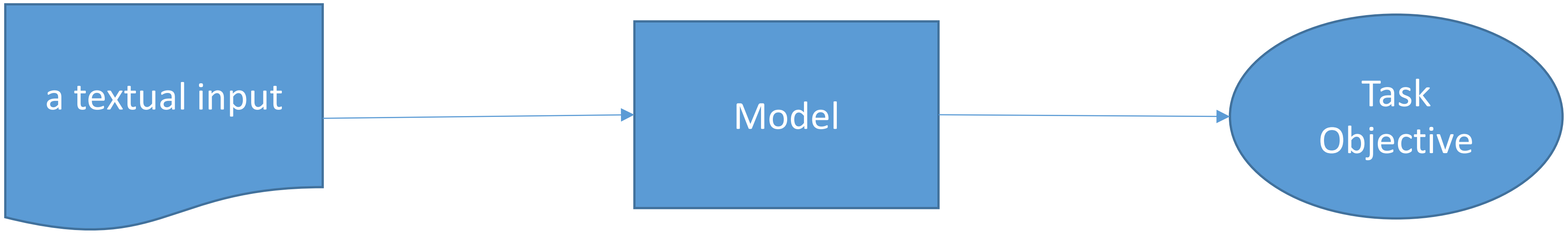
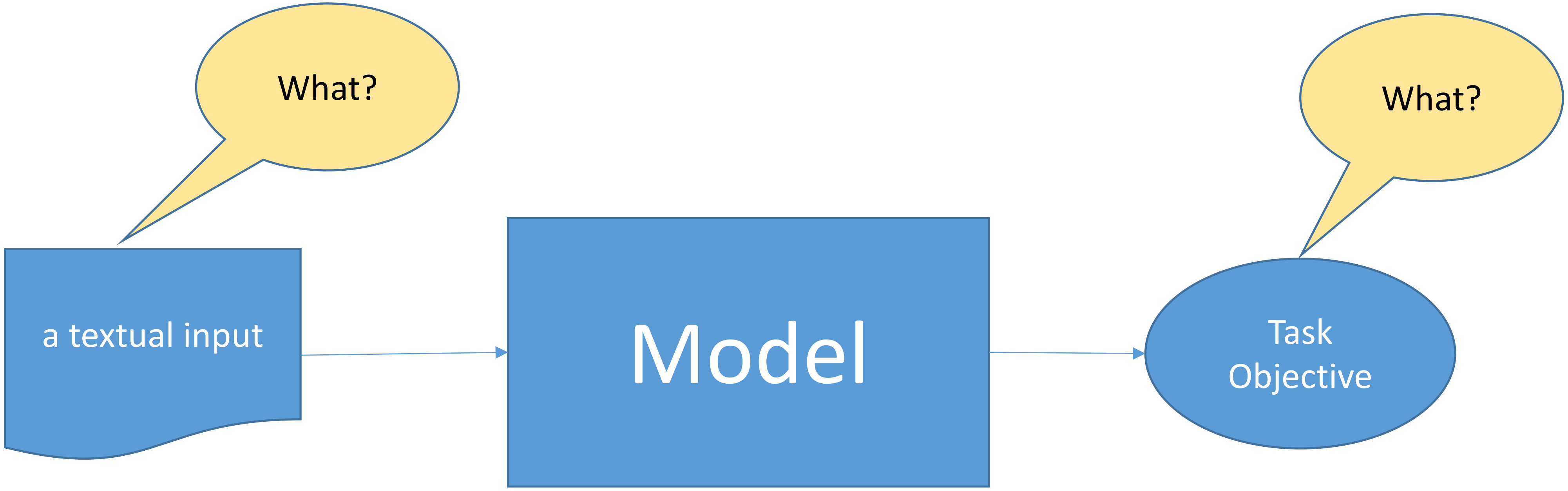
Text Representation for Classification

LÊ ANH CƯỜNG
Faculty of Information Technology
Ton Duc Thang University, Ho Chi Minh city, Vietnam
2020

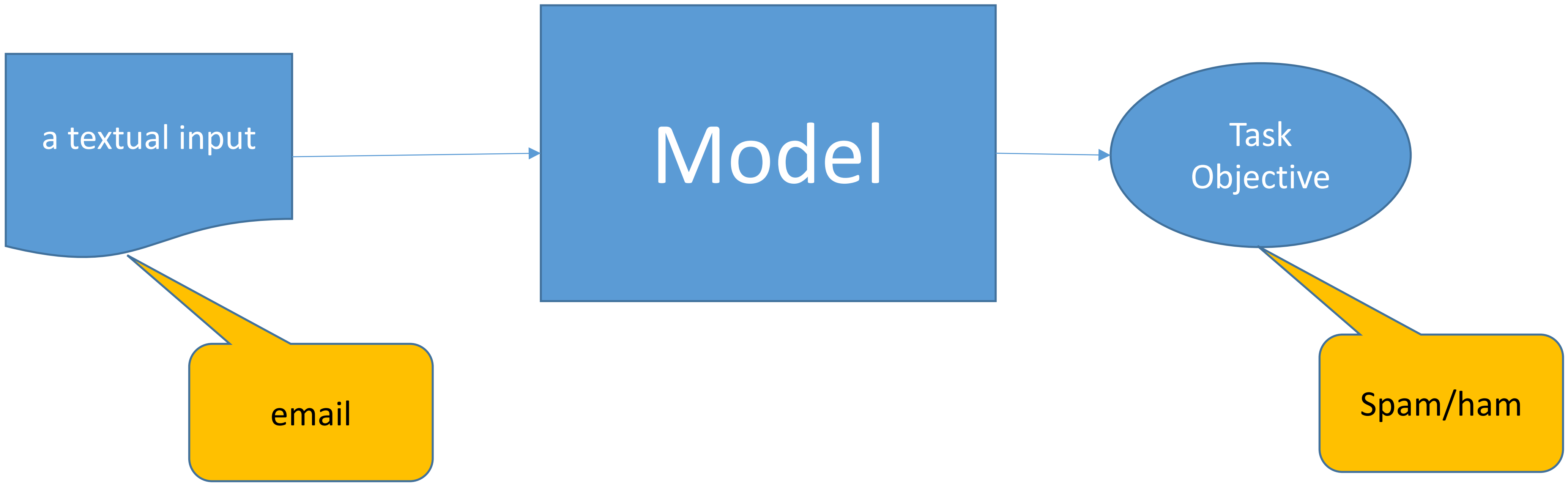
Natural Language Processing: a General Scheme



Natural Language Processing: a General Scheme



Natural Language Processing: a General Scheme



Tasks with inputs/outputs

Task	Input	output
Document classification	A textual document	Classes/labels
Parsing	A sentence	Parsed tree
Machine translation	A source sentence	A target sentence
Named entity recognition	A sentence	NE tags
…	…	…

How represent the Input for NLP tasks?

- A sentence: a sequence of Words
- A document: a sequence of sentences
- A word: a sequence of characters

- The word is the smallest unit of language that has a meaning

Work on Documents?

- Document representation
- Document generation
- Document classification
- Document summarization, multi-document summarization

Types of Word Representation

1. Word as unique, indivisible unit
2. Word as Distributed Semantic Representation

Word as unique, indivisible unit

- Each word is represented as *this word*
- Each word is represented by an unique key or an index in the vocabulary dictionary.
- Each word is represented as one-hot vector

```
                 Paris
        Rome                          word V
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

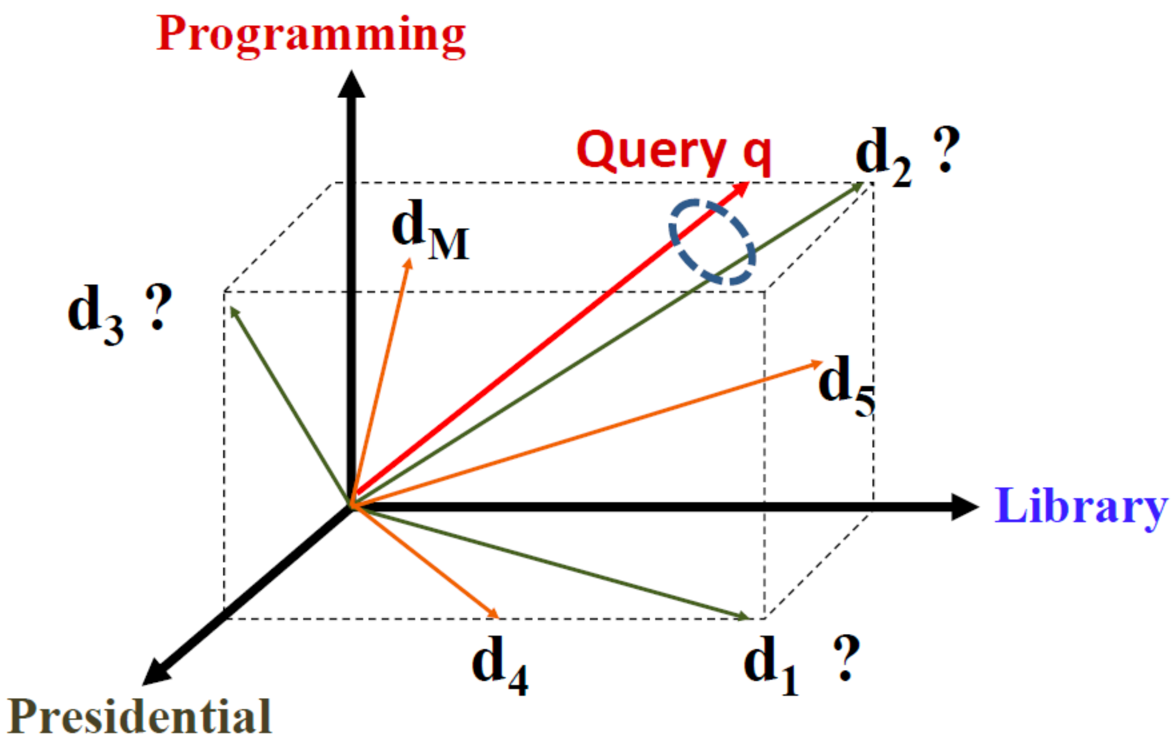
Text Representation

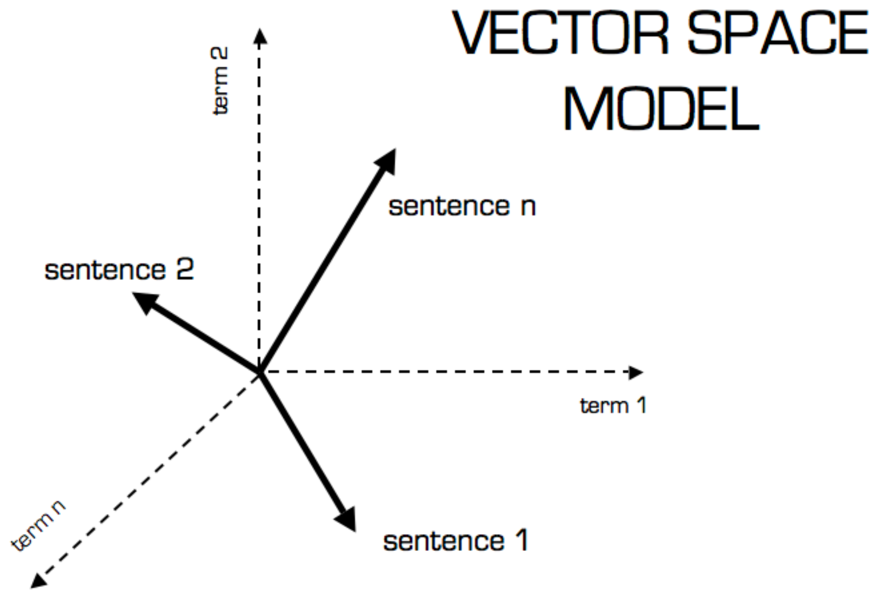
Documents		Terms			
		data	result	statistics	analysis
	Document1	0	1	0	1
	Document2	1	0	1	0
	Document3	0	0	1	0
	Document4	1	1	0	0

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

- Represent each document as a feature vector of Vector Space Model (Term Vector Model)
- Term: word, n-gram, …

Vector Space Model and Bag-Of-Words for Text Representation





TF-IDF weights

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

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	$K + (1 - K) \frac{f_{t,d}}{\max f_{t,d}}$

TF-IDF example

Document-term matrix (no need to normalize, every word occurs just once)

	angeles	los	new	post	times	york
d1	0	0	1	0	1	1
d2	0	0	1	1	0	1
d3	1	1	0	0	1	0

tf-idf

	angeles	los	new	post	times	york
d1	0	0	0.584	0	0.584	0.584
d2	0	0	0.584	1.584	0	0.584
d3	1.584	1.584	0	0	0.584	0

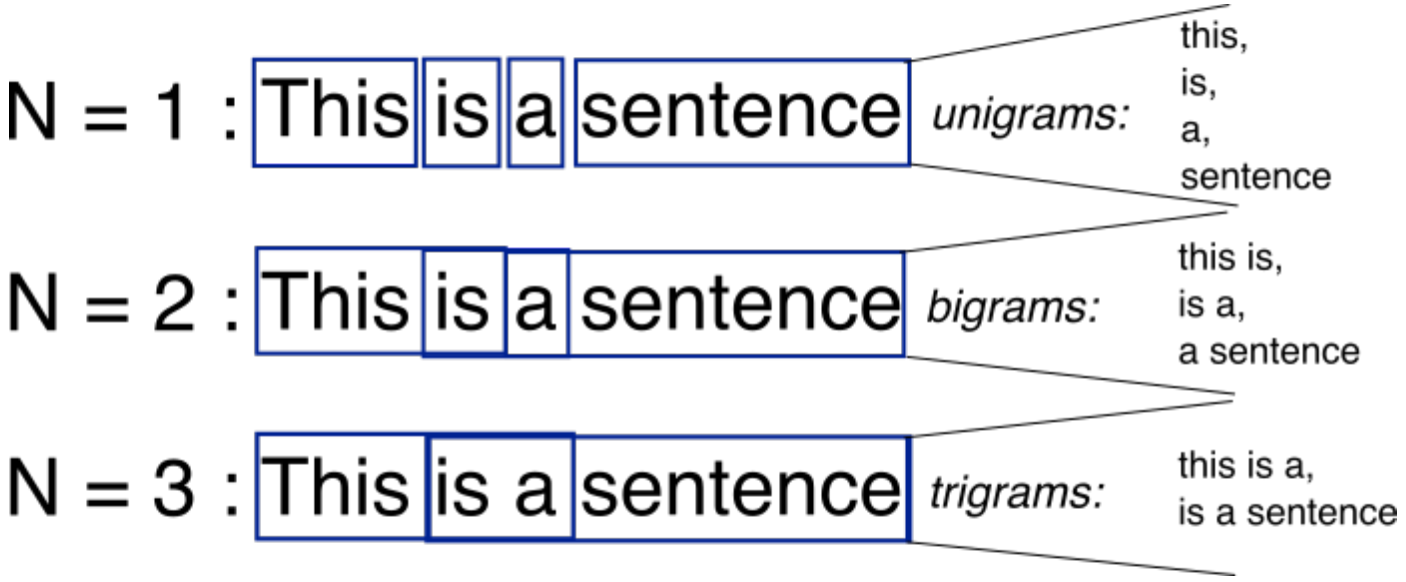
TF-IDF Variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}, \alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

► **Figure 6.7** SMART notation for tf-idf variants. Here *CharLength* is the number of characters in the document.

Terms ?

- N-gram



- Other kinds: POS tags, Syntactic sub-structures, ...

Applications

- Text Similarity
- Text Classification
- Word Similarity
- Information Retrieval

Term-context matrix for word similarity

- Two words are similar in meaning if their context vector are similar

context

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

words

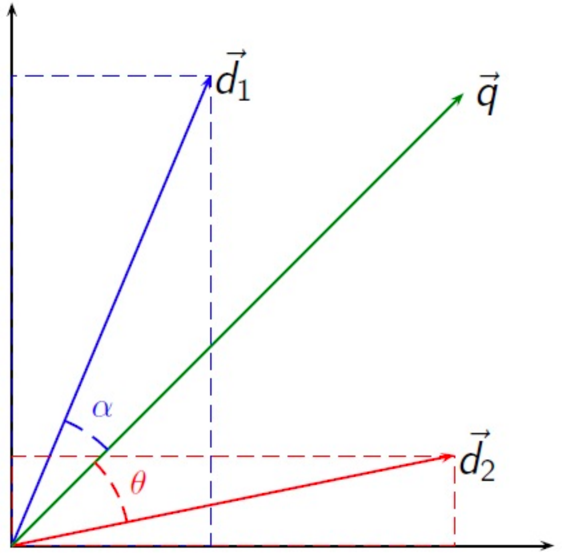
Cosine Similarity for Information Retrieval, as well as for Text Similarity

Documents and queries are represented as vectors.

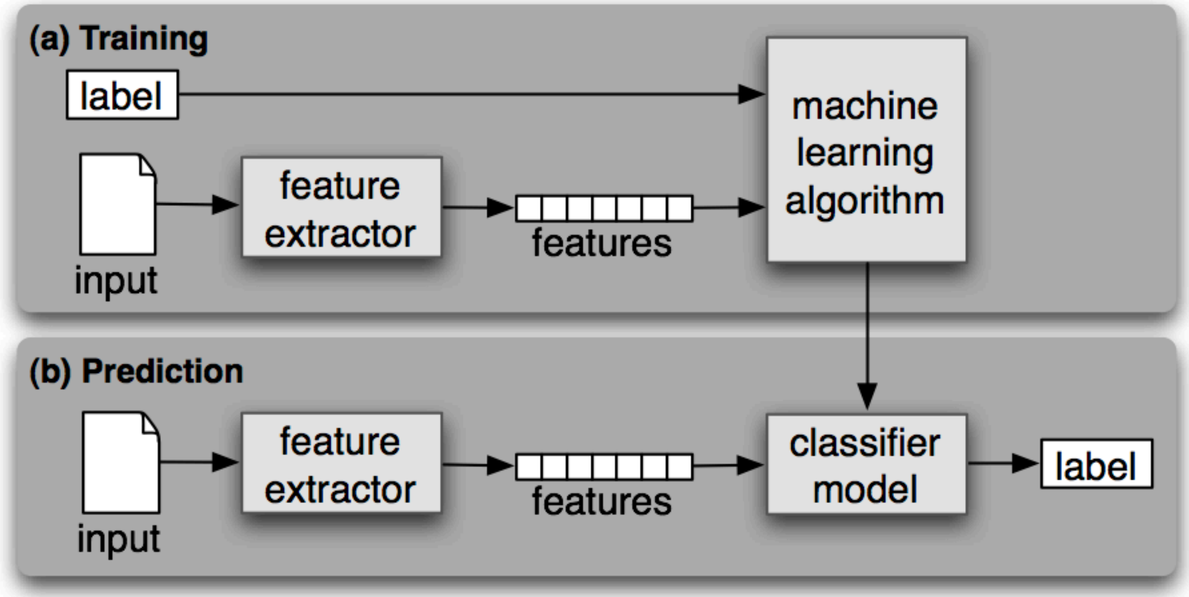
$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$

Using the cosine the similarity between document d_j and query q can be calculated as:

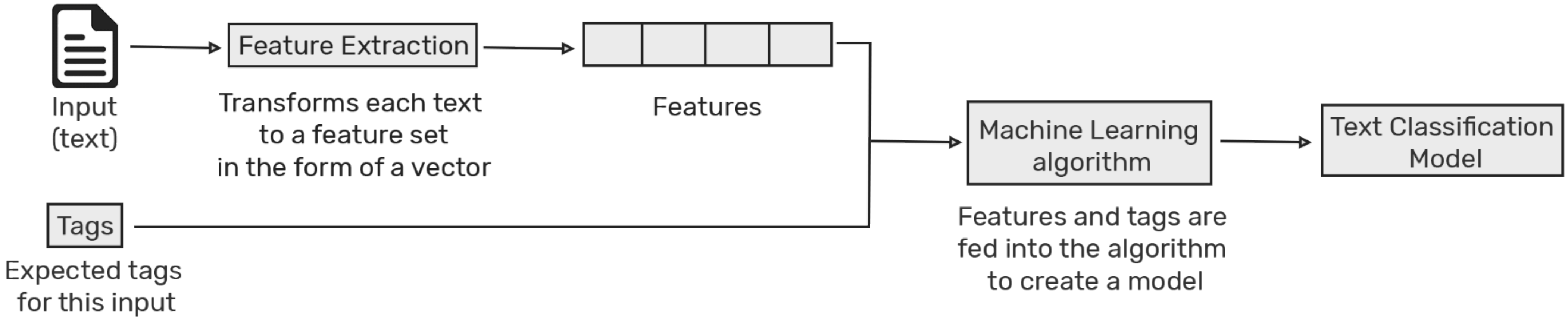
$$\cos(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$



Machine Learning for Text Classification



Machine Learning for Text Classification



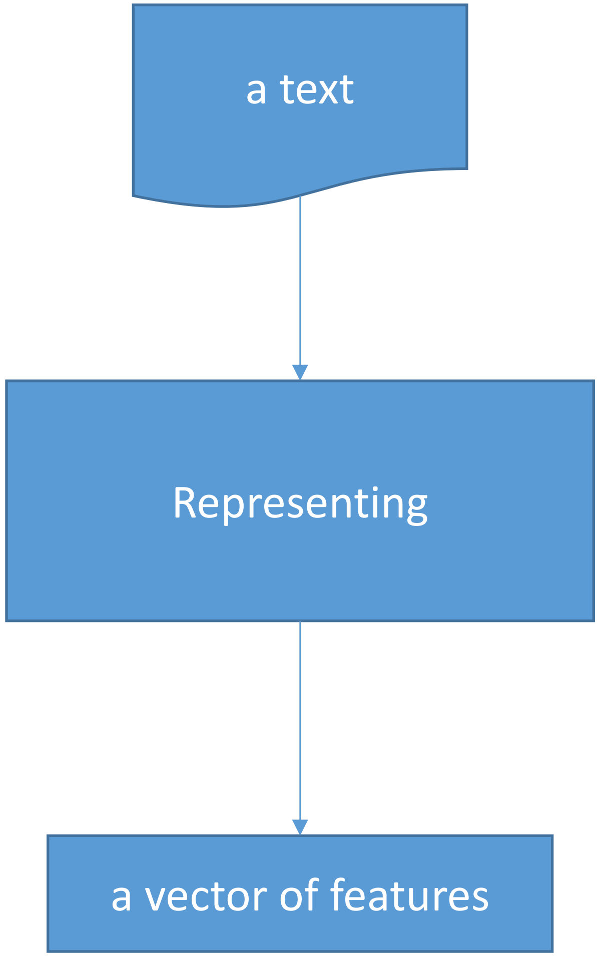
Projects

1. Document classification
2. Word Similarity

- Requirement:
  - Using "vector space model"
  - Using different representations of document and word context:
    - Binary matrix
    - Word Frequency
    - tf.idf
    - N-gram

Text preprocessing

- Token segmentation/split
- Sentence segmentation
- Get n-gram of token sequences
- Vocabulary of tokens
- Texts to sequences
- Sequences to texts
- Convert sentence to vector
    - Binary mode
    - Counting mode
    - Tfidf mode



Text Preprocessing: using nltk

Input: document d

- Sentence segmentation: s1, s2, …, sn
- For each sentence si:
- Token segmentation: w1, w2, …, wk
- Vocabulary dictionary
- Convert word to number
- Convert sequence (i.e.) to vector

```
>>> from nltk import tokenize
>>> p = "Good morning Dr. Adams. The patient is waiting for you in room r

>>> tokenize.sent_tokenize(p)
['Good morning Dr. Adams.', 'The patient is waiting for you in room numbe
```

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
```

```
from nltk.tokenize import word_tokenize
from gensim.corpora.dictionary import Dictionary

sometext = "hello how are you doing?"

tokens = word_tokenize(sometext)
my_vocab = Dictionary([tokens])

print(my_vocab.token2id['hello'])
```

```python
from nltk.util import ngrams

text = "Hi How are you? i am fine and you"

n = int(input("ngram value = "))

n_grams = ngrams(text.split(), n)

for grams in n_grams :

   print(grams)
```

Text preprocessing: using tensorflow.keras.preprocessing.text

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

- text
  - Overview
  - hashing_trick
  - one_hot
  - text_to_word_sequence
  - **Tokenizer**
  - tokenizer_from_json

- text
  - Overview
  - hashing_trick
  - one_hot
  - **text_to_word_sequence**
  - Tokenizer
  - tokenizer_from_json

tf.keras.preprocessing.text.Tokenizer

https://keras.io/api/preprocessing/text/

Tokenizer class

```
tf.keras.preprocessing.text.Tokenizer(
    num_words=None,
    filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
    lower=True,
    split=" ",
    char_level=False,
    oov_token=None,
    document_count=0,
    **kwargs
)
```

tf.keras.preprocessing.text.Tokenizer

Tokenizer class

```
tf.keras.preprocessing.text.Tokenizer(
    num_words=None,
    filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
    lower=True,
    split=" ",
    char_level=False,
    oov_token=None,
    document_count=0,
    **kwargs
)
```

- **num_words**: the maximum number of words to keep, based on word frequency. Only the most common `num_words-1` words will be kept.
- **filters**: a string where each element is a character that will be filtered from the texts. The default is all punctuation, plus tabs and line breaks, minus the `'` character.
- **lower**: boolean. Whether to convert the texts to lowercase.
- **split**: str. Separator for word splitting.
- **char_level**: if True, every character will be treated as a token.
- **oov_token**: if given, it will be added to word_index and used to replace out-of-vocabulary words during text_to_sequence calls

tf.keras.preprocessing.text.Tokenizer

```
from keras.preprocessing.text import Tokenizer
```

```
Using TensorFlow backend.
```

```
t  = Tokenizer()
text = 'The earth is an awesome place live'
tokens = text.split(' ')
t.fit_on_texts(tokens)
print("word_index : ",t.word_index)
```

```
word_index :  {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5, 'place': 6, 'live': 7}
```

```
test_text = ['The', 'earth', 'live']
sequences = t.texts_to_sequences(test_text)
print('sequences : ',sequences,'\n')
```

```
sequences :  [[1], [2], [7]]
```

tf.keras.preprocessing.text.Tokenizer

```
from keras.preprocessing.text import Tokenizer
```

```
Using TensorFlow backend.
```

```
t  = Tokenizer()
text = ['The earth is an awesome place live']
t.fit_on_texts(text)
print("word_index : ",t.word_index)
```

```
word_index :  {'the': 1, 'earth': 2, 'is': 3, 'an': 4, 'awesome': 5, 'place': 6, 'live': 7}
```

```
test_text = ['The', 'earth', 'live']
sequences = t.texts_to_sequences(test_text)
print('sequences : ',sequences,'\n')
```

```
sequences :  [[1], [2], [7]]
```

tf.keras.preprocessing.text.Tokenizer

```
docs = ['Well done!',
     'Good work',
     'Great effort',
     'nice work',
     'Excellent!']

# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
t.word_index.items()
```

```
dict_items([('work', 1), ('well', 2), ('done', 3), ('good', 4), ('great', 5), ('effort', 6),
('nice', 7), ('excellent', 8)])
```

```
test_text = ['Good nice work']
sequences = t.texts_to_sequences(test_text)
print('sequences : ',sequences,'\n')
```

```
sequences :  [[4, 7, 1]]
```

tf.keras.preprocessing.text.Tokenizer sequence to texts

```
test_text = ['Good nice work']
sequences = t.texts_to_sequences(test_text)
print('sequences : ',sequences,'\n')
```

```
sequences :  [[4, 7, 1]]
```

```
text = t.sequences_to_texts(sequences)
print(text)
```

```
['good nice work']
```

```
text = t.sequences_to_texts([[7]])
print(text)
```

```
['nice']
```

tf.keras.preprocessing.text. text_to_word_sequence

```
tf.keras.preprocessing.text.text_to_word_sequence(
    text, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', lower=True, split=' '
)
```

tf.keras.preprocessing.text. text_to_word_sequence

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.text import text_to_word_sequence
max_words = 10000

text = 'Decreased glucose-6-phosphate dehydrogenase activity along with oxidative stress affect
text = text_to_word_sequence(text)
print(text)
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(text)
sequences = tokenizer.texts_to_sequences(text)
print(sequences)
```

```
['decreased', 'glucose', '6', 'phosphate', 'dehydrogenase', 'activity', 'along', 'with', 'oxi
dative', 'stress', 'affects', 'visual', 'contrast', 'sensitivity', 'in', 'alcoholics']
[[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]]
```

Convert to vector

fit_on_texts

```
fit_on_texts(
    texts
)
```

Updates internal vocabulary based on a list of texts.

In the case where texts contains lists, we assume each entry of the lists to be a token.

Required before using `texts_to_sequences` or `texts_to_matrix`.

texts_to_matrix

```
texts_to_matrix(
    texts, mode='binary'
)
```

Convert a list of texts to a Numpy matrix.

Arguments

```
texts: list of strings.
mode: one of "binary", "count", "tfidf", "freq".
```

tf.keras.preprocessing.text.Tokenizer

```
docs = ['Well done!',
     'Good work',
     'Great effort',
     'nice work',
     'Excellent!']

# create the tokenizer
t = Tokenizer()

# fit the tokenizer on the documents
t.fit_on_texts(docs)
print(t)
encoded_docs = t.texts_to_matrix(docs, mode='count')
print(encoded_docs)
print(t.word_index.items())
```

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

tf.keras.preprocessing.text.Tokenizer

tf.idf

```
docs = ['Well done!',
     'Good work',
     'Great effort',
     'nice work',
     'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
encoded_docs = t.texts_to_matrix(docs, mode='tfidf')
print(encoded_docs)
print(t.word_index.items())
```

```
[[0.         0.         1.25276297 1.25276297 0.         0.
  0.         0.         0.        ]
 [0.         0.98082925 0.         0.         1.25276297 0.
  0.         0.         0.        ]
 [0.         0.         0.         0.         0.         1.25276297
  1.25276297 0.         0.        ]
 [0.         0.98082925 0.         0.         0.         0.
  0.         1.25276297 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         1.25276297]]
dict_items([('work', 1), ('well', 2), ('done', 3), ('good', 4), ('great', 5), ('effort', 6),
('nice', 7), ('excellent', 8)])
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
vectorizer = TfidfVectorizer(max_features=300)
vectorizer = vectorizer.fit(X_train)

df_train = vectorizer.transform(X_train)

print(df_train.shape)
print(df_train)
```

```
(900, 300)
  (0, 133)	1.0
  (1, 89)	0.7403480856124964
  (1, 65)	0.6722237069085794
  (2, 246)	0.30211756738541284
  (2, 238)	0.6850871121270957
```

```python
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
```

https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/#.Xsij-RMzbEY