# Text Mining and Sentiment Analysis

**Prof. Annamaria Bianchi**
**A.Y. 2024/2025**

Lecture 13

7 April 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Scienze Aziendali

# Outline

SA with amplifiers, de-amplifiers and negators

Lemmatisation and stemming

Packages: **base, tidyverse, tidytext, udpipe**

# Example with bos.airbnb data

We implement a dictionary-based sentiment analysis using Bing. We consider two cases:

- A basic analysis using Bing;
- A more sophisticated analysis using Bing and including negators, amplifiers and de-amplifiers.

```
> sent_bing <- txt_sentiment(x = output, + term = "token",
+ polarity_terms = bing_dict, + polarity_negators = "",
+ polarity_amplifiers = "",
+ polarity_deamplifiers = "",
+ amplifier_weight = 0,
+ n_before = 0,
+ n_after = 0,
+ constrain = F)
> data$bing = sent_bing$overall$sentiment_polarity
```

# Example with bos.airbnb data

```
> sent_udpipe <- txt_sentiment(x = output,
+ term = "token",
+ polarity_terms = bing_dict,
+ polarity_negators = c("not", "neither", "no", "without"),
+ polarity_amplifiers = c("really", "very", "definitely", "super"),
+ polarity_deamplifiers =c ("barely", "hardly"),
+ amplifier_weight = 0.8,
+ n_before = 2,
+ n_after = 0,
+ constrain = F)

> data$udpipe = sent_udpipe$overall$sentiment_polarity
```

# Comparison

**Exercise**. Built two variables, named bing_pol and udpipe_pol, identifying three categories in the polarity scores (positive, negative, and neutral).

# Comparison

```
> table(BING=data$bing_pol,UDPIPE=data$udpipe_pol)
          UDPIPE
BING       negative neutral positive
  negative       16       0        3
  neutral         1      63        5
  positive        1       2      909
```

# Lemmatization and Stemming

Term **normalization** is the process of transforming a term into a single, standardized form. Lemmatization and stemming are two **alternative methodologies** for word normalization:

- **Lemmatization**: is the process of determining the lemma of a word based on the context and identifying the part of speech of each word. This is done by implementing Machine Learning algorithms and can be computationally expensive;

- **Stemming**: is the process of reducing the word to its «stem» (or root) eliminating the suffix. It is less computational expensive than lemmatization;

Both methods are used in order to reduce the dimensionality and speed up the estimation processes when implementing more sophisticated analyses (Topic Modeling, Machine learning algrithms).

# Lemmatization and Stemming

- **Stemming** works well with languages such as **English**, where words have a common root, but
- **Lemmatization** is better for **interpretability**.
- For example:

| Word | Lemma | Stem |
|---|---|---|
| Caring | Care | Car |
| Running | Run | Run |

# udpipe and SnowballC

In order to **compare stemming and lemmatization** we consider the udpipe and snowballC packages.

- `udpipe::udpipe(data, model)`: implements several NPL functions, including tokenization, lemmatization, part of speech (POS) tagging, and some additional information including dependences between words. To do that, it uses some pre-trained models. For the english language, the most common model is «`english-gum`». It does not include stemming.

- `SnowballC::wordStem(words,language)`: is a popular stemmer based on Porter's word stemming algorithm.

# Part-of-Speech (POS) tagging

- POS tagging finds grammatical tags based on ML models.
- It is the process of marking up the words with a particular part of speech.
- This is done based on the word context: relationship with adjacent and related words in the sentence (or paragraph).
- Categories include: noun, verb, article, adjective, preposition, pronoun, adverb, conjuction, etc.
- They are in the `upos` (universal pos taggers) column of the output `udpipe()` table.
- The function also returns `xpos` tags, which are based on a different classification

# Database

We keep working with the Boston Airbnb comments. We will be using the dataset output.rds, that was obtained in Lecture 12 using the function `udpipe()`.

# udpipe function

The data.frame obtained with the `udpipe()` function has the following fields:

- `doc_id`: The document identifier
- `token`: The token.
- `lemma`: The lemma of the token.
- `upos`: The universal parts of speech tag of the token.
- `xpos`: The treebank-specific parts of speech tag of the token.
- `feats`: The morphological features of the token, separated by |.

# udpipe function

```
> View(output)
```

| token_id | token | lemma | upos | xpos | feats |
|---|---|---|---|---|---|
| 1 | My | my | PRON | PRP$ | Number=Sing\|Person=1\|Poss=Yes\|F |
| 2 | daughter | daughter | NOUN | NN | Number=Sing |
| 3 | and | and | CCONJ | CC | *NA* |
| 4 | I | I | PRON | PRP | Case=Nom\|Number=Sing\|Person=1 |
| 5 | had | have | VERB | VBD | Mood=Ind\|Tense=Past\|VerbForm=F |

# Stem

The function `SnowballC::wordStem()` extracts the stems of each of the given words in the vector.

```
wordStem(words, language = "porter")
```

| | |
|---|---|
| `words` | a character vector of words whose stems are to be extracted. |
| `language` | the name of a recognized language, as returned by getStemLanguages, or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes). |

The output is a character vector with as many elements as there are in the input vector with the corresponding elements being the stem of the word.

# Stem

We add the stem to the output dataset in order to compare stems with the lemmas for Boston Airbnb reviews

```
> output = output |>
+ mutate(stem = wordStem(token))
> View(output)
```

# Comparing lemmas and stems

We can compare words (token), lemmas and stems of verbs and adjectives:

```
> output %>%
+   filter(upos=="VERB") %>%
+   distinct(token, .keep_all = T) %>%
+   select(token, lemma, stem) %>%
+   slice(1:15)
          token       lemma     stem
1           had        have      had
2          kept        keep     kept
3      arriving      arrive    arriv
4      charming      charme    charm
5         asked         ask      ask
6          laid         lay     laid
7          stay        stay     stai
8     decorated    decorate    decor
9       located      locate    locat
10         come        come     come
11          say         say      sai
12 appreciate appreciate appreci
13     staying        stay     stai
14        camp        camp     camp
15          is          be        i
>
```

```
> output %>%
+   filter(upos=="ADJ") %>%
+   distinct(token, .keep_all = T) %>%
+   select(token, lemma, stem) %>%
+   slice(1:15)
           token        lemma     stem
1      wonderful    wonderful   wonder
2          close        close    close
3          whole        whole    whole
4           warm         warm     warm
5       eclectic     eclectic   eclect
6           nice         nice     nice
7          great        great    great
8         better         good   better
9   confortable  confortable  confort
10      optimal      optimal    optim
11     terrific     terrific   terrif
12       lovely       lovely     love
13    excellent    excellent    excel
14      walking         walk     walk
15        small        small    small
>
```

# Exercises for you

**Exercise 1**. With reference to the Airbnb data, for some reviews different outcome was obtained implementing a basic analysis using Bing or a more sophisticated analysis using Bing and including negators, amplifiers and de-amplifiers (Slide 6). Focusing on reviews n. 186 and n. 401, try to understand how the polarity are computed in the two cases.
Hint: it is convenient to look at the overall dataframe created as output from the function `txt_sentiment()`.

# Exercise for you

**Exercise 2**. The file data.csv contains 220 reviews about an hotel. The true sentiment is in the column sentiment. Write proper R code to perform the following tasks:

1. Apply the `udpipe` function in order to tokenize the text.
2. Apply the dictionary-based sentiment analysis. Use tokens to perform the basic sentiment analysis with bing. Then, repeat the tasks adding polarity negators, amplifiers, deamplifiers. Consider an amplifier weight equal to 0.8 and look for 3 words before.
3. Compare the overall sentiment obtained with the different methods and also with the true score.
4. Check the number of reviews that have received reverted polarity using the different methods and with respect to the true score.

# Exercise for you

**Exercise 3**. With reference to the same data used in the previous exercise (output from `udpipe()` function), write proper R code to perform the following tasks:
1. Add a column with the stem
2. Compare the token (word), lemma and stem for the first 15 observations, considering only nouns first and next only adjectives.
3. Plot the 10 most common stems which are noun
4. Plot the top 10 adjectives, considering the lemma and by the true sentiment.
5. Apply the dictionary sentiment analysis. Use lemmas to perform the basic sentiment analysis with bing. Then, repeat the tasks adding polarity negators, amplifiers, deamplifiers. Consider an amplifier weight equal to 0.8 and look for 3 words before.
6. Compare the results using the different methods and with the true score.