

Machine Learning

Lecture 5: Text Classification

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Topics

- Text Preprocessing
 - Regular Expression
 - Tokenization
 - Normalization
 - Stopword Removal
 - POS Tagging
- Text Classification

Text Preprocessing

Part 1

Text Classification

Part 2

Loading data

```
import nltk
f = open('harry_7books.txt', encoding='utf8')
raw = f.read()
tokens = nltk.word_tokenize(raw)
```

```
tokens[1000:1100]
```

```
['',
 'eating',
 'banquets',
 'in',
 'the',
 'Great',
 'Hall',
 '',
 'sleep\xading',
 'in',
 'his',
 'four-poster',
```

Regular expression for detecting word patterns

- Many analysis tasks involve pattern matching
- Regular expressions give us a powerful and flexible method for describing the character patterns we are interested in.
- To use regular expressions in Python we need to import the re library using: `import re`.

Using basic meta-characters

\$ end of word

```
tokens = [w.lower() for w in tokens]
```

```
import re
```

```
[w for w in tokens if re.search('ed$', w)]
```

```
['reserved',  
'published',  
'registered',  
'related',  
'reproduced',  
'stored',  
'transmitted',  
'opened',  
'released',
```

Using basic meta-characters

```
[w for w in tokens if re.search('^..j..t..$', w)]
```

```
['dejected',  
'adjusted',  
'rejected',  
'rejected',  
'dejected',  
'adjusted',  
'adjusted',  
'adjusted',  
'adjusted']
```

- . A wildcard symbol matches any single character

- ^ A caret symbol matches the start of a string

Using basic meta-characters ? the preceding is optional

```
[w for w in tokens if re.search('^spo.ts?$', w)]
```

```
['sport',  
'sports',  
'sport',  
'sport',  
'sports',  
'sport',  
'sports',  
'sports',  
'sports',  
'sports',  
'sports',  
'sports',  
'sports',  
'sports',  
'sports',
```

Ranges and Closures

Range

```
[w for w in tokens if re.search('^[b-f]', w)]
```

```
['chamber',  
 'by',  
 'by',  
 'books',  
 'for',  
 'f.',  
 'driver',  
 'foul-weather',  
 'friend',  
 'copyright',  
 'by',
```

Ranges and Closures

[] possible options

```
[w for w in tokens if re.search('^[bdf][aue][rts]$', w)]
```

```
['but',  
 'but',  
 'far',  
 'but',  
 'but',  
 'but',  
 'fat',  
 'but',  
 'fat',
```

Ranges and Closures

+ * closure

```
set([w for w in tokens if re.search('noo+$', w)])
```

```
{'nooo',  
 'noooo',  
 'nooooo',  
 'noooooo',  
 'nooooooo',  
 'noooooooo',  
 'nooooooooo',  
 'noooooooooo',  
 'nooooooooooo',  
 'nooooooooooo'}
```

```
set([w for w in tokens if re.search('noo*$', w)])
```

```
{'albino',  
 'dunno',  
 'gemino',  
 'inferno',  
 'n-no',  
 'no',  
 'nooo',  
 'noooo',
```

+ one or more
instances

* zero or more
instances

\ special character

```
[w for w in tokens if re.search('\.+$', w)]
```

```
['j.',  
'k.',  
'a.',  
'p.',  
'f.',  
'.',  
'j.',  
'k.',  
'.',  
'bros.',
```

`{` number of character

```
[w for w in tokens if re.search('^[0-9]{4}$', w)]
```

```
['1999',  
'1999',  
'1920',  
'1999',  
'1999',  
'1875',  
'1492',  
'1289',  
'2007',  
'2007',  
'1966',
```

{a,b} number of repeats from a to b

```
nltk.download('treebank')  
wsj = sorted(set(nltk.corpus.treebank.words()))  
[w for w in wsj if re.search('^[0-9]+-[a-z]{3,5}$', w)]
```

```
[nltk_data] Downloading package treebank to  
[nltk_data] C:\Users\santitham\AppData\Roaming\nltk_data...  
[nltk_data] Unzipping corpora\treebank.zip.
```

```
['10-day',  
 '10-lap',  
 '10-year',  
 '100-share',  
 '12-point',  
 '12-year',  
 '14-hour',  
 '15-day',  
 '150-point',  
 '190-point',
```

{a,} number of repeats from a
{,b} number of repeats up to b

```
[w for w in wsj if re.search('^[a-z]{5,}-[a-z]{2,3}-[a-z]{,6}$', w)]
```

```
['black-and-white',  
'bread-and-butter',  
'father-in-law',  
'machine-gun-toting',  
'savings-and-loan']
```


| match one of specified strings
() scope of operator

```
set([w for w in tokens if re.search('(ed|ing)$', w)])
```

```
{'wording',  
 'darted',  
 'okay-looking',  
 'cir\xadcling',  
 'performed',  
 'display\xading',  
 'underlying',  
 'resurfaced',  
 'scorching',
```

Regular expression symbol

Symbol	Function
<code>\b</code>	Word boundary (zero width)
<code>\d</code>	Any decimal digit (equivalent to <code>[0-9]</code>)
<code>\D</code>	Any non-digit character (equivalent to <code>[^0-9]</code>)
<code>\s</code>	Any whitespace character (equivalent to <code>[\t\n\r\f\v]</code>)
<code>\S</code>	Any non-whitespace character (equivalent to <code>[^ \t\n\r\f\v]</code>)
<code>\w</code>	Any alphanumeric character (equivalent to <code>[a-zA-Z0-9_]</code>)
<code>\W</code>	Any non-alphanumeric character (equivalent to <code>[^a-zA-Z0-9_]</code>)
<code>\t</code>	The tab character
<code>\n</code>	The newline character

NLTK regular expression tokenizer

```
s = ("Good muffins cost $3.88\nin New York.  Please buy me\n"
     "two of them.\n\nThanks.")
s2 = ("Alas, it has not rained today. When, do you think, "
     "will it rain again?")
s3 = ("<p>Although this is <b>not</b> the case here, we must "
     "not relax our vigilance!</p>")
```

s2

'Alas, it has not rained today. When, do you think, will it rain again?'

```
nltk.regexp_tokenize(s2, r'[.,\.\?!"]\s*', gaps=True)
```

```
['Alas',
 'it has not rained today',
 'When',
 'do you think',
 'will it rain again']
```

NLTK regular expression tokenizer

```
s3
```

```
'<p>Although this is <b>not</b> the case here, we must not relax our vigilance!</p>'
```

```
nltk.regexp_tokenize(s3, r'</?.>', gaps=False)
```

```
['<p>', '<b>', '</b>', '</p>']
```

```
nltk.regexp_tokenize(s3, r'</?.>', gaps=True)
```

```
['Although this is ',  
 'not',  
 ' the case here, we must not relax our vigilance!']
```

Regular expression tokenizer

Removing all punctuations

```
tokenizer = nltk.RegexpTokenizer(r"\w+")  
tokens_0 = tokenizer.tokenize(raw)  
print(tokens_0[2000:2100])
```

```
['him', 'at', 'all', 'Neither', 'of', 'them', 'had', 'written', 'to'  
'said', 'he', 'was', 'going', 'to', 'ask', 'Harry', 'to', 'come', 'a'  
'on', 'the', 'point', 'of', 'unlocking', 'Hedwig', 's', 'cage', 'by'  
rmione', 'with', 'a', 'letter', 'but', 'it', 'wasn', 't', 'worth', '  
d', 'to', 'use', 'magic', 'outside', 'of', 'school', 'Harry', 'hadn'  
t', 'was', 'only', 'their', 'terror', 'that', 'he', 'might', 'turn',  
d', 'them', 'from', 'locking', 'him', 'in']
```

Normalization

- British vs. American spellings:
 - colour vs. color.
- Multiple formats for dates, times:
 - 09/30/2013 vs. Sep 30, 2013.
- Asymmetric expansion:
 - Enter: **window** Search: **window, windows**
 - Enter: **windows** Search: **Windows, windows, window**
 - Enter: **Windows** Search: **Windows**

Stemming

- Reduce tokens to “root” form of words to recognize morphological variation.
 - “computer”, “computational”, “computation” all reduced to same token “compute”
- Correct morphological analysis is language specific and can be complex.
- Stemming “blindly” strips off known affixes (prefixes and suffixes) in an iterative fashion.

for example compressed and compression are both accepted as equivalent to compress.



for example compress and compress are both accepted as equivalent to compress.

Porter Stemmer

- Simple procedure for removing known affixes in English without using a dictionary.
- Can produce unusual stems that are not English words:
 - “computer”, “computational”, “computation” all reduced to same token “comput”
- May conflate (reduce to the same token) words that are actually distinct.
- Not recognize all morphological derivations.

Typical rules in Porter

- *sses* → *ss*
 - *ies* → *i*
 - *ational* → *ate*
 - *tional* → *tion*
-
- See class website for link to “official” Porter stemmer site
 - Provides Python ready to use implementations

Porter Stemmer Errors

- Errors of “comission”:
 - organization, organ → organ
 - police, policy → polic
 - arm, army → arm
- Errors of “omission”:
 - cylinder, cylindrical
 - create, creation
 - Europe, European

Case normalization

```
tokens = [w.lower() for w in tokens]
```

```
tokens
```

```
['harry',  
 'potter',  
 'and',  
 'the',  
 'chamber',  
 'of',  
 'secrets',  
 'by',  
 'j.',  
 'k.',  
 'rowling',  
 'illustrations',  
 'by',  
 'mary',  
 'grandpré',
```

Clean \xad

8.1 Replacing \xad

```
set([w.replace('\xad', '') for w in tokens if w.startswith('Gry') ])
```

```
{'Gryff',  
 'Gryffindor',  
 'Gryffindor-Ravenclaw',  
 'Gryffindor-Slytherin',  
 'Gryffindor.',  
 'Gryffindors',  
 'Gryffindors.',  
 'Gryffndor'}
```

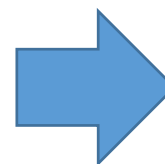
```
token_1 = [w.replace('\xad', '') for w in tokens]
```

Stemming

```
porter = nltk.PorterStemmer()  
tokens_2 = [porter.stem(t) for t in tokens_1]
```

tokens_1[1000:1100]

```
['',  
'eating',  
'banquets',  
'in',  
'the',  
'Great',  
'Hall',  
'',  
'sleeping',  
'in',  
'his',  
'four-poster',  
'bed',  
'in',  
'the',  
'tower',  
'dormitory',
```



tokens_2[1000:1100]

```
['',  
'eat',  
'banquet',  
'in',  
'the',  
'great',  
'hall',  
'',  
'sleep',  
'in',  
'hi',  
'four-post',  
'bed',  
'in',  
'the',  
'tower',  
'dormitori',
```

Lemmatization

- Reduce inflectional/variant forms to base form
- Direct impact on vocabulary size
- E.g.,
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*

Lemmatization

- How to do this?
 - Need a list of grammatical rules + a list of irregular words
 - Children → child, spoken → speak ...
 - Practical implementation: use WordNet's morphstr function
 - Perl: WordNet::QueryData (first returned value from validForms function)
 - Python: NLTK.stem
 - R: wordnet

Wordnet Lemmatization

```
nltk.download('wordnet')  
wnl = nltk.WordNetLemmatizer()  
  
print("rocks :", wnl.lemmatize("rocks"))  
print("corpora :", wnl.lemmatize("corpora"))  
print("running :", wnl.lemmatize("running", pos='v'))
```

```
rocks : rock  
corpora : corpus  
running : run
```


Application of text normalization

- Widely used in Text mining for data preparation
- Use to reduce variations/fragmentations in data
- Replace/improve by deep learning and embedding

Listing stopwords

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'whom', 'didn', 'its', 'couldn', 'mustn't', 'his', 'itself', 'what', 'a', 'hims',  
'or', 'did', 'such', 'be', 'had', 'wasn't', 'shouldn', 'here', 'their', 'herself',  
'o', 'ain', 'will', 'but', 'was', 'both', 'no', 'there', 'from', 'very', 'didn't',  
her', 'wouldn't', 'doesn't', 'isn't', 'until', 'nor', 'yours', 'do', 'hasn't',  
'can', 'you'd', 'are', 'themselves', 'mustn', 'you're', 'against', 'to', 'these',  
n't', 'other', 'those', 'after', 'than', 'her', 'weren', 'have', 'theirs', 'havi  
han't', 'how', 'in', 'under', 'why', 'haven't', 'mightn't', 'they', 'once', 'jus  
e's", "that'll", 'because', 's', 'wasn', 'with', 'haven', 'has', 'll', 'my', 'by  
re', 'at', 'of', "won't", 'yourself', 'being', 'only', "should've", 'as', "it's",  
'mightn', 'myself', 'some', 'your', 'and', 'am', 'before', 'hadn', 'me', 'been',  
ing', 'him', 'who', 'doesn', 'isn', 'while', 'needn', 'again', 'out', 'each', 'i  
h', 'so', 'we', 'd', 'ma', 'between', 'should', 'own', "you've", 'this', 'that'}
```

Stopword removal

```
print(tokens[2000:2100])
```

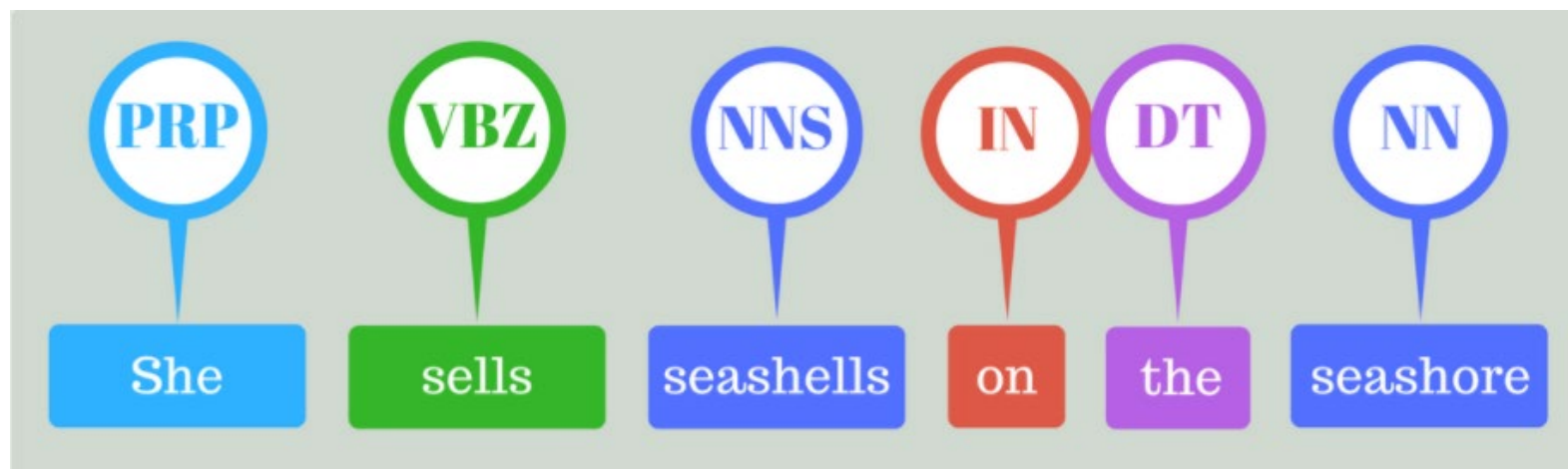
```
['i', '', 'll', 'be', 'in', 'my', 'room', ',', 'making', 'no', 'noise', 'and',  
, 'said', 'harry', 'dully', '.', '"', 'precisely', '.', 'now', ',', 'we'  
w', 'good', 'compli\xadments', 'at', 'dinner', '.', 'petunia', ',', 'any', 'idea  
u', '', 're', 'a', 'wonderful', 'golfer', ',', 'mr.', 'mason', '.', '...', 'do',  
'dress', ',', 'mrs.', 'mason', '.', '...', '"', '"', 'perfect', '...', 'dudley', '?'  
d', 'to', 'write', 'an', 'essay', 'about', 'our', 'hero', 'at', 'school', ',']
```

```
tokens_4 = [w for w in tokens if not w in stop_words]  
print(tokens_4[2000:2100])
```

```
['house', '.', '"', 'muuum', '!', '', 'know', '!', '"', 'harry', 'paid', 'dear  
'hedge', 'way', 'hurt', ',', 'aunt', 'petunia', 'knew', '', 'really', 'done', '  
'blow', 'head', 'soapy', 'frying', 'pan', '.', 'gave', 'work', ',', 'promise', '  
led', 'around', 'watching', 'eating', 'ice', 'cream', ',', 'harry', 'cleaned', '  
'lawn', ',', 'trimmed', 'flowerbeds', ',', 'pruned', 'watered', 'roses', ',', 'r  
'blazed', 'overhead', ',', 'burning', 'back', 'neck', '.', 'harry', 'knew', '',  
'said', 'thing', 'harry', 'thinking', '...', 'maybe']
```

Part of speech (POS) tagging

- Tagging of words in a corpus with the correct part of speech
- Early automatic POS taggers were rule-based
- Modern POS taggers used a combination of rule-based and machine learning



NLTK POS Tagging

```
nltk.download('averaged_perceptron_tagger')  
tokens_5 = nltk.pos_tag(tokens)
```

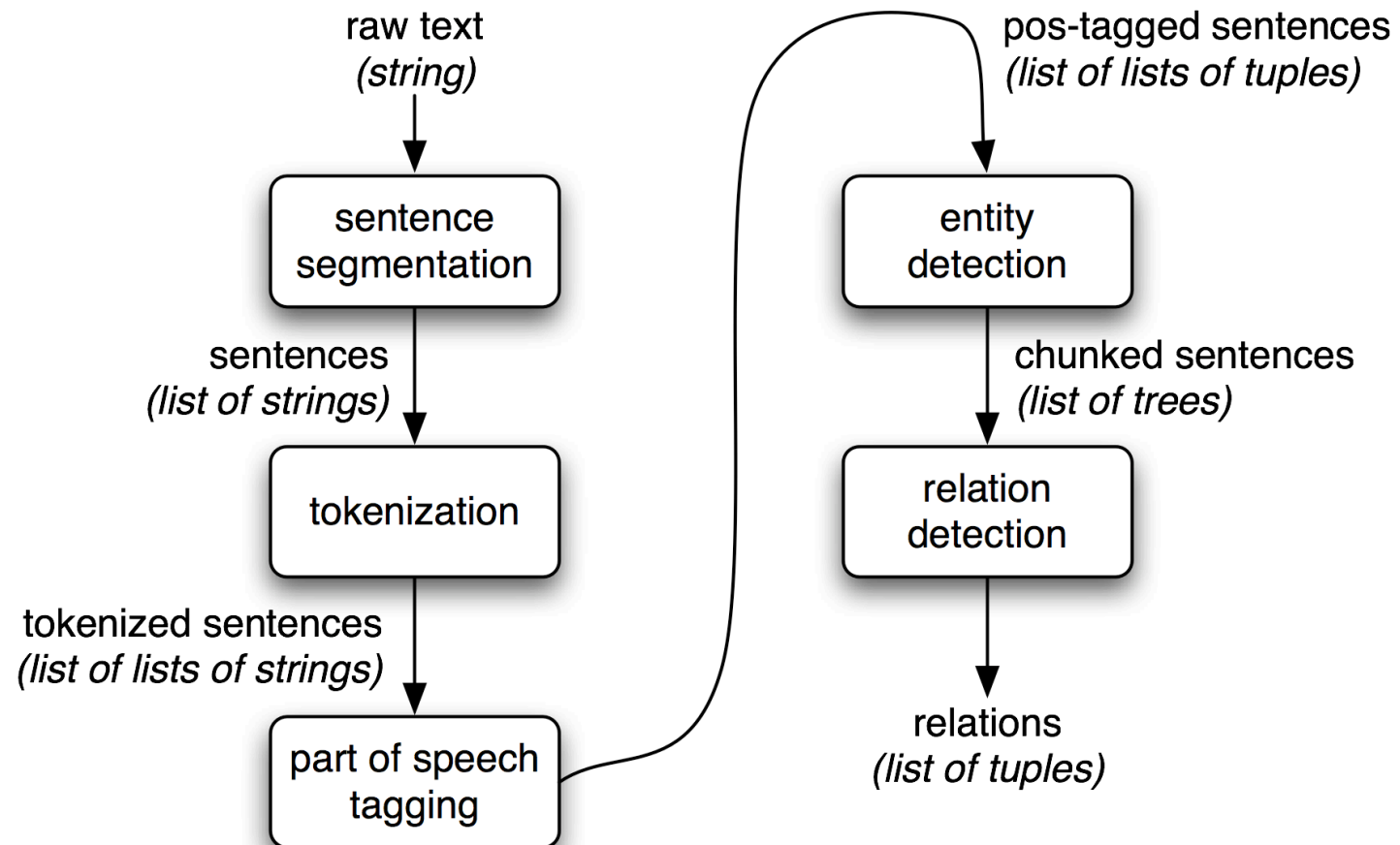
```
[nltk_data] Downloading package averaged_perceptron_tagger to  
[nltk_data] C:\Users\santitham\AppData\Roaming\nltk_data...  
[nltk_data] Unzipping taggers\averaged_perceptron_tagger.zip.
```

```
tokens_5[2000:2100]
```

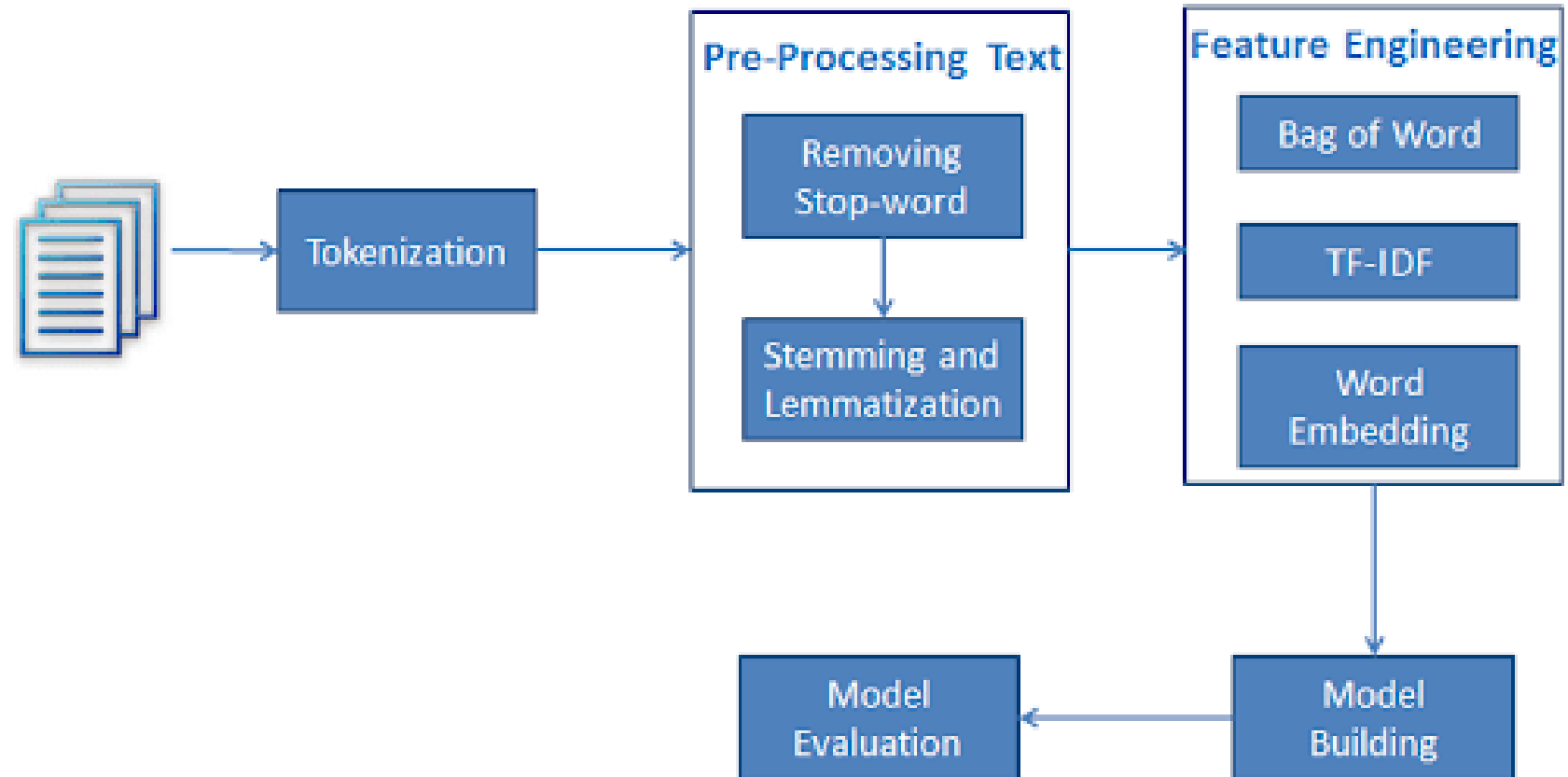
```
[('i', 'JJ'),  
 ('', 'NNP'),  
 ('ll', 'NN'),  
 ('be', 'VB'),  
 ('in', 'IN'),  
 ('my', 'PRP$'),  
 ('room', 'NN'),  
 (',', ','),  
 ('making', 'VBG'),  
 ('no', 'DT'),  
 ('noise', 'NN'),  
 ('and', 'CC'),
```

Next steps

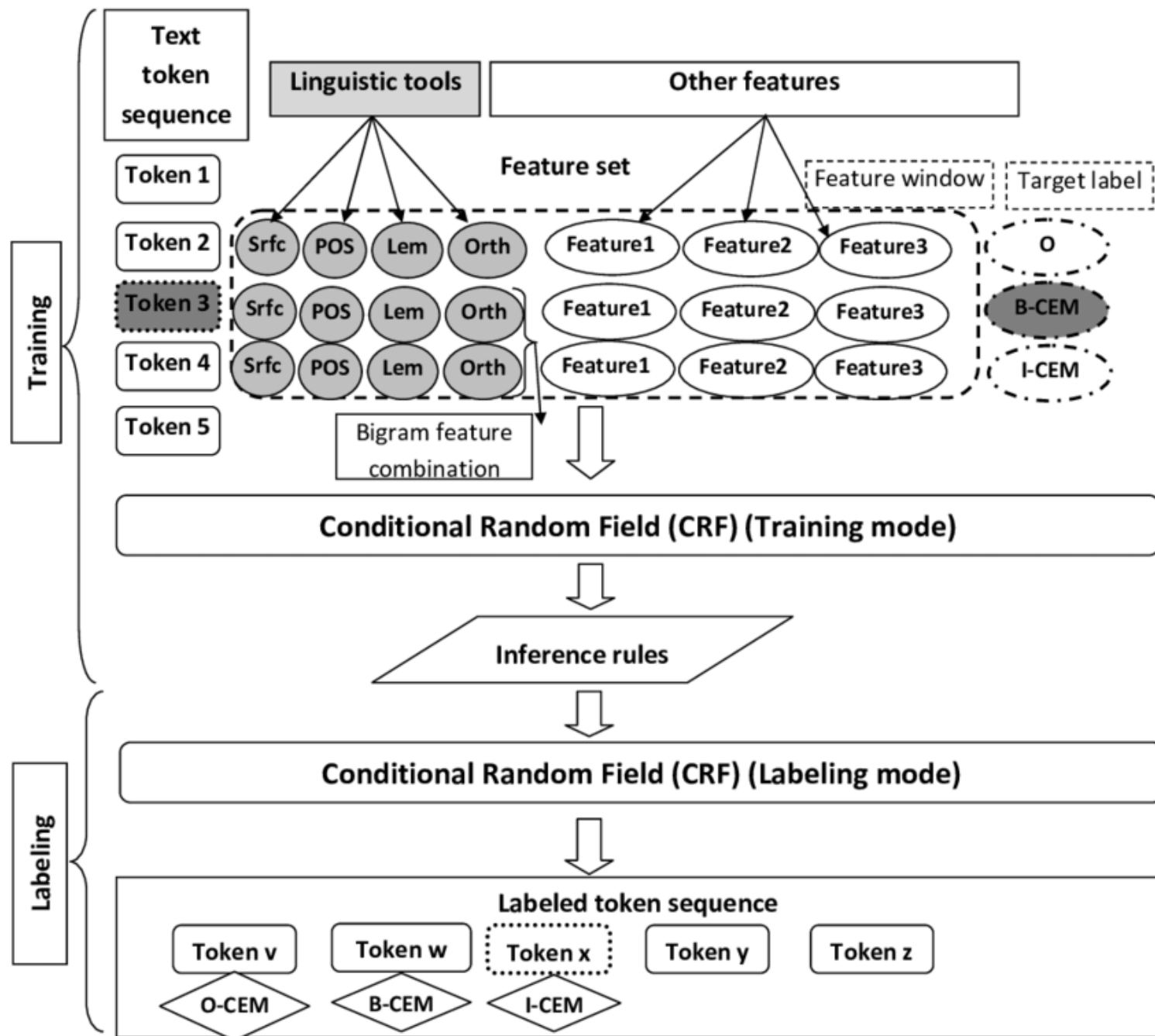
Information extraction



Next steps Classification



Name Entity Recognition (NER)



Text Classification in General

- Given a text item, which class / category should it belong to?
- Task:

$$D \Rightarrow c$$

where $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ is a set of input text, and
 $c = \{c_1, c_2, \dots\}$ is a set of class/topic labels

Main Types of Classification

- Binary: Each item belong to one of the two classes
 - e.g. spam/ham, positive/negative sentiment
- Single-Label Multi-Class: Each item belong to one of the multiple classes.
 - e.g. assigning chat into an intent group (inquiry, purchase, support, information)
- Multi-Label Multi-Class: Each item can have multiple labels
 - e.g. assigning an article to classes of topics (mathematics, biology, chemistry, ecology, medical, etc.)

Hard vs Soft Classifications

- Hard classification:
 - Good for automated systems, require constant tuning
 - Train a soft classifier, $f(d, c)$
 - Use a specific threshold, t , to define a class
 - if $f(d, c) > t$, d is predicted to be the class c_1
 - if $f(d, c) > t$, d is predicted to be the class c_2
- Soft classification
 - Good when more than one outputs are required
 - Probability of d belong to c_i
 - Give only ranking/probabilities

Objective

- Topics: most frequent cases
- Sentiment: useful in market research, online reputation
- CRM, social sciences, political science
- Language: in search engine
- Genre ; e.g., sport, crime, drama, etc.
- etc

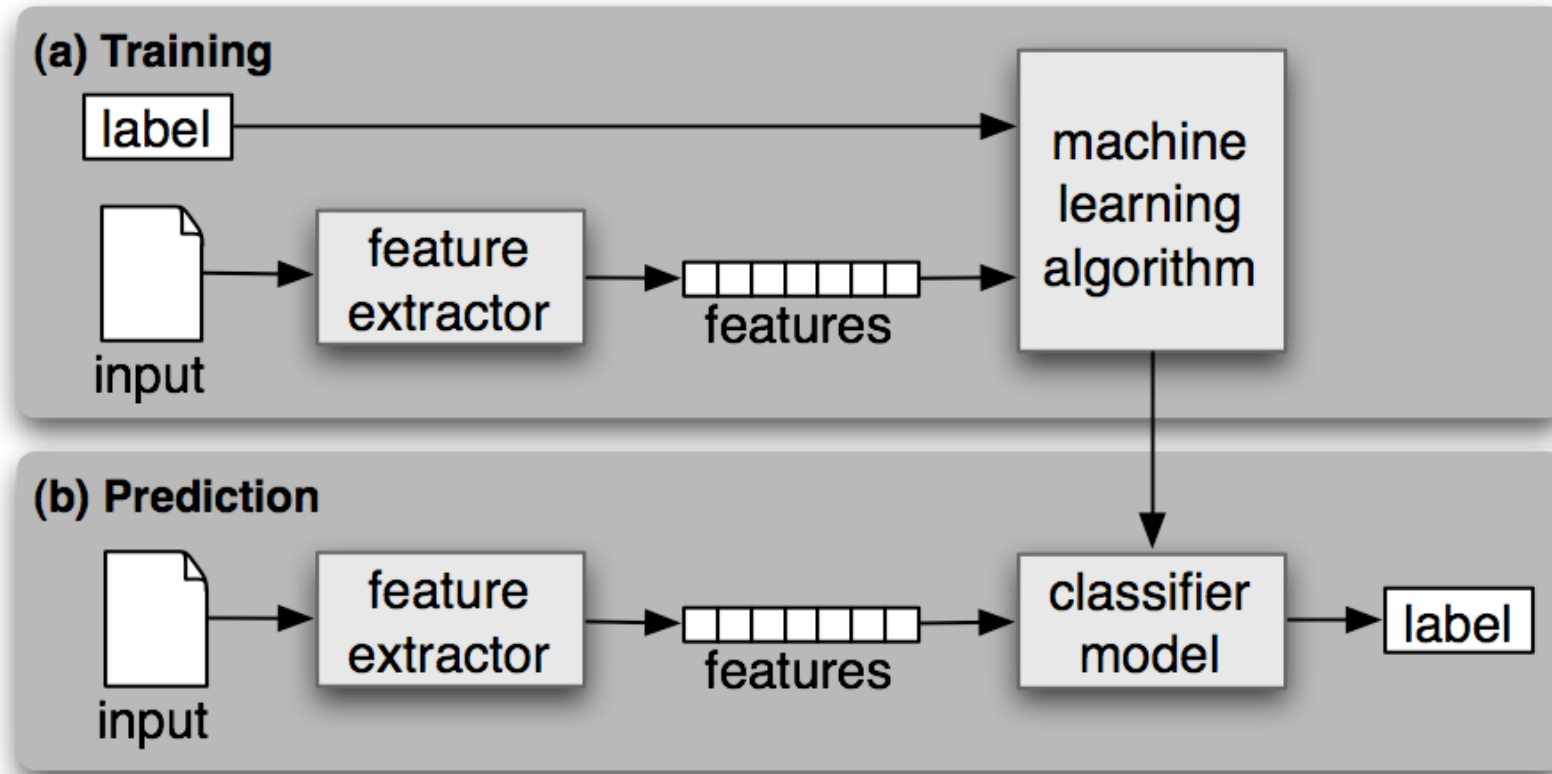
Use cases

- Document classification
 - Classify a website into categories
 - Classify types of documents
- Filtering
 - Filter out spam messages
 - Detect unsuitable contents
- Information retrieval
 - Classify intent of the chat message
 - Detect name entities

Scope

- Document-level:
 - Topics of the e-mail / news / articles
- Sentence-level
 - Topics of the headline / title
- Sub-sentence-level
 - Topics in the product reviews

ML-based topic classification



ML approaches

Traditional ML

- Naïve Bayes
- SVM
- etc.

Modern ML

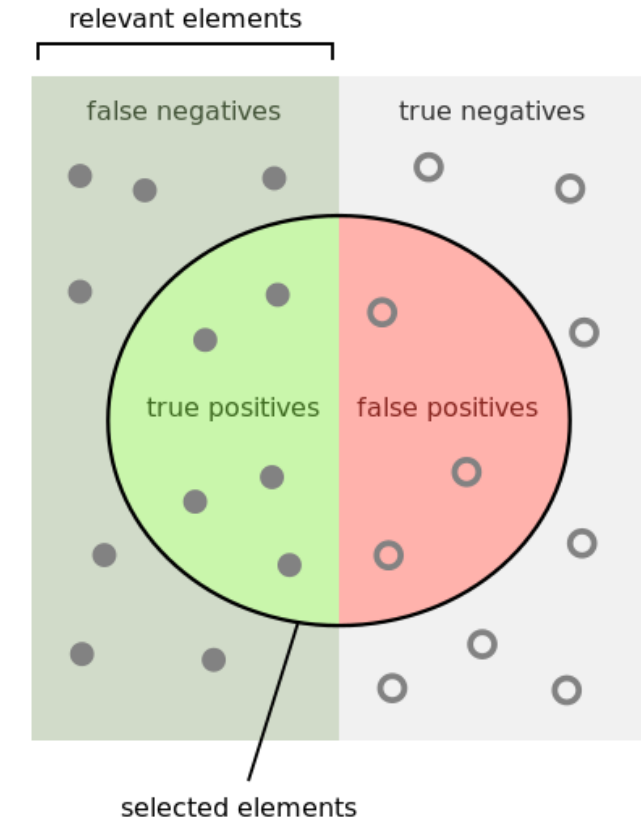
- Deep Learning
- Text Representation

Metrics

	Actual Positive (p)	Actual Negative (n)
The model says “Yes” = positive (y)	True positives	False positives
The model says “No” = not positive (n)	False negatives	True negatives


- Accuracy = $(TP + TN) / (TP + FP + TN + FN)$
- Recall (Completeness) = true positive rate = $TP / (TP + FN)$
- Precision (Exactness) = the accuracy over the cases predicted to be positive, $TP / (TP + FP)$
- F-measure = the harmonic mean of precision and recall
 = the balance between recall and precision

$$= 2 \cdot \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$



How many selected items are relevant?

Precision = $\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$



How many relevant items are selected?

Recall = $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$



End of Lecture 5

Question?