서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

Text

문서를 어떻게 수치데이터로 바꾸는가?

- Text Vectorization
  - TF-IDF
  - Bag of Words
- Word Embedding with Word2Vec
- Deep Learning

	D1	D2	D3	D4
linux	3	4	1	0
modem	4	3	0	1
the	3	4	4	3
clutch	0	1	4	3
steering	2	0	3	3
petrol	0	1	3	4

Document Term Matrix - Bag of Words

TF-IDF

- 포함되어 있는 단어의 중요성에 따라서 단어와 document의 연관성을 계산하는 방법

- Term frequency – inverse document frequency

- Can be used to *query* a *corpus* by calculating *normalized scores* that express the *relative importance* of *terms* in documents

Corpus and Terms

```
corpus = {
 'a' : "Mr. Green killed Colonel Mustard in the study with the candlestick. \
Mr. Green is not a very nice fellow.",
 'b' : "Professor Plum has a green plant in his study.",
 'c' : "Miss Scarlett watered Professor Plum's green plant while he was away \
from his office last week."
}

terms = {
 'a' : [ i.lower() for i in corpus['a'].split() ],
 'b' : [ i.lower() for i in corpus['b'].split() ],
 'c' : [ i.lower() for i in corpus['c'].split() ]
}
```

Term Frequency for "Mr. Green"

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

Document	tf(mr.)	tF(green)	Sum
corpus['a']	2/19	2/19	4/19 (0.21)
corpus['b']	0/9	1/9	1/9 (0.11)
corpus['c']	0/16	1/16	1/16 (0.06)

Term Frequency for “the green plant”

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

Document	tf(the)	tf(green)	tf(plant)	Sum
corpus[‘a’]	2/19	2/19	0/19	4/19 (0.21)
corpus[‘b’]	0/9	1/9	1/9	2/9 (0.22)
corpus[‘c’]	0/16	1/16	1/16	2/16 (0.13)

뭐가 문제인가?

Inverse Document Frequency

- 모든 문서에 공통적으로 나오는 단어는 변별력이 없다.
- 많은 문서에 나오는 단어를 penalize해 줘야 한다.

- idf (term, D) = 1 + log (전체 document의 수 / term을 포함하고 있는 document의 수)

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

그러나 term이 document에 없는 경우가 있을 수 있다.

이경우를 대비하기 위해 분모를 다음과 같게 사용하는 경우도 많다.

$$1 + |\{d \in D : t \in d\}|$$

다양한 변종 IDF

Variants of TF weight

weighting scheme	TF weight
binary	{0,1}
raw frequency	$f_{t,d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5\frac{f_{t,d}}{\max f_{t,d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max f_{t,d}}$

Variants of IDF weight

weighting scheme	IDF weight
unary	1
inverse frequency	$\log \frac{N}{n_t}$
inverse frequency smooth	$\log(1 + \frac{N}{n_t})$
inverse frequency max	$\log\left(1 + \frac{\max_t n_t}{n_t}\right)$
probabilistic inverse frequency	$\log \frac{N - n_t}{n_t}$

Inverse Document Frequency

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

idf(mr.)	idf(green)	idf (the)	idf(plant)
1+log(3/1)	1 + log(3/3)	1+log(3/1)	1+log(3/2)
2.0986	1.0	2.0986	1.4055

“green” vs. “mr. green” vs. “the green plant”

Document	tf(mr.)	tf(green)	tf(the)	tf(plant)
corpus['a']	0.1053 (2/19)	0.1053	0.1053	0
corpus['b']	0	0.1111 (1/9)	0	0.1111
corpus['c']	0	0.0625 (1/16)	0	0.0625

idf(mr.)	idf(green)	idf (the)	idf(plant)
1+log(3/1)	1 + log(3/3)	1+log(3/1)	1+log(3/2)
2.099	1.0	2.099	1.406

Document	tf-idf(mr.)	tf-idf(green)	tf-idf(the)	tf-idf(plant)
corpus[‘a’]	0.1053*2.099 0.2209	0.1053*1.0 0.1053	0.1053*2.099 0.2209	0*1.406 0
corpus[‘b’]	0*2.099 0	0.1111*1.0 0.1111	0*2.099 0	0.1111*1.406 0.1562
corpus[‘c’]	0*2.099 0	0.0625*1.0 0.0625	0*2.099 0	0.0625*1.406 0.0878

Summed TF-IDF for Sample Queries

Document	tf-idf(mr.)	tf-idf(green)	tf-idf(the)	tf-idf(plant)
corpus['a']	0.1053*2.099 0.2209	0.1053*1.0 0.1053	0.1053*2.099 0.2209	0*1.406 0
corpus['b']	0*2.099 0	0.1111*1.0 0.1111	0*2.099 0	0.1111*1.406 0.1562
corpus['c']	0*2.099 0	0.0625*1.0 0.0625	0*2.099 0	0.0625*1.406 0.0878

Query	corpus['a']	corpus['b']	corpus['c']
Green	0.1053	**0.1111**	0.0625
Mr. green	0.2209 + 0.1053 = **0.3262**	0 + 0.1111 = 0.1111	0 + 0.0625 = 0.0625
The green plant	0.2209 + 0.1053 + 0 = **0.3262**	0 + 0.1111 + 0.1562 = 0.2673	0 + 0.0625 + 0.0878 = 0.15 03

- Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.
- Professor Plum has a green plant in his study.
- Miss Scarlett watered Professor Plum's green plant while he was away from his office last week.

NLTK를 이용

```
import nltk

# Download ancillary nltk packages if not already installed
nltk.download('stopwords')

all_content = " ".join( [corpus[index] for index in corpus] )    // 모든 텍스트를 한줄로 합침

# Approximate bytes of text
print len(all_content)                                            // 텍스트의 길이

tokens = all_content.split()                                      // 텍스트를 스페이스로 자름
text = nltk.Text(tokens)                                          // nltk가 사용할 수 있는 자료구조로 변환

# Examples of the appearance of the word "open"
text.concordance("open")                                          // 해당 token 주위의 context를 보여줌

# Frequent collocations in the text (usually meaningful phrases)
text.collocations()                                               // 같이 연달아 나오는 단어들

# Frequency analysis for words of interest
fdist = text.vocab()                                              // 빈도, 각 token이 몇번이나 나오는가?
fdist["green"]                                                    // dictionary 형태로 조회할 수 있다
fdist["mr."]
fdist["the"]
```

```
>>> text = nltk.Text(tokens)
>>> type(text)
<class 'nltk.text.Text'>
>>> text[1024:1062]
['CHAPTER', 'I', 'On', 'an', 'exceptionally', 'hot', 'evening', 'early', 'in',
 'July', 'a', 'young', 'man', 'came', 'out', 'of', 'the', 'garret', 'in',
 'which', 'he', 'lodged', 'in', 'S.', 'Place', 'and', 'walked', 'slowly',
 ',', 'as', 'though', 'in', 'hesitation', ',', 'towards', 'K.', 'bridge', '.']
>>> text.collocations()
Katerina Ivanovna; Pyotr Petrovitch; Pulcheria Alexandrovna; Avdotya
Romanovna; Rodion Romanovitch; Marfa Petrovna; Sofya Semyonovna; old
woman; Project Gutenberg-tm; Porfiry Petrovitch; Amalia Ivanovna;
great deal; Nikodim Fomitch; young man; Ilya Petrovitch; n't know;
Project Gutenberg; Dmitri Prokofitch; Andrey Semyonovitch; Hay Market
```

```
>>> tokens = tokens[110:390]
>>> text = nltk.Text(tokens)
>>> text.concordance('gene')
Displaying 5 of 5 matches:
hey say too few people now carry the gene for blondes to last beyond the next
blonde hair is caused by a recessive gene . In order for a child to have blond
have blonde hair , it must have the gene on both sides of the family in the g
ere is a disadvantage of having that gene or by chance . They do n't disappear
des would disappear is if having the gene was a disadvantage and I do not thin
```

NLTK Utilities

```
# Number of words in the text
len(tokens)

# Number of unique words in the text

len(fdist.keys())

# Common words that aren't stopwords
[w for w in fdist.keys()[:100] \
   if w.lower() not in nltk.corpus.stopwords.words('english')]

# Long words that aren't URLs
[w for w in fdist.keys() if len(w) > 15 and not w.startswith("http")]

# Number of URLs
len([w for w in fdist.keys() if w.startswith("http")])

# Enumerate the frequency distribution
for rank, word in enumerate(fdist):
    print rank, word, fdist[word]
```

기본 자연 언어 처리

- Tokenize
  - 문장을 단어 단위로 자른다
- Stemming
  - Playing vs. play
  - Apples vs. apple
  - 영어의 경우에 어근을 찾아서 분석하는 것이 좋다
  - 어근을 찾는 것을 stemming / 형태소 분석 이라 한다
- Stopword
  - 많이 사용되는 단어를 분석에 넣어야 하는가?
  - 영어에서 가장 많이 사용되는 단어는?
  - 분석에서 제외되는 단어를 stopword라고 한다
- Part of speech (품사부착)
  - 문장에서 명사, 동사, 형용사등을 알아낸다

Tokenize

```
>>> import nltk
>>> txt = "Mr. Green killed Colonel Mustard in the study with the  \
... candlestick. Mr. Green is not a very nice fellow."
>>> txt = "Mr. Green killed Colonel Mustard in the study with the \
... candlestick. Mr. Green is not a very nice fellow."
>>> sentences = nltk.tokenize.sent_tokenize(txt)
>>> sentences
['Mr. Green killed Colonel Mustard in the study with the candlestick.',
 'Mr. Green is not a very nice fellow.']
```

```
>>> tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
>>> tokens
[['Mr.', 'Green', 'killed', 'Colonel', 'Mustard', 'in', 'the', 'study',
  'with', 'the', 'candlestick', '.'],
 ['Mr.', 'Green', 'is', 'not', 'a', 'very', 'nice', 'fellow', '.']]
```

Stopwords

```
import nltk
nltk.download('stopwords')                # 한번만 해주면 됨. 해당 자료를 다운로드 받음

from nltk.corpus import stopwords
stop_words = stopwords.words('english') + ['.', ',', '--', '\'s', '?', ')', '(', ':',  '\'',  '\'re', '"',
'-', '}', '{', u'—', 'rt', 'http', 't', 'co', ]
```

기본 **stopwords**이외에 많이 사용되는 단어들을 분석에서 제외한다.

Stemming

```
>>> from nltk.stem.porter import *
```

Create a new Porter stemmer.

```
>>> stemmer = PorterStemmer()
```

Test the stemmer on various pluralised words.

```
>>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
...            'died', 'agreed', 'owned', 'humbled', 'sized',
...            'meeting', 'stating', 'siezing', 'itemization',
...            'sensational', 'traditional', 'reference', 'colonizer',
...            'plotted']
>>> singles = []
```

```
>>> for plural in plurals:
...     singles.append(stemmer.stem(plural))
```

```
>>> singles
[u'caress', u'fli', u'die', u'mule', u'deni', u'die', u'agre', u'own',
 u'humbl', u'size', u'meet', u'state', u'siez', u'item', u'sensat',
 u'tradit', u'refer', u'colon', u'plot']
```

http://www.nltk.org/howto/stem.html

All together

```
status_texts = [ ‘xxxx’, ‘yyyy’, …]

tokens = []
for s in status_texts:
    tokens += nltk.tokenize.word_tokenize(s.lower())

from nltk.corpus import stopwords
stop_words = stopwords.words('english') + [
    '.', ',', '--', '\'s', '?', '!', ')', '(', ':', '\",
    '\'re', "", '-', '}', '{', u'—', 'rt', 'http', 't', 'co', '@', '#',
    ]

from nltk.stem import PorterStemmer
stemmer=PorterStemmer()

stemmed = []
for token in tokens:
    if token is '' or token in stop_words:
        continue
    stemmed.append(stemmer.stem(token))
```

Text를 단어로 잘라서 tokens list에 넣는다

영어 stopword와 트위터에서 사용하는 RT등을 stop word list에 넣는다

Porter Stemmer를 사용하여 각 token들의 어근을 구해서 stemmed list에 넣는다

한글

- KoNLPy
  - http://konlpy.org/en/v0.4.4/
  - https://www.lucypark.kr/courses/2015-dm/text-mining.html
- MeCab
  - https://bitbucket.org/eunjeon/mecab-ko

TF-IDF의 문제점은?

- Bag of Words
  - Context를 사용하지 못한다
  - 문장의 의미가 없어짐
- Homonym
  - 동음이의어
- Link Structure가 있다면?
  - PageRank

Basic Classification

Text Classification

- How can we identify particular features of language data that are salient for classifying it?
  - Classify하려는데 무슨 language feature를 사용해야 하는가?
- How can we construct models of language that can be used to perform language processing tasks automatically?
  - 어떠한 language model을 사용해야 하는가?
- What can we learn about language from these models?
  - 이러한 모델은 language에 대해서 어떤 것을 알려주는가?
- Naïve Bayesian 방법과 Decision Tree방법을 배운다

Supervised Classification

- Classification
  - the task of choosing the correct class label for a given input.
  - each input is considered in isolation from all other inputs
  - the set of labels is defined in advance

- Examples
  - Deciding whether an email is spam or not.
  - Deciding what the topic of a news article is, from a fixed list of topic areas
    - "sports," "technology," or "politics."
  - Deciding whether a given occurrence of the word bank is used
    - a river bank
    - a financial institution
    - the act of tilting to the side
    - or the act of depositing something in a financial institution

Training and Prediction (Supervised)

Basic Probabilities

- Conditional Probability – P (A | B)
  - Given B, the probability of A
  - P(quick | good) = 0.6666

('Nobody owns the water.',	'good')
('the quick rabbit jumps fences',	'good')
('buy pharmaceuticals now',	'bad')
('make quick money at the online casino',	'bad')
('the quick brown fox jumps',	'good')

- 우리가 원하는 것은?
  - Label Prediction
  - P(Category | Document)

Bayes' Theorem

- P(Category | Document)
  - Given a document, what is the probability of the category?
  - 우리가 결과로써 원하는 것
- P(Document | Category)
  - E.g. P(quick | good) : the word "quick" appears in xx% of your "bad" document
- P(Category)
  - A randomly selected document will be in this category
- P(Document)
  - Do we need it?
  - 왜 그럴까?

$$Pr(A \mid B) = Pr(B \mid A) \times Pr(A)/Pr(B)$$

In the example, this becomes:

$$Pr(Category \mid Document) = Pr(Document \mid Category) \times Pr(Category) / Pr(Document)$$

Naïve Classifier

- Bayes' Theorem을 직접적으로 이용
- 각 단어들이 독립적이라고 가정
  - 올바른 가정은 아니나 계산이 간편해서 많이 사용된다.
  - 왜 올바른 가정이 아닐까?

- P(Python | Bad) = 0.2
  - Python이 20%의 Bad document에 등장
- P(Casino | Bad) = 0.8
  - Casino가 80% 의 Bad document에 등장

- P(Python & Casino | Bad) = 0.2 X 0.8 = 0.16
  - 모든 단어에 대하여 조건부 확률을 그대로 곱한다
  - 독립이라는 가정을 하므로 가능

Document의 Label예상

- P(quick | good) = 2/3
- P(rabbit | good)
- P(good) = 0.6 (전체 다섯개 중 세개가 good)

- P(good | quick rabbit)

  = P(quick rabbit | good) * P(good)

  = P(quick | good) * P(rabbit | good) * P(good)

  = 2/3 * 1/3 * 3/5 = 2/15

  – 참고로 P(quick rabbit)으로 나누는 것은 생략되었음.

('Nobody owns the water.', 'good')

('the **quick rabbit** jumps fences', 'good')

('buy pharmaceuticals now', 'bad')

('make quick money at the online casino','bad')

('the **quick** brown fox jumps', 'good')

NLTK – http://www.nltk.org/book/ch06.html

```
import nltk
nltk.download()
```

- Download “movie_reviews” & “names” in “Corpora”

Name Corpus in NLTK

```
>>> names = nltk.corpus.names
>>> names.fileids()
['female.txt', 'male.txt']
>>> male_names = names.words('male.txt')
>>> female_names = names.words('female.txt')
>>> [w for w in male_names if w in female_names]
['Abbey', 'Abbie', 'Abby', 'Addie', 'Adrian', 'Adrien', 'Ajay', 'Alex', 'Alexis',
'Alfie', 'Ali', 'Alix', 'Allie', 'Allyn', 'Andie', 'Andrea', 'Andy', 'Angel',
'Angie', 'Ariel', 'Ashley', 'Aubrey', 'Augustine', 'Austin', 'Averil', ...]
```

Data Set & Train Classifier

```
>>> from nltk.corpus import names
>>> import random
>>> names = ([(name, 'male') for name in names.words('male.txt')] +
...          [(name, 'female') for name in names.words('female.txt')])
>>> import random
>>> random.shuffle(names)
```

```
>>> def gender_features(word):
...     return {'last_letter': word[-1]}
>>> gender_features('Shrek')
{'last_letter': 'k'}
```

```
>>> featuresets = [(gender_features(n), g) for (n,g) in names]
>>> train_set, test_set = featuresets[500:], featuresets[:500]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Classification and Accuracy

```
>>> classifier.classify(gender_features('Neo'))
'male'
>>> classifier.classify(gender_features('Trinity'))
'female'
```

```
>>> print nltk.classify.accuracy(classifier, test_set)
0.758
```

이름의 제일 끝자리 알파벳만으로 76%의 정확도를 가지는 classifier를 만들 수 있다.

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
             last_letter = 'a'            female : male   =     38.3 : 1.0
             last_letter = 'k'              male : female =     31.4 : 1.0
             last_letter = 'f'              male : female =     15.3 : 1.0
             last_letter = 'p'              male : female =     10.6 : 1.0
             last_letter = 'w'              male : female =     10.6 : 1.0
```

Choosing the Right Features

```
def gender_features2(name):
    features = {}
    features["firstletter"] = name[0].lower()
    features["lastletter"] = name[-1].lower()
    for letter in 'abcdefghijklmnopqrstuvwxyz':
        features["count(%s)" % letter] = name.lower().count(letter)
        features["has(%s)" % letter] = (letter in name.lower())
    return features
```

```
>>> gender_features2('John')
{'count(j)': 1, 'has(d)': False, 'count(b)': 0, ...}
```

```
>>> featuresets = [(gender_features2(n), g) for (n,g) in names]
>>> train_set, test_set = featuresets[500:], featuresets[:500]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
0.748
```

Using more features did not help.
It actually lowered the accuracy

Overfitting:

너무 많은 feature를 사용하면 training data의 특징적인 지엽적인 부분에만 집착 새로운 자료에 일반화되어 적용이 안되는 경우가 많다.

Dev-Test Set

```
>>> train_names = names[1500:]
>>> devtest_names = names[500:1500]
>>> test_names = names[:500]
```

```
>>> train_set = [(gender_features(n), g) for (n,g) in train_names]
>>> devtest_set = [(gender_features(n), g) for (n,g) in devtest_names]
>>> test_set = [(gender_features(n), g) for (n,g) in test_names]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set) ❶
>>> print nltk.classify.accuracy(classifier, devtest_set) ❷
0.765
```

Examining Errors

```
>>> errors = []
>>> for (name, tag) in devtest_names:
...     guess = classifier.classify(gender_features(name))
...     if guess != tag:
...         errors.append( (tag, guess, name) )
```

```
>>> for (tag, guess, name) in sorted(errors):
...     print 'correct=%-8s guess=%-8s name=%-30s' % (tag, guess, name)
  ...
correct=female   guess=male     name=Cindelyn
  ...
correct=female   guess=male     name=Katheryn
correct=female   guess=male     name=Kathryn
  ...
correct=male     guess=female   name=Aldrich
  ...
correct=male     guess=female   name=Mitch
  ...
correct=male     guess=female   name=Rich
  ...
```

Using New Features

```
>>> def gender_features(word):
...     return {'suffix1': word[-1:],
...             'suffix2': word[-2:]}
```

```
>>> train_set = [(gender_features(n), g) for (n,g) in train_names]
>>> devtest_set = [(gender_features(n), g) for (n,g) in devtest_names]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, devtest_set)
0.782
```

정확도가 올라갔다. 하지만 어떤 feature를 써야 하는지는 아직도 연구 주제이다.

Movie Review Data Set

- 2k movie reviews with sentiment polarity classification

```
from nltk.corpus import movie_reviews
movie_reviews.fileids()
movie_reviews.categories('fileidxxx.txt')
movie_reviews.words('fileidxxx.txt')
movie_reviews.words()
```

```
>>> movie_reviews.categories('pos/cv995_21821.txt')
[u'pos']
>>>
>>> movie_reviews.words('pos/cv995_21821.txt')
[u'wow', u'!', u'what', u'a', u'movie', u'.', u'it', ...]
>>>
```

Document Classification

```
>>> from nltk.corpus import movie_reviews
>>> documents = [(list(movie_reviews.words(fileid)), category)
...              for category in movie_reviews.categories()
...              for fileid in movie_reviews.fileids(category)]
>>> random.shuffle(documents)
```

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = all_words.keys()[:2000] ❶

def document_features(document): ❷
    document_words = set(document) ❸
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

```
>>> print document_features(movie_reviews.words('pos/cv957_8737.txt'))
{'contains(waste)': False, 'contains(lot)': False, ...}
```

Classify and Accuracy

```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

```
>>> print nltk.classify.accuracy(classifier, test_set) ❶
0.81
>>> classifier.show_most_informative_features(5) ❷
Most Informative Features
   contains(outstanding) = True              pos : neg    =     11.1 : 1.0
        contains(seagal) = True              neg : pos    =      7.7 : 1.0
   contains(wonderfully) = True              pos : neg    =      6.8 : 1.0
         contains(damon) = True              pos : neg    =      5.9 : 1.0
        contains(wasted) = True              neg : pos    =      5.8 : 1.0
```

Confusion Matrix

```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print nltk.classify.accuracy(classifier, test_set)
classifier.show_most_informative_features(5)

gold = [tag for (features, tag) in test_set]
test = [classifier.classify(features) for features, tag in test_set]

cm = nltk.ConfusionMatrix(gold, test)
print cm.pp()
print cm.pp(sort_by_count=True, show_percents=True)
```

```
>>>
>>> print cm.pp(sort_by_count=True)
    |  p  n |
    |  o  e |
    |  s  g |
----+-------+
pos |<40>11 |
neg |  6<43>|
----+-------+
(row = reference; col = test)
```

Size of Train Set & Test Set

- Test set은 작아도 된다
- Train set이 문제
  - 자주 나오는 term이 적어도 50번 정도는 나오게 train해야 한다.
  - 독립적이지 않은 예를 사용해야 하면 더욱 크기를 늘려야 한다. (Naïve 이기 때문에)
  - 보통의 경우 90%를 train set으로 10%를 test set으로 사용한다.

- Cross Validation
  - In order to evaluate our models, we must reserve a portion of the annotated data for the test set.
  - If the test set is too small, then our evaluation may not be accurate.
  - Making the test set larger usually means making the training set smaller, which can have a significant impact on performance
  - To perform multiple evaluations on different test sets, then to combine the scores from those evaluations
  - 데이타를 N개로 나누어서 그중 하나를 test set으로 나머지를 train set으로 사용.
  - 이를 N번 수행
  - E.g. Ten-fold cross validation

Precision, Recall, F-Score

- 얼마나 정확한지 평가

- Precision = TP / (TP + FP)
- Recall = TP / (TP + FN)

- F-score는 precision과 recall의 조화평균
  - 조화평균 H = n / (1/A + 1/B + ⋯ 1/Z)
    - 비율의 평균을 구하는데 사용

보조 자료

Computing TF-IDF

```python
from pyspark.mllib.feature import HashingTF, IDF

# Load documents (one per line).
documents = sc.textFile("data/mllib/kmeans_data.txt").map(lambda line: line.split(" "))

hashingTF = HashingTF()
tf = hashingTF.transform(documents)

# While applying HashingTF only needs a single pass to the data, applying IDF needs two passes:
# First to compute the IDF vector and second to scale the term frequencies by IDF.
tf.cache()
idf = IDF().fit(tf)
tfidf = idf.transform(tf)
```

HashingTF vs. CountVectorizer

```
Output of hashingTF function
+-----------------------------------------+-------------------------------------------------------+
|words                                    |Features(vocab_size,[index],[tf])                      |
+-----------------------------------------+-------------------------------------------------------+
|[hi, i, heard, about, spark]             |(100,[5,29,57,60,77],[1.0,1.0,1.0,1.0,1.0])            |
|[i, wish, java, could, use, case, classes]|(100,[9,13,29,42,67,89,95],[1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
|[logistic, regression, models, are, neat] |(100,[4,38,86,93,95],[1.0,1.0,1.0,1.0,1.0])            |
+-----------------------------------------+-------------------------------------------------------+

Output of CountVectorizer function
+-----------------------------------------+-----------------------------------------------------+
|words                                    |Features(vocab_size,[index],[tf])                    |
+-----------------------------------------+-----------------------------------------------------+
|[hi, i, heard, about, spark]             |(16,[0,1,11,12,13],[1.0,1.0,1.0,1.0,1.0])            |
|[i, wish, java, could, use, case, classes]|(16,[0,2,4,6,7,10,14],[1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
|[logistic, regression, models, are, neat] |(16,[3,5,8,9,15],[1.0,1.0,1.0,1.0,1.0])              |
+-----------------------------------------+-----------------------------------------------------+

Vocabulary from CountVectorizerModel is
[u'i', u'wish', u'use', u'are', u'logistic', u'case', u'heard', u'classes', u'neat', u'java', u'models', u'spark',
 u'hi', u'about', u'could', u'regression']
```

Reading Cleaned Tweet

```
df = spark.read.option("header", "true") \
                    .option("nullValue", "?") \
                    .option("inferSchema", "true") \
                    .csv('../pyspark/clean_tweet.csv’)
df = df.dropna()

df.count()
1596753

(train_set, val_set, test_set) = df.randomSplit([0.98, 0.01, 0.01], \
                 seed = 2000)
```

TF IDF

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer, CountVectorizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="words")

hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf’)

idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) # remove sparse terms

label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label" )

pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
```

Logistic Regression

```
from pyspark.ml.classification import LogisticRegression
```

```
train_df = pipelineFit.transform(train_set)
val_df = pipelineFit.transform(val_set)
```

```
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
|_c0|                text|target|               words|                  tf|            features|label|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
|  0|awww that bummer ...|     0|[awww, that, bumm...|(65536,[8436,8847...|(65536,[8436,8847...|  0.0|
|  1|is upset that he ...|     0|[is, upset, that,...|(65536,[1444,2071...|(65536,[1444,2071...|  0.0|
|  2|dived many times ...|     0|[dived, many, tim...|(65536,[2548,2888...|(65536,[2548,2888...|  0.0|
|  3|my whole body fee...|     0|[my, whole, body,...|(65536,[158,11650...|(65536,[158,11650...|  0.0|
|  4|no it not behavin...|     0|[no, it, not, beh...|(65536,[1968,4488...|(65536,[1968,4488...|  0.0|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
only showing top 5 rows
```

```
lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)
```

```
predictions = lrModel.transform(val_df)
```

평가

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
# rawPrediction은 tranform에 의해서 자동생성됨

evaluator.evaluate(predictions)

evaluator.getMetricName()

accuracy = predictions.filter(predictions.label == predictions.predictio
n).count() / float(val_set.count())
```