COMP30810
Intro to Text Analytics

Dr. Binh Thanh Le

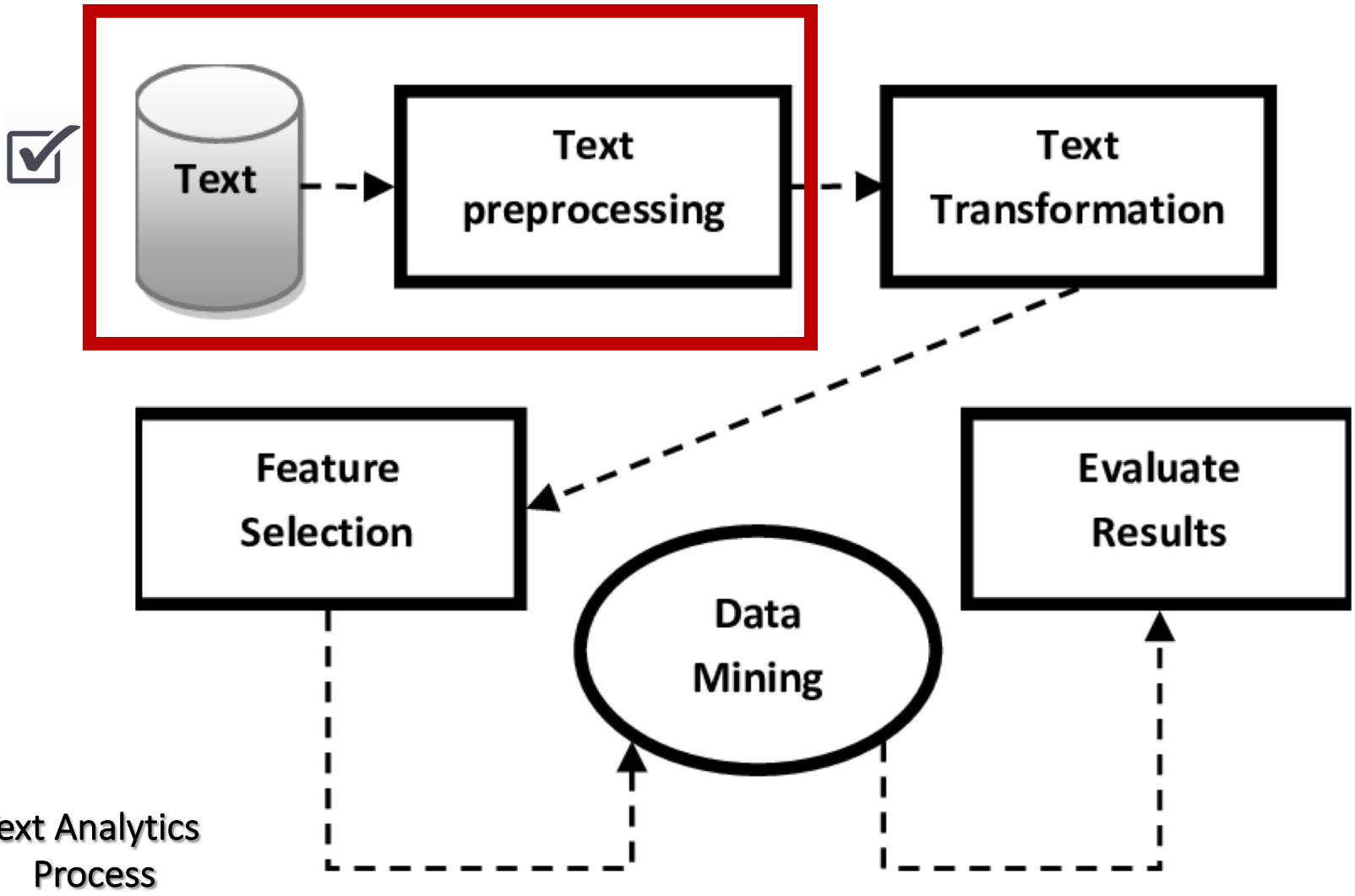
thanhbinh.le@ucd.ie

Insight Centre for Data Analytics

School of Computer Science

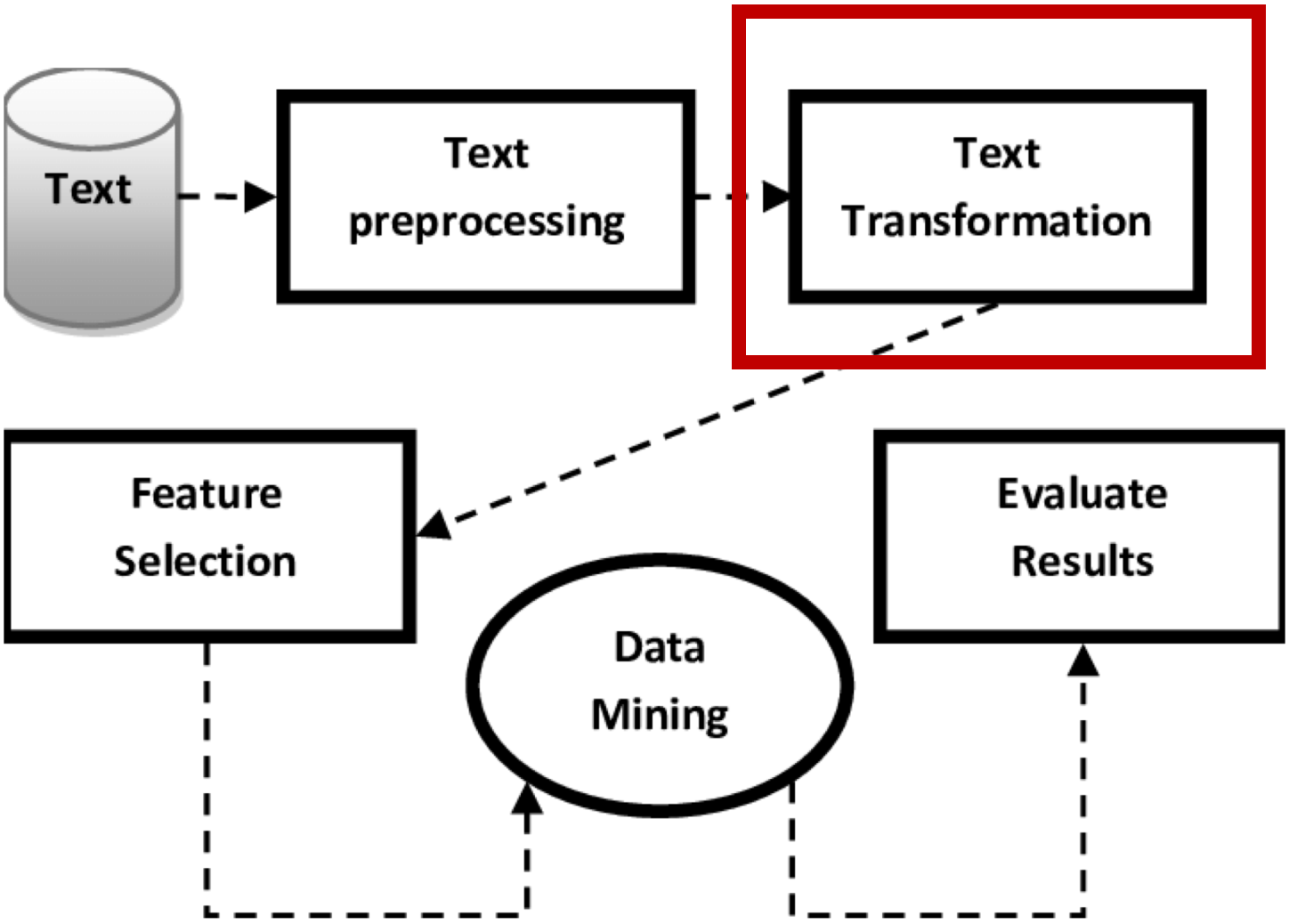
University College Dublin

Last lecture …



Text Analytics Process

Today task


Text
Text preprocessing
Text Transformation
Feature Selection
Data Mining
Evaluate Results

In [77]:

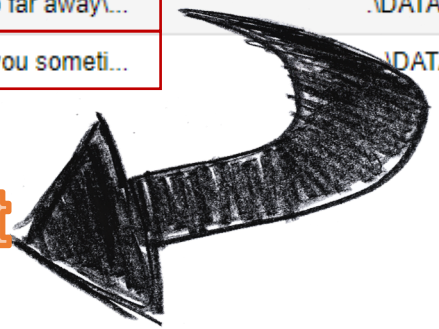
```
1  import os
2
3  path = '.\DATA\Kaggle\poetry'
4  df = pd.DataFrame([],columns=['filename','content','path'])
5
6  for root, directories, files in os.walk(path):
7      for filename in files:
8          _name = os.path.splitext(filename)[0]
9
10         filepath = os.path.join(root, filename)
11         f = open(filepath,"r",encoding="utf8")
12         content = f.read()
13         pieces = {'filename': _name, 'content': content, 'path': filepath}
14         df = df.append(pieces,ignore_index=True)
15 df
```

executed in 187ms, finished 16:17:03 2018-07-25

Out[77]:

	filename	content	path
0	adele	Looking for some education\nMade my way into t...	.\DATA\Kaggle\poetry\adele.txt
1	al-green	Let's stay together I, I'm I'm so in love with...	.\DATA\Kaggle\poetry\al-green.txt
2	alicia-keys	Ooh..... New York x2 Grew up in a town that ...	.\DATA\Kaggle\poetry\alicia-keys.txt
3	amy-winehouse	Build your dreams to the stars above\nBut when...	.\DATA\Kaggle\poetry\amy-winehouse.txt
4	beatles	Yesterday, all my troubles seemed so far away\...	.\DATA\Kaggle\poetry\beatles.txt
5	bieber	What do you mean?\nOh, oh, oh\nWhen you someti...	.\DATA\Kaggle\poetry\bieber.txt

We will handle this content



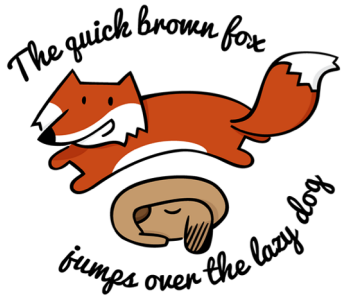
The Quick Brown Fox -- The Rest of the Story

The quick brown fox jumps over the lazy dog.

He lands head first on a rotting maple log.

Knocked unconscious, fox sleeps with shallow breath

until the lazy dog awakes and worries him to death.

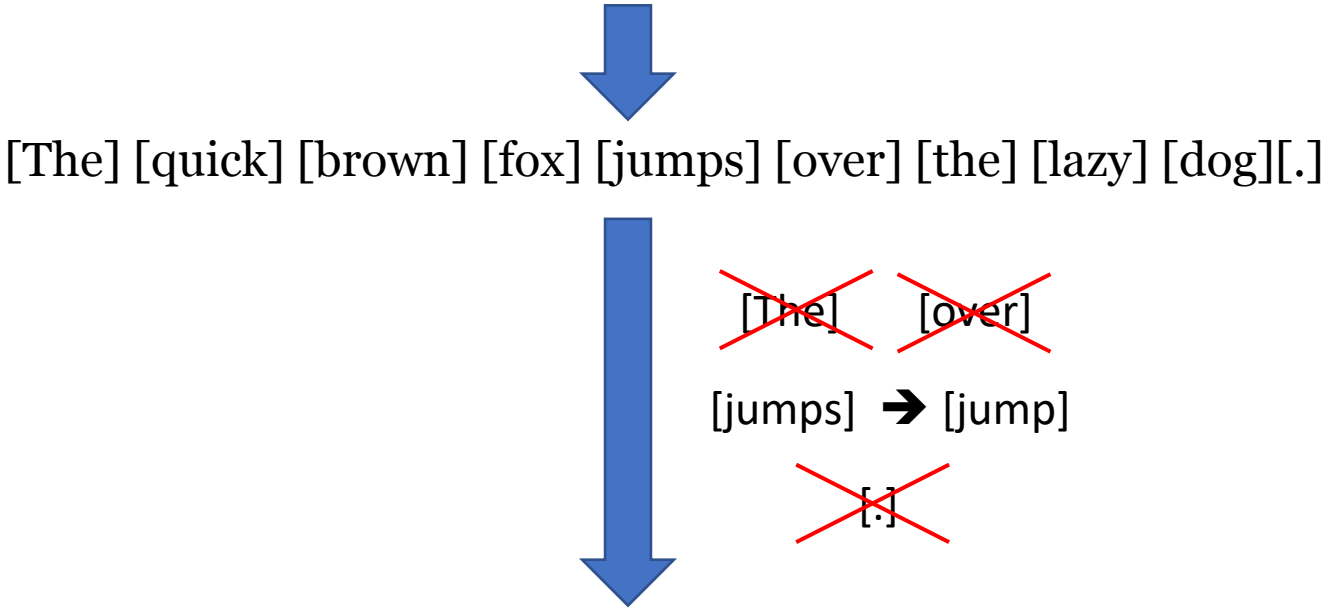


https://allpoetry.com/poem/1063961-The-Quick-Brown-Fox----The-Rest-of-the-Story-by-JennyLee

Today Goals:

- From text → List of tokens

The quick brown fox jumps over the lazy dog.



1) Splitting text into words

Python code

```
3  # split the sentence into words
4  words = text.split()
```

Split words using <space>

The quick brown fox jumps over the lazy dog.

[The] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog.]

Python code

```
1  text = "The quick brown fox jumps over the lazy dog."
2
3  # split the sentence into words
4  words = text.split()
5  print(words)
```

executed in 4ms, finished 14:52:04 2018-07-30

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.']
```

Better way: Tokenization

The quick brown fox jumps over the lazy dog.

[The] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog][.]

Python code

```
1  text = "The quick brown fox jumps over the lazy dog."
2
3  # split the sentence into words
4  import nltk
5  words = nltk.word_tokenize(text)
6  print(words)
```

executed in 6ms, finished 11:59:46 2018-07-31

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']
```

Even better way: Tokenization + Filtering

The quick brown fox jumps over the lazy dog.

[The] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

Python code

```
1  from nltk.tokenize import RegexpTokenizer
2
3  pattern = r'\w+'
4  tokenizer = RegexpTokenizer(pattern)
5  words = tokenizer.tokenize(text)
6  words
```

executed in 6ms, finished 12:04:52 2018-07-31

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

Regular-Expression Tokenizers

REG_EXP_cheatsheet.jpg

A **regular expression**, **regex** or **regexp**[1] (sometimes called a **rational expression**)[2][3] is, in theoretical computer science and formal language theory, a sequence of characters that define a *search pattern*. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

'\w+' : Refer to word only

r'abc' = raw string "abc"
u'abc' = unicode string "abc"

Example:

RAW text

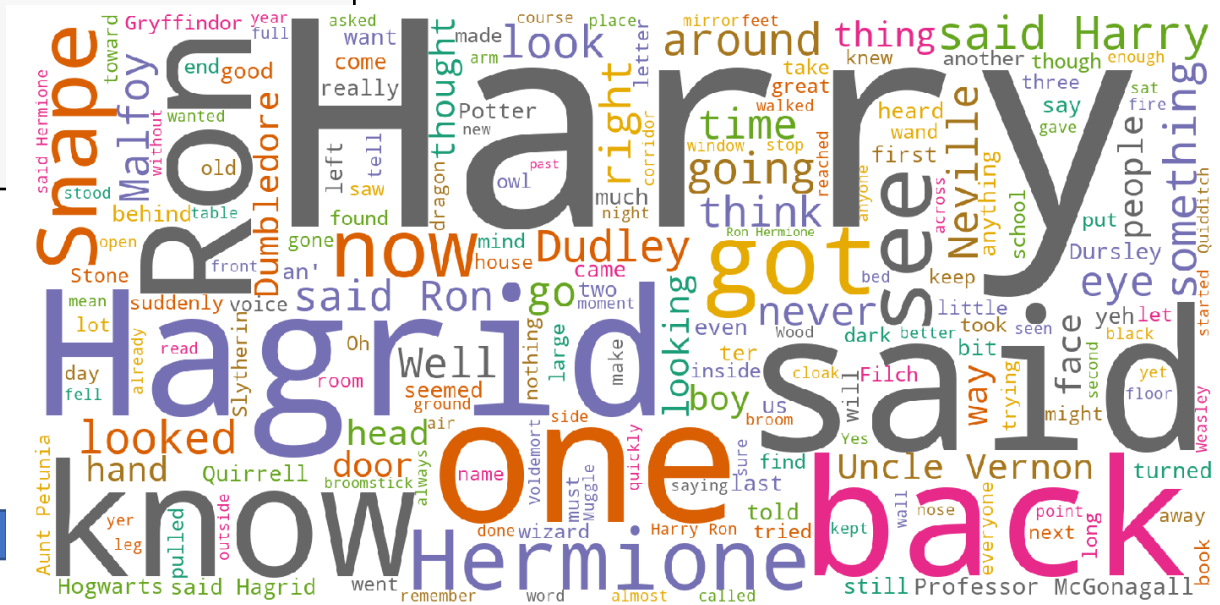
Harry Potter

Harry Potter and the Sorcerer's Stone CHAPTER ONE THE BOY WHO LIVED and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was t hin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time cranin g over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no fin er boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's s ister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never eve n seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like tha t. When and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Dursley hummed as he picked out his m ost boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Dursley picked up his briefcase, pecked Mrs. Durs ley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls. "Little tyke," chortled Dursley as he left the house. He got into his car and backed out of number four's drive. It was on the corner of the street that he noticed the first sign of something peculiar -- a cat reading a map. For a second, Dursley didn't realize what he had seen -- then he jerked his head around to look again. There was a tabby cat standing on the corner of Privet Drive, but there wasn't a map in sight. What could he have been thinking of? It must have been a trick of the light. Dursley blinked and stared at the cat. It stared back. As Dursley drove around the corner and up the road, he watched the cat in his mirror. It was now reading the sign that said Privet Drive -- no, looking at the sign; cats couldn't read maps o r signs. Dursley gave himself a little shake and put the cat out of his mind. As he drove toward town he thought of nothing ex cept a large order of drills he was hoping to get that day. But on the edge of town, drills were driven out of his mind by some

Words (wordcloud)

```
from wordcloud import WordCloud
wordcloudplot = WordCloud(width=1600, height=800).generate(HarryText)

import matplotlib.pyplot as plt
plt.figure( figsize=(20,10),facecolor='w')
plt.axis("off")
plt.tight_layout(pad=0)
plt.imshow(wordcloudplot)
plt.show()
```



We need to clean the words

2) Decapitalizing the words

- This step helps to turn words in the **same form**

[The] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

[the] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

Python code

```
1  # decapitalize
2  dep_words = [word.lower() for word in words]
3  print(dep_words)
```

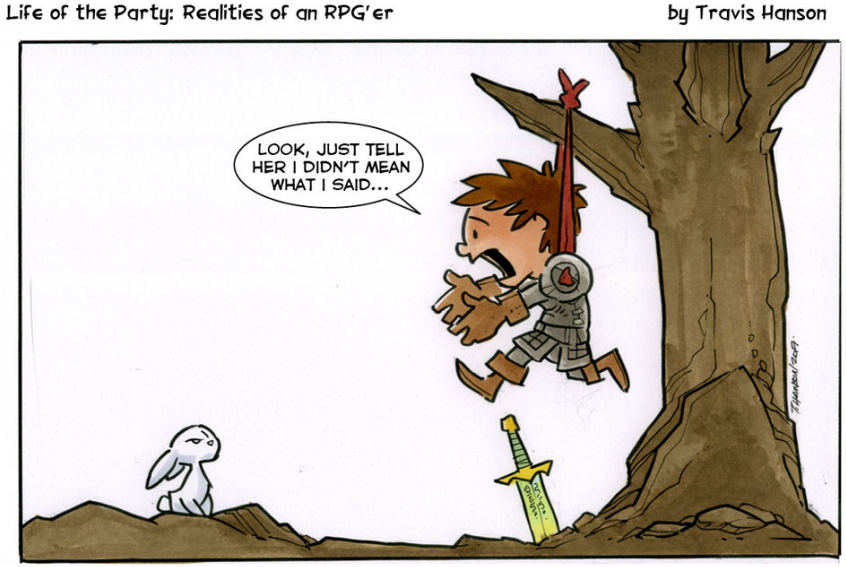
executed in 5ms, finished 12:07:21 2018-07-31

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

2) Decapitalizing the words

- This step helps to turn words in the **same form**.
- Be careful about the **proper nouns**

The Thing is going to help them.



The thing you just said was wrong

3.1) Stemming

Stemming is a technique used to find out the root/stem of a word.

Python code

```
1  from nltk.stem.porter import PorterStemmer
2  porter_stemmer = PorterStemmer()
```

executed in 3ms, finished 11:14:20 2018-07-30

```
1  print(porter_stemmer.stem('talk'))
2  print(porter_stemmer.stem('talked'))
3  print(porter_stemmer.stem('talking'))
4  print(porter_stemmer.stem('talkative'))
```

executed in 6ms, finished 11:14:20 2018-07-30

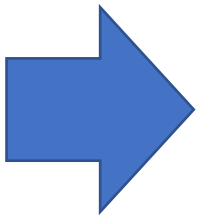
```
talk
talk
talk
talk
```

talk

talked

talking

talkative



talk

3.2) Lemmatization

Lemmatization is a technique used to find out the lemma of a word.

Python code

```
1  from nltk.stem import WordNetLemmatizer
2   wnl = WordNetLemmatizer()
```

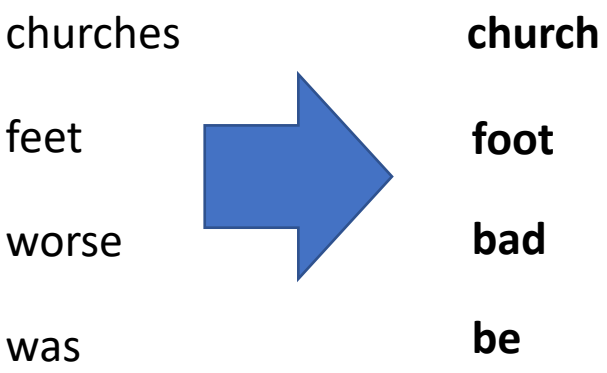
executed in 4ms, finished 14:09:53 2018-07-30

```
1  print(wnl.lemmatize("churches",'n'))
2  print(wnl.lemmatize('feet','n'))
3  print(wnl.lemmatize('worse','a'))
4  print(wnl.lemmatize('was','v'))
```

executed in 6ms, finished 14:17:40 2018-07-30

```
church
foot
bad
be
```

part-of-speech



3.3) Stemming vs Lemmatization

Stemmer

- Words → word's form
- Removing prefixes and suffixes

➔ **The result might not be an actual dictionary word.**

- Faster than *lemmatizers*

```
1  print(porter_stemmer.stem('churches'))
2  print(porter_stemmer.stem('feet'))
3  print(porter_stemmer.stem('worse'))
4  print(porter_stemmer.stem('was'))
5  print(porter_stemmer.stem('thinking'))
6  print(porter_stemmer.stem('cats'))
7
```

executed in 6ms, finished 14:21:31 2018-07-30

```
church
feet
wors
wa
think
cat
```

Lemmatizer

- Words → word's form
- Use a corpus

➔ **The result is always a dictionary word.**

- Need extra info about the part of speech

```
1  print(wnl.lemmatize("churches",'n'))
2  print(wnl.lemmatize('feet','n'))
3  print(wnl.lemmatize('worse','a'))
4  print(wnl.lemmatize('was','v'))
5  print(wnl.lemmatize('thinking','n'))
6  print(wnl.lemmatize('thinking','v'))
7  print(wnl.lemmatize('cats','n'))
```

executed in 8ms, finished 14:22:01 2018-07-30

```
church
foot
bad
be
thinking
think
cat
```

3.4) TAG and LEM

- Get the TAG of part-of-speech (POS)

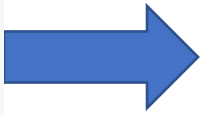
```
1  nltk.pos_tag(dep_words)
```

executed in 5ms, finished 15:37:12 2018-08-01

```
[('the', 'DT'),
 ('quick', 'JJ'),
 ('brown', 'NN'),
 ('fox', 'NN'),
 ('jumps', 'VBZ'),
 ('over', 'IN'),
 ('the', 'DT'),
 ('lazy', 'JJ'),
 ('dog', 'NN')]
```

Python code

```
## Function to transfer pos to tag
def transfer_tag(treebank_tag):
        if treebank_tag.startswith('j' or 'J'):
            return 'a'
        elif treebank_tag.startswith('v' or 'V'):
            return 'v'
        elif treebank_tag.startswith('n' or 'N'):
            return 'n'
        elif treebank_tag.startswith('r' or 'R'):
            return 'r'
        else:
            # As default pos in lemmatization is Noun
            return 'n'
```



Transfer POS into word's tag
(adjective, verb, noun, adverb)

3.4) TAG and LEM

- Get the TAG of part-of-speech
- Return the lemma with corresponding TAG

Python code

dep_words = [the] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

```
1  # Lemmatization nltk
2  from nltk import pos_tag
3  from nltk.stem import WordNetLemmatizer
4  wnl = WordNetLemmatizer()
5
6  lemma_words = []
7  for word, tag in nltk.pos_tag(dep_words):
8      firstletter = tag[0].lower() # -> get the first letter of tag and put them decapitalized form
9      wtag = transfer_tag(firstletter) # -> extract the word's tag (noun, verb, adverb, adjective)
10     if not wtag:
11         lemma_words.extend([word])
12     else:
13         lemma_words.extend([wnl.lemmatize(word, wtag)]) # -> get lemma for word with tag
14 print(lemma_words)
```

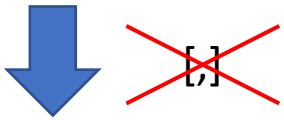
executed in 12ms, finished 15:26:39 2018-08-01

```
['the', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazy', 'dog']
```

4) Remove punctuations (if necessary)

- Split and remove will give you the exactly words in list.
- If we used the Regular-Expression Tokenizers, we do not need to worry about this

[knocked] [unconscious][,] [fox] [sleeps] [with] [shallow] [breath]



[knocked] [unconscious][fox] [sleeps] [with] [shallow] [breath]

Python code

```
1  # remove punctuation
2  import string
3  table = str.maketrans('','', string.punctuation)
4  # change punctuation into empty string
5  stripped_words = [w.translate(table) for w in dep_words]
6  # remove empty string in list
7  stripped_words = [x for x in stripped_words if str(x) != '']
8  print(stripped_words)
```

executed in 6ms, finished 12:04:03 2018-07-31

```
['knocked', 'unconscious', 'fox', 'sleeps', 'with', 'shallow', 'breath']
```

5) Remove stop words (if necessary)

Stop words are words which are filtered out before or after processing of text.

Python code

```
1 from nltk.corpus import stopwords
2 stop_words = set(stopwords.words('english'))
```

[the] [quick] [brown] [fox] [jumps] [over] [the] [lazy] [dog]

[quick] [brown] [fox] [jumps] [lazy] [dog]

Remove stop words: Better way !

We can combine the stop words we have in NLTK with other stop words list we find online.

https://github.com/6/stopwords-json

Python code

```
1  from nltk.corpus import stopwords
2  stopwords_json = {"af":["'n","aan","af","al","as","baie","by","daar","dag","dat",
3  stopwords_json_en = set(stopwords_json['en'])
4  stopwords_nltk_en = set(stopwords.words('english'))
5  # Combine the stopwords. Its a lot longer so I'm not printing it out...
6  stoplist_combined = set.union(stopwords_json_en, stopwords_nltk_en)
7
8  # check and remove stopwords
9  rmstopwords = ([word for word in lemma_words if word not in stoplist_combined])
10 print(rmstopwords)
```

executed in 1.44s, finished 15:27:53 2018-08-01

```
['quick', 'brown', 'fox', 'jump', 'lazy', 'dog']
```

Why do we need to decapitalize before removing stop words?

Answer:

Python code

```
1  from nltk.corpus import stopwords
2  stopwords_json = {"af":["'n","aan","af","al","as","baie","by","daar","dag","dat",
3  stopwords_json_en = set(stopwords_json['en'])
4  stopwords_nltk_en = set(stopwords.words('english'))
5  # Combine the stopwords. Its a lot longer so I'm not printing it out...
6  stoplist_combined = set.union(stopwords_json_en, stopwords_nltk_en)
7
8  # check and remove stopwords
9  rmstopwords = ([word for word in lemma_words if word not in stoplist_combined])
10 print(rmstopwords)
```

executed in 1.43s, finished 15:31:03 2018-08-01

```
['The', 'quick', 'brown', 'fox', 'jump', 'lazy', 'dog']
```

Why do we need to decapitalize before removing stop words?

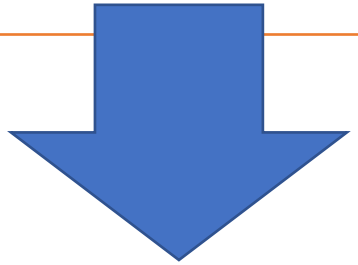
Finally we get…

The quick brown fox jumps over the lazy dog.

He lands head first on a rotting maple log.

Knocked unconscious, fox sleeps with shallow breath

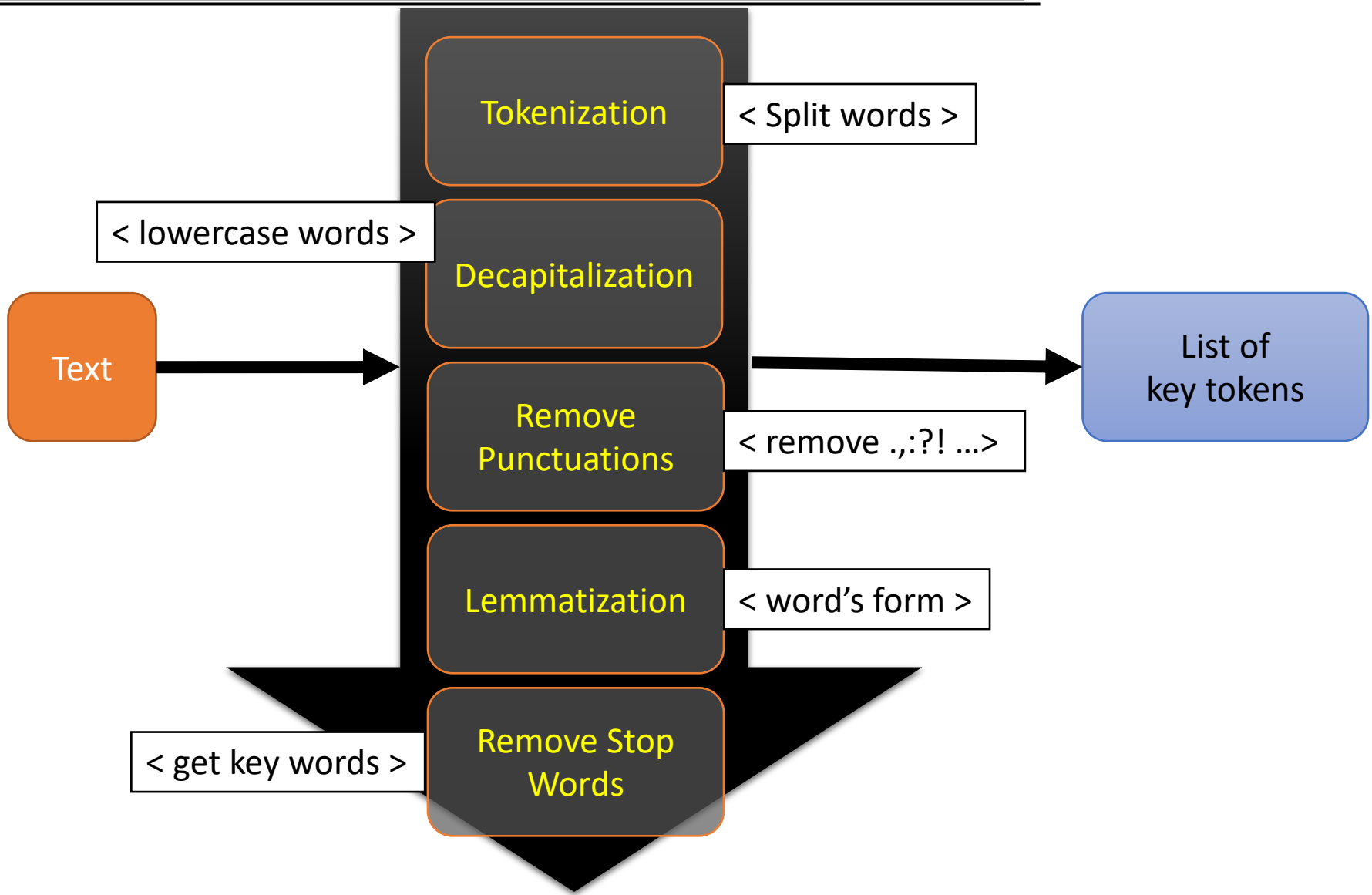
until the lazy dog awakes and worries him to death.



List of key tokens

```
['quick', 'brown', 'fox', 'jump', 'lazy', 'dog', 'land', 'head', 'rotting', 'maple', 'log', 'knock',
'unconscious', 'fox', 'sleep', 'shallow', 'breath', 'lazy', 'dog', 'awakes', 'worry', 'death']
```

Summary

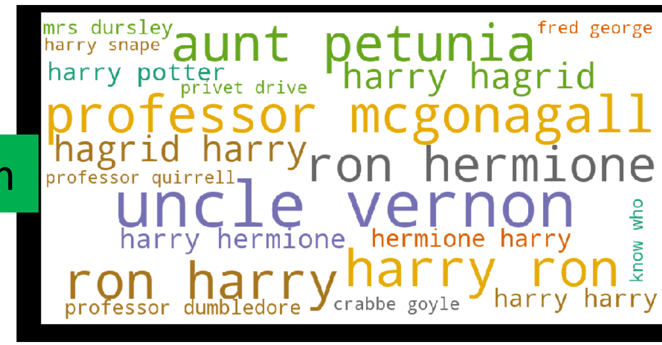

Tokenization
< Split words >
< lowercase words >
Decapitalization
Text
List of
key tokens
Remove
Punctuations
< remove .,:?! …>
Lemmatization
< word's form >
< get key words >
Remove Stop
Words

Back to *Harry Potter* :

Harry Potter and the Sorcerer's Stone CHAPTER ONE THE BOY WHO LIVED and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that. When and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair. None of them noticed a large, tawny owl flutter past the window. At half past eight, Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls. "Little tyke," chortled Dursley as he left the house. He got into his car and backed out of number four's drive. It was on the corner of the street that he noticed the first sign of something peculiar -- a cat reading a map. For a second, Dursley didn't realize what he had seen -- then he jerked his head around to look again. There was a tabby cat standing on the corner of Privet Drive, but there wasn't a map in sight. What could he have been thinking of? It must have been a trick of the light. Dursley blinked and stared at the cat. It stared back. As Dursley drove around the corner and up the road, he watched the cat in his mirror. It was now reading the sign that said Privet Drive -- no, looking at the sign; cats couldn't read maps or signs. Dursley gave himself a little shake and put the cat out of his mind. As he drove toward town he thought of nothing except a large order of drills he was hoping to get that day. But on the edge of town, drills were driven out of his mind by some

Next lecture:

N-gram



Lexical Dispersion Plot