Learning from Text Data

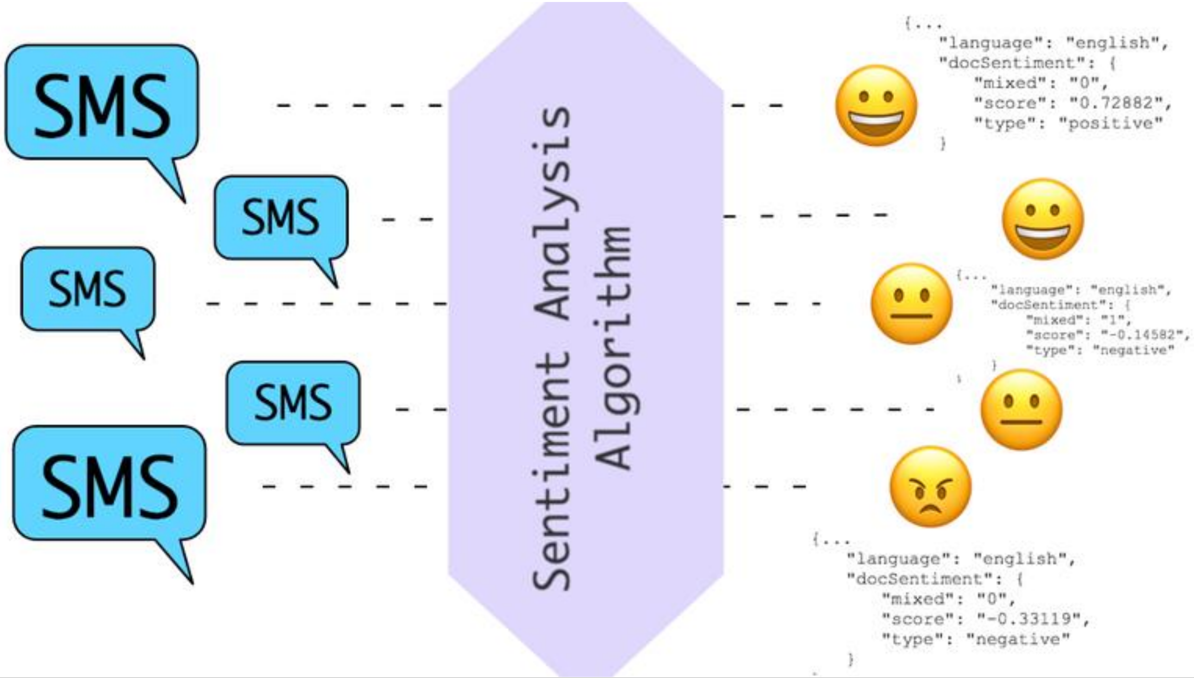
Machine Learning

Contents

- Preprocessing texts
  - Cleaning
  - Tokenizing
- Vectorization : the bag-of-words Model
  - Transforming documents into term frequency vectors
  - Transforming documents into TF-IDF vectors
- Training a model for document classification

Applying Machine Learning to Sentiment Analysis

- Sentiment analysis using text data
  - Sentiment Analysis(Opinion Mining) : subfield of natural language processing (NLP)
    - Learn how to use machine learning algorithms to classify documents based on their polarity: the attitude of the writer



Applying Machine Learning to Sentiment Analysis

- IMDb(Internet Movie Database) movie review dataset(Eng)
  - The dataset consists of 50,000 polar movie reviews that are labeled as either $positive$(1) or $negative$(0)
    - $positive$: a movie was rated with more than 6 stars on IMDb
    - $negative$: a movie was rated with fewer than 5 stars on IMDb
  - A compressed archive of the movie review dataset(84.1 MB) can be downloaded from http://ai.stanford.edu/~amaas/data/sentiment/ as a gzip-compressed tarball archive

Preprocessing texts

- Load the IMDb movie review data
  - IMDb Movie Review Dataset

review	sentiment
In 1974, the teenager Martha Moxley (Maggie Grace) moves to th	1
OK... so... I really like Kris Kristofferson and his usual easy going d	0
SPOILER Do not read this, if you think about watching that	0
hi for all the people who have seen this wonderful movie im sure	1
I recently bought the DVD, forgetting just how much I hated the	0
Leave it to Braik to put on a good show. Finally he and Zorak are	1
Nathan Detroit (Frank Sinatra) is the manager of the New York's I	1
To understand "Crash Course" in the right context, you must und	1
I've been impressed with Chavez's stance against globalisation fo	1
This movie is directed by Renny Harlin the finnish miracle. Stallor	1
I once lived in the u.p and let me tell you what. I didn't have the	0

movie_data.csv

Preprocessing texts

- Load the IMDb movie review data

```python
import pandas as pd

df = pd.read_csv('movie_data.csv', encoding='utf-8')

df.shape
```

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr...	1
1	OK... so... I really like Kris Kristofferson a...	0
2	***SPOILER*** Do not read this, if you think a...	0

```
(50000, 2)
```

Preprocessing texts

- Cleaning text data

```
# cleaning texts using regular expression
import re
def preprocessor(text):
    text = re.sub('<[^>]*>', '', text) # remove <...> (tags)
    text = re.sub('[\W]+', ' ', text) # remove all non-words
    text = text.lower() # change to lower cases
    return text

preprocessor("</a>This is a $100 TEST!!! ^^")
```

```
'this is a 100 test '
```

Preprocessing texts

- Cleaning text data

```
# review 1 text
df.loc[1, 'review']
```

```
"OK... so... I really like Kris Kristofferson and his usual easy
going delivery of lines in his movies. Age has helped him with his
soft spoken low energy style and he will steal a scene
effortlessly. But, Disappearance is his misstep. Holy Moly, this
was a bad movie! <br /><br />I must give kudos to the
cinematography and and the actors, including Kris, for trying their
darndest to make sense from this goofy, confusing story!
```

```
# review 1 text
preprocessor(df.loc[1, 'review'])
```

```
'ok so i really like kris kristofferson and his usual easy going
delivery of lines in his movies age has helped him with his soft
spoken low energy style and he will steal a scene effortlessly but
disappearance is his misstep holy moly this was a bad movie i must
give kudos to the cinematography and and the actors including kris
for trying their darndest to make sense from this goofy confusing
story
```

Preprocessing texts

- Processing documents into tokens(English)

```
text = 'The sun is shining, the weather is sweet, and she likes RUNNING!'
print(text)
# cleaning
text_prep = preprocessor(text)
print(text_prep)
```

```
The sun is shining, the weather is sweet, and she likes RUNNING!
the sun is shining the weather is sweet and she likes running
```

```
# tokenizing
import nltk
nltk.download('punkt') # tokenizer
text_tokens = nltk.word_tokenize(text_prep)
print(text_tokens)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\dwkim\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['the', 'sun', 'is', 'shining', 'the', 'weather', 'is', 'sweet', 'and', 'she',
'likes', 'running']
```

Preprocessing texts

- Processing documents into tokens(English)

```
# stemming
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()

def tokenizer_porter(text):
    text_tokens = nltk.word_tokenize(text)
    return [porter.stem(word) for word in text_tokens]

text_stems = tokenizer_porter(text_prep)
print(text_stems)
```

```
['the', 'sun', 'is', 'shine', 'the', 'weather', 'is', 'sweet', 'and', 'she',
'like', 'run']
```

Preprocessing texts

- Processing documents into tokens(English)

```
# stopwords
from nltk.corpus import stopwords
nltk.download('stopwords')
stop = stopwords.words('english')
print(stop)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\dwkim\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because',
```

```
# removing stopwords
def remove_stopwords(text):
    return [w for w in text if w not in stop]
text_stems = remove_stopwords(tokenizer_porter(text_prep))
text_stems
```

```
['sun', 'shine', 'weather', 'sweet', 'like', 'run']
```

Preprocessing texts

- POS Tagging(Enlgish)

```
# POS tagging
from nltk.tag import pos_tag
nltk.download('averaged_perception_tagger') # POS tagger
tagged_text = pos_tag(nltk.word_tokenize(text_prep))
tagged_text
```

```
[nltk_data] Error loading averaged_perception_tagger: Package
[nltk_data]     'averaged_perception_tagger' not found in index
[('the', 'DT'),
 ('sun', 'NN'),
 ('is', 'VBZ'),
 ('shining', 'VBG'),
 ('the', 'DT'),
 ('weather', 'NN'),
 ('is', 'VBZ'),
 ('sweet', 'JJ'),
 ('and', 'CC'),
 ('she', 'PRP'),
 ('likes', 'VBZ'),
 ('running', 'VBG')]
```

Preprocessing texts

- POS Tagging(Korean)

```
# Korean movie reviews
df_kor = pd.read_csv("kor_movie.csv", encoding='utf-8')
df_kor.head(3)
```

	review	sentiment
0	아 더빙.. 진짜 짜증나네요 목소리	0
1	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	너무재밓었다그래서보는것을추천한다	0

```
df_kor.shape
```

(200000, 2)

```
# review 1 text
df_kor.loc[1,'review']
```

'흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나'

```
# cleaning review 1 text
preprocessor(df_kor.loc[1,'review'])
```

'흠 포스터보고 초딩영화줄 오버연기조차 가볍지 않구나'

Preprocessing texts

- POS Tagging(Korean)

```
# tokenizing - Okt
from konlpy.tag import Okt
okt = Okt()

text = '하늘을 나는 아름다운 꿈을 꾸었습니다!'

# simple split() method is not appropriate
print('<split() method>')
print(text.split())

print('<Okt word tokenizer>')
print(okt.morphs(text))
```

<split() method>
['하늘을', '나는', '아름다운', '꿈을', '꾸었습니다!']
<Okt word tokenizer>
['하늘', '을', '나', '는', '아름다운', '꿈', '을', '꾸었습니다', '!']

Preprocessing texts

- POS Tagging(Korean)

```
# POS tagging (형태소 분석))
tagged_text = okt.pos(text)
tagged_text
```

```
[('하늘', 'Noun'),
 ('을', 'Josa'),
 ('나', 'Noun'),
 ('는', 'Josa'),
 ('아름다운', 'Adjective'),
 ('꿈', 'Noun'),
 ('을', 'Josa'),
 ('꾸었습니다', 'Verb'),
 ('!', 'Punctuation')]
```

Preprocessing texts

- POS Tagging(Korean)

```
# tokenizing - Kkma
from konlpy.tag import Kkma
kkma = Kkma()
print('<Kkma word tokenizer>')
print(kkma.morphs(text))
```

<Kkma word tokenizer>
['하늘', '을', '날', '는', '아름답', 'ㄴ', '꿈', '을', '꾸', '었', '습니다', '!']

```
# POS tagging (형태소 분석) - Kkma
tagged_text = kkma.pos(text)
tagged_text
```

[('하늘', 'NNG'),
('을', 'JKO'),
('날', 'VV'),
('는', 'ETD'),
('아름답', 'VA'),
('ㄴ', 'ETD'),
('꿈', 'NNG'),
('을', 'JKO'),
('꾸', 'VV'),
('었', 'EPT'),
('습니다', 'EFN'),
('!', 'SF')]

Preprocessing texts

- POS Tagging(Korean)

```
# Stemming
def tokenizer_porter_kor(text):
    return okt.morphs(text, norm=True, stem=True)
```

```
# tokenizing only
okt.morphs(text)
```

['하늘', '을', '나', '는', '아름다운', '꿈', '을', '꾸었습니다', '!']

```
# tokenizing + stemming
tokenizer_porter_kor(text)
```

['하늘', '을', '나', '는', '아름답다', '꿈', '을', '꾸다', '!']

```
# nouns only
okt.nouns(text)
```

['하늘', '나', '꿈']

```
# 띄어쓰기 오류인 경우도 가능
okt.nouns('아버지가방에들어가신다')
```

['아버지', '가방']

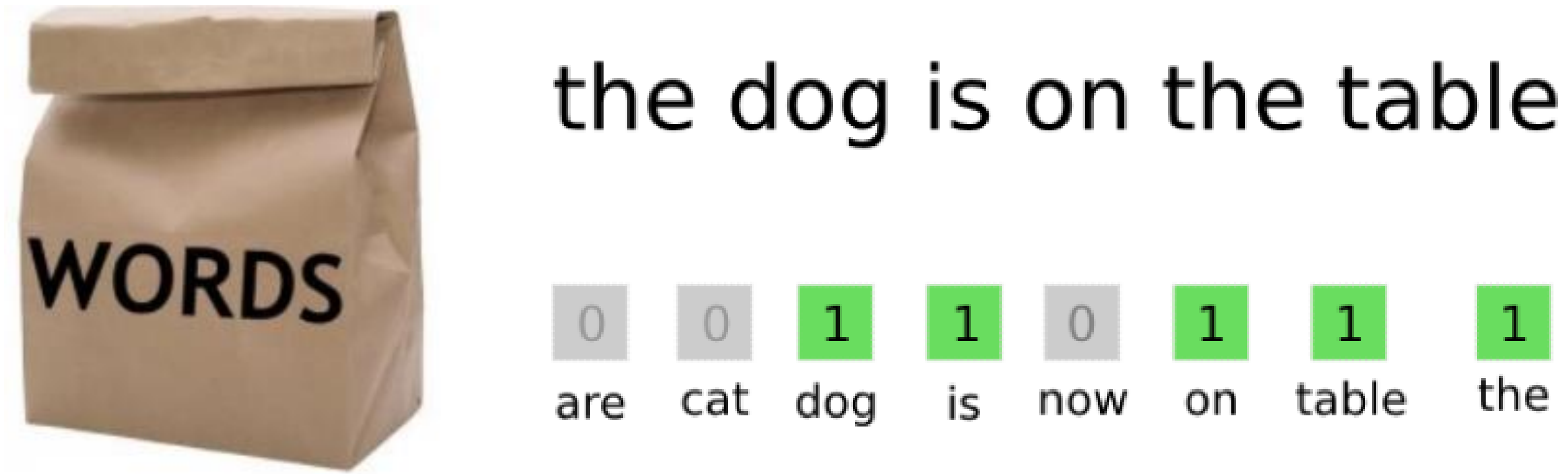
KoNLPy : Korean NLP in Python

- Korean Morphological Analyzers
  - **Okt(Open Korean Text / previous name : Twitter)**
    - Second fastest tokenizer
    - Include twitter hashtag, Korean emoticon(e.g. ㅋㅋ)
  - **Kkma(꼬꼬마)**
    - Accurate part-of-speech information
  - **Komoran(코모란)**
    - When accuracy and time matter both
  - **Hannanum(한나눔)**
    - Robust to misspelling & separated consonants and vowels(e.g. 한 -> ㅎㅏㄴ)

https://iostream.tistory.com/144
https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/05/10/postag/

Vectorization : the bag-of-words model

- Introducing the bag-of-words model
  - Bag-of-words : allows us to represent text as numerical feature vectors
    - Create a vocabulary of unique tokens – for example, words – from the entire set of documents
    - Construct a feature vector from each document that contains the counts of how often each word occurs in the particular document



Vectorization : the bag-of-words model

- Transforming documents into term frequency vectors

```
# vectorize texts - Document-Term Matrix
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer()

docs = np.array([
        'The sun is shining',
        'The weather is sweet',
        'The sun is shining, the weather is sweet, and she likes RUNNING!'])
bag = count.fit_transform(docs)
```

```
# vocabulary
print(count.vocabulary_)
```

```
{'the': 8, 'sun': 6, 'is': 1, 'shining': 5, 'weather': 9, 'sweet': 7, 'and': 0,
'she': 4, 'likes': 2, 'running': 3}
```

```
# Document-Term Matrix(DTM)
print(bag.toarray())
```

```
[[0 1 0 0 0 1 1 0 1 0]
 [0 1 0 0 0 0 0 1 1 1]
 [1 2 1 1 1 1 1 1 2 1]]
```

Vectorization : the bag-of-words model

- TF-IDF(Term Frequency – Inverse Document Frequency)
  - Frequently occurring words typically don't contain useful or discriminatory information("the", "is", etc.)

term(word) document

  - TF(Term Frequency)
    - The number of times a term occurs in a document

$$TF(t, d)$$

total number of documents

  - IDF(Inverse Document Frequency)
    - Inverse function of the number of documents in which it occurs

$$IDF(t) = \log \frac{n_d}{1 + DF(t)}$$

The number of documents d that contain the term t

  - TF-IDF(Term Frequency – Inverse Document Frequency)
    - Down-weight those frequently occurring words

$$TF\text{–}IDF(t, d) = TF(t, d) \times IDF(t)$$

dongguk UNIVERSITY

Vectorization : the bag-of-words model

- TF-IDF(Term Frequency – Inverse Document Frequency)
  - Scikit-learn' Style
    - Inverse Document Frequency(idf)

$$idf(t,d) = \log \frac{1 + n_d}{1 + df(d,t)}$$

    - TF-IDF

$$tf - idf(t,d) = tf(t,d) \times (idf(t,d) + 1)$$

    - Typically normalize the raw term frequencies before calculating the $tf - idf$, the *TfidfTransformer* normalizes the $tf - idf$ directly

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} = \frac{v}{\left(\sum_{i=1}^{n} v_i^2\right)^{1/2}}$$

Vectorization : the bag-of-words model

- Transforming documents into TF-IDF vectors

```
np.set_printoptions(precision=2)
# vectorize texts - TF-IDF Matrix
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()
docs_vector = tfidf.fit_transform(docs)
```

```
# vocabulary
print(tfidf.vocabulary_)
```

```
{'the': 8, 'sun': 6, 'is': 1, 'shining': 5, 'weather': 9, 'sweet': 7, 'and': 0,
'she': 4, 'likes': 2, 'running': 3}
```

```
# TF-IDF Matrix (normalized)
print(docs_vector.toarray())
```

```
[[0.   0.43 0.   0.   0.   0.56 0.56 0.   0.43 0.  ]
 [0.   0.43 0.   0.   0.   0.   0.   0.56 0.43 0.56]
 [0.33 0.39 0.33 0.33 0.33 0.25 0.25 0.25 0.39 0.25]]
```

Vectorization : the bag-of-words model

- Transforming documents into TF-IDF vectors

```
# vectorize texts - TF-IDF Matrix (with preprocessing, stemming, stopwords)
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=preprocessor,  # preprocessing
                        tokenizer=tokenizer_porter, # stemming
                        stop_words=stop             # removing stopwords
                        )
docs_vector = tfidf.fit_transform(docs)
```

```
# vocabulary
print(tfidf.vocabulary_)
```

```
{'sun': 3, 'shine': 2, 'weather': 5, 'sweet': 4, 'like': 0, 'run': 1}
```

```
# TF-IDF Matrix (normalized)
print(docs_vector.toarray())
```

```
[[0.   0.   0.71 0.71 0.   0.  ]
 [0.   0.   0.   0.   0.71 0.71]
 [0.48 0.48 0.37 0.37 0.37 0.37]]
```

Vectorization : the bag-of-words model

- Transforming documents into TF-IDF vectors

```
# TFIDF example
docs = np.array([
           'The sun is shining',
           'The weather is sweet',
           'The sun is shining, the weather is sweet, and she likes RUNNING!'])
n_docs = 3

# Tf-Idf score of "is" in doc 1
tf = 1
df = 3
idf = np.log((n_docs+1) / (df+1))
tfidf = tf * (idf+1)
print('tf-idf of term "is" = %.2f' % tfidf)

# Tf-Idf score of "sun" in doc 1
tf = 1
df = 1
idf = np.log((n_docs+1) / (df+1))
tfidf = tf * (idf+1)
print('tf-idf of term "sun" = %.2f' % tfidf)
```

```
tf-idf of term "is" = 1.00
tf-idf of term "sun" = 1.69
```

Vectorization : the bag-of-words model

- Overall Process

Raw Data

```
0    In 1974, the teenager Martha Moxley (Maggie Gr...
1    OK... so... I really like Kris Kristofferson a...
2    ***SPOILER*** Do not read this, if you think a...
```

↓

Cleaning

```
0    in 1974 the teenager martha moxley maggie grac...
1    ok so i really like kris kristofferson and his...
2     spoiler do not read this if you think about w...
```

↓

TF-IDF weighted DTM

```
array([[0.  , 0.  , 0.07, 0.  , 0.  , 0.  , 0.07, 0.07, 0.  , 0.  , 0.  ,
        0.  , 0.  , 0.  , 0.  , 0.07, 0.  , 0.  , 0.07, 0.  , 0.  , 0.  ,
        0.  , 0.07, 0.07, 0.  , 0.  , 0.  , 0.07, 0.  , 0.  , 0.07, 0.  ,
        0.07, 0.  , 0.  , 0.  , 0.  , 0.07, 0.05, 0.  , 0.  , 0.07, 0.  ,
        0.  , 0.07, 0.  , 0.07, 0.  , 0.14, 0.  , 0.  , 0.07, 0.07, 0.  ,
        0.14, 0.  , 0.07, 0.07, 0.  , 0.07, 0.07, 0.  , 0.  , 0.  , 0.21,
        ……
```

Training a model for document classification

- Load the IMDb movie review data

```
import pandas as pd

df = pd.read_csv('movie_data.csv', encoding='utf-8')

df.shape
```

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr...	1
1	OK... so... I really like Kris Kristofferson a...	0
2	***SPOILER*** Do not read this, if you think a...	0

```
(50000, 2)
```

Training a model for document classification

- Preprocessing

```
# use 1000 texts for training and test
X_train = df.loc[0:999, 'review'].values
y_train = df.loc[0:999, 'sentiment'].values
X_test = df.loc[49000:, 'review'].values
y_test = df.loc[49000:, 'sentiment'].values
X_train.shape
```

```
(1000,)
```

```
X_train[0]
```

```
'In 1974, the teenager Martha Moxley (Maggie Grace) moves to the high-class area
of Belle Haven, Greenwich, Connecticut. On the Mischief Night, eve of Halloween,
she was murdered in the backyard of her house and her murder remained unsolved.
Twenty-two years later, the writer Mark Fuhrman (Christopher Meloni), who is a
former LA detective that has fallen in disgrace for perjury in O.J. Simpson trial
and moved to Idaho, decides to investigate the case with his partner Stephen
Weeks (Andrew Mitchell) with the purpose of writing a book. The locals squirm and
d …
```

Training a model for document classification

- Preprocessing

```
# vectorize to TF-IDF Matrix
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(strip_accents=None,
                lowercase=False,
                preprocessor=preprocessor,
                # Below two steps are needed, but it takes long time,
                # so, We'll skip this processes
                #tokenizer=tokenizer_porter,
                #stop_words=stop,
                max_df=0.1, # ignore terms occured in more than 10% of docs
                (stop words)
                )
X_train_vector = tfidf.fit_transform(X_train)
X_test_vector = tfidf.transform(X_test)
```

```
# automatic stop words
print(tfidf.stop_words_)
```

```
{'always', 'time', 'if', 'you', 'better', 'any', 'films', 'more', 'does', 'very',
'look', 'far', 'years', 'something', 'has', 'real', 'that', 'even', 'then',
'actually', 'find', 'not', 'story', 'bad', 'your', 'so', 'way', 'while', 'saw',
'work', 've', 'cast', 'show', 'best', 'from', 'just', 'between', 'will', 'watch',
'such', 'on', 'are', 'into', 'in', 'watching', 'because', 'role', 'was', 'seen',
'first', 'scene', 'director', 'is', 'much', 'didn', 'only', 'up', 'big',
'thought', 'anything', 'old', 'love', 'f
```

Training a model for document classification

- Preprocessing

```
# data dimension
X_train_vector = X_train_vector.toarray()
X_test_vector = X_test_vector.toarray()
X_train_vector.shape
```

```
(1000, 18452)
```

Training a model for document classification

- Preprocessing

```
# vectorize to TF-IDF Matrix - remove rare terms too
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=preprocessor,
                        # Below two steps are needed, but it takes long time,
                        # so, We'll skip this processes
                        #tokenizer=tokenizer_porter,
                        #stop_words=stop,
                        max_df=0.1, # ignore terms occured in more than 10% of docs
                        (stop words)
                        min_df=10 # ignore terms occured in less than 10 docs
                        )
X_train_vector = tfidf.fit_transform(X_train)
X_test_vector = tfidf.transform(X_test)
```

```
# data dimension
X_train_vector = X_train_vector.toarray()
X_test_vector = X_test_vector.toarray()
X_train_vector.shape
```

```
(1000, 1827)
```

Training a model for document classification

- Preprocessing

```
# text 0
print(X_train[0])
```

```
In 1974, the teenager Martha Moxley (Maggie Grace) moves to the high-class area
of Belle Haven, Greenwich, Connecticut. On the Mischief Night, eve of Halloween,
she was murdered in the backyard of her house and her murder remained unsolved.
Twenty-two years later, the writer Mark Fuhrman (Christopher Meloni), who is a
former LA detective that has fallen in disgrace for perjury in O.J. Simpson trial
and moved to Idaho, decides to investigate the case with his partner Stephen
Weeks (Andrew Mitchell) with the purpose of writing a book. The locals squirm and
do not welcome them, but with the support of the retired detect …
```

```
# tfidf vector of text 0
import numpy as np
np.set_printoptions(threshold=np.inf)
print(X_train_vector[0])
```

```
[0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.11 0.   0.   0.   0.08 0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.   0.   0.   0.   0.12 0.   0.   0.   0.   0.   0.   0.   0.  …
```

Training a model for document classification

- Logistic Regression

```
# train using Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty="l2", verbose=1)
lr.fit(X_train_vector, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=-1, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=1,
                   warm_start=False)
```

```
# train score
lr.score(X_train_vector, y_train)
```

```
0.964
```

```
# test score
lr.score(X_test_vector, y_test)
```

```
0.811
```

```
# sentiment prediction example
tweets = ["this movie is garbage",
      "I loved it very much",
      "what a fantastic film!"]
tweets_tfidf = tfidf.transform(tweets)
lr.predict(tweets_tfidf)
```

```
array([0, 1, 1], dtype=int64)
```

Training a model for document classification

- Decision Tree

```
# train using Decision Tree
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=20)
tree.fit(X_train_vector, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# train score
tree.score(X_train_vector, y_train)
```

```
0.892
```

```
# test score
tree.score(X_test_vector, y_test)
```

```
0.717
```

Training a model for document classification

- Decision Tree

```
# finding most important terms
importances = tree.feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(10):
print("%2d. %-30s %f" % (f+1,
[w for w, n in tfidf.vocabulary_.items() if n == indices[f]],
importances[indices[f]]))
```

```
 1. ['worst']                          0.110662
 2. ['waste']                          0.069570
 3. ['awful']                          0.065943
 4. ['boring']                         0.051438
 5. ['terrible']                       0.043975
 6. ['crap']                           0.041877
 7. ['script']                         0.041736
 8. ['worse']                          0.029585
 9. ['flick']                          0.028862
10. ['wonderful']                      0.028122
```

Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week10_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week10_StudentNum_Name.ipynb”
  Ex) **Week10_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz : Naver Movie Review Classification(Korean)

- Use movie review dataset "kor_movie.csv"
  - class : 0, 1 (neg, pos)
  - data size : 200,000 - use first 1,000 texts
  - use 70% as training set
  - Preprocess text using Okt to make TFIDF vectors - ignore terms occured in more than 10% of texts
  - Build model using Logistic Regression, Decision Tree, and Neural Network. Check the accuracies
  - Find most important 20 terms using DT

https://github.com/e9t/nsmc