Multilingual Resources

Lexical Resources - Lectures 7 & 8

December 5^{th}, 2018

Contents of the lecture

Encoding

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)

Useful python library for detecting encoding: chardet

Word-level resources

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 data is frequently used in NLP, both in multilingual and in monolingual contexts

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

Translations [edit]

- ± **large, bulky, corpulent**
- ± **lusty, vigorous**
- ± **proud, haughty**
- ± **firm, resolute**
- ± **materially strong**
- **obstinate** — *see* **obstinate**

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

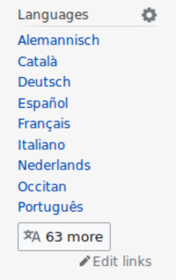
```
====Translations====
{{trans-top|large, bulky, corpulent}}
* Finnish: {{t|fi|pönäkkä}}, {{t+|fi|tanakka}}
* Greek: {{t+|el|}}
* Irish: {{t|ga|alpartha}}
```

(dumps are available here: https://dumps.wikimedia.org/)

- but it's actually hard work.

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

Wordnet

The nltk implementation of wordnet boasts multilingual support

- 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.

Exercises

Multilingual Wordnet

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. **Homework (due Tuesday noon)** : Write a function that takes a language code as a parameter, and returns all synsets that match an existing lemma in this language.
5. **Homework (due Tuesday noon)** : Write a function to compute the number and the proportion of synsets that have lemmas in multiple languages

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.

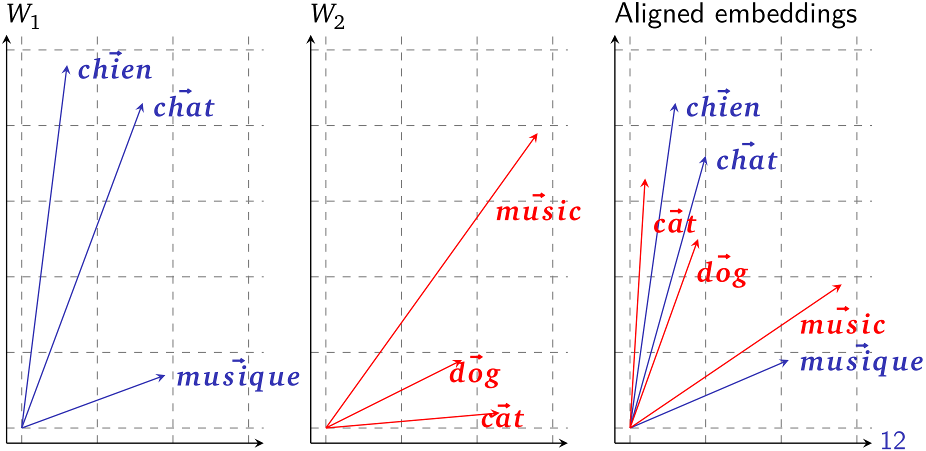
- Babelnet is a collection "Babel synsets", mapping of wordnet synsets to wikipedia pages.
- The mapping is initialized by first aligning pages and synsets which are monosemous (wikipedia pages with no disambiguation page associated, and synsets with only one lemmas).
- Redirections are mapped to the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

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

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.

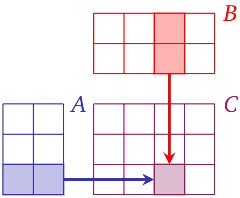
If these are available, various algorithms can be used to transfer the word embeddings in a common space, we'll focus on SVD (Smith et al., 2017).

Word-level resources

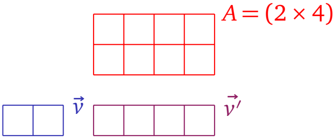
Matrix multiplication as a vector function

- 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^{th} **row** vector in A and the j^{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 $(1 \times d')$

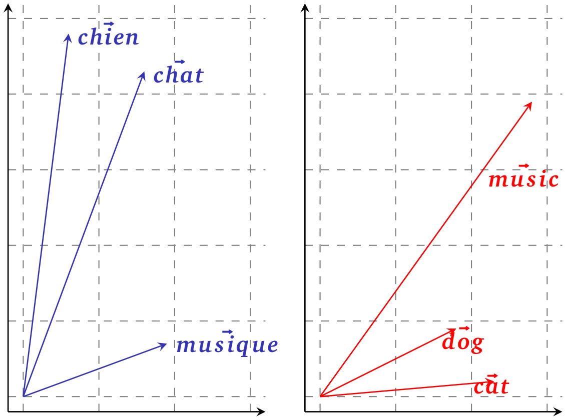


- Thus a matrix of shape $(d \times d')$ can be seen as a function mapping vectors from a given space of dimension d to another space of dimension d'.

Word-level resources

Cross-lingual embeddings

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.

Word-level resources

Cross-lingual embeddings

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, Σ and V, such as $M = U\ \Sigma\ V^T$
- Σ is a diagonal matrix, and U and V are unitary matrix, ie. $U\ U^T = U^T\ U = I$ and $V\ V^T = V^T\ V = I$
- When M is a square matrix (of shape $K \times K$), U and V^T can be seen as rotations and Σ 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 apply the rotation V, representing W_2, to W_1 and the rotation U, a proxy for W_1, to W_2

Word-level resources I

Cross-lingual embeddings

Smith et al. (2017) propose 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 the very specific case where W_1 and W_2 are normalized, P holds the cosine values for all vectors in W_1^D and W_2^D:

$$\begin{aligned} P_{ij} &= \sum_k {W_1^{DT}}_{ik} \times W_{2kj} \\ &= \vec{W_1}j \cdot \vec{W_2}i \\ &= \frac{\vec{W_1}j \cdot \vec{W_2}i}{|\vec{W_1}j||\vec{W_2}i|} \\ &= \cos(\vec{W_1}j, \vec{W_2}i) \end{aligned}$$

Since both $|\vec{W_1}j|$ and $|\vec{W_2}i|$ are equal to 1.

Word-level resources II

Cross-lingual embeddings

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

IV Apply the second rotation to W_1, and the first to W_2 : $W_1' = W_1 V^T$ and $W_2' = W_2 U^T$.
This is akin to rotating both word embedding spaces so that they are projected in the same space: we use the transformation V on the embedding space W_1, the superset of W_1^D because the one relates to the rows of the dimension-wise product $P = W_1^{DT}\ W_2^D$ and the other to the columns; likewise we use U on W_2. This allows us to mesh together the semantic spaces.

Exercises

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

6. write a functions that returns a list of pairs of strings such all of them are a possible translation of the other.

   You can either use Wordnet, or simply, like what Smith & al did, return strings present in the lookuup dictionaries that match exactly

7. write a function that takes a list of elements, and returns two lists l_1 and l_2 such that 9 out of 10 elements in the original list are in the first list l_1, and the remaining ones are in the second list l_2.
8. write a function that takes a list of pairs, and return a pair of lists.
9. write a function that takes a list of words and the path to a word2vec file, and returns the associated list of vectors.
10. write a function that turns a list of vectors into a matrix.

Exercises

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

11. write a function that takes two matrices M and N and returns the rotations based on SVD. Use numpy to do this:
    11.1 first compute the product of $M^T \times N$ using the function `numpy.matmul(M,N)`
    11.2 then compute the singular value decomposition : compute U, Σ and V^T by applying the function `numpy.linalg.svd(P)` to the previously computed product,
    11.3 finally, compute and return the necessary rotations U^T and V^T.
12. Write a function taking a transformation matrix and a vector space as parameters and applies the transformation to the vector space. Once again, use `numpy.matmul()`.

Exercises

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

13. Download French and English word embedding spaces from fasTText: https://fasttext.cc/docs/en/crawl-vectors.html.
    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.
14. write a function to perform and evaluate the alignment from start to end:
    - 14.1 Using the function written in 6, compute a bilingual lexicon for English and French.
    - 14.2 Using the function written in 7, split this lexicon in two.
    - 14.3 Using the functions written in 8, 9 and 10, compute the matrices representing the vector spaces based on the list containing 90% of the lexicon defined in the previous step.
    - 14.4 compute the rotations using the function in 11. Use the French vector space as M and the English vector space as N.
    - 14.5 compute the whole word vector spaces based on the full files. Remember to compute a lookup dictionary as well.
    - 14.6 apply each rotation to the corresponding whole word vector space.
    - 14.7 using the remaining 10% bilingual examples that were set aside in 14.2, compute the average of the cosines of the (transformed) French word vectors and their (transformed) English counterparts.

Exercises

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

15. **Homework** (due Tuesday noon): Rewrite the function in 6 so that it only contains monosemous lemmas. Do you get different results?
16. **Homework** (advanced, due whenever): Try another algorithm! Can you rewrite the function from 11 so that it uses a Stochastic Gradient Descent with Mean Squared Error loss? Do you get better results?

Sentence-level multilingual resources

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

see for instance the corpora available at WMT-18 :
http://www.statmt.org/wmt18/translation-task.html

Sentence-level resources

Parallel corpora

What does a parallel corpus look like? Europarl En ↔ 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.

- 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 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

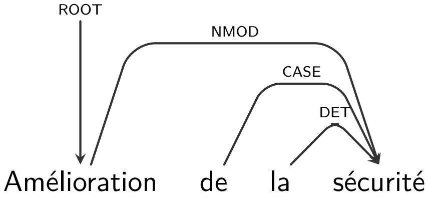
The `.conllu` format is a widely adopted format for dependency tree banks

- 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

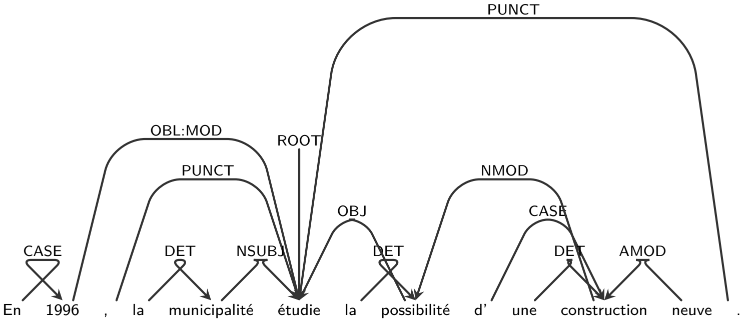
Sentence-level resources

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

Universal dependency

17. Retrieve and unzip the Sequoïa UD-treebank corpus available on the github for this lecture.
18. write a function that turns the string for a token into the list of its fields
19. write a function that turns the string of a well-formed sentence into a list of parsed tokens
20. write a function that read a `.conllu` file and returns a list of parsed sentences
21. Test your code: open the train file for the sequoïa corpus. How many sentences are there? how many tokens?
22. **Homework** (due Tuesday noon): 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?

Advanced Exercises

23. **Advanced homework** (due Tuesday noon): 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.
24. **Advanced homework** (due whenever): 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.

Language detection

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

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

25. 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.

26. 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.

27. 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.

Exercises

Detecting Language using character n-grams

28. 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.
29. compute the probability distribution of trigrams stored in a dictionary by normalizing their counts : $P(\text{tri}_t, \text{lang}_L) = \frac{\#\text{tri}_t \wedge \text{lang}_L}{\sum_{\text{tri}_{t'}} \#\text{tri}_{t'} \wedge \text{lang}_L}$
30. write a function that computes the **statistical divergence** between two probabilities of tri-grams using the **total variation distance** :

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

31. 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)

Exercises

Detecting Language using character n-grams

32. Test your code!
    32.1 Retrieve a few books in a few different languages from the Gutenberg project : http://www.gutenberg.org/wiki/Category:Bookshelf
    32.2 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.
    32.3 use the remainder 9 out 10 sentences for computing a distribution probability for each language.
    32.4 compute **precision** and **recall** for each language over the 10% sentences that were set aside for testing:

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

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

- 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.

Advanced exercises

Detecting Language using character n-grams

33. **Homework** (due Tuesday noon) : 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?
34. **Homework** (due Tuesday noon) : Other than using statistical divergence, you can try training a simple SVM classifier using probability distributions over trigrams as feature vectors.
35. **Homework** (due Tuesday noon) : Rather than using basic count-based probabilities, try to use smoothing. You can use Laplace smoothing :

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