# Working with many languages

T. Mickus

Dec $16^{\text{th}}$, 2020

# Outline

# Outline

# Encoding

## General caveat

- Multilingual NLP applications must deal with variations in writing:
  - Multiple alphabets or writing systems (latin, cyrillic, arabic, chinese...)
  - Multiple variants of similar writing systems: diacritics, etc.

# Encoding
## General caveat

- Multilingual NLP applications must deal with variations in writing:
  - Multiple alphabets or writing systems (latin, cyrillic, arabic, chinese...)
  - Multiple variants of similar writing systems: diacritics, etc.

- Computers deal with these variations by using different **encodings**: latin1, UTF-8, CP1251 ...

# Encoding

## General caveat

- Multilingual NLP applications must deal with variations in writing:
  - Multiple alphabets or writing systems (latin, cyrillic, arabic, chinese...)
  - Multiple variants of similar writing systems: diacritics, etc.

- Computers deal with these variations by using different **encodings**: latin1, UTF-8, CP1251 ...

- Each encoding is a specific mapping of characters to binary representations, and vice-versa: low level text representation is done by manipulating the text in binary or byte format.

# Encoding

## General caveat

- Multilingual NLP applications must deal with variations in writing:
  - Multiple alphabets or writing systems (latin, cyrillic, arabic, chinese...)
  - Multiple variants of similar writing systems: diacritics, etc.
- Computers deal with these variations by using different **encodings**: latin1, UTF-8, CP1251 ...
- Each encoding is a specific mapping of characters to binary representations, and vice-versa: low level text representation is done by manipulating the text in binary or byte format.
- Encodings generally cover a specific set of characters : latin1 covers only basic Latin characters, CP1251 contains both Latin and Cyrillic letters, etc. The unicode standard defines both UTF-8 and UTF-16 encodings and try to represent any possible character.

# Encoding

## General caveat

- Multilingual NLP applications must deal with variations in writing:
  - Multiple alphabets or writing systems (latin, cyrillic, arabic, chinese...)
  - Multiple variants of similar writing systems: diacritics, etc.

- Computers deal with these variations by using different **encodings**: latin1, UTF-8, CP1251 ...

- Each encoding is a specific mapping of characters to binary representations, and vice-versa: low level text representation is done by manipulating the text in binary or byte format.

- Encodings generally cover a specific set of characters : latin1 covers only basic Latin characters, CP1251 contains both Latin and Cyrillic letters, etc. The unicode standard defines both UTF-8 and UTF-16 encodings and try to represent any possible character.

- The good practice is to keep track of the encoding of files, and, as much as possible, use **unicode** encoding (used by default in python 3)
  **NB**: Useful python library for detecting encoding: `chardet`

# Outline

- Wiktionary is a collaboratively edited multilingual web-based project.

# Word-level Resources

## Wiktionary

- Wiktionary is a collaboratively edited multilingual web-based project.
- The aim is to produce dictionaries for all the world's languages, currently it covers 171 languages

- Wiktionary is a collaboratively edited multilingual web-based project.
- The aim is to produce dictionaries for all the world's languages, currently it covers 171 languages
- Wiktionary data is frequently used in NLP, both in multilingual and in monolingual contexts
  **NB**: cf. for instance GLAWI: `http://redac.univ-tlse2.fr/lexiques/glawi.html` which is a freely distributed resource for French, mapping morphological annotations from GLÀFF to definitions from the French wiktionary.

- Wiktionary is a collaboratively edited multilingual web-based project.

- The aim is to produce dictionaries for all the world's languages, currently it covers 171 languages

- Wiktionary data is frequently used in NLP, both in multilingual and in monolingual contexts
  **NB**: cf. for instance GLAWI: `http://redac.univ-tlse2.fr/lexiques/glawi.html` which is a freely distributed resource for French, mapping morphological annotations from GLÀFF to definitions from the French wiktionary.

- As a consequence of the collaborative nature of the project, Wiktionary is generally deemed to have broad coverage, but unsystematic definitions.

# Word-level Resources
## Wiktionary

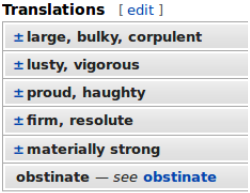
- Many dictionaries include relevant multilingual information that can be exploited in linguistic applications and experiments.

# Word-level Resources

## Wiktionary

- Many dictionaries include relevant multilingual information that can be exploited in linguistic applications and experiments.

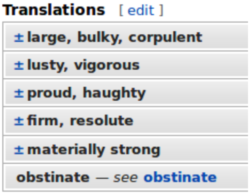
- In our case, wiktionary entries often have a "**Translations**" subsection



# Word-level Resources
## Wiktionary

- Many dictionaries include relevant multilingual information that can be exploited in linguistic applications and experiments.

- In our case, wiktionary entries often have a "**Translations**" subsection



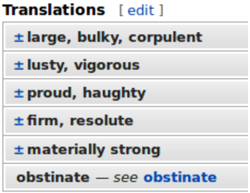

- … which can be retrieved by parsing the XML dump

```
====Translations====
{{trans-top|large, bulky, corpulent}}
* Finnish: {{t|fi|pönäkkä}}, {{t+|fi|tanakka}}
* Greek: {{t+|el|}}
* Irish: {{t|ga|alpartha}}
```

**NB**: dumps are available here: `https://dumps.wikimedia.org/`

# Word-level Resources
## Wiktionary

- Many dictionaries include relevant multilingual information that can be exploited in linguistic applications and experiments.
- In our case, wiktionary entries often have a "**Translations**" subsection



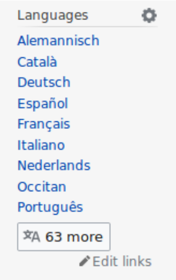

- ... which can be retrieved by parsing the XML dump

```
====Translations====
{{trans-top|large, bulky, corpulent}}
* Finnish: {{t|fi|pönäkkä}}, {{t+|fi|tanakka}}
* Greek: {{t+|el|}}
* Irish: {{t|ga|alpartha}}
```

**NB**: dumps are available here: `https://dumps.wikimedia.org/`

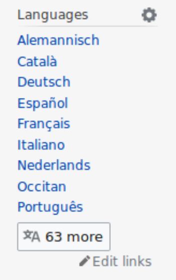
- but it's actually hard work.

- More generally, Wiki-based resources such as Wikipedia, Wiktionary, Wikimedia, etc. often display “interlanguage links” :



# Word-level resources
## Wiki

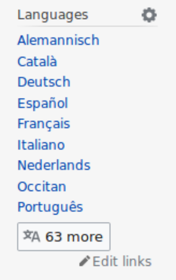
- More generally, Wiki-based resources such as Wikipedia, Wiktionary, Wikimedia, etc. often display “interlanguage links” :




- Likewise, they can be retrieved by parsing the XML dump and it can quickly become a time-consuming task.

# Word-level resources
## Wiki

- More generally, Wiki-based resources such as Wikipedia, Wiktionary, Wikimedia, etc. often display “interlanguage links” :

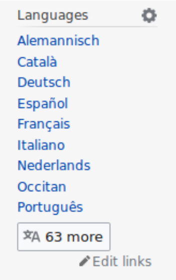



- Likewise, they can be retrieved by parsing the XML dump and it can quickly become a time-consuming task.

- More info : `https://en.wikipedia.org/wiki/Help:Interlanguage_links`

# Word-level resources
## Wiki

- More generally, Wiki-based resources such as Wikipedia, Wiktionary, Wikimedia, etc. often display “interlanguage links” :




- Likewise, they can be retrieved by parsing the XML dump and it can quickly become a time-consuming task.

- More info : `https://en.wikipedia.org/wiki/Help:Interlanguage_links`

**NB**: On that note, have a look at the project Omega Wiki: `http://omegawiki.org/`

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

# Word-level resources

## Wordnet

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

- The list of all codes for supported languages can be found using `wn.langs()`.

# Word-level resources

## Wordnet

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

- The list of all codes for supported languages can be found using **`wn.langs()`**.
- Synsets can be queried with the **`lang`** keyword :

```
>>> wn.synsets('cane', lang='ita')
[Synset('dog.n.01'), Synset('cramp.n.02'),
Synset('hammer.n.01'), Synset('bad_person.n.01'),
Synset('incompetent.n.01')]
```

# Word-level resources

## Wordnet

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

- The list of all codes for supported languages can be found using **`wn.langs()`**.
- Synsets can be queried with the **`lang`** keyword :

```
>>> wn.synsets('cane', lang='ita')
[Synset('dog.n.01'), Synset('cramp.n.02'),
Synset('hammer.n.01'), Synset('bad_person.n.01'),
Synset('incompetent.n.01')]
```

- It's possible to retrieve lemmas in a given language with the functions **`synset.lemma_names()`** and **`synset.lemmas()`** :

```
>>> dog = wn.synset('dog.n.01')
>>> dog.lemmas('ita')
[Lemma('dog.n.01.cane'), Lemma('dog.n.01.Canis_familiaris')]
>>> dog.lemma_names('ita')
['cane', 'Canis_familiaris']
```

# Word-level resources

## Wordnet

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

- The list of all codes for supported languages can be found using **wn.langs()**.
- Synsets can be queried with the **lang** keyword :

```
>>> wn.synsets('cane', lang='ita')
[Synset('dog.n.01'), Synset('cramp.n.02'),
Synset('hammer.n.01'), Synset('bad_person.n.01'),
Synset('incompetent.n.01')]
```

- It's possible to retrieve lemmas in a given language with the functions **synset.lemma_names()** and **synset.lemmas()** :

```
>>> dog = wn.synset('dog.n.01')
>>> dog.lemmas('ita')
[Lemma('dog.n.01.cane'), Lemma('dog.n.01.Canis_familiaris')]
>>> dog.lemma_names('ita')
['cane', 'Canis_familiaris']
```

- Lemmas have a **lemma.lang()** function that maps to their language code:

```
>>> lemma  = dog.lemmas('ita')[0]
>>> lemma.lang()
'ita'
```

# Word-level resources

## Wordnet

The nltk implementation of Open Multilingual Wordnet yields multilingual support for Wordnet

- The list of all codes for supported languages can be found using `wn.langs()`.
- Synsets can be queried with the `lang` keyword :

```
>>> wn.synsets('cane', lang='ita')
[Synset('dog.n.01'), Synset('cramp.n.02'),
Synset('hammer.n.01'), Synset('bad_person.n.01'),
Synset('incompetent.n.01')]
```

- It's possible to retrieve lemmas in a given language with the functions `synset.lemma_names()` and `synset.lemmas()` :

```
>>> dog = wn.synset('dog.n.01')
>>> dog.lemmas('ita')
[Lemma('dog.n.01.cane'), Lemma('dog.n.01.Canis_familiaris')]
>>> dog.lemma_names('ita')
['cane', 'Canis_familiaris']
```

- Lemmas have a `lemma.lang()` function that maps to their language code:

```
>>> lemma  = dog.lemmas('ita')[0]
>>> lemma.lang()
'ita'
```

- The function `wn.all_lemma_names()` can be restricted to a specific language using the `lang` keyword.

1. Write a function that takes a string as a parameter and checks whether it is a valid language code.
2. Write a function that takes a word and a language code as a parameter, and returns all possible translations of this word in that language according to Wordnet.
3. Write a function that takes a word as a parameter, and returns a dictionary mapping all language codes to synsets for this word in the corresponding language.
4. Write a function that takes a language code as a parameter, and returns all synsets that match an existing lemma in this language.
5. Write a function to compute the number and the proportion of synsets that have lemmas in multiple languages

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

# Word-level resources
## Babelnet

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

- Babelnet is a collection "Babel synsets", mapping of wordnet synsets to wikipedia pages.

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

- Babelnet is a collection “Babel synsets”, mapping of wordnet synsets to wikipedia pages.
- The mapping is initialized by first aligning pages and synsets which are monosemous (wikipedia pages with no disambiguation page associated, and synsets with only one lemmas).

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

- Babelnet is a collection "Babel synsets", mapping of wordnet synsets to wikipedia pages.
- The mapping is initialized by first aligning pages and synsets which are monosemous (wikipedia pages with no disambiguation page associated, and synsets with only one lemmas).
- Redirections are mapped to the synset they redirect to.

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

- Babelnet is a collection “Babel synsets”, mapping of wordnet synsets to wikipedia pages.
- The mapping is initialized by first aligning pages and synsets which are monosemous (wikipedia pages with no disambiguation page associated, and synsets with only one lemmas).
- Redirections are mapped to the synset they redirect to.
- The rest of the mapping is computed by selecting the most probable sense in wordnet based on the content of the wikipedia page.

# Word-level resources
## Babelnet

Babelnet (https://babelnet.org/) is a network of concept mostly based on the integration of wikipedia, wiktionnary and wordnet, with an official Java API. It makes full use of the multilingual structures of Wordnet and Wiki-resources.

- Babelnet is a collection “Babel synsets”, mapping of wordnet synsets to wikipedia pages.
- The mapping is initialized by first aligning pages and synsets which are monosemous (wikipedia pages with no disambiguation page associated, and synsets with only one lemmas).
- Redirections are mapped to the synset they redirect to.
- The rest of the mapping is computed by selecting the most probable sense in wordnet based on the content of the wikipedia page.

*There are concerns raised about the quality of Babelnet.*

# Word-level resources

## Cross-lingual embeddings requirements

In some NLP applications, different sets of word embeddings from multiple languages are used jointly.

# Word-level resources

## Cross-lingual embeddings requirements

In some NLP applications, different sets of word embeddings from multiple languages are used jointly.

- To do this we need to project them in a **shared semantic space**, ie. we “**align**” them

# Word-level resources

## Cross-lingual embeddings requirements

In some NLP applications, different sets of word embeddings from multiple languages are used jointly.

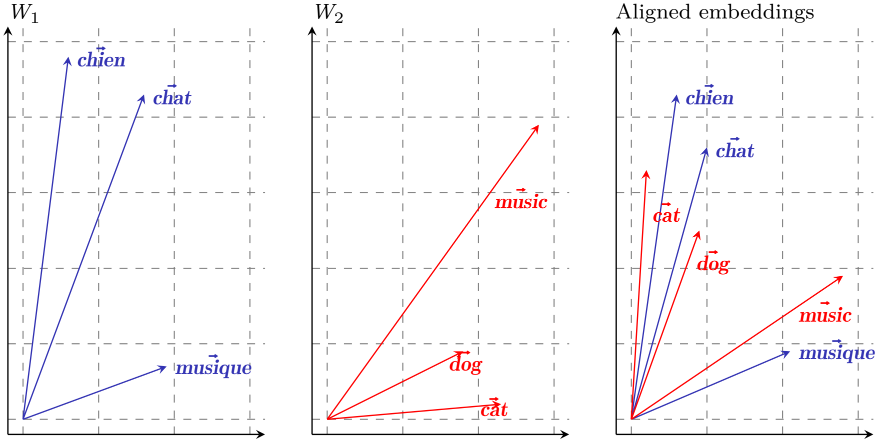
- To do this we need to project them in a **shared semantic space**, ie. we “**align**” them
- we want to make sure that items with a similar meanings are near one another: even more so when it comes to translation pairs

# Word-level resources

## Cross-lingual embeddings requirements

In some NLP applications, different sets of word embeddings from multiple languages are used jointly.

- To do this we need to project them in a **shared semantic space**, ie. we "**align**" them
- we want to make sure that items with a similar meanings are near one another: even more so when it comes to translation pairs



Aligning word embeddings for two different languages $L_1$ and $L_2$ require :

# Word-level resources

## Cross-lingual embeddings requirements

Aligning word embeddings for two different languages $L_1$ and $L_2$ require :

- a set of embeddings for each of the two languages $L_1$ and $L_2$, Aligning word embeddings for two different languages $L_1$ and $L_2$ require :

- a set of embeddings for each of the two languages $L_1$ and $L_2$,
- a set of word pairs $(w_1, w_2)$ such that $w_1$ is a word of $L_1$ and $w_2$ is a translation for $w_1$ in $L_2$.

# Word-level resources

## Cross-lingual embeddings requirements

Aligning word embeddings for two different languages $L_1$ and $L_2$ require :

- a set of embeddings for each of the two languages $L_1$ and $L_2$,
- a set of word pairs $(w_1, w_2)$ such that $w_1$ is a word of $L_1$ and $w_2$ is a translation for $w_1$ in $L_2$.

If these are available, various algorithms can be used to transfer the word embeddings in a common space, we'll focus on SVD (Smith et al., 2017).

# Word-level resources

## Cross-lingual embeddings requirements

Aligning word embeddings for two different languages $L_1$ and $L_2$ require :

- a set of embeddings for each of the two languages $L_1$ and $L_2$,
- a set of word pairs $(w_1, w_2)$ such that $w_1$ is a word of $L_1$ and $w_2$ is a translation for $w_1$ in $L_2$.

If these are available, various algorithms can be used to transfer the word embeddings in a common space, we'll focus on SVD (Smith et al., 2017).

- The general idea is to learn a linear transformation for vectors from $L_1$ to vectors from $L_2$.

# Word-level resources

## Matrix multiplication as a vector function

- The multiplication $C = A\ B$ of a matrix $A$ of shape $(M \times N)$ and a matrix $B$ of shape $(N \times P)$ is of shape $(M \times P)$.

# Word-level resources

## Matrix multiplication as a vector function

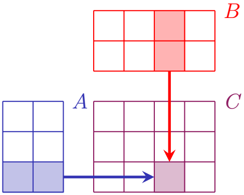
- The multiplication $C = A\ B$ of a matrix $A$ of shape $(M \times N)$ and a matrix $B$ of shape $(N \times P)$ is of shape $(M \times P)$.
  The cell $\langle i, j \rangle$ in $C$ will have as value the dot product between the $i^{\text{th}}$ **row** vector in $A$ and the $j^{\text{th}}$ **column** vector in $B$:

# Word-level resources

## Matrix multiplication as a vector function

- The multiplication $C = A\ B$ of a matrix $A$ of shape $(M \times N)$ and a matrix $B$ of shape $(N \times P)$ is of shape $(M \times P)$.
  The cell $\langle i, j \rangle$ in $C$ will have as value the dot product between the $i^{\text{th}}$ **row** vector in $A$ and the $j^{\text{th}}$ **column** vector in $B$:

$$C_{ij} = \sum_{k=1}^{N} A_{ik} \times B_{kj}$$

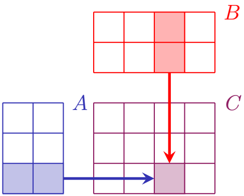


# Word-level resources

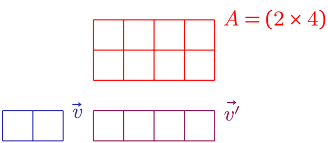
## Matrix multiplication as a vector function

- The multiplication $C = A\ B$ of a matrix $A$ of shape $(M \times N)$ and a matrix $B$ of shape $(N \times P)$ is of shape $(M \times P)$.
  The cell $\langle i, j \rangle$ in $C$ will have as value the dot product between the $i^{\text{th}}$ **row** vector in $A$ and the $j^{\text{th}}$ **column** vector in $B$:

$$C_{ij} = \sum_{k=1}^{N} A_{ik} \times B_{kj}$$



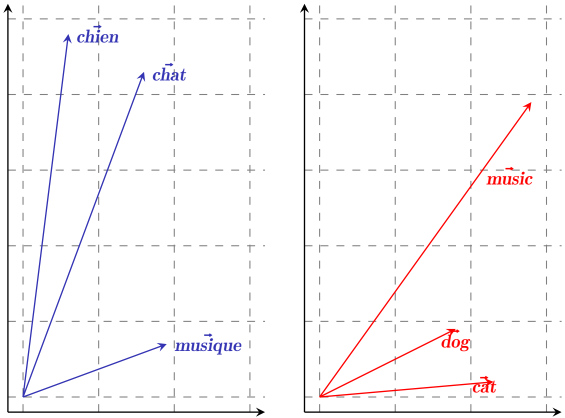

- Therefore the multiplication of a vector $\vec{v}$ of shape $(1 \times d)$ and a matrix $A$ of shape $(d \times d')$ is a vector $\vec{v'} = \vec{v}A$, of shape $(1 \times d')$




- Thus a matrix of shape $(d \times d')$ can be seen as a linear transformation, ie. a function mapping vectors from a space of dimension $d$ to another space of dimension $d'$.

We first need to assess what sort of function is needed to map one space to the other : we do that using the **matrix product**.

We first need to assess what sort of function is needed to map one space to the other : we do that using the **matrix product**.




In this example, high values on the $y$ axis in $W_1$ map to low values on the $y'$ axis in $W_2$. This will be captured in the dimension wise-product :

$$P = W_1^T W_2$$

This product defines the conjunction of the distributional descriptions of word vectors : the cell $\langle y, y' \rangle$ corresponding to the importance of $y$ in $W_1$ to compute $y'$ in $W_2$ will be given a low coefficient. Rows in $P$ will correspond to dimensions in $W_1$, and columns to dimensions in $W_2$.

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

- In linear algebra, SVD is a factorization of a Matrix $M$ in three terms, $U$, $\Sigma$ and $V$, such as $M = U\ \Sigma\ V^T$

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

- In linear algebra, SVD is a factorization of a Matrix $M$ in three terms, $U$, $\Sigma$ and $V$, such as $M = U\ \Sigma\ V^T$
- $\Sigma$ is a diagonal matrix, and $U$ and $V$ are unitary matrix, ie. $U\ U^T = U^T\ U = I$ and $V\ V^T = V^T\ V = I$
  **NB**: Unitary matrices have useful properties, notably their eigenspaces are orthogonal

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

- In linear algebra, SVD is a factorization of a Matrix $M$ in three terms, $U$, $\Sigma$ and $V$, such as $M = U\ \Sigma\ V^T$
- $\Sigma$ is a diagonal matrix, and $U$ and $V$ are unitary matrix, ie. $U\ U^T = U^T\ U = I$ and $V\ V^T = V^T\ V = I$
  **NB**: Unitary matrices have useful properties, notably their eigenspaces are orthogonal
- When $M$ is a square matrix (of shape $K \times K$), $U$ and $V^T$ can be seen as rotations and $\Sigma$ as a scaling factor.

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

- In linear algebra, SVD is a factorization of a Matrix $M$ in three terms, $U$, $\Sigma$ and $V$, such as $M = U\ \Sigma\ V^T$
- $\Sigma$ is a diagonal matrix, and $U$ and $V$ are unitary matrix, ie. $U\ U^T = U^T\ U = I$ and $V\ V^T = V^T\ V = I$
  **NB**: Unitary matrices have useful properties, notably their eigenspaces are orthogonal
- When $M$ is a square matrix (of shape $K \times K$), $U$ and $V^T$ can be seen as rotations and $\Sigma$ as a scaling factor.
- $U$ is usually matched to the rows in $M$, and $V$ to the columns in $M$

# Word-level resources
## Understanding SVD alignment

. We can then use SVD to see how we would need to rotate the two spaces so that they match.

- In linear algebra, SVD is a factorization of a Matrix $M$ in three terms, $U$, $\Sigma$ and $V$, such as $M = U\ \Sigma\ V^T$
- $\Sigma$ is a diagonal matrix, and $U$ and $V$ are unitary matrix, ie. $U\ U^T = U^T\ U = I$ and $V\ V^T = V^T\ V = I$
  **NB**: Unitary matrices have useful properties, notably their eigenspaces are orthogonal
- When $M$ is a square matrix (of shape $K \times K$), $U$ and $V^T$ can be seen as rotations and $\Sigma$ as a scaling factor.
- $U$ is usually matched to the rows in $M$, and $V$ to the columns in $M$
- in the case of our two semantic spaces $W_1$ and $W_2$, if we define $M$ as the conjunction of the effects in $W_1$ and $W_2$, ie. $M = W_1^T W_2$, we can therefore see $U$ as the rotation mapping $W_1^D$ to its natural description in a shared semantic space, and an approximation of the necessary rotation for $W_1$; likewise, we can see $V$ as a natural description of $W_2$.

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

# Word-level resources

## Understanding SVD alignment

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

- First compute $W_1^D$ and $W_2^D$, the subsets of the matrices $W_1$ and $W_2$ containing only vectors of words present in $D$.

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

I First compute $W_1^D$ and $W_2^D$, the subsets of the matrices $W_1$ and $W_2$ containing only vectors of words present in $D$.

II Compute the matrix product $P = W_1^{DT}\ W_2^D$, which can be seen as pairing up $W_1^D$ and $W_2^D$ based on the vectors components.

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

I. First compute $W_1^D$ and $W_2^D$, the subsets of the matrices $W_1$ and $W_2$ containing only vectors of words present in $D$.

II. Compute the matrix product $P = W_1^{DT}\ W_2^D$, which can be seen as pairing up $W_1^D$ and $W_2^D$ based on the vectors components.

III. Then retrieve the rotations by computing the SVD: $P = U\Sigma V^T$

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

I First compute $W_1^D$ and $W_2^D$, the subsets of the matrices $W_1$ and $W_2$ containing only vectors of words present in $D$.

II Compute the matrix product $P = W_1^{D\,T}\ W_2^D$, which can be seen as pairing up $W_1^D$ and $W_2^D$ based on the vectors components.

III Then retrieve the rotations by computing the SVD: $P = U \Sigma V^T$

IV Apply the first rotation to $W_1$, and the second to $W_2$ : $W_1' = W_1 U$ and $W_2' = W_2 V$.

# Word-level resources

## Understanding SVD alignment

In detail, Smith et al. (2017) use the following procedure to align two embedding matrices $W_1$ and $W_2$, using a bilingual lexicon $D = \langle w_1^i, w_2^i \rangle$ :

I First compute $W_1^D$ and $W_2^D$, the subsets of the matrices $W_1$ and $W_2$ containing only vectors of words present in $D$.

II Compute the matrix product $P = W_1^{D\,T}\ W_2^D$, which can be seen as pairing up $W_1^D$ and $W_2^D$ based on the vectors components.

III Then retrieve the rotations by computing the SVD: $P = U \Sigma V^T$

IV Apply the first rotation to $W_1$, and the second to $W_2$ : $W_1' = W_1 U$ and $W_2' = W_2 V$.

This is akin to rotating both word embedding spaces so that they are projected in the same space: we use the transformation $U$ on the embedding space $W_1$, the superset of $W_1^D$ as both relates to the rows of the dimension-wise product $P = W_1^{D\,T}\ W_2^D$; likewise we use $V$ on $W_2$. This allows us to mesh together the semantic spaces.

# Exercises I

## Using SVD to align word embeddings, step-by-step

- To align two word embedding spaces, we need to compute for each of them a linear transformation that projects the vectors into a shared space
- Here's a python function that does this :

```
def compute_transformations(WE_fr_paired, WE_en_paired):
    P = WE_fr_paired.T @ WE_en_paired
    u,s,vT = np.linalg.svd(P)
    return u, vT.T
```

- it supposes that `WE_fr_paired` and `WE_en_paired` are two word embeddings matrices where the $i^{\text{th}}$ row in the one corresponds to the translation of the word for the $i^{\text{th}}$ row in the other
- it uses SVD to return two matrices `u` and `vT.T` which are linear transformations and that must be applied to the whole set of vectors (not just the paired ones that were used to compute these transformations)

With that it mind, let's get to it! The following instructions refer to English and French, but feel free to work on any two language you want.

6. write a functions that returns a list of pairs of strings such all of them are a possible translation of the other. Make sure all words are present in the corresponding lookup!

   *You can either use Wordnet, or simply extract strings present in both lookuup dictionaries that match exactly*

# Exercises II

## Using SVD to align word embeddings, step-by-step

7. Download French and English word embedding spaces from fastText: `https://fasttext.cc/docs/en/crawl-vectors.html`.

   **NB**: If your computer doesn't have the resources to handle the full embedding spaces, you can download and use the French and English vector spaces from the github page for this lecture.

8. write a function that takes an embedding matrix, its lookup, and a list of words as parameters, and returns as sub-matrix containing only the rows corresponding to words in the list.

9. Perform the alignment from start to end:
   - 9.1 Using the function written in 6, compute a bilingual lexicon for English and French.
   - 9.2 split this lexicon in a 90/10 ratio; use the list containing 90% of the examples in the remainder of this exercise.
   - 9.3 compute the embedding matrices and the lookup dictionaries for the two vector spaces using the function written in the previous lecture.
   - 9.4 using the function from 8, compute the sub-matrices where the $i^{\text{th}}$ row in the one is the (vectorized) translation of the $i^{\text{th}}$ row of the other.
   - 9.5 compute the transformations. You can use the above `compute_transformations()`.
   - 9.6 apply each transformation to the corresponding whole word vector space (simply use `numpy.matmul` or its alias `@`).

10. using the remaining 10% bilingual examples that were set aside in 9.2, compute
    - 10.1 the average of the cosines of the French word vectors and their English counterparts before applying the transformations
    - 10.2 the average of the cosines of the (transformed) French word vectors and their (transformed) English counterparts after applying the transformations

# Exercises III

Using SVD to align word embeddings, step-by-step

11. what can you conclude from your results on this held out test set?
12. Adapt the function to retrieve the $k$ top candidates written in the previous lecture, so as to retrieve the $k$ most plausible translations for a given word.
13. Rewrite the function in exercise 6 so that it only contains monosemous lemmas. Do you get different results?
14. Try another algorithm! Can you rewrite the alignment function so that it uses a Stochastic Gradient Descent with Mean Squared Error loss? Do you get better results?

# Outline

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level
- Today, most statistical & neural MT systems rely on parallel corpora of sentences in natural language

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level
- Today, most statistical & neural MT systems rely on parallel corpora of sentences in natural language
- This entails that many sentence-level parallel corpora can be found

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level
- Today, most statistical & neural MT systems rely on parallel corpora of sentences in natural language
- This entails that many sentence-level parallel corpora can be found

**NB**: for instance the WMT corpora: `http://www.statmt.org/wmt18/translation-task.html`

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level
- Today, most statistical & neural MT systems rely on parallel corpora of sentences in natural language
- This entails that many sentence-level parallel corpora can be found

**NB**: for instance the WMT corpora: `http://www.statmt.org/wmt18/translation-task.html`

**NB**: ... or the opensubtitles datasets : `http://opus.nlpl.eu/OpenSubtitles.php`

# Sentence-level resources

## Parallel corpora

Machine translation has been a long standing goal of NLP (The term was first coined by Warren Weaver in 1949)

- Machine translation requires parallel data: linguistic elements from a given source language must be mapped to another target language
- This alignment can be made at any linguistic level
- Today, most statistical & neural MT systems rely on parallel corpora of sentences in natural language
- This entails that many sentence-level parallel corpora can be found

**NB**: for instance the WMT corpora: `http://www.statmt.org/wmt18/translation-task.html`

**NB**: ... or the opensubtitles datasets : `http://opus.nlpl.eu/OpenSubtitles.php`

**NB**: you can even find English-Inuktitut parliamentary parallel data : `http://www.inuktitutcomputing.ca/NunavutHansard/info.php`

What does a parallel corpus look like?

What does a parallel corpus look like? Europarl En $\leftrightarrow$ De

| **Source** `europarl-v7.de-en.en` | **Target** `europarl-v7.de-en.de` |
|---|---|
| `1 Resumption of the session` | `1 Wiederaufnahme der Sitzungsperiode` |
| `2 I declare resumed the session of the European Parliament adjourned on Friday 17 December 1999, and I would like once again to wish you a happy new year in the hope that you enjoyed a pleasant festive period.` | `2 Ich erkläre die am Freitag, dem 17. Dezember unterbrochene Sitzungsperiode des Europäischen Parlaments für wiederaufgenommen, wünsche Ihnen nochmals alles Gute zum Jahreswechsel und hoffe, da Sie schöne Ferien hatten.` |
| `3 Although, as you will have seen, the dreaded ...` | `3 Wie Sie feststellen konnten, ist der gefürchtete ...` |

# Sentence-level resources

## Universal annotations

Another type of multilingual resources are those concerned with universal annotation schemes.

Another type of multilingual resources are those concerned with universal annotation schemes.

- Although Chinese and Japanese have classifier whereas French doesn't, one can try to make an inventory of all the possible PoS-tags, and use it consistently across languages

Another type of multilingual resources are those concerned with universal annotation schemes.

- Although Chinese and Japanese have classifier whereas French doesn't, one can try to make an inventory of all the possible PoS-tags, and use it consistently across languages
- This idea has been seriously considered : cf. for instance the Universal PoS tagset of Petrov, Das, and McDonald (2012)

# Sentence-level resources

## Universal annotations

Another type of multilingual resources are those concerned with universal annotation schemes.

- Although Chinese and Japanese have classifier whereas French doesn't, one can try to make an inventory of all the possible PoS-tags, and use it consistently across languages
- This idea has been seriously considered : cf. for instance the Universal PoS tagset of Petrov, Das, and McDonald (2012)
- Likewise, efforts have been made to consistently annotate morphosyntactical features across languages (for instance Interset by Zeman (2008) has been used to map features across languages, by deriving an "interlingua" representation)

# Sentence-level resources

## Universal annotations

Another type of multilingual resources are those concerned with universal annotation schemes.

- Although Chinese and Japanese have classifier whereas French doesn't, one can try to make an inventory of all the possible PoS-tags, and use it consistently across languages
- This idea has been seriously considered : cf. for instance the Universal PoS tagset of Petrov, Das, and McDonald (2012)
- Likewise, efforts have been made to consistently annotate morphosyntactical features across languages (for instance Interset by Zeman (2008) has been used to map features across languages, by deriving an "interlingua" representation)
- Lastly, much research has been made to present a cross-lingual dependency annotation scheme, called "**Universal Dependencies**" (UD, cf. `http://universaldependencies.org`).

# Sentence-level resources

## Universal Dependencies

- the UD project is an open collaboration, mainly coordinated by Joakim Nivre

- the UD project is an open collaboration, mainly coordinated by Joakim Nivre
- the UD project proposes dependency corpora in 79 languages across many linguistic phyla, from Akkadian to Yoruba and from Old French to Swedish Sign language.

# Sentence-level resources

## Universal Dependencies

- the UD project is an open collaboration, mainly coordinated by Joakim Nivre
- the UD project proposes dependency corpora in 79 languages across many linguistic phyla, from Akkadian to Yoruba and from Old French to Swedish Sign language.
- the annotation scheme is based on an integration of the Stanford Dependency annotations, the universal PoS tagset and the Interset interlingua for morphological features.

- the UD project is an open collaboration, mainly coordinated by Joakim Nivre
- the UD project proposes dependency corpora in 79 languages across many linguistic phyla, from Akkadian to Yoruba and from Old French to Swedish Sign language.
- the annotation scheme is based on an integration of the Stanford Dependency annotations, the universal PoS tagset and the Interset interlingua for morphological features.
- the main idea of the UD project is to be both accessible to the non-specialist (learner, human annotator or NLP engineer) and linguistically accurate for each language (despite being universal).

- the UD project is an open collaboration, mainly coordinated by Joakim Nivre
- the UD project proposes dependency corpora in 79 languages across many linguistic phyla, from Akkadian to Yoruba and from Old French to Swedish Sign language.
- the annotation scheme is based on an integration of the Stanford Dependency annotations, the universal PoS tagset and the Interset interlingua for morphological features.
- the main idea of the UD project is to be both accessible to the non-specialist (learner, human annotator or NLP engineer) and linguistically accurate for each language (despite being universal).
- the corpora are each split in three (train, dev and test), and are available both as raw `.txt` format and as `.conllu` format

# Sentence-level resources

## `.conllu` format

The **`.conllu`** format is a widely adopted format for dependency tree banks

## Sentence-level resources

### `.conllu` format

The `.conllu` format is a widely adopted format for dependency tree banks

- Each sentence is represented as the list of its tokens, eventually preceded by meta-information (eg. sentence ID or plain text) signalled by a # character at the start.

# Sentence-level resources

## `.conllu` format

The **`.conllu`** format is a widely adopted format for dependency tree banks

- Each sentence is represented as the list of its tokens, eventually preceded by meta-information (eg. sentence ID or plain text) signalled by a # character at the start.

- Each token contains fields or "columns", separated by tabs, listed in a specific order :

The `.conllu` format is a widely adopted format for dependency tree banks

- Each sentence is represented as the list of its tokens, eventually preceded by meta-information (eg. sentence ID or plain text) signalled by a # character at the start.

- Each token contains fields or "columns", separated by tabs, listed in a specific order :

    1. **ID**: its index in the sentence
    2. **FORM**: its word form
    3. **LEMMA**: its lemma, if available
    4. **UPOS**: its universal PoS tag
    5. **XPOS**: its language-specific PoS tag
    6. **FEATS**: its morphosyntactic features
    7. **HEAD**: the index of its head, or 0 if it is the root
    8. **DEPREL**: the dependency relation that it holds with respect to its head
    9. **DEPS**: an enhanced graph annotation
    10. **MISC**: any remaining miscellaneous annotation

    Any unspecified or missing information is represented using the _ character. ID cannot be missing. In UD tree banks, the UPOS, HEAD and DEPREL columns must not be unspecified or missing.

# Sentence-level resources

## .conllu format

The `.conllu` format is a widely adopted format for dependency tree banks

- Each sentence is represented as the list of its tokens, eventually preceded by meta-information (eg. sentence ID or plain text) signalled by a # character at the start.

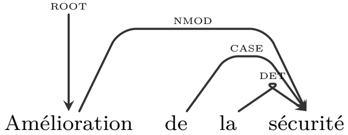
- Each token contains fields or "columns", separated by tabs, listed in a specific order :

  1. **ID**: its index in the sentence
  2. **FORM**: its word form
  3. **LEMMA**: its lemma, if available
  4. **UPOS**: its universal PoS tag
  5. **XPOS**: its language-specific PoS tag
  6. **FEATS**: its morphosyntactic features
  7. **HEAD**: the index of its head, or 0 if it is the root
  8. **DEPREL**: the dependency relation that it holds with respect to its head
  9. **DEPS**: an enhanced graph annotation
  10. **MISC**: any remaining miscellaneous annotation

  Any unspecified or missing information is represented using the _ character. ID cannot be missing. In UD tree banks, the UPOS, HEAD and DEPREL columns must not be unspecified or missing.

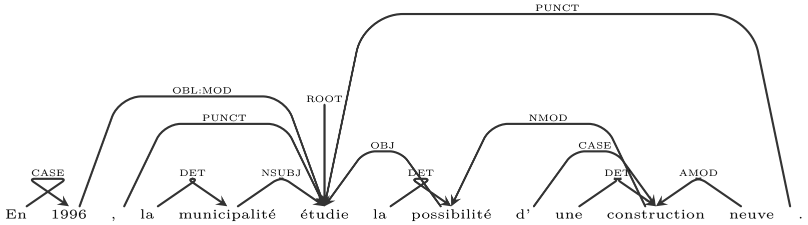
- blank lines separate sentences

# Sentence-level resources
## UD example I




```
# sent_id = annodis.er_00007
# text = Amélioration de la sécurité
1       Amélioration    amélioration    NOUN    _       Gender=Fem|Number=Sing  0       root    _       _
2       de      de      ADP     _       _       4       case    _       _
3       la      le      DET     _       Definite=Def|Gender=Fem|Number=Sing|Pron    Type=Art        4       det     _       _
4       sécurité        sécurité        NOUN    _       Gender=Fem|Number=Sing  1       nmod    _       _
```

# Sentence-level resources




```
# sent_id = annodis.er_00029
# text = En 1996, la municipalité étudie la possibilité d'une construction neuve.
1	En	en	ADP	_	_	2	case	_	_
2	1996	1996	NUM	_	NumType=Card	6	obl:mod	_	SpaceAfter=No
3	,	,	PUNCT	_	_	6	punct	_	_
4	la	le	DET	_	Definite=Def|Gender=Fem|Number=Sing|PronType=Art	5	det	_	_
5	municipalité	municipalité	NOUN	_	Gender=Fem|Number=Sing	6	nsubj	_	_
6	étudie	étudier	VERB	_	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root	_	_
7	la	le	DET	_	Definite=Def|Gender=Fem|Number=Sing|PronType=Art	8	det	_	_
8	possibilité	possibilité	NOUN	_	Gender=Fem|Number=Sing	6	obj	_	_
9	d'	de	ADP	_	_	11	case	_	SpaceAfter=No
10	une	un	DET	_	Definite=Ind|Gender=Fem|Number=Sing|PronType=Art	11	det	_	_
11	construction	construction	NOUN	_	Gender=Fem|Number=Sing	8	nmod	_	_
12	neuve	neuf	ADJ	_	Gender=Fem|Number=Sing	11	amod	_	SpaceAfter=No
13	.	.	PUNCT	_	_	6	punct	_	_
```

# Exercises I

## Universal dependency

15. Retrieve and unzip the Sequoïa UD-treebank corpus available on the github for this lecture.
16. write a function that turns the string for a token into the list of its fields
17. write a function that turns the string of a well-formed sentence into a list of parsed tokens
18. write a function that read a `.conllu` file and returns a list of parsed sentences
19. Test your code: open the train file for the sequoïa corpus. How many sentences are there? how many tokens?
20. write a function that takes the path of a UD tree bank, and returns a list of all the parsed sentences where the root is not a verb. Test it on UD tree banks for different languages. What different results do you get?
21. write a function that takes the path of a UD tree bank, and a token-level query, and returns all the sentences that match this query. The query can be represented as a list of length 10, where each element is the value required for the corresponding column of the token, or `None` if no value is required for that column.
22. write a function that takes the path of a UD tree bank, creates a SQL database and populates it accordingly. Your SQL schema should at least contain tables for sentences, tokens and features (FEATS column of the tokens). Try to conserve all the meta-annotations (starting with #) for sentences as well. Try writing a SQL query to retrieve sentences where the root is not a verb.

# Outline

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages
  - For instance, POS-tagging French texts does not require a part of speech for counters, whereas it is an essential part of Chinese POS-tagsets.

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages
  - For instance, POS-tagging French texts does not require a part of speech for counters, whereas it is an essential part of Chinese POS-tagsets.
  - Other issues include cross-language homographs : compare French and English “pour”.

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages
  - For instance, POS-tagging French texts does not require a part of speech for counters, whereas it is an essential part of Chinese POS-tagsets.
  - Other issues include cross-language homographs : compare French and English “pour”.
- Documents in different languages do not “look the same”. We can classify documents according to their language based on “how they look”.

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages
  - For instance, POS-tagging French texts does not require a part of speech for counters, whereas it is an essential part of Chinese POS-tagsets.
  - Other issues include cross-language homographs : compare French and English “pour”.
- Documents in different languages do not “look the same”. We can classify documents according to their language based on “how they look”.
  - We can look at the distribution of their words : a document containing the word “the” is likely to be in English

# Detecting Language

- Multilingual NLP applications generally have specific processes for specific languages
  - For instance, POS-tagging French texts does not require a part of speech for counters, whereas it is an essential part of Chinese POS-tagsets.
  - Other issues include cross-language homographs : compare French and English “pour”.
- Documents in different languages do not “look the same”. We can classify documents according to their language based on “how they look”.
  - We can look at the distribution of their words : a document containing the word “the” is likely to be in English
  - We can look at the distribution of their characters : a document containing the sequence “kno” is most likely not in French. The distribution of characters is actually specific to each language.

# Exercise I

## Detecting Language using character n-grams, step-by-step

23. write a function that takes a word as parameters and returns the list of **trigrams** it contains : eg. the word “banana” should return the list `['##b', '#ba', 'ban', 'ana', 'nan', 'ana', 'na#', 'a##']`
    As a variation you can also modify this function by adding a parameter $n$ and returning the n-grams for the word.
24. write a function that takes a sentence as a parameter, that computes trigrams for each word in it and returns a dictionary that maps trigrams to their number of occurrences in the text: eg. the sentence “i like ike” should return the dictionary `{"##i" : 2, "#i#" : 1, "i##" : 1, "##l" : 1, "#li" : 1, "lik" : 1, "ike" : 2, "ke#" : 2, "e##" : 2, "#ik" : 1}`
    Use the word tokenizer from nltk to split the sentence into words : download the punkt package and import the function word_tokenize from the module nltk.tokenize.
25. write a functions that takes two dictionaries $d_1$ and $d_2$, mapping trigrams to counts, and returns a dictionary mapping trigrams to the sum of their counts in $d_1$ and $d_2$.
26. write a function that receives a list of sentences paired to their language, and returns a dictionary mapping each language to a trigram count based on the relevant sentences.
27. compute the probability distribution of trigrams stored in a dictionary by normalizing their counts : $P(\mathrm{tri}_t, \mathrm{lang}_L) = \frac{\#\mathrm{tri}_t \wedge \mathrm{lang}_L}{\sum_{\mathrm{tri}_{t'}} \#\mathrm{tri}_{t'} \wedge \mathrm{lang}_L}$

# Exercise II

## Detecting Language using character n-grams, step-by-step

28. write a function that computes the **statistical divergence** between two probabilities of tri-grams using the **total variation distance** :

$$\mathrm{tvd}(P,Q) = \frac{1}{2} \sum_{\mathrm{tri}_t} |P(\mathrm{tri}_t) - Q(\mathrm{tri}_t)|$$

29. write a function that receives a text and a dictionary mapping languages to trigram probability distributions, and returns the language that is the most likely for the text (ie. the language for which the tvd yields the lowest value)
30. Test your code!
    - 30.1 Retrieve a few books in a few different languages from the Gutenberg project : `http://www.gutenberg.org/wiki/Category:Bookshelf`
    - 30.2 For each language, write a function to retrieve test from the books, split the texts into sentences and reserve 1 out of every 10 sentences for testing.
    - 30.3 use the remainder 9 out 10 sentences for computing a distribution probability for each language.
    - 30.4 compute **precision** and **recall** for each language over the 10% sentences that were set aside for testing:

$$\mathrm{precision} = \frac{\#\text{True positives}}{\#\text{True positives} + \#\text{False positives}}$$

$$\mathrm{recall} = \frac{\#\text{True positives}}{\#\text{True positives} + \#\text{False negatives}}$$

- true positives for language $L$ are the sentences that are written in $L$ and predicted as such,
- false positives are sentences predicted to be written in $L$ although they aren't,
- false negatives are sentences not predicted to be written in $L$ although they are.

# Exercise III

Detecting Language using character n-grams, step-by-step

31. Does word frequency matter? Modify your code so that the trigrams of a word are only evaluated once for a language. Do you get different results?
32. Other than using statistical divergence, you can try training a simple SVM classifier using probability distributions over trigrams as feature vectors.
33. Rather than using basic count-based probabilities, try to use smoothing. You can use Laplace smoothing :

$$\hat{P}(\mathrm{tri}_t, \mathrm{lang}_L) = \frac{1 + (\#\mathrm{tri}_t \wedge \mathrm{lang}_L)}{\#\mathrm{tri} + \sum_{\mathrm{tri}_{t'}} \#\mathrm{tri}_{t'} \wedge \mathrm{lang}_L}$$