# COMP90042 Workshop Week 2

## Introduction and Pre-processing

Zenan Zhai

The University of Melbourne

9 March 2014

# Table of Contents

# Table of Contents

# Contact

Canvas - Discussion Board

Subject Coordinator

- Dr. Jey Han Lau (laujh@unimelb.edu.au)

Me

- Zenan Zhai (zenan.zhai@unimelb.edu.au)
- Workshop slides available at https://zenanz.github.io/comp90042-2020/

# Table of Contents

# Pre-processing Pipeline

1. Formatting
2. Sentence Segmentation
3. Tokenisation
4. Normalisation
5. Stopword Removal

# Formatting

Web page you see ☺

# Formatting

Source code you get ☹

```
<!DOCTYPE html>
<html lang="en-gb" class="b-pw-1280 no-touch" id="responsive-news">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="google-site-verification" content="Tk6bx1127nACXoqt94L4-D-Of1fdr5gxrZ7u2Vtj9YI">
    <link href="//static.bbc.co.uk" rel="preconnect" crossorigin>
    <link href="//m.files.bbci.co.uk" rel="preconnect" crossorigin>
    <link href="//nav.files.bbci.co.uk" rel="preconnect" crossorigin>
    <link href="//ichef.bbci.co.uk" rel="preconnect" crossorigin>
    <link rel="dns-prefetch" href="//mybbc.files.bbci.co.uk">
    <link rel="dns-prefetch" href="//ssl.bbc.co.uk/">
    <link rel="dns-prefetch" href="//sa.bbc.co.uk/">
    <link rel="dns-prefetch" href="//ichef.bbci.co.uk">

    <script type="text/javascript">var domain = "co.uk";var edition = "";var prettyEdition = edition;if (window.NewsPage && window.NewsPage.edition) {edition =
window.NewsPage.edition;prettyEdition = edition === "northernireland" ? "Northern Ireland" :edition.charAt(0).toUpperCase() + edition.slice(1);}var pathEdition = edition.length
> 0 ? "/?" + edition.toLowerCase() : "";var _sf_async_config = _sf_async_config || {};var _sf_startpt=(new Date()).getTime();_sf_async_config.domain = "www" + ".bbc." +
domain;_sf_async_config.uid = 50924;_sf_async_config.title = window.document.title.replace("##edition##", prettyEdition);_sf_async_config.path = "bbc." + domain +
"/news/live/world-51796781" + pathEdition;_sf_async_config.sections = "News, News - world, News - LIV, News - world - LIV, News - news-category";_sf_async_config.mobileApp =
undefined;</script>

    <title>Coronavirus: Lowest increase in Chinese cases since crisis began - BBC News</title>

    <meta name="description" content="While the growth in China and South Korea is slowing, cases increase in the US and Europe.">
    <meta name="robots" content="NOODP,NOYDIR">

    <link rel="canonical" href="https://www.bbc.co.uk/news/live/world-51796781">

    <link rel="alternate" hreflang="en-gb" href="https://www.bbc.co.uk/news">
<link rel="alternate" hreflang="en" href="https://www.bbc.com/news">
```

The title is here!

Off-the-shelf packages come to aid (e.g. `beautifulsoup`)

# Word-level Tokenisation

‘Coronavirus: Lowest virus cases in China since crisis began.’

$\Downarrow$

[‘Coronavirus’, ‘:’, ‘Lowest’, ‘cases’, ‘in’, ‘China’, ‘since’, ‘crisis’, ‘began’, ‘.’]

- Tokenisation
  - Rule-based / Machine Learning
  - Subject to languages/domains (e.g. Medicine Chemistry)
- Off-the-shelf implementations
  - NLTK
    `https://www.nltk.org/`
  - OpenNLP
    `https://opennlp.apache.org/`
  - StanfordNLP
    `https://stanfordnlp.github.io/stanfordnlp/`

# Word-level Tokenisation

‘Coronavirus: Lowest virus cases in China since crisis began.’

$\Downarrow$

[‘Coronavirus’, ‘:’, ‘Lowest’, ‘cases’, ‘in’, ‘China’, ‘since’, ‘crisis’, ‘began’, ‘.’]

- Tokenisation
  - Rule-based / Machine Learning
  - Subject to languages/domains (e.g. Medicine Chemistry)
- Off-the-shelf implementations
  - NLTK
    `https://www.nltk.org/`
  - OpenNLP
    `https://opennlp.apache.org/`
  - StanfordNLP
    `https://stanfordnlp.github.io/stanfordnlp/`

Wait, why did you skip sentence segmentation?

# Byte-pair encoding

Why tokenisation?

# Byte-pair encoding

## Why tokenisation?

- Easier for machine to understand.

# Byte-pair encoding

## Why tokenisation?

- Easier for machine to understand.

## Is there a better unit of text than word?

- Let's take the word 'Coronavirus' as an example.
  - Do you know this word before the outbreak?
  - If so, do you understand it when you first saw it?

# Byte-pair encoding

Okay, now I know sub-words are fantastic.
Tell me how to get sub-word vocab?

Byte-pair Encoding

1. Break the entire piece of text into single characters tokens.
2. Count frequency of two tokens being together.
3. Merge most frequent pair of characters into one token.
4. Repeat from step 2.

# BPE in action

Coronavirus: Lowest virus cases in China since crisis began.

```
Vocab = defaultdict(<class 'int'>, {'C o r o n a v i r u s : </w>': 1, 'L o w e s t </w>': 1, 'v i r u s </w>': 1,
'c a s e s </w>': 1, 'i n </w>': 1, 'C h i n a </w>': 1, 's i n c e </w>': 1, 'c r i s i s </w>': 1, 'b e g a n .
</w>': 1})
==========
Tokens Before BPE
Tokens: defaultdict(<class 'int'>, {'C': 2, 'o': 3, 'r': 4, 'n': 5, 'a': 4, 'v': 2, 'i': 7, 'u': 2, 's': 8, ':': 1,
'</w>': 9, 'L': 1, 'w': 1, 'e': 4, 't': 1, 'c': 3, 'h': 1, 'b': 1, 'g': 1, '.': 1})
Number of tokens: 20
==========
Iter: 0
Best pair: ('s', '</w>')
Tokens: defaultdict(<class 'int'>, {'C': 2, 'o': 3, 'r': 4, 'n': 5, 'a': 4, 'v': 2, 'i': 7, 'u': 2, 's': 5, ':': 1,
'</w>': 6, 'L': 1, 'w': 1, 'e': 4, 't': 1, 'c': 3, 'h': 1, 'b': 1, 'g': 1, '.': 1, 's</w>': 3})
Number of tokens: 21
==========
Iter: 1
Best pair: ('i', 'n')
Tokens: defaultdict(<class 'int'>, {'C': 2, 'o': 3, 'r': 4, 'n': 2, 'a': 4, 'v': 2, 'i': 4, 'u': 2, 's': 5, ':': 1,
'</w>': 6, 'L': 1, 'w': 1, 'e': 4, 't': 1, 'c': 3, 'h': 1, 'b': 1, 'g': 1, '.': 1, 's</w>': 3, 'in': 3})
Number of tokens: 22
==========
Iter: 2
Best pair: ('v', 'i')
Tokens: defaultdict(<class 'int'>, {'C': 2, 'o': 3, 'r': 4, 'n': 2, 'a': 4, 'v': 0, 'i': 2, 'u': 2, 's': 5, ':': 1,
'</w>': 6, 'L': 1, 'w': 1, 'e': 4, 't': 1, 'c': 3, 'h': 1, 'b': 1, 'g': 1, '.': 1, 's</w>': 3, 'in': 3, 'vi': 2})
Number of tokens: 23
```

# Normalisation

## Normalisation techniques

- Lower casing
- Spelling correction
- Abbreviation expansion
- Removing Morphology
- ...

## What is normalisation?

# Normalisation

## Normalisation techniques

- Lower casing
- Spelling correction
- Abbreviation expansion
- Removing Morphology
- ...

## What is normalisation?

- Converting words to a standard format

## Why do we want normalisation?

# Normalisation

## Normalisation techniques

- Lower casing
- Spelling correction
- Abbreviation expansion
- Removing Morphology
- ...

## What is normalisation?

- Converting words to a standard format

## Why do we want normalisation?

# Normalisation

## Normalisation techniques

- Lower casing
- Spelling correction
- Abbreviation expansion
- Removing Morphology
- ...

## What is normalisation?

- Converting words to a standard format

## Why do we want normalisation?

- Reduce noises

# Normalisation

## Normalisation techniques

- Lower casing
- Spelling correction
- Abbreviation expansion
- Removing Morphology
- ...

## What is normalisation?

- Converting words to a standard format

## Why do we want normalisation?

- Reduce noises
- Reduce data sparsity

# Morphology

- Inflectional Morphology
  - Grammatical variants
- Derivational morphology
  - Another word with different meaning

Inflectional Morphology

began → begin

cases → case

Derivational morphology

Ethiopia → Ethiopian

# Lemmatization and Stemming

Rule-based deterministic algorithm for normalisation

| Lemmatisation | Stemming |
|---|---|
| Remove all inflections | Remove all suffixes |
| Matches with lexicons | No matching required |
| Product: Lemma | Product: Stem |

```
import nltk

sentence = ['Coronavirus', ':', 'Lowest', 'virus', 'cases', 'in', 'China', 'since', 'crisis', 'began', '.']
lemmatiser = nltk.stem.wordnet.WordNetLemmatizer()
stemmer = nltk.stem.porter.PorterStemmer()

# Code below from ...
def lemmatise(word):
    lemma = lemmatiser.lemmatize(word, 'v')
    if lemma == word:
        lemma = lemmatiser.lemmatize(word, 'n')
    return lemma
# End of copied code

lemmatised_sent = [lemmatise(word) for word in sentence ]
stemmed_sent = [stemmer.stem(word) for word in sentence ]

print('Lemmatised Sentence: ', lemmatised_sent)
print('Stemmed Sentence: ', stemmed_sent)
```

```
Lemmatised Sentence:  ['Coronavirus', ':', 'Lowest', 'virus', 'case', 'in', 'China', 'since', 'crisis', 'begin', '.']
Stemmed Sentence:  ['coronaviru', ':', 'lowest', 'viru', 'case', 'in', 'china', 'sinc', 'crisi', 'began', '.']
```

# Porter Stemmer

## Symbols

- Case sensitive

$V \rightarrow$ sequence of vowels $\quad C \rightarrow$ sequence of consonants
$v \rightarrow$ a single vowel $\quad c \rightarrow$ a single consonant

## Measure

1. Convert STEM of the word in the form of $[C](VC)^m[V]$
2. Take $m$ as measure

## Rules

- Example: $(m > 0$ *not* $^{\star}o)$ e $\rightarrow$ NULL
- $^{\star}o$ = `stem ends cvc and second c is not w, x or y (e.g. -HIL, -HOP)`

# Porter Stemmer - Exercise

## Rules

1. $(m > 0)$ `ational` $\rightarrow$ `ate`
2. $(m > 1)$ `ate` $\rightarrow$ `NULL`

`computational`

# Porter Stemmer - Exercise

## Rules

1. $(m > 0)$ `ational` $\rightarrow$ `ate`
2. $(m > 1)$ `ate` $\rightarrow$ `NULL`

## `computational`

| Step | Rule | Stem | Form | $m$ | Result |
|---|---|---|---|---|---|
| 1 | 1 | `comput` | $[C](VC)^2$ | 2 | `computate` |

# Porter Stemmer - Exercise

## Rules

1. $(m > 0)$ `ational` $\rightarrow$ `ate`
2. $(m > 1)$ `ate` $\rightarrow$ `NULL`

## `computational`

| Step | Rule | Stem | Form | $m$ | Result |
|---|---|---|---|---|---|
| 1 | 1 | `comput` | $[C](VC)^2$ | 2 | `computate` |
| 2 | 2 | `comput` | $[C](VC)^2$ | 2 | `comput` |

What about `national`?

# Stopword

## Stopword

- Short functional words that are very common
- Examples (NLTK): me, what, by, with, into, above ...

How would stopwords affect text classification?

# Take away

1. Pre-processing pipeline

2. Tokenisation
   - word level
   - sub-word level (BPE)

3. Normalization
   - Morphology (inflectional v.s. derivational)
   - Lemmatisation v.s. Stemming