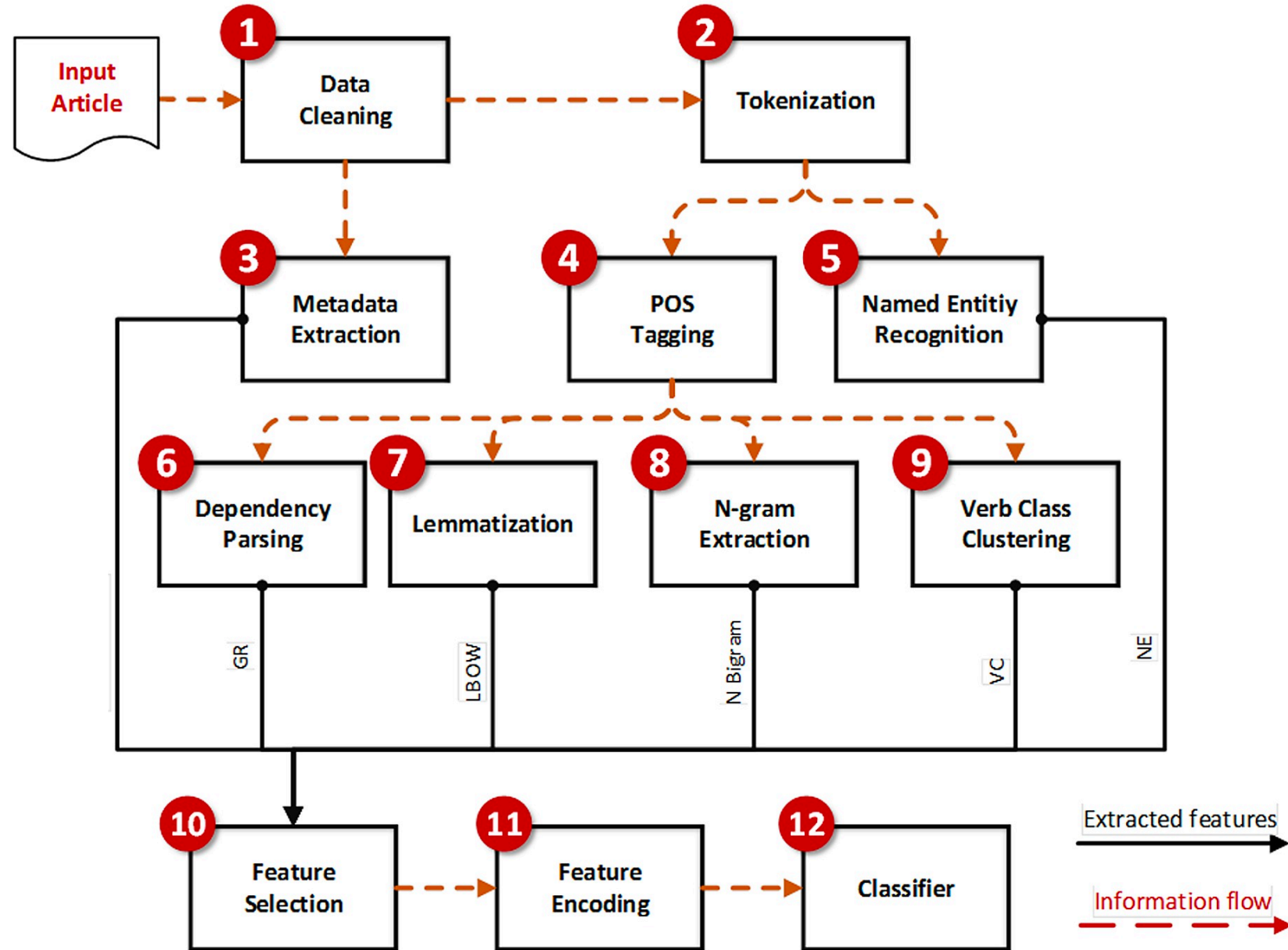


Text Wrangling Pipeline

Machine Learning algorithms requires **Numeric Input Features**

GR: Grammatical Relations,
LBOW: Lemmatized Bag of Words,
N.Bigrams: Noun Bigrams,
VC: Verb Clusters,
NE: Named Entities

An Example of an NLP Pipeline for Classification Tasks



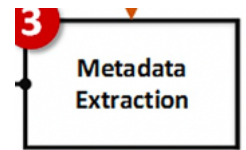
Text Wrangling Pipeline



Pre-process retrieved data, automatically cleaning the text (encoding errors, HTML tags).



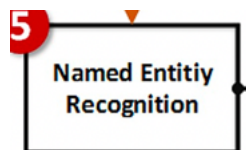
Segment the text by words and then by sentences (Natural Language Tool Kit NLTK or Spacy).



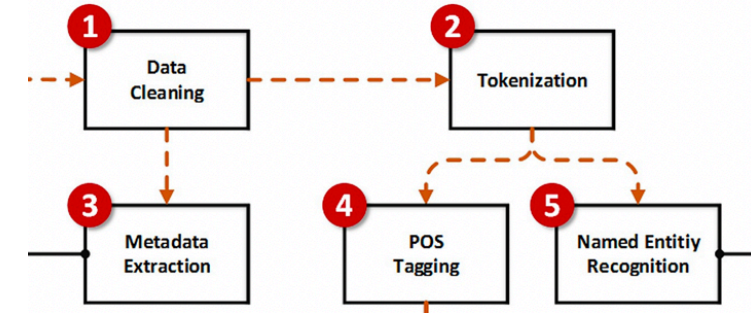
Documents are often accompanied with metadata that capture additional descriptive categorization information about the document, such as tags or keywords.



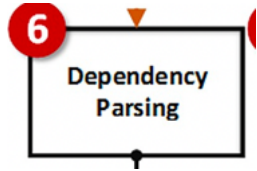
Label each word in the text with a lexical category label (Penn Treebank).



Identify and extract named entities.



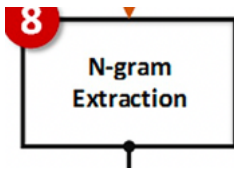
Text Wrangling Pipeline



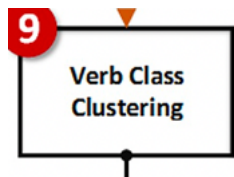
Extract syntactic structures (trees) that encode grammatical dependency relations between words in the sentence.



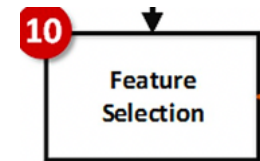
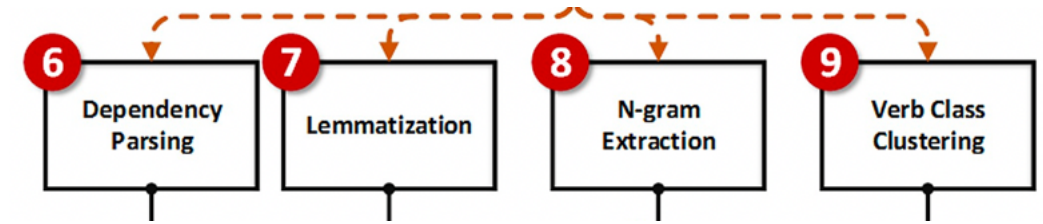
Lemmatize the text in order to reduce words sparsity.



Extract noun compound bigrams, as they can represent a concept in the text (“drinking water”).



Group semantically similar verb predicates together for example, verbs like “stimulate” and “activate” would be clustered together.



Remove rare or very common features. Apply thresholds.

Text Wrangling Pipeline – Step 1 Data Cleaning

```
data = requests.get('http://www.gutenberg.org/cache/epub/8001/pg8001.html')
```

```
content = data.content
```

```
print(content[1163:2200])
```

```
b'content="Ebookmaker 0.4.0a5 by Marcello Perathoner <webmaster@gutenberg.org>" name="generator"/>\r\n</head>\r\n <body><p id="id00000">Project Gutenberg EBook The Bible, King James, Book 1: Genesis</p>\r\n\r\n<p id="id00001">Copyright laws are changing all over the world. Be sure to check the\r\ncopyright laws for your country before downloading or redistributing\r\nthis or any other Project Gutenberg eBook.</p>\r\n\r\n<p id="id00002">This header should be the first thing seen when viewing this Project\r\nGutenberg file. Please do not remove it. Do not change or edit the\r\nheader without written permission.</p>\r\n\r\n<p id="id00003">Please read the "legal small print," and other information about the\r\neBook and Project Gutenberg at the bottom of this file. Included is\r\nimportant information about your specific rights and restrictions in\r\nhow the file may be used. You can also find out about how to make a\r\ndonation to Project Gutenberg, and how to get involved.</p>\r\n\r\n<p id="id00004" style="margin-top: 2em">**Welcome To The World of F'
```

BeautifulSoup a popular HTML parser


BeautifulSoup does not fetch the web page for you, you can use requests.get.

```
soup = BeautifulSoup(content, "html.parser")
```


Text Wrangling Pipeline – Step 1 Data Cleaning

Replace carriage return by new lines

Remove script and frames



```
def strip_html_tags(text):  
    soup = BeautifulSoup(text, "html.parser")  
    [s.extract() for s in soup(['iframe', 'script'])]  
    stripped_text = soup.get_text()  
    stripped_text = re.sub(r'[\r|\n|\r\n]'+, '\n', stripped_text)  
    return stripped_text
```



```
stripped_text = strip_html_tags(content)
```

Note: Text Cleaning Is Task- and text-/language- Specific.

Does your text have obvious typos or spelling mistakes?

What punctuation is used?

Do you have dashes (-)?

Does your text have apostrophes (') and quotes?

Do you have dates and units (\$) that require specific handling?

Does your text have names and titles (Mr.)

Substituting and correcting tokens

doesn't?

does not?

does n't?

```
replacement_patterns = [
    (r'won\\'t', 'will not'),
    (r'can\\'t', 'cannot'),
    (r'i\\'m', 'i am'),
    (r'ain\\'t', 'is not'),
    (r'(\w+)\\"ll', '\g<1> will'),
    (r'(\w+)\n\\'t', '\g<1> not'),
    (r'(\w+)\\"ve', '\g<1> have'),
    (r'(\w+)\\"s', '\g<1> s'),
    (r'it\\'s', '\g<1> is'),
    (r'(\w+)\\"re', '\g<1> are'),
    (r'(\w+)\\"d', '\g<1> would')
]
```

```
class RegexpReplacer(object):
    def __init__(self, patterns=replacement_patterns):
        self.patterns = [(re.compile(regex), repl) for (regex, repl) in patterns]
    def replace(self, text):
        s = text
        for (pattern, repl) in self.patterns:
            (s, count) = re.subn(pattern, repl, s)
        return s
```

(Deepti et al., 2016)

```
sample_text = "I've seen most powerful supercomputer. It's over twice as fast..."
```

```
replacer = RegexpReplacer()
replacer.replace(sample_text.lower())
```

```
'i have seen most powerful supercomputer. it is over twice as fast...'
```

Define patterns

Useful String Functions

```
sample_text.lower()
```

```
"i've seen most powerful supercomputer. it's over twice as fast..."
```

```
sample_text.upper()
```

```
"I'VE SEEN MOST POWERFUL SUPERCOMPUTER. IT'S OVER TWICE AS FAST..."
```

```
sample_text.title()
```

```
"I'Ve Seen Most Powerful Supercomputer. It'S Over Twice As Fast..."
```

```
sample_text.split(' ')
```

```
["I've",  
 'seen',  
 'most',  
 'powerful',  
 'supercomputer.',  
 "It's",  
 'over',  
 'twice',  
 'as',  
 'fast...']
```

<code>s.join(text)</code>	Combine the words of the text into a string using <code>s</code> as the glue
<code>s.split(t)</code>	Split <code>s</code> into a list wherever a <code>t</code> is found (whitespace by default)
<code>s.splitlines()</code>	Split <code>s</code> into a list of strings, one per line
<code>s.lower()</code>	A lowercased version of the string <code>s</code>
<code>s.upper()</code>	An uppercased version of the string <code>s</code>
<code>s.title()</code>	A titlecased version of the string <code>s</code>
<code>s.strip()</code>	A copy of <code>s</code> without leading or trailing whitespace
<code>s.replace(t, u)</code>	Replace instances of <code>t</code> with <code>u</code> inside <code>s</code>

Text Wrangling Pipeline – Step 2 Tokenization

Sentence Segmentation

sent_tokenize



Default Sentence Tokenizer

Segment text into meaningful sentences, using specific delimiters.

period (.)
new line (\n)
semicolon (;)
exclamation (!)
question (?)
Capitalization

Note: You need to install NLTK corpora also

```
default_st = nltk.sent_tokenize  
alice_sentences = default_st(text=alice)
```

```
print('Total sentences in alice:', len(alice_sentences))  
print('First 2 sentences in alice:-')  
print(np.array(alice_sentences[0:2]))
```

Total sentences in alice: 1625

First 2 sentences in alice:-

```
["[Alice's Adventures in Wonderland by Lewis Carroll :  
"Down the Rabbit-Hole\n\nAlice was beginning to get \\  
o: once or twice she had peeped into the\nbook her sis  
s the use of a book,' thought Alice 'without pictures
```

```
nltk.corpus.gutenberg.fileids()
```

```
['austen-emma.txt',  
'austen-persuasion.txt',  
'austen-sense.txt',  
'bible-kjv.txt',  
'blake-poems.txt',  
'bryant-stories.txt',  
'burgess-busterbrown.txt',  
'carroll-alice.txt',  
'chesterton-ball.txt',  
'chesterton-brown.txt',  
'chesterton-thursday.txt',  
'edgeworth-parents.txt',  
'melville-moby_dick.txt',  
'milton-paradise.txt',  
'shakespeare-caesar.txt',  
'shakespeare-hamlet.txt',  
'shakespeare-macbeth.txt',  
'whitman-leaves.txt']
```


Text Wrangling Pipeline – Step 2 Tokenization

Word Tokenization

word_tokenize

Split or segment sentences into their constituent words.

period (.)
new line (\n)
semicolon (;)
exclamation (!)
question (?)
Capitalization

```
default_wt = nltk.word_tokenize  
words = default_wt(alice)  
np.array(words[0:5])
```

```
array(['[', 'Alice', '"s"', 'Adventures', 'in'], dtype='<U10')
```

```
tokens_sentences = [default_wt(t) for t in default_st(alice)]  
print(tokens_sentences[2:5])
```

```
['So', 'she', 'was', 'considering', 'in', 'her', 'own', 'mind', '(', 'as', 'well', 'as', 'she', 'could', ',', 'for', 'the', 'hot', 'day', 'made', 'her', 'feel', 'very', 'sleepy', 'and', 'stupid', ')', ',', 'whether', 'the', 'pleasure', 'of', 'making', 'a', 'daisy-chain', 'would', 'be', 'worth', 'the', 'trouble', 'of', 'getting', 'up', 'and', 'picking', 'the', 'daisies', ',', 'when', 'suddenly', 'a', 'White', 'Rabbit', 'with', 'pink', 'eyes', 'ran', 'close', 'by', 'her', '.'], ['There', 'was', 'nothing', 'so', 'VERY', 'remarkable', 'in', 'that', ';', 'nor', 'did', 'Alice', 'think', 'it', 'so', 'VERY', 'much', 'out', 'of', 'the', 'way', 'to', 'hear', 'the', 'Rabbit', 'say', 'to', 'itself', ',', '"Oh", 'dear', '!'], ['Oh', 'dear', '!']]
```

sentence 1

sentence 2

sentence 3

Text Wrangling Pipeline – Step 2 Tokenization – Some Stats

Top-10 Words

1. Remove Punctuation *from nltk import RegexpTokenizer*
2. Lower case
3. Count tokens

Any One or More Words

```
tokenizer = RegexpTokenizer(r'\w+') #only words
tokens_sentences = [tokenizer.tokenize(t) for t in default_st(alice)]
words = [word.lower() for sentence in tokens_
```

```
word_counts = collections.Counter(words)
word_counts.most_common(10)
```

```
[('the', 1642),
 ('and', 872),
 ('to', 729),
 ('a', 632),
 ('it', 595),
 ('she', 553),
 ('i', 543),
 ('of', 514),
 ('said', 462),
 ('you', 411)]
```

```
all_chars = len(alice)
num_chars = len(alice.translate(str.maketrans('', '', whitespace)))
num_words = len(words)
num_sents = len(tokens_sentences)
num_vocab = len(set(words))
print(int(all_chars/num_words), int(num_chars/num_words),
      int(num_words/num_sents),
      int(num_words/num_vocab))
```

Indeed, the story is about Alice

Text Wrangling Pipeline – Step 2 Tokenization – Some Stats

Average

1. Average Word Length
2. Average Sentence Length
3. Lexical Diversity

a general property of a language

characteristics of particular authors

counting all chars including white spaces

```
all_chars = len(alice)
num_chars = len(alice.translate(str.maketrans('', '', whitespace)))
num_words = len(words)
num_sents = len(tokens_sentences)
num_vocab = len(set(words))
print(int(all_chars/num_words), int(num_chars/num_words),
      int(num_words/num_sents),
      int(num_words/num_vocab))
```

5 4 16 10

Lexical Diversity

“the range of different words used in a text, with a greater range indicating a higher diversity” (McCarthy and Jarvis 2010: 381).