Week 2 Tutorial: Regular Expressions

POP77032 Quantitative Text Analysis for Social Scientists

Exercise 1: Working with Regular Expressions

- In this exercise we will consider parliamentary debates using the US Congressional Record data.
- A common task when working with unstructured text data is to preprocess it to extract relevant information.
- Here we will focus on extracting speakers and speeches from the Congressional Record HTML files.
- As an example, we will consider the debate on the second impeachment of Donald Trump that took place on 13 January 2021.
- After downloading the texts of the debates, our first goal would be to extract the names of individual speakers and speeches using regular expressions.
- The empirical question that we are trying to answer is: how many unique speakers took part in the debate?
- The key part of this exercise is to build a regular expression that can accurately identify speakers' names.
- Take some time to analyse the structure of speakers' names and what aspects distinguish those from surrounding text.

```r
library("rvest")
library("stringr")
```

```r
# As the full impeachment debate is split across two sections,
# we will first download both HTML files and then combine them in single text
impeachment1 <- rvest::read_html(
  "https://www.govinfo.gov/content/pkg/CREC-2021-01-13/html/CREC-2021-01-13-pt1-PgH151-8.htm"
)
impeachment2 <- rvest::read_html(
  "https://www.govinfo.gov/content/pkg/CREC-2021-01-13/html/CREC-2021-01-13-pt1-PgH165.htm"
)
```

```r
# Extract text from the first URL
txt1 <- rvest::html_text(rvest::html_nodes(impeachment1, "pre"))
nchar(txt1)
```

```
[1] 126613
```

```r
# Extract text from the second URL
txt2 <- rvest::html_text(rvest::html_nodes(impeachment2, "pre"))
nchar(txt2)
```

```
[1] 231550
```

```r
txt <- paste0(txt1, "\n", txt2)
nchar(txt)
```

```
[1] 358164
```

String Distances

String Distance

- While regular expressions allow us to capture different strings,
- Sometimes it is useful to know just *how* different these strings are.
- The simplest way is to consider difference between individual words.
- Consider these words:

cut

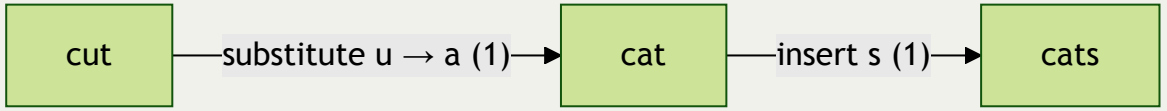
and

cats

- How different are they?

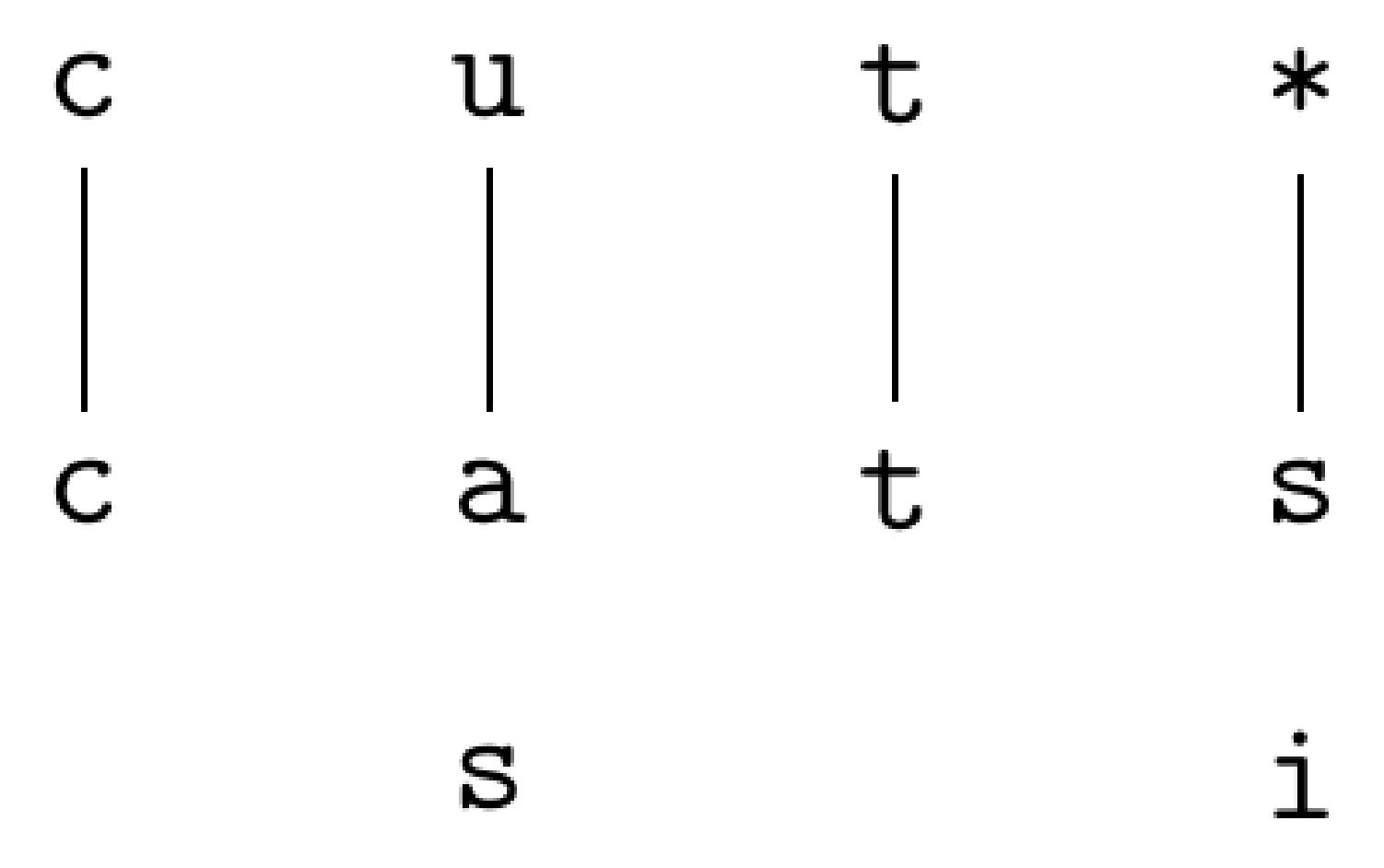
Edit Distance

- One way to think about the difference is to count the number of editing operations required to transform one string into another.
- Think about what it takes to fix typo(s) in text.
- Most common operations would be:
  - **Insertion**
  - **Deletion**
  - **Substitution**
- In our example this is what it might look like:



Alignment

- An alternative way of representing the difference between two strings is through their **alignment**.
- This is a correspondence between substrings of the two sequences.



Levenstein Distance

- The only remaining aspect of calculating the edit distance is to assign *costs* to each of the operations.
- In the simplest case we can assign a cost of 1 to each operation.
- This is also known as the **Levenshtein distance**.

```
1 import textdistance
2 textdistance.levenshtein.distance("cut", "cats")
```

```
2
```

```
1 library("stringdist")
2 stringdist::stringdist("cut", "cats", method = "lv")
```

```
[1] 2
```

Exercise 2: Working with String Distances

- Now that we have extracted individual speeches, we might be interested in finding similar speeches.
- One way to measure similarity between texts is to use string distance metrics.
- Try using one of the common string distance metrics such as Levenshtein distance, Jaccard distance, etc. to find the most similar speeches in the debate.
- In R you can use the `stringdist` package to calculate various string distance metrics.
- In Python you can use the `textdistance` library to calculate string distances.
- What kind of speeches would you expect to be most similar to each other?