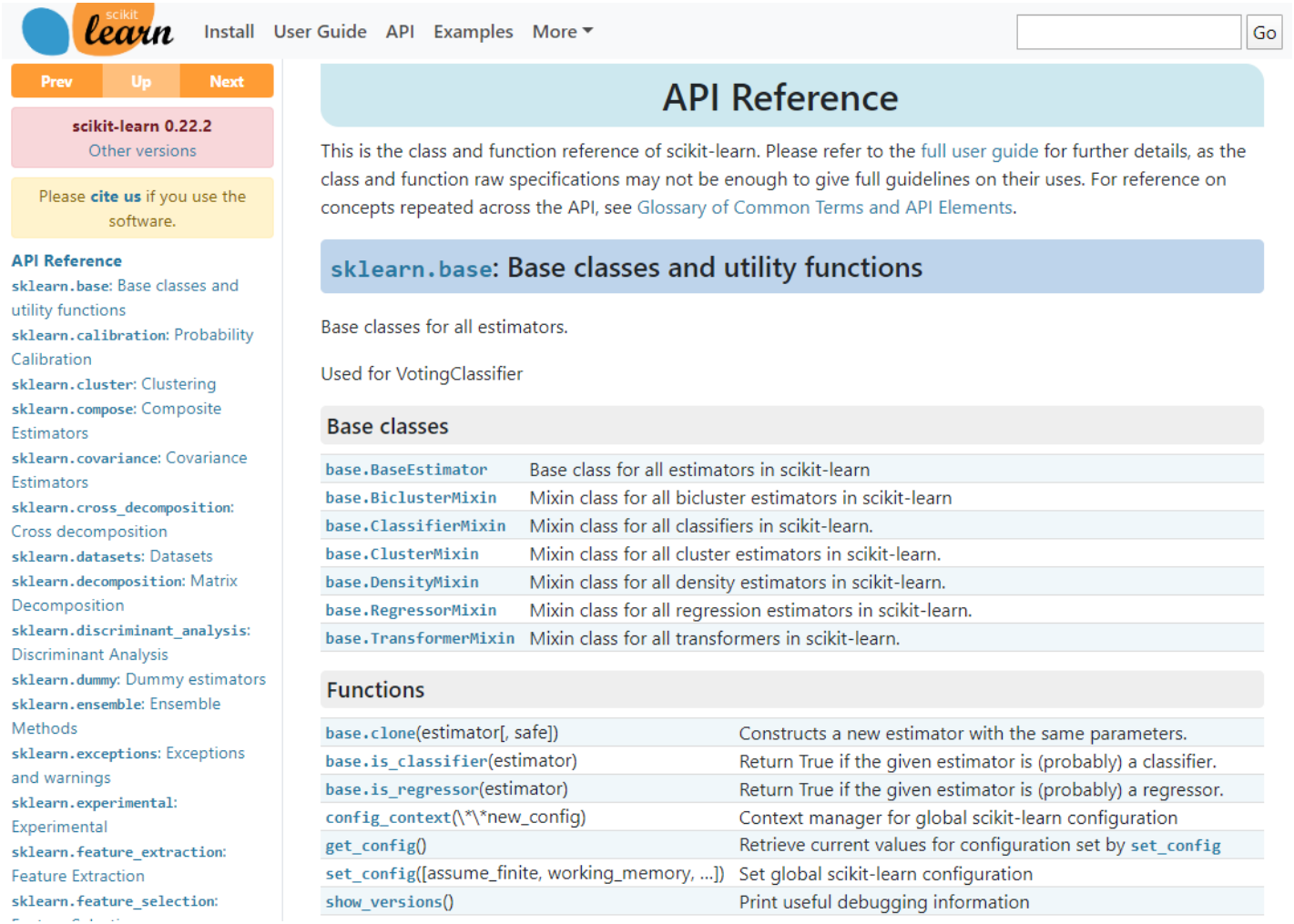
https://scikit-learn.org/stable/

Tutorial

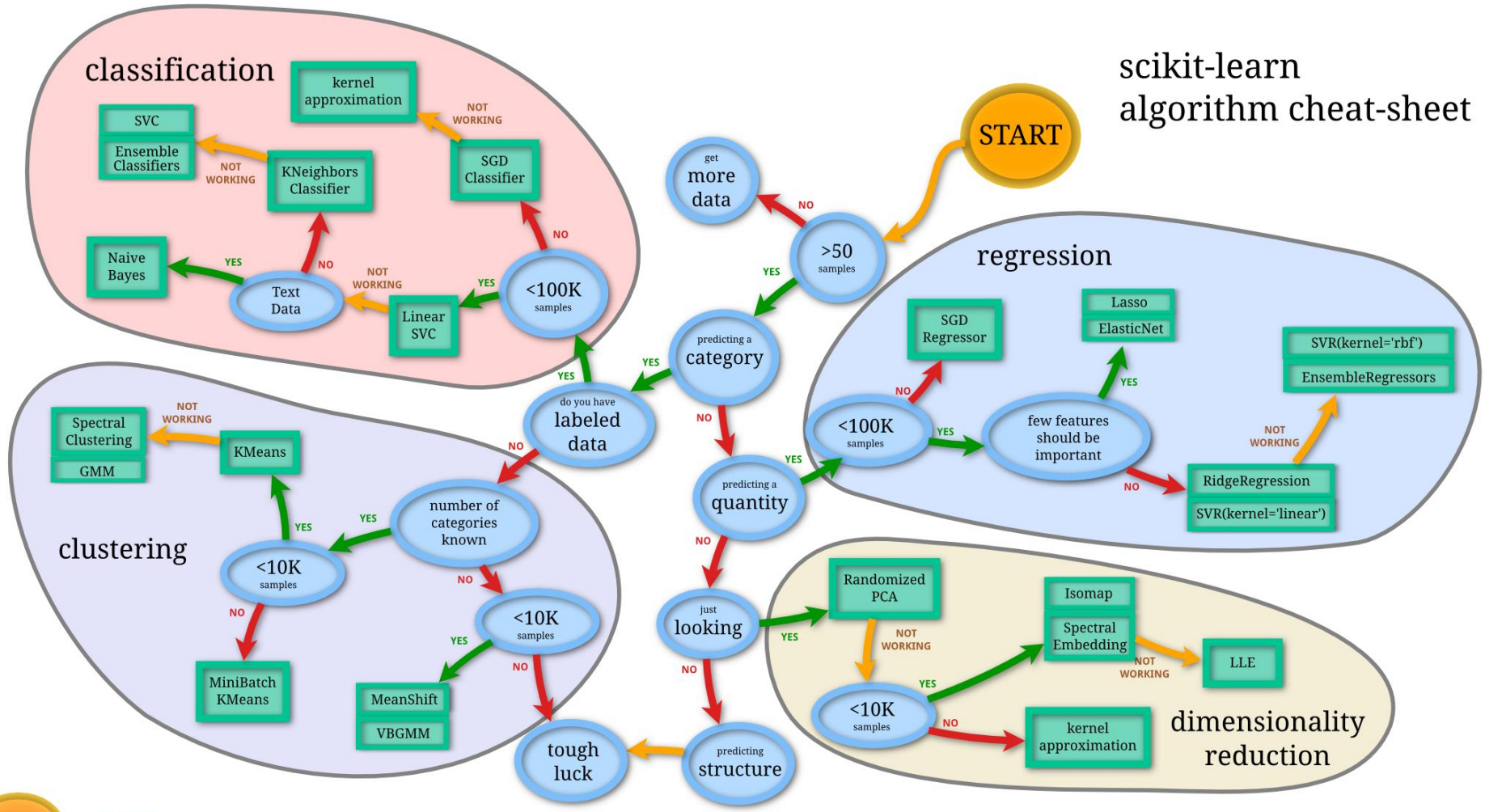


https://scikit-learn.org/stable/tutorial/index.html#tutorial-menu

API



https://scikit-learn.org/stable/modules/classes.html


scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
kernel approximation
SVC
Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K samples
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
<100K samples
few features should be important
RidgeRegression
SVR(kernel='linear')
clustering
Spectral Clustering
GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
<10K samples
MeanShift
VBGMM
dimensionality reduction
Randomized PCA
Isomap
Spectral Embedding
LLE
<10K samples
kernel approximation
YES
NO
NOT WORKING

Back

Training a Model

- Task
  - Classify x = (x1, x2, x3) to y = 1 or 0
- Training dataset
  - 5 instances (example data) with known labels

```
X = np.array([[0, 1, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]])
y = np.array([1, 0, 1, 1, 0])
```

features label

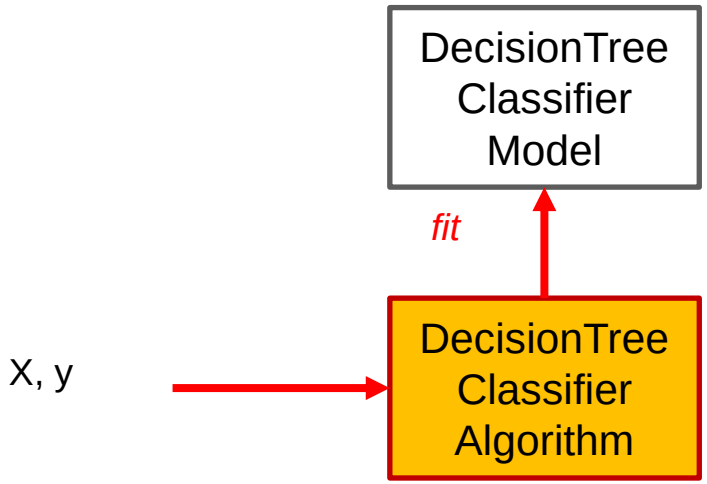
5 instances

X			y
0,	1,	1	1
1,	0,	1	**0**
1,	1,	1	**1**
0,	1,	1	**1**
0,	0,	1	**0**

Training a Model

- Training (learning) - fit
  - Learning Decision Tree Classifier model with the training dataset

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf = clf.fit(X, y)
```



Predicting using the Model

- Test dataset
  - 2 new instances

```
X_test = np.array([[0, 0, 0], [1, 1, 0]])
y_test = np.array([0, 1])
```

new instances

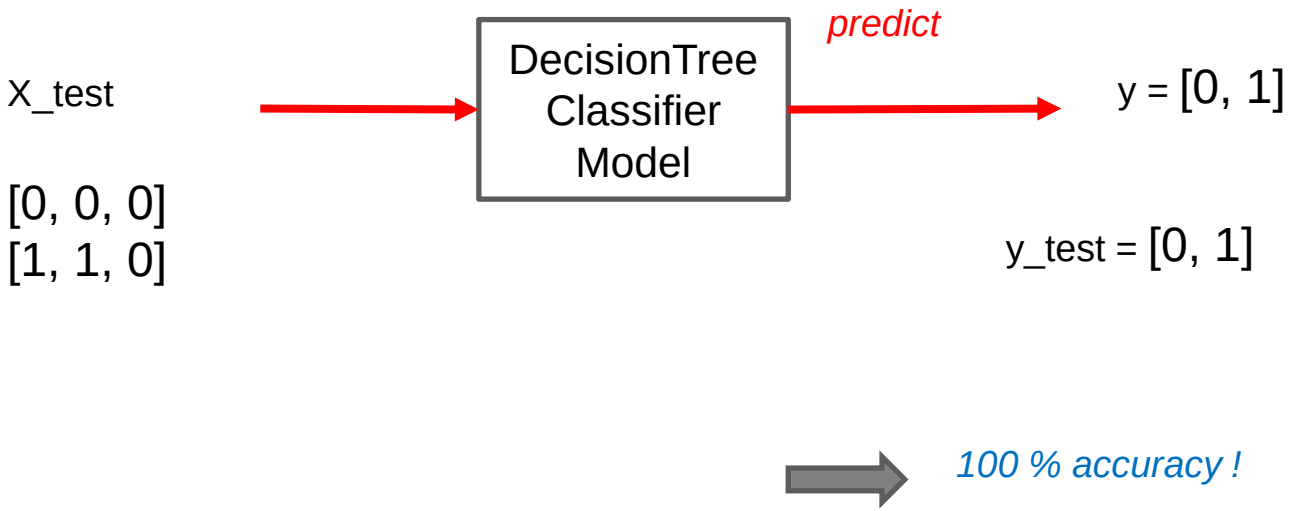
X			y
0,	0,	0	
1,	1,	0	

Predicting using the Model

- Predicting labels - predict
  - Predict the label of new x using the learned model

```
predicted = clf.predict(X_test)
predicted
```

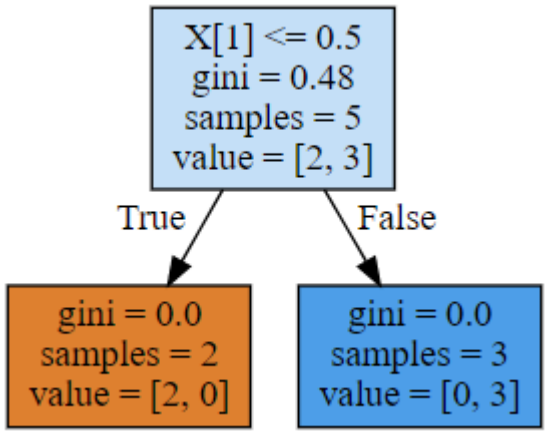
```
array([0, 1])
```



Visualize the Model

- Visualizing Decision Tree model using graphviz

```
from sklearn import tree
import graphviz

dot_data = tree.export_graphviz(clf, filled=True, out_file=None)
graph = graphviz.Source(dot_data)
graph
```



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

```
import os

import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
DATA_IN_PATH = './data_in/'
DATA_OUT_PATH = './data_out/'
TRAIN_CLEAN_DATA = 'train_clean.csv'

RANDOM_SEED = 42
TEST_SPLIT = 0.2
```

```
train_data = pd.read_csv( DATA_IN_PATH + TRAIN_CLEAN_DATA )
```

```
reviews = list(train_data['review'])
sentiments = list(train_data['sentiment'])
```

```
vectorizer = TfidfVectorizer(min_df = 0.0, analyzer="char", sublinear_tf=True,
                             ngram_range=(1,3), max_features=5000)

X = vectorizer.fit_transform(reviews)
y = np.array(sentiments)
```

```
X
```

```
<25000x5000 sparse matrix of type '<class 'numpy.float64'>'
        with 17862871 stored elements in Compressed Sparse Row format>
```

```
features = vectorizer.get_feature_names()
```

```
X_train, X_eval, y_train, y_eval = train_test_split(X, y, test_size=TEST_SPLIT,
                                                    random_state=RANDOM_SEED)
```

```
lgs = LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
```

```
LogisticRegression(class_weight='balanced')
```

```
predicted = lgs.predict(X_eval)
```

```
print("Accuracy: %f" % lgs.score(X_eval, y_eval))
```

```
Accuracy: 0.859800
```

```
TEST_CLEAN_DATA = 'test_clean.csv'

test_data = pd.read_csv(DATA_IN_PATH + TEST_CLEAN_DATA)
```

```
testDataVecs = vectorizer.transform(test_data['review'])
```

```
test_predicted = lgs.predict(testDataVecs)
print(test_predicted)
```

```
[1 0 1 ... 0 1 0]
```

Word2Vec Feature Example

```
import os
import re

import pandas as pd
import numpy as np

from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
DATA_IN_PATH = './data_in/'
TRAIN_CLEAN_DATA = 'train_clean.csv'

RANDOM_SEED = 42
TEST_SPLIT = 0.2
```

```
train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
```

```
reviews = list(train_data['review'])
sentiments = list(train_data['sentiment'])
```

```
sentences = []
for review in reviews:
    sentences.append(review.split())
```

```
num_features = 300
min_word_count = 40
num_workers = 4
context = 10
downsampling = 1e-3
```

```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',\
   level=logging.INFO)
```

```
from gensim.models import word2vec

model = word2vec.Word2Vec(sentences, workers=num_workers, \
          vector_size=num_features, min_count = min_word_count, \
           window = context, sample = downsampling)
```

```python
def get_features(words, model, num_features):
    feature_vector = np.zeros((num_features),dtype=np.float32)

    num_words = 0
#     index2word_set = set(model.wv.index2word)
    index2word_set = set(model.wv.index_to_key)

    for w in words:
        if w in index2word_set:
            num_words += 1
            feature_vector = np.add(feature_vector, model.wv[w])

    feature_vector = np.divide(feature_vector, num_words)
    return feature_vector
```

```python
def get_dataset(reviews, model, num_features):
    dataset = list()

    for s in reviews:
        dataset.append(get_features(s, model, num_features))

    reviewFeatureVecs = np.stack(dataset)

    return reviewFeatureVecs
```

```python
test_data_vecs = get_dataset(sentences, model, num_features)
```

```
from sklearn.model_selection import train_test_split
import numpy as np

X = test_data_vecs
y = np.array(sentiments)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SPLIT,
                                                    random_state=RANDOM_SEED)
```

```
from sklearn.linear_model import LogisticRegression

lgs = LogisticRegression(class_weight='balanced')
lgs.fit(X_train, y_train)
```

• • •

```
print("Accuracy: %f" % lgs.score(X_test, y_test))
```

```
Accuracy: 0.870200
```

Random Forest

패키지 및 데이터 불러오기 불러오기

```
import pandas as pd
import numpy as np
import os
from sklearn.feature_extraction.text import CountVectorizer
```

```
DATA_IN_PATH = './data_in/'
DATA_OUT_PATH = './data_out/'
TRAIN_CLEAN_DATA = 'train_clean.csv'
TEST_SIZE = 0.2
RANDOM_SEED = 42
```

```
train_data = pd.read_csv(DATA_IN_PATH + TRAIN_CLEAN_DATA)
```

```
reviews = list(train_data['review'])
y = np.array(train_data['sentiment'])
```

CountVectorizer를 활용한 벡터화

```
vectorizer = CountVectorizer(analyzer = "word", max_features = 5000)

train_data_features = vectorizer.fit_transform(reviews)
```

학습과 검증 데이터 분리

```
from sklearn.model_selection import train_test_split
```

```
train_input, eval_input, train_label, eval_label = train_test_split(
    train_data_features, y, test_size=TEST_SIZE, random_state=RANDOM_SEED)
```

모델 구현 및 학습

```
from sklearn.ensemble import RandomForestClassifier


# 랜덤 포레스트 분류기에  100개 의사 결정 트리를 사용한다.
forest = RandomForestClassifier(n_estimators = 1000)

# 단어 묶음을 벡터화한 데이터와 정답 데이터를 가지고 학습을 시작한다.
forest.fit( train_input, train_label )
```

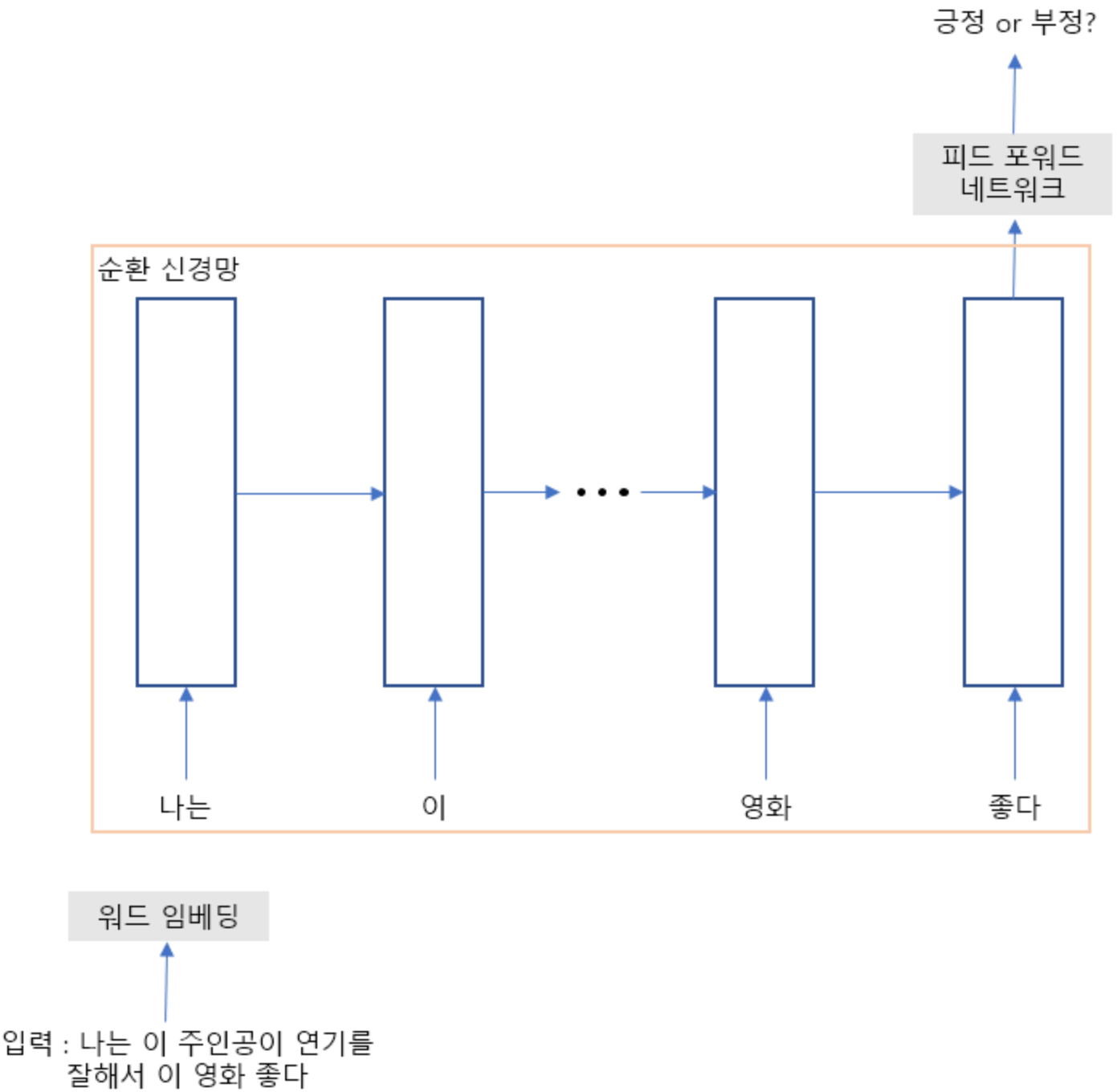
```
RandomForestClassifier(n_estimators=1000)
```

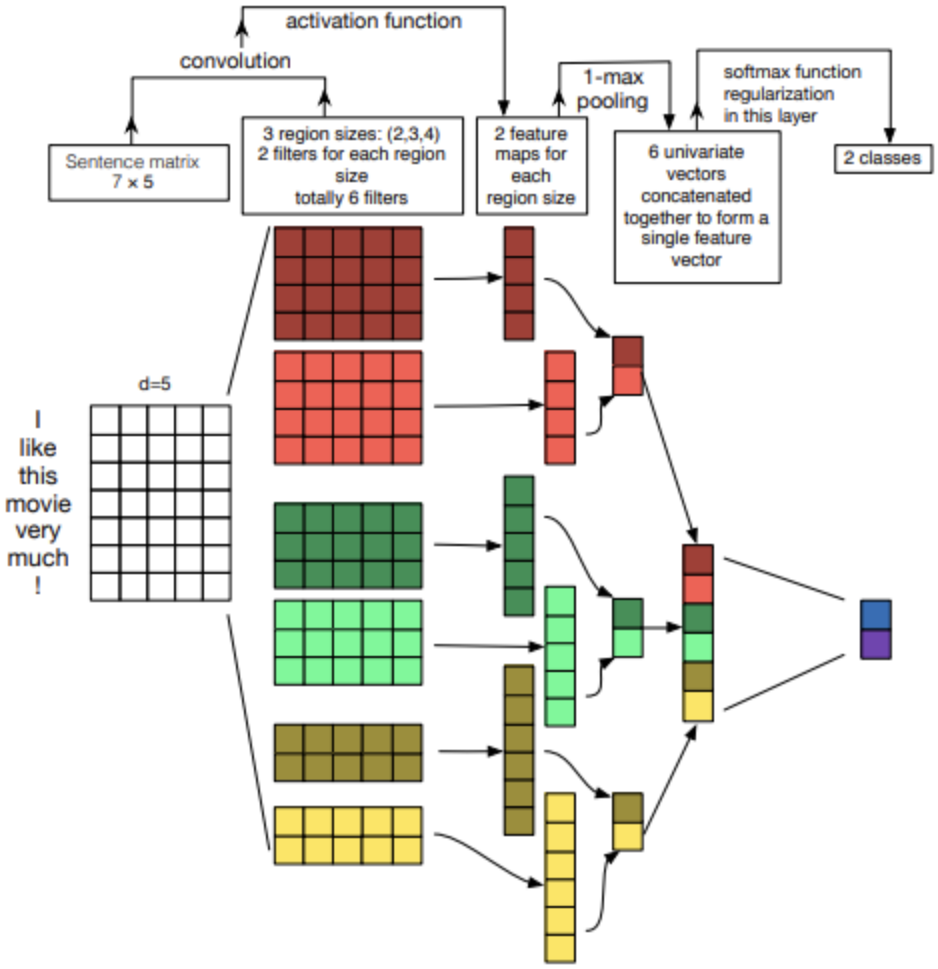
```
print("Accuracy: %f" % forest.score(eval_input, eval_label))  # 검증함수로 정확도 측정
```

```
Accuracy: 0.849400
```

```
print("Accuracy: %f" % forest.score(eval_input, eval_label))  # 검증함수로 정확도 측정
```

```
Accuracy: 0.853000
```


긍정 or 부정?
피드 포워드
네트워크
순환 신경망
나는
이
영화
좋다
워드 임베딩
입력 : 나는 이 주인공이 연기를
잘해서 이 영화 좋다


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

네이버 영화 리뷰 감정 분석

> 데이터 전처리 및 분석

github.com/e9t/nsmc

e9t/nsmc

Naver sentiment movie corpus. Contribute to e9t/nsmc development by creating an account on GitHub.

github.com

위 URL에서 데이터를 받는다.

- ratings.txt : 전체 리뷰 데이터, 약 20만 개
- ratings_train.txt : 학습 데이터. 총 15만 개
- ratings_test.txt : 평가 데이터. 총 5만 개

데이터 전처리

```
import numpy as np
import pandas as pd
import re
import json
from konlpy.tag import Okt
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
from tensorflow.python.keras.preprocessing.text import Tokenizer
```

```
DATA_IN_PATH ='./data_in/'

train_data = pd.read_csv(DATA_IN_PATH + 'ratings_train.txt', header=0, delimiter='\t', quoting=3 )

print(train_data.head())
```

```
         id                                           document  label
0   9976970                                아 더빙.. 진짜 짜증나네요 목소리      0
1   3819312                  흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나      1
2  10265843                                  너무재밓었다그래서보는것을추천한다      0
3   9045019                      교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정      0
4   6483659  사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...      1
```

```
review_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]", "", train_data['document'][0])
print(review_text)
```

```
아 더빙 진짜 짜증나네요 목소리
```

```
okt=Okt()
review_text = okt.morphs(review_text, stem=True)
print(review_text)
```

```
['아', '더빙', '진짜', '짜증나다', '목소리']
```

```
stop_words = set(['은', '는', '이', '가', '하', '아', '것', '들','의', '있', '되', '수', '보', '주', '등',
                  '한'])
clean_review = [token for token in review_text if not token in stop_words]
print(clean_review)
```

```
['더빙', '진짜', '짜증나다', '목소리']
```

```
stop_words = set(['은', '는', '이', '가', '하', '아', '것', '들','의', '있', '되', '수', '보', '주', '등',
                  '한'])
```

```
def preprocessing(review, okt, remove_stopwords = False, stop_words = []):
    # 함수의 인자는 다음과 같다.
    # review : 전처리할 텍스트
    # okt : okt 객체를 반복적으로 생성하지 않고 미리 생성후 인자로 받는다.
    # remove_stopword : 불용어를 제거할지 선택 기본값은 False
    # stop_word : 불용어 사전은 사용자가 직접 입력해야함 기본값은 비어있는 리스트

    # 1. 한글 및 공백을 제외한 문자 모두 제거.
    review_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]", "", review)

    # 2. okt 객체를 활용해서 형태소 단위로 나눈다.
    word_review = okt.morphs(review_text, stem=True)

    if remove_stopwords:
        # 불용어 제거(선택적)
        word_review = [token for token in word_review if not token in stop_words]

    return word_review
```

```
stop_words = [ '은', '는', '이', '가', '하', '아', '것', '들','의', '있', '되', '수', '보', '주', '등', '한']
okt = Okt()
clean_train_review = []

for review in train_data['document']:
    # 비어있는 데이터에서 멈추지 않도록 string인 경우만 진행
    if type(review) == str:
        clean_train_review.append(preprocessing(review, okt, remove_stopwords = True, stop_words=stop_words)
    else:
        clean_train_review.append([])  #string이 아니면 비어있는 값 추가
```

감사합니다.