5. Sentiment Analysis

- Lexicon based
- machine learning based

Sogang University Department of Business Administration
Professor Myung Suk Kim

Sentiment analysis

◆ Uses of Sentiment Analysis

- It is necessary to analyze the emotions and sentiments contained in the text using machines without bias.
- Spam or malicious messages can be extracted and removed.
- A chatbot can analyze the emotional level of a message sent by a human and create a human-like response corresponding to it.

◆ Sentiment analysis approach

The following two methods are used

(1) using human-written rule-based algorithms;

It is based on a dictionary (lexicon) containing pairs of specific words and sentiment scores, and the VADER algorithm (Scikit Learn) has exist

(2) using machine learning models where computers learn directly from data

Create rules by training a machine learning model using a set of sentences or documents with

Sentiment analysis

1. English Emotional Glossary

(One) AFINN: assigns a score between -5 and 5 (positive/ negative)

(2) Bing : positive / negative of words in binary format classification

(3) NRC: 10 types emotion Use a glossary of terms

{fear, anger, anticipation, trust, surprise, positive, negative, sadness, disgust, joy}

http://jonathansoma.com/lede/algorithms-2017/classes/more-text-analysis/nrc-emotional-lexicon/

(4) VADER

Score calculation for

(negative+neutral+positive =1)

(https://towardsdatascience.com/religion-on-twitter-5f7b84062304)

The compound score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive).

- Typical threshold values:

positive sentiment: compound score >= 0.05

neutral sentiment: (compound score > -0.05) and (compound score < 0.05)

negative sentiment: compound score <= -0.05

(5) SentiWordNet

SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity.

https://github.com/aesuli/SentiWordNet

Sentiment analysis

(6) TextBlob

It provides common NLP tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. (https://textblob.readthedocs.io/en/dev/)

# Negative : polatiry [-1 to 1]

# Subjectivity : subjectivity [objective:0 ~subjective:1]

(7) SentimentR

2. Emotional terms in Korean dictionary

For Korean, a free downloadable glossary is available. Incomplete .

the Korean-ko-NRC-Emotion-Intensity-Lexicon-v1.txt file from

google Applying the English sentiment analysis method using a translator is one method .

Sentiment Analysis : Lexicon

1. IMDB movie review download and sentiment analysis (Using the glossary)

```
!pip install Afinn

import pandas as pd
import nltk
from afinn import afinn
nltk.download (' stopwords ')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
import numpy as np
import matplotlib.pyplot as plt

# Download below data from Kaggle
# https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/IMDB Dataset.csv"
review = pd.read_csv ( file_name , engine="python")
review. head (10)
```

Sentiment Analysis : Lexicon

2. AFINN Lexicon

Calculate sentiment score using AFINN glossary (AFINN: consists of -5 to 5 points for each word)

```
afinn = Afinn ()
pos_review = review['review'][1] # Print only one positive sentence
neg_review = review['review'][3] # print only one negative sentence
print( afinn. score ( pos_review ))
print( afinn. score ( neg_review ))

afn = Afinn (emoticons=True)

# Parse only the first n sentences
n=100
index = []
for row in review['review'][0:n]:
    index.append (row)
print( len (index), 'Predicted Sentiment polarity:', afn. score (row))
```

Sentiment Analysis : Lexicon

3. NRC Vocabulary

```
# Sentiment classification using NRC terminology (NRC: classifying each word into 10 sentiments )
Download NRC : Click (only the NRC Word-Emotion Association Lexicon) at
https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
Using NRC-Emotion-Lexicon-Wordlevel-v0.92.txt file

# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount('/content/gdrive')

# Specify file path
file_name1 = "/content/gdrive/My Drive/Colab Notebooks/Textmining/download/NRC-Emotion-Lexicon-Wordlevel-
v0.92.txt"

NRC = pd.read_csv(file_name1, engine="python", header=None, sep="₩t")
NRC.head(20)

NRC = NRC[(NRC != 0).all(1)]
NRC.head(10)
# Column 0 : ( Applicable word ), Column 1 : (10 sentiments ), Column 2 : ( Applicable )
# Example ) abacus corresponds to the emotion of trust , and abadon corresponds to the three types of fear, negative,
and sadness .

# reset index number
NRC = NRC. reset_index (drop=True)
NRC. head (10)

list(NRC[0])
```

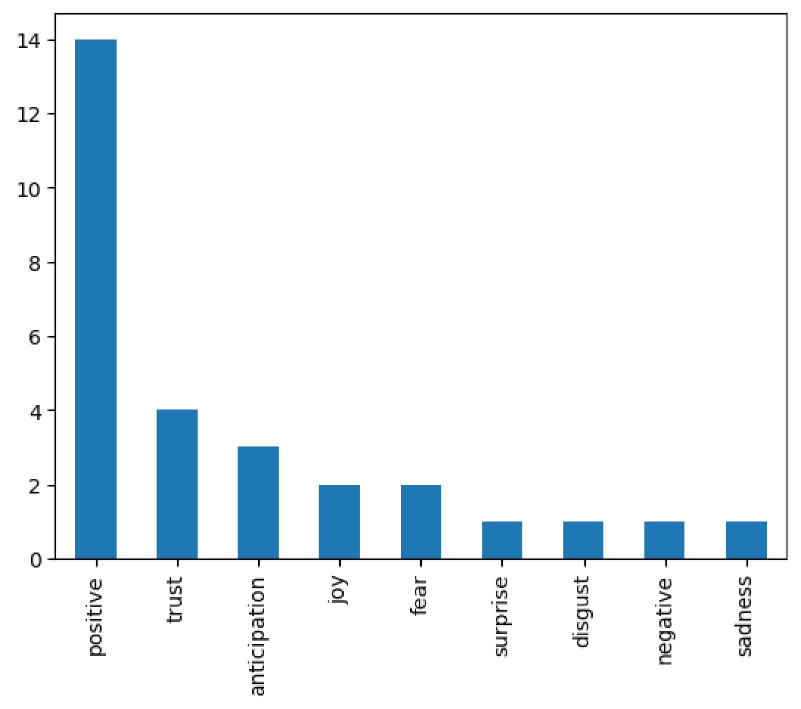
Sentiment Analysis : Lexicon

```
# Sentiment analysis using NRC for a specific sentence 1
tokenizer = RegexpTokenizer ('[₩w]+')
stop_words = stopwords.words (' english ')
p_stemmer = PorterStemmer ()

raw = pos_review.lower () # the sentence you want to analyze
tokens = tokenizer. tokenize (raw)
stopped_tokens = [i for i in tokens if not i in stop_words] # remove stop words
match_words = [x for x in stopped_tokens if x in list(NRC[0])] # match with dictionary

emotion=[]
for i in match_words:
    temp = list(NRC.iloc[np.where(NRC[0] == i)[0],1])
    for j in temp:
        emotion.append (j)

sentiment_result1 = pd.Series (emotion). value_counts ()
print(sentiment_result1, sentiment_result1.plot.bar())
```

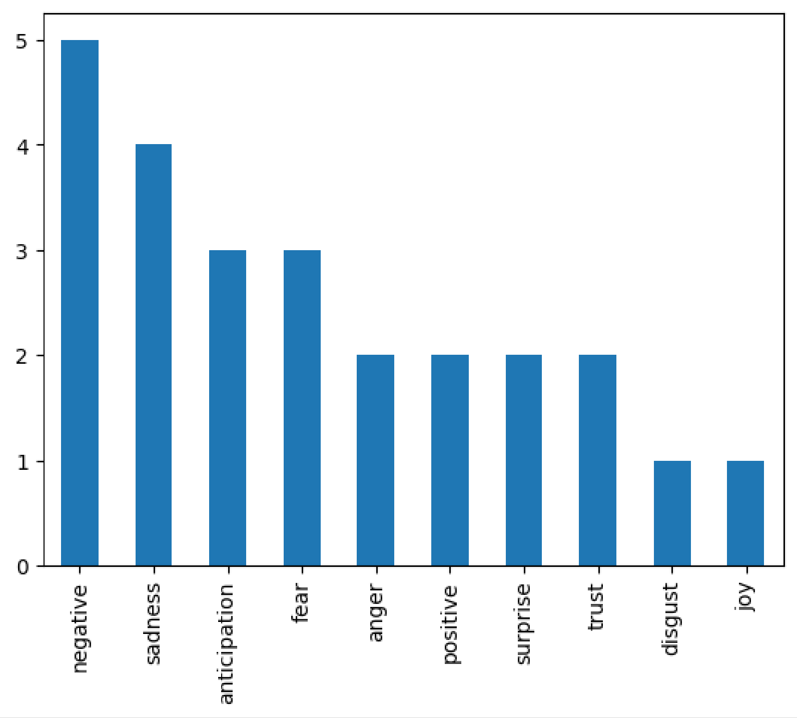


Sentiment Analysis : Lexicon

```
# Sentiment analysis using NRC for a specific sentence 2
raw = neg_review.lower () # the sentence you want to analyze
tokens = tokenizer. tokenize (raw)
stopped_tokens = [i for i in tokens if not i in stop_words] # remove stop words
match_words = [x for x in stopped_tokens if x in list(NRC[0])] # match w/ dictionary

emotion=[]
for i in match_words:
    temp = list(NRC.iloc[np.where(NRC[0] == i)[0],1])
    for j in temp:
        emotion.append(j)

sentiment_result2 = pd.Series(emotion).value_counts()
print(sentiment_result2, sentiment_result2.plot.bar())
```



Sentiment Analysis : Lexicon

4. VADER Lexicon

```
#https://statkclee.github.io/nlp2/nlp-sentiment.html

! pip install vaderSentiment
import nltk
nltk.download (' vader_lexicon ')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

analyser = SentimentIntensityAnalyzer()
example = review['review'][0]
score = analyser.polarity_scores(example)
print(score)

def vader_polarity(text):
    """ Transform the output to a binary 0/1 result """
    score = analyser.polarity_scores(text)
    return 1 if score['pos'] > score['neg'] else 0

# Parse only the first n sentences
n=10
index = []
for row in review['review'][0:n]:
    index.append (row)
print( len (index), 'Predicted Sentiment polarity:', analyser. polarity_scores (row))
print( len (index), 'Predicted Sentiment polarity Class:', vader_polarity (row))
```

Sentiment Analysis : Lexicon

5. SentiWordNet lexicon:

```
# https://statkclee.github.io/nlp2/nlp-sentiment.html
import nltk
nltk.download('wordnet')
nltk.download('sentiwordnet')

from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn
from nltk import sent_tokenize, word_tokenize, pos_tag
lemmatizer = WordNetLemmatizer()

def penn_to_wn(tag):
    """Convert between the PennTreebank tags to simple Wordnet tags"""
    if tag.startswith('J'):
        return wn.ADJ
    elif tag.startswith('N'):
        return wn.NOUN
    elif tag.startswith('R'):
        return wn.ADV
    elif tag.startswith('V'):
        return wn.VERB
    return None

 def clean_text(text):
    text = text.replace("<br />", " ") #     text = text.decode("utf-8")
    return text
```

Sentiment Analysis : Lexicon

```
def swn_polarity(text):
    """Return a sentiment polarity: 0 = negative, 1 = positive"""
    sentiment = 0.0
    tokens_count = 0
    text = clean_text(text)
    raw_sentences = sent_tokenize(text)
    for raw_sentence in raw_sentences:
        tagged_sentence = pos_tag(word_tokenize(raw_sentence))
        for word, tag in tagged_sentence:
            wn_tag = penn_to_wn(tag)
            if wn_tag not in (wn.NOUN, wn.ADJ, wn.ADV):
                continue
            lemma = lemmatizer.lemmatize(word, pos=wn_tag)
            if not lemma:
                continue
            synsets = wn.synsets(lemma, pos=wn_tag)
            if not synsets:
                continue
            synset = synsets[0] # Take the first sense, the most common
            swn_synset = swn.senti_synset(synset.name())
            sentiment += swn_synset.pos_score() - swn_synset.neg_score()
            tokens_count += 1
    if not tokens_count:   # judgment call ? Default to positive or negative
        return 0
    if sentiment >= 0:  # sum greater than 0 => positive sentiment
        return 1
    return 0   # negative sentiment
```

Sentiment Analysis : Lexicon

```
# Parse only the first n sentences
n=10
index = []
for row in review['review'][0:n]:
    index.append (row)
#print( len (index), 'Sentiment:', row['sentiment'])
print('Predicted Sentiment polarity:', swn_polarity (row))
```

Sentiment Analysis : Lexicon

6. TextBlobs

```
#https://textblob.readthedocs.io/en/dev/
# positive~negative: polatiry [-1 ~ 1]
# subjectivity [objective:0 ~subjective:1]

!pip install -U textblob
!python -m textblob.download_corpora

from textblob import TextBlob
text = '''
The titular threat of The Blob has always struck me as the ultimate movie monster: an insatiably hungry, amoeba-
like mass able to penetrate virtually any safeguard, capable of--as a doomed doctor chillingly describes it--
"assimilating flesh on contact.
Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario proposed by technological theorists fearful of
artificial intelligence run rampant.
'''
blob = TextBlob(text)
blob.tags           # [('The', 'DT'), ('titular', 'JJ'),    #  ('threat', 'NN'), ('of', 'IN'), ...]

blob.noun_phrases   # WordList(['titular threat', 'blob',
# 'ultimate movie monster', # 'amoeba-like mass', ...])

# Calculate polarity for 2 sentences
for sentence in blob. sentences :
   print( sentence. sentiment. polarity )
```

Sentiment Analysis : Lexicon

```
#https://stackabuse.com/sentiment-analysis-in-python-with- textblob
from textblob import TextBlob

# sentence 1
sentence1 = '''The platform provides universal access to the world's best education, partnering with top universities an
d organizations to offer courses online.'''

# polarity and subjectivity
analysis = TextBlob(sentence1).sentiment
print(analysis)

analysisPol = TextBlob(sentence1).polarity      # 긍부정
analysisSub = TextBlob(sentence1).subjectivity # 주객관성
print(analysisPol)
print(analysisSub)



# sentence2
sentence2 = '''This phone's camera image is very good. But, the life time of battery is too short'''
analysis = TextBlob(sentence2).sentiment
print(analysis)

blob2 = TextBlob(sentence2)
for sentence in blob2.sentences:
    print(sentence.sentiment)
```

Sentiment Analysis : Lexicon

7. Hangul Sentiment Analysis (NRC Korean processing)

# Sentiment classification using NRC terminology (NRC: classifying each word into 10 sentiments)
Download NRC : Click (only the NRC Word-Emotion Association Lexicon) at
https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
Using Korean-ko-NRC-Emotion-Intensity-Lexicon-v1.txt file

```
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name2 = "/content/gdrive/My Drive/Colab Notebooks/Textmining/download/Korean-ko-NRC-Emotion-Intensity-
Lexicon-v1.txt"
NRC = pd.read_csv(file_name2, engine="python", header=None, sep="₩t")
NRC.head(20)

NRC1 = NRC.drop([0],axis=0)  # delete first row
NRC1.head(10)
NRC2 = NRC1.drop([0],axis=1)  # delete first column
NRC2.head(10)
list(NRC2[1])
########################################
NRC2.iloc[:,0:1] # iloc [x, y] gets the value corresponding to row x and column y of the dataframe
NRC2.loc[6154] # loc [x] gets the value corresponding to the index x ( row number ) of the data frame

NRC3 = NRC2.iloc[:,0:2]
NRC4 = NRC3.drop_duplicates()
len (NRC4[1])
print(NRC4[1])
```

Sentiment Analysis : Lexicon

```
# Sentiment analysis
review1="재미없다 지루하고. 같은 음식 영화인데도 바베트의 만찬하고 넘 차이남....바베트의 만찬은 이야기도 있고 음식
보는재미도 있는데 ; 이건 볼게없다 음식도 별로 안나오고, 핀란드 풍경이라도 구경할랬는데 그것도 별로 안나옴 ——"

import numpy as np
tokenizer = RegexpTokenizer ('[₩w]+')
tokens = tokenizer. tokenize (review1[22])
print(tokens)

match_words = [x for x in tokens if x in list(NRC4[1])] # 사전과 매칭
print(match_words)
len(match_words)

emotion=[]
for i in match_words:
    temp = list(NRC4.iloc[np.where(NRC4[1]  == i)[0],1])
    for j in temp:
        emotion.append(j)
print(emotion)
########
NRC4.iloc[ np.where (NRC4[1] == match_words [0])[0],0]
NRC4.iloc[ np.where (NRC4[1] == match_words [0])[0],1]
list(NRC4.iloc[ np.where (NRC4[1] == match_words [0])[0],1])[0]
NRC4.loc[6154]
########

sentiment_result1 = pd.Series (emotion). value_counts ()
print(sentiment_result1, sentiment_result1.plot.bar())
```

Sentiment Analysis : Lexicon

8. Google translate (Korean to English)

```
#https://translate.google.com/?hl=en&tab=TT
# data limit: 10MB

text = [
 'Ah dubbing.. Really annoying voice',
 'It was so funny, so I recommend watching it',
 'Its a prison story ..Honestly, its not fun .. Rating adjustment',
 'A movie with Simon Peggs humorous acting that stood out! Kirsten Dunst, who only looked old in Spider-
Man, looked so pretty',
 'A movie for 8-year-
olds who have just started walking from the age of 3 to the 1st year of elementary school. Hahaha... Its not even wort
h it.',
 'I couldnt properly revive the tension of the original.',
 'One of the few movies that is interesting even without action',
 'Why is the rating so low? Its quite a sight to behold. Are you too accustomed to Hollywood-style glamour?',
 'Gyan Infinite is the best. Its really cool ♥',
 'Every time I see it I will die of tears! The nostalgic stimulation of the 90s!! Jinho Heo is a master of emotionally restra
ined melodies~'
 'I almost ran out when I crossed the crosswalk with my hands raised crying, I cant show off Lee Beom-soos acting',
 'Goodbye Lenin, I understand that this is plagiarism, but why does it get less interesting the further back you go',
 'This is a really good mix of real casting and refreshing content that isnt sticky!!♥',
'Excuse for the looter, ya . Those guys aren't good guys at all.',
'It seems to have a profound meaning. Its never just a movie in which students play with their teachers',
'I would like to say that it is a masterpiece, not an ordinary movie.',
'The subject is good, but it gets boring from the middle'
]
```

Sentiment Analysis : Lexicon

```
# Sentiment classification using NRC terminology (NRC: classifying each word into 10 sentiments )
Download NRC : Click (only the NRC Word-Emotion Association Lexicon) at
https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
Using NRC-Emotion-Lexicon-Wordlevel-v0.92.txt file

# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
file_name1 = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/NRC-Emotion-Lexicon-Wordlevel-v0.92.txt"

NRC = pd.read_csv (file_name1, engine="python", header=None, sep ="₩t")
NRC. head (20)

NRC = NRC[(NRC != 0).all(1)]
NRC. head (10)
# Column 0 : ( Applicable word ), Column 1 : (10 sentiments ), Column 2 : ( Applicable )
# Example ) abacus corresponds to the emotion of trust , and abadon corresponds to the three types of fear, negative, and sadness .

# reset index number
NRC = NRC. reset_index (drop=True)
NRC. head (10)

list(NRC[0])
```

Sentiment Analysis : Lexicon

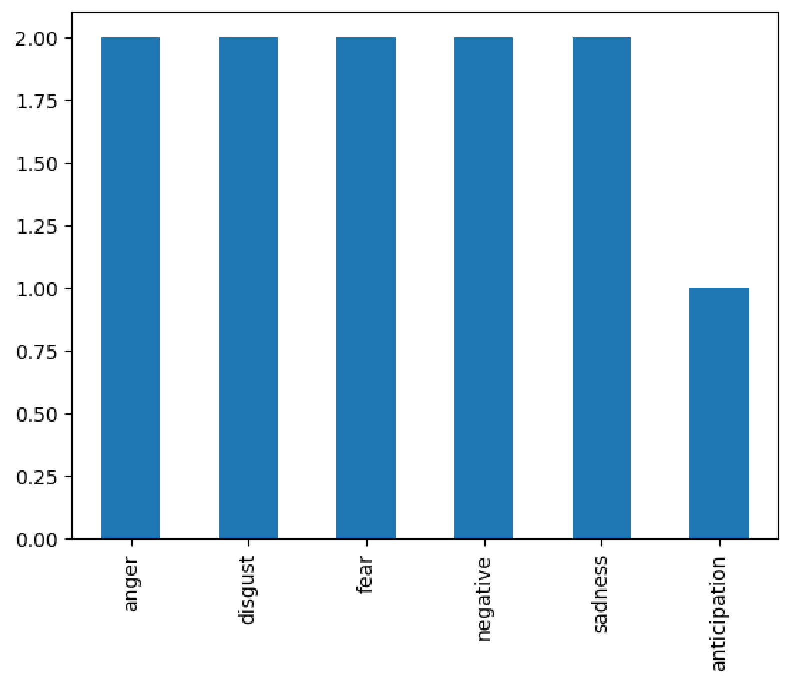
```
# Sentiment analysis using NRC for a specific sentence 1
tokenizer = RegexpTokenizer('[₩w]+')
stop_words = stopwords.words('english')
p_stemmer = PorterStemmer()

text1 = text[19]
raw = text1.lower()   # target sentence
tokens = tokenizer.tokenize(raw)
stopped_tokens = [i for i in tokens if not i in stop_words] # remove stop words
match_words = [x for x in stopped_tokens if x in list(NRC[0])] # match w/ dictionary

emotion=[]
for i in match_words:
    temp = list(NRC.iloc[np.where(NRC[0] == i)[0],1])
    for j in temp:
        emotion.append(j)

sentiment_result1 = pd.Series(emotion).value_counts()

print(sentiment_result1, sentiment_result1.plot.bar())
```



Sentiment Analysis : Machine Learning

❖ Example (Normalized TF-IDF)

Doc1: the fox chases the rabbit
Doc2: the rabbit ate the cabbage
Doc3: the fox caught the rabbit

	Doc1	Doc2	Doc3
the	1.70084	1.70084	1.70084
fox	0.37796	-0.944911	0.37796
rabbit	0.37796	0.37796	0.37796
chases	0.37796	-0.944911	-0.944911
caught	-0.944911	-0.944911	0.37796
cabbage	-0.944911	0.37796	-0.944911
ate	-0.944911	0.37796	-0.944911

	the	fox	rabbit	chases	caught	cabbage	ate
Doc1	1.70084	0.37796	0.37796	0.37796	-0.944911	-0.94491	-0.944911
Doc2	1.70084	-0.944911	0.37796	-0.944911	-0.944911	0.37796	0.37796
Doc3	1.70084	0.37796	0.37796	-0.944911	0.37796	-0.944911	-0.944911

Sentiment Analysis : Machine Learning

❖**Model evaluation method**

For binary response variable

noon classification		prediction group	
		Y=1	Y=0
real collective	Y=1	f11 (true positive)	f12(false negative)
	Y=0	f21 (false positive)	f22(true negative)

① Accuracy or correct classification rate:

As a correctly predicted proportion of the total The closer (f11+f22)/n is to 1 , the better.

② Sensitivity:

Proportion of predicting (classifying) what is actually true as true (true positive).

The closer f11/(f11+f12) is to 1 , the more desirable it

③ Specificity :

Proportion of correctly predicting (classifying) true false as false (true negative)

The closer f22/(f21+f22) is to 1 , the better.

Correct classification rate = (f11+f12)/n X sensitivity + (f21+f22)/n X specificity

Error rate = 1 – accuracy

Sentiment Analysis : Machine Learning

❖ Considerations when comparing models

(1) Cross validation

Training (train) set to be used for model construction and evaluation (test) to be used for prediction evaluation.

If the amount of data is large enough , for prediction : for evaluation Randomly divided by 50:50 and applied.

a) K - fold method

① If the amount of data is not sufficient, the entire datasets are divided into K pieces.

Building a model with k-1 pieces, and make predictions on the remaining one piece.

② Evaluation by repeating k times-> obtain the average prediction performances.

b) Leave one out method

① Thinking that K = n , proceed . In other words , after excluding one data, building a model with the rest, in one execution of predictions.

② Evaluation by repeating n times-> obtain the average prediction performances.

Sentiment Analysis : Machine Learning

❖ Considerations when comparing models

(2) How to use the ROC (Receiver Operating Characteristic) curve

(mainly used for discrete response variables (0, 1))

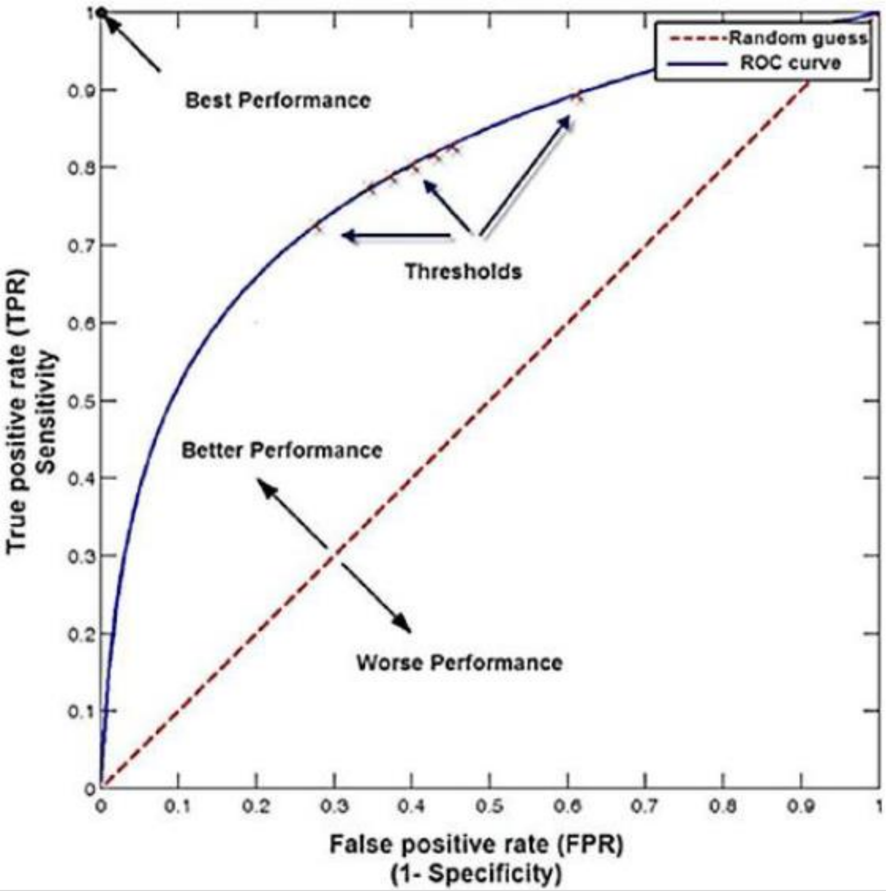
- After sorting the predicted values (mainly continuous variables) for the validation data in descending order, then selecting the reference value (value between 0 and 1) for classification, a confusion matrix is obtained.

- The ROC curve is based on a graph drawn by converting values between 0 and 1, obtaining false positive (1- specificity) and true positive (sensitivity) values from the confusion matrix , and connecting these values on the X, Y coordinates. The area under the curve is called the c statistic or AUC (area under curve), and the lar ger the area, the better the performance of the model.

Sentiment Analysis : Machine Learning

❖ Considerations when comparing models

(2) ROC (Receiver Operating Characteristic) curve (continued)



➢ AUC 점수 체계(rule of thumb)

0.9~1.0: 탁월하다

0.8~0.9: 뛰어나다

0.7~0.8: 괜찮다

0.6~0.7: 형편없다

0.5~0.6: 가치없다

Sentiment Analysis : ML

1. Sentiment analysis using IMDB movie review (using word appearance frequency)

```
!pip install Afinn

import pandas as pd
import nltk
from afinn import Afinn
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount ('/content/ gdrive ‘)


# Specify the file name and save path
file_name = "/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /download/IMDB Dataset.csv"
review = pd.read_csv ( file_name , engine="python")
review. head (10)
len (review)
review['review'][0]
```

Sentiment Analysis : ML

```
#https://duckkkk.com/entry/ Kaggle -IMDB-%EA%B0%90%EC%A0%95-%EB%B6%84%EC%84%9D-Part-1
########## Preprocessing ##########
# Install module to remove HTML tags
from bs4 import BeautifulSoup

# Analyze only the first n reviews
n = 100 # there are 50000 total
reviews = []
for row in review['review'][0:n]:
review1 = BeautifulSoup (row, "html5lib"). get_text ()
  reviews. append (review1)

print(reviews) # get
len (reviews)

# Install the module to use regular expressions
import re

# ^: means start , extracts only letters starting with lowercase letters of the alphabet
review_list = []
for row1 in reviews:
review2 = re.sub ('[^a- zA -z]',' ',row1)
review3 =review2.lower() # Convert all to lower case
  review_list. append (review3)

print( review_list )
len ( review_list )
```

Sentiment Analysis : ML

```
########## Handling Tokens ##########
token_list = []
for row2 in review_list :
review4 = row2.split() # Tokenize
 token_list. append (review4)

print( token_list )
len(token_list)

# remove stopwords
sentence_words = [w for w in token_list if not w in stopwords.words('english')]
len(sentence_words)
type(sentence_words)

clean_review = []
for sentence in sentence_words:
  s = ' '
  clean_review.append ( s.join (sentence))

clean_review
```

Sentiment Analysis : ML

```
########### Vectorize documents by frequency of occurrence of words : CounterVectorizer #############
### Convert tokens from reviews to features
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

# Change the parameter value differently from the tutorial
vectorizer = CountVectorizer (analyzer = 'word’,
                              tokenizer = None,
                              preprocessor=None,
                              stop_words = None;
                              min_df = 2, # minimum number of documents for token to appear
                              ngram_range =(1, 1), # ngra_range = ( min , max )
                              max_features = 20000)

# Improved to use pipelines for speed improvement
pipeline = Pipeline([(' vect ', vectorizer ),])

# vectorize
data_features = pipeline.fit_transform ( clean_review )
data_features.shape
data_features.toarray ()

vocab = vectorizer.get_feature_names_out ()

import pandas as pd
df = pd.DataFrame ( data_features.toarray ())
print( df )
```

Sentiment Analysis : ML

```
# random forest classification
from sklearn.ensemble import RandomForestClassifier

k = data_features.shape [0]
forest = RandomForestClassifier ( n_estimators = 100, n_jobs = -1, random_state =2018)
forest = forest. fit ( data_features , review['sentiment'][0:k])

# k-fold test
from sklearn.model_selection import cross_val_score

# for each k-fold, ROC AUC value
cross_val_score(forest, data_features,review['sentiment'][0:k], cv=10, scoring='roc_auc')

# for all k-fold, ROC AUC average
score = np.mean(cross_val_score(forest, data_features,review['sentiment'][0:k], cv=10, scoring='roc_auc'))
print(score)
```

Sentiment Analysis: ML

2. Sentiment analysis using IMDB movie reviews (Using TF-IDF vector)

```
from sklearn.model_selection import train_test_split
n = 1000
review = review[0:n]

x_input = review['review']
y_output = review['sentiment']

x_train , x_test , y_train , y_test = train_test_split ( x_input , y_output , stratify=y, test_size =0.2, random_state =15)

print( x_train.shape , x_test.shape ) # Check ratio of training set and test set
np.unique ( y_train , return_counts =True) # Check the targets ( labels ) of the training set


from sklearn.feature_extraction.text import TfidfVectorizer
stop_words = stopwords.words (' english ')
len ( stop_words )
stop_words

# Convert to document - word matrix via TF-IDF weights
vect = TfidfVectorizer(stop_words=stop_words).fit(x_train)
x_train_vectorized = vect.transform(x_train)

x_train_vectorized
print(x_train_vectorized)
```

Sentiment Analysis: ML

(1) Logistic regression

```
from sklearn.linear_model import LogisticRegression, SGDClassifier

model = LogisticRegression()
model.fit(x_train_vectorized, y_train)
print(model.score(x_train_vectorized, y_train))

print(model.score(vect.transform(x_test), y_test))
```

(2) Decision tree

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier ()
clf.fit ( x_train_vectorized , y_train )
print( clf. score ( x_train_vectorized , y_train ))

print( clf. score ( vect. transform ( x_test ), y_test ))
```

Sentiment Analysis : ML

3. Naver movie emotional corpus data (using word appearance frequency)

```
#https://cyc1am3n.github.io/2018/11/10/classifying_korean_movie_review.html

# Download the data below from github # https://github.com/e9t/nsmc/
# ratings_train.txt, ratings_test.txt
# Preprocessing command to save file in colab ( set path )
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
folder_name = "/content/gdrive/My Drive/Colab Notebooks/Textmining/download/"

def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('₩t') for line in f.read().splitlines()]
        # in txt file, remove header except for (id document label)
      data = data[1:]
    return data

train_data = read_data (folder_name+'ratings_train.txt')
test_data = read_data (folder_name+'ratings_test.txt')

train_data [0:10]
len ( train_data )
##### Resizing data : try using only part of it
train_data = train_data [0:800]
test_data = test_data [0:200]
```

Sentiment Analysis : ML

```
#konlpy _ practice
import konlpy
konlpy .__version__

from konlpy.tag import Okt
okt = Okt ()
print( okt.pos (u'이 밤 그날의 반딧불을 당신의 창 가까이 보낼게요'))



import json
import os
from pprint import pprint

###### Part of speech extraction using konlpy 's okt.pos
def tokenize(doc): # norm indicates normalization , stem indicates root expression
   return ['/'.join(t) for t in okt.pos (doc, norm=True, stem=True)]
```

Sentiment Analysis : ML

```
###### create train_docs and test_docs
if os.path.isfile('../datasets/nsmc/train_docs.json'):
    with open('../datasets/nsmc/train_docs.json') as f:
        train_docs = json.load(f)
    with open('../datasets/nsmc/test_docs.json') as f:
        test_docs = json.load(f)
else:
    train_docs = [(tokenize(row[1]), row[2]) for row in train_data]
    test_docs = [(tokenize(row[1]), row[2]) for row in test_data]
    # JSON 파일로 저장
   with open('../datasets/nsmc/train_docs.json', 'w', encoding="utf-8") as make_file:
        json.dump(train_docs, make_file, ensure_ascii=False, indent="₩t")
    with open('../datasets/nsmc/test_docs.json', 'w', encoding="utf-8") as make_file:
        json.dump(test_docs, make_file, ensure_ascii=False, indent="₩t")

# print
pprint(train_docs[0])

##### Output Korean tokens
tokens = [t for d in train_docs for t in d[0]]
print(len(tokens))
```

Sentiment Analysis : ML

```
# Detailed token analysis
import nltk
text = nltk.Text (tokens, name='NMSC')
print(text)

# total number of tokens
print( len ( text. tokens ))

# number of tokens excluding duplicates
print( len (set( text. tokens )))

# Top 10 tokens with high occurrence frequency
pprint ( text.vocab (). most_common (10))
```

Sentiment Analysis : ML

```
#### CountVectorization : Generate document vectors based on word frequency
# Get top 500 vocab from train data
selected_words = [f[0] for f in text.vocab (). most_common (500)]

# Count the top 500 word occurrences per document
def term_frequency (doc):
return [ doc. count (word) for word in selected_words ]

# apply to the document
train_x = [ term_frequency (d) for d, _ in train_docs ]
test_x = [ term_frequency (d) for d, _ in test_docs ]
train_y = [c for _, c in train_docs]
test_y = [c for _, c in test_docs]

#
train_docs[0]
train_docs[0][0]
term_frequency(train_docs[0][0])
#

# to real number
import numpy as np
x_train = np.asarray(train_x).astype('float32')
x_test = np.array ( test_x ). astype ('float32')
y_train = np.asarray ( train_y ). astype ('float32')
y_test = np.array ( test_y ). astype ('float32')
```

Sentiment Analysis : ML

```
# (1) Logistic regression
from sklearn.linear_model import LogisticRegression , SGDClassifier
import numpy as np

model = LogisticRegression()
model.fit(x_train, y_train)
# model estimation
print(model.score(x_train, y_train))
# prediction
print(model.score(x_test, y_test))



# (2) Decision tree
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
# model estimation
print(clf.score(x_train, y_train))
# prediction
print(clf.score(x_test, y_test))
```

Sentiment Analysis : ML

4. Naver movie emotion corpus data (tf-idf use)

```
from google.colab import drive
drive.mount ('/content/ gdrive ')

# Specify the file name and save path
folder_name = "/content/gdrive/My Drive/Colab Notebooks/Textmining/download/"

def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('₩t') for line in f.read().splitlines()]
        # txt 파일의 헤더(id document label)는 제외하기
      data = data[1:]
    return data

train_data = read_data (folder_name+'ratings_train.txt')
test_data = read_data (folder_name+'ratings_test.txt')

train_data [0:3]
```

Sentiment Analysis : ML

```
# convert to dataframe
new_train = pd.DataFrame ( train_data [0:800])
new_test = pd.DataFrame ( test_data [0:200])

##### Resizing data : try using only part of it
x_train = new_train.iloc[:,1]
y_train = new_train.iloc[:,2]
x_test = new_test.iloc[:,1]
y_test = new_test.iloc[:,2]


#### tf-idf application
from sklearn.feature_extraction.text import TfidfVectorizer
stop_words = stopwords.words('english')

# Convert to document - word matrix via TF-IDF weights
vect = TfidfVectorizer ( stop_words = stop_words ).fit( x_train )
# Convert to document - word matrix via TF-IDF weights
x_train_vectorized = vect.transform ( x_train )

x_train_vectorized
print( x_train_vectorized )
```

Sentiment Analysis : ML

```
# (1) Logistic regression
from sklearn.linear_model import LogisticRegression, SGDClassifier

model = LogisticRegression()
model.fit(x_train_vectorized, y_train)
# model estimation
print(model.score(x_train_vectorized, y_train))
# prediction
print(model.score(vect.transform(x_test), y_test))



# (2) Decision tree
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(x_train_vectorized, y_train)
# model estimation
print(clf.score(x_train_vectorized, y_train))
# prediction
print( clf. score ( vect. transform ( x_test ), y_test ))
```