INTRODUCTION TO NLP

Session 1

David Buchaca Prats

2022

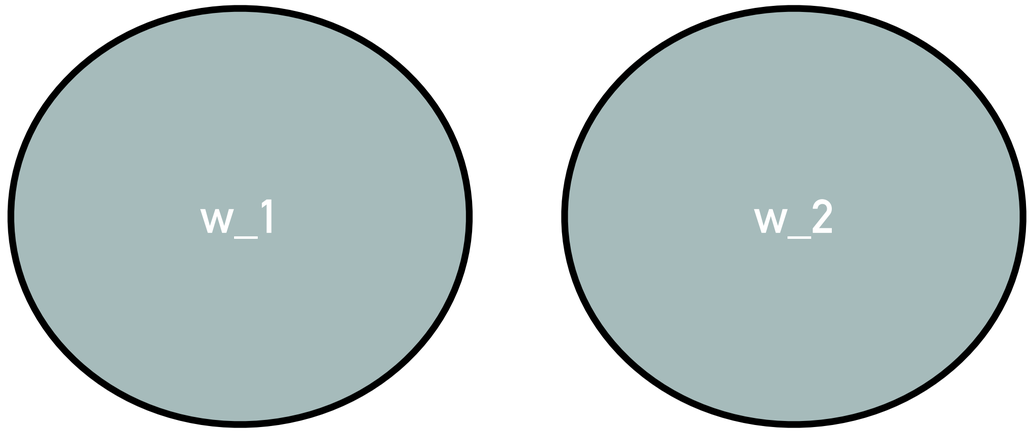
WHAT YOU WILL LEARN (OR ALREADY KNOW)

- Work efficiently with strings
- Perform sensible preprocessing of text
- Extract features fro text (and for words)
- Search efficiently raw text data
- Perform tagging problems using structured prediction
- Language models

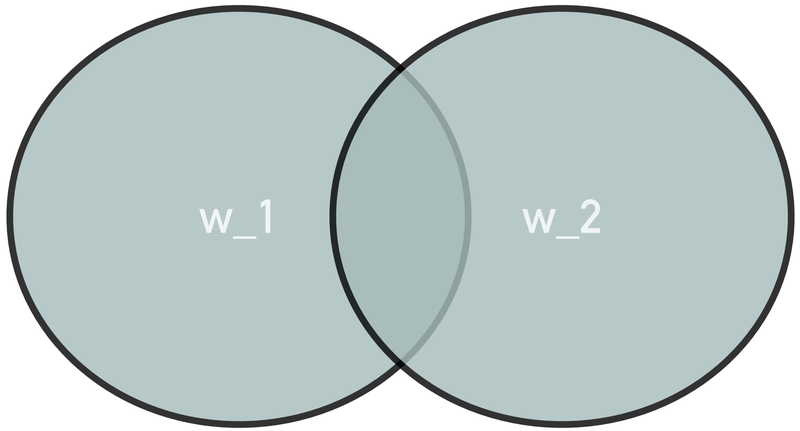
WHY SHOULD YOU BE INTERESTED FROM A PRACTICAL POINT OF VIEW

- The world is full of web services based on text data.
  - Ebay, Amazon, Aliexpress, Wallapop
  - Most services already use NLP (or could be heavily improved using NLP)
- Let's look a a typical problem with key-word matching techniques for search

Words have very distinct meaning



Words have similar meaning



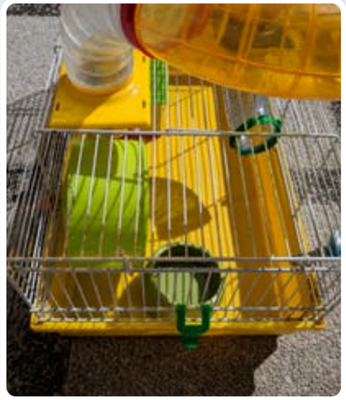
LOOKING “HAMSTER” IN WALLAPOP

10 €
La pandilla hamster

Juego de cooperación de 4 a 8 años.

15 €
Jaula para hámster con...

¡La nueva casita de tu amiguito peludo! El bebedero es ideal para que tu hámster apague su sed...

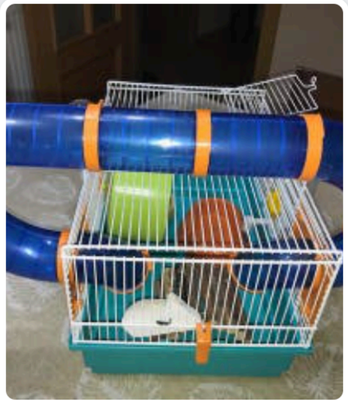


20 €
Jaula grande de hámst...

Jaula grande con comedero, bebedero, casita, rueda y tubos para tu hámster. Si necesitas otro tamaño de...

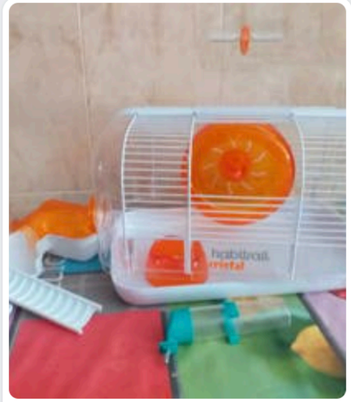
35 €
jaula

JAULA Hámster con accesorios



30 €
Jaula hamster

Jaula de hamster con juego de tubos, rueda, comedero, bebedero y casita.



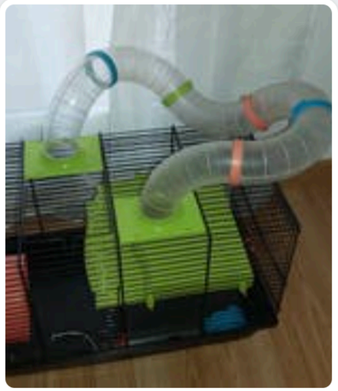
25 €
Jaula para hámster

Jaula para hámster con muy poco uso, menos de 1 mes y está completamente limpia. Viene con los...

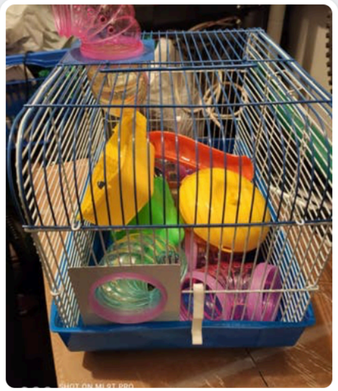
11 €
Jaula Hámster

Jaula marca Hábitall hcon tobogán, comederos....y de regalo una bola de actividad para el Hámster,...

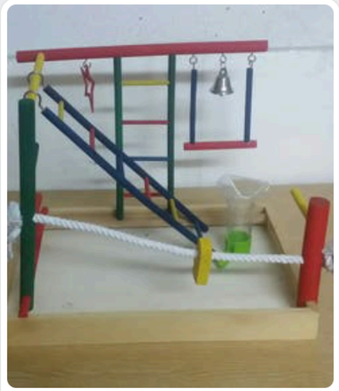
LOOKING “JAMSTER” IN WALLAPOP



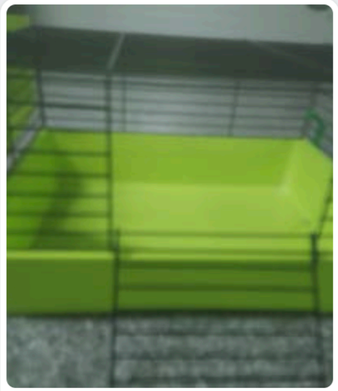
25 €
jaula jamster.
jaula jamster, contiene bebedero.



20 €
Jaula jamster
Vendo jaula de jamster por habernos abandonado. No tiene más de un mes de uso. Tiene tubos, rueda de...



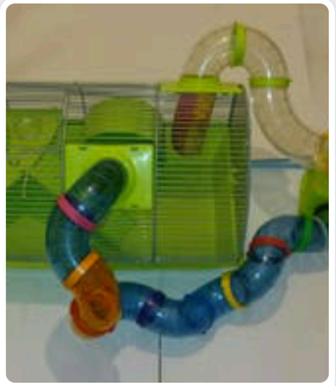
5 €
Juego para Jamster
Juego para Jamster



8 €
jaula para jamster
esta perfecta, sin apenas uso. La usaba sólo cuando me iba de viaje, en casa usaba una más grande. La...

3 €
Jaula Jamster
Nueva, sin uso. Medidas 21x29x20

15 €
Jaula de Jamster comp...
Jaula de Jamster completa y en perfecto estado.

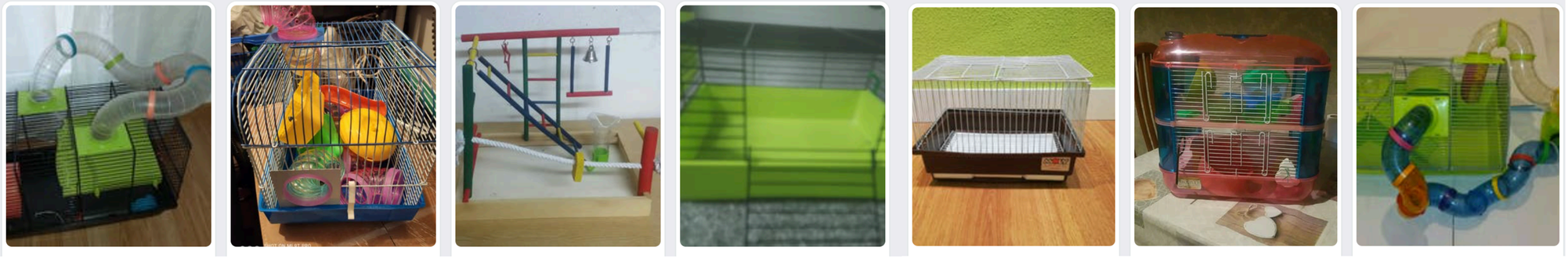


40 €
jaula Hamster y mucho...
jaula de hámster Cunipic 119C medidas 32.5X35.5, choza, comedero, noria para correr, altillo con...

HAMSTER VS JAMSTER

Hamster

Jamster

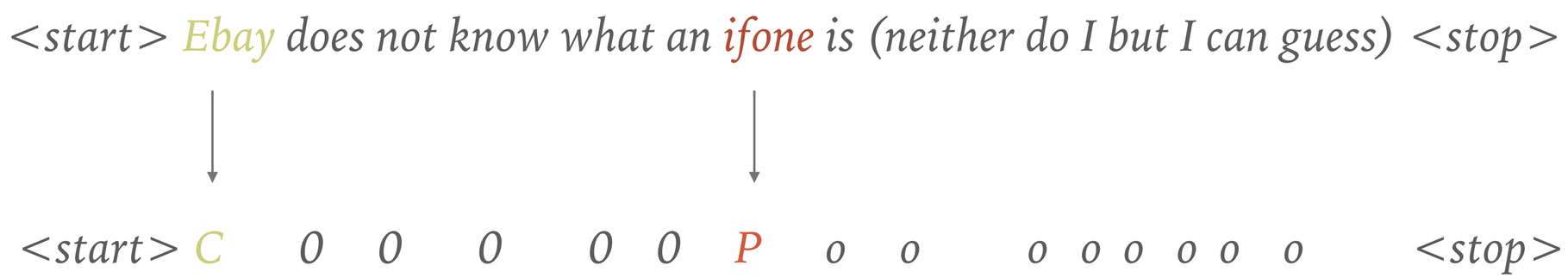


WHY YOU MIGHT BE INTERESTED FROM A THEORETICAL POINT OF VIEW

- We will see a different paradigm of Machine Learning with techniques that have to deal with sequential structured data
- In particular we will deal with independence assumptions in our models that can provide huge speedup benefits (if implemented correctly) with good tradeoff in quality.
- Example: Let us consider P(X=x, Y=y) where x and y are sequences.
  - Assume x and y have length 10
  - Assume Y can have 12 possible values
  - How can you compute P(X=x) ?
  - 12^2 = 61.917.364.224 sums

PROBLEMS THAT WE WILL FACE

- Detecting similarity between documents
- Tagging words as Named Entities (Person, Location, Company …)



- Detecting a particular language from a given input string
- Predicting the next word within a stream of words
- Predicting the probability of a given string

EVALUATION

- 70 % Deliverables
  - Two projects have to be delivered. These projects consist on working on a “challenge”.
  - You will be given a task (dataset and metric) and you will have some freedom to explore different strategies to solve the task.

- 30 % Final Exam
  - A final exam will evaluate all the material in the course.

REGULAR EXPRESSIONS

REGULAR EXPRESSIONS

- Generate a regular expression using `p = re.compile(r'our_regular_expression')`
  - `p` will be a `re.Pattern` object that we can apply to any string.
  - `p.findall(s)` returns a list containing all substrings within `s` that satisfy our regular expression.
  - `p.finditer(s)` returns a generator. Each element of the generator is a `SRE_Match` which contains
    - `pan(x,y)` indicates the start position `x` and end position `y` (of `s`).
    - `match='...'` indicates the string that satisfies the regular expression.

QUANTIFIERS

- Quantifiers are operators that are applied to the preceding symbol.

- `'*'` previous symbol appears 0 or more matches.
- `'+'` previous symbol appears 1 or more matches.
- `'?'` previous symbol appears at most one (0 or 1).
- `'{k}'` previous symbol repeated k exact matches.
- `'{min,max}'` previous symbol appears any number between min and max.
  - For example `'{2,8}'` would match numbers between 2 and 8

QUANTIFIERS

- `"r'a.*'"` the `'*'` refers to `'.'` making this expression get trigered when `'a'` is followed by any character.
- `"r'\d{4}'"` the `'{4}'` refers to `'\d'` making this regular expression get triggered with 4 consecutive digits.

- Example:

```
aux =  "The girl who loved the black cat ended up with a catwomen costume.”
```

- `p = re.compile(r'cat\s')`
  `p.findall(aux)`
  - `['cat ']`

- `p = re.compile(r'cat\s*')`
  `p.findall(aux)`
  - `['cat ', 'cat']`

SPECIAL REGEX CHARACTERS

- `'.'` Matches any character except new line.
- `'\d'` Matches any digit (0-9). Equivalent to `'[0-9]'`.
- `'\D'` Matches any NON digit. Equivalent to `'[^0-9]'`.
- `'\w'` Matches any "word character". Equivalent to `'[a-zA-Z0-9_]'`
- `'\W'` Matches any NON alphanumeric character.
  - Equivalent to `'[^a-zA-Z0-9_]'`
- `'\s'` Matches any whitespace (space, tab, newline).
  - Equivalent to `'[\t\n\r\f\v]'`
- `'\S'` Matches any NON whitespace (space, tab, newline).
  - Equivalent to `'[^ \t\n\r\f\v]'`

REGEX EXAMPLE

- Let us consider the following example

```
aux =  "The girl who loved the black cat ended up with a catwomen costume.”
```

- ```
  p = re.compile(r'cat')
  p.findall(aux)
  ```
- ['cat', 'cat']

- ```
  p = re.compile(r'cat\s')
  p.findall(aux)
  ```
- [‘cat ']

- ```
  p = re.compile(r'cat.*')
  p.findall(aux)
  ```
- ['cat ended up with a catwomen costume.']
  ```

REGEX EXAMPLE

- Let us consider the following example, we want to match any word with cat prefix

```
aux =  "The girl who loved the black cat ended up with a catwomen costume.”
```

- ```
  p = re.compile(r’cat\w+’)
  p.findall(aux)
  ```
- ```
  ['catwomen']
  ```

- ```
  p = re.compile(r’cat\w*’)
  p.findall(aux)
  ```
- ```
  ['cat', 'catwomen']
  ```

WE CAN USE REGEX FOR TOKENIZATION

```
aux =  'The girl who loved the black cat, bought a catwomen costume'
```

- `p = aux.split(‘ ’)`

- `['The', 'girl', 'who', 'loved',`
  `‘the', 'black', 'cat,', 'bought',`
  `'a', 'catwomen', 'costume']`

In [ ]:

WE CAN USE REGEX FOR TOKENIZATION

```
aux =  'The girl who loved the black cat, bought a catwomen costume'
```

- ```
  p = re.compile(r'\w+')
  p.findall(aux)
  ```

- ```
  ['The', 'girl', 'who', 'loved',
   'the', 'black', 'cat', 'bought',
   'a', 'catwomen', 'costume']
  ```

In [ ]:

ANCHERS

- `'\b'` Matches any word boundary.
- `'\b'` Matches any NON word boundary.
- `'^'` Matches beginning of a string
- `'[^a-e]'` negates the character set. That is matches any character outside `'a-e'`.
- `'$'` Matches a position that is end of a string.

CHARACTER SETS

- `'[]'` allows us to specify sets of symbols.
- `'[ab3-]'` matches any character `'a'` or `'b'`or `'3'` or `'-'`.
- `'[a-g]'` matches any character `'a'` to `'g'` such as `'b','c','d',...,'g'`
- `'[A-G]'` matches any character `'A'` to `'G'` such as `'B','C','D',...,'G'`
- `'[a-zA-G]'` matches any lowerchase character and any uppercase character from `‘A'` to `'G'`.
- `'[d1-d2]'` matches any digit between `'d1'` and `'d2'`. For example `'[0-5]'` matches `'0','1','2','3','4','5'`

EXAMPLES

- `abc*` matches a string that has `ab` followed by zero or more `c`
- `abc+` matches a string that has ab followed by one or more `c`
- `abc?` matches a string that has ab followed by zero or one `c`
- `abc{2}` matches a string that has ab followed by 2 `c`
- `abc{2,}` matches a string that has ab followed by 2 or more `c`
- `abc{2,5}` matches a string that has ab followed by 2 up to 5 `c`
- `a(bc)*` matches a string that has a followed by zero or more copies of the sequence `bc`
- `a(bc){2,5}` matches a string that has a followed by 2 up to 5 copies of the sequence `bc`

GROUPS

- Groups are created between parenthesis

- `'(abc)?'` Checks if the group `'abc'` appears or not.