Word Classes and Embeddings

due 23 December, 8:00 pm

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

1. Word Classes

Word Classes

Objectives:

- download and tokenize data from the Guthenberg project
- implement the word classes algorithm
- compute the word class hierarchy on frequent words
- compare the results between two languages

Obtaining Data

The Guthenberg project (https://www.gutenberg.org/) is a collection of public domain books, downloadable in plain text -- handy for NLP experiments!

- Download Dostoevsky's "The house of the dead" in plaintext
  - Available in both English and Czech

```
!wget -O "guthenberg-cs.txt" "https://www.gutenberg.org/cache/epub/34225/pg34225.txt"
!wget -O "guthenberg-en.txt" "https://www.gutenberg.org/cache/epub/37536/pg37536.txt"
```

- Tokenize using `MosesTokenizer` from the `sacremoses` library
  - same as in Assignment 1
- Note down token counts and unique token counts for submission form

Implement Word Classes Algorithm

- English
  - Take the first **30000** tokens, discard the rest
  - Compute class hierarchy for tokens occurring at least **50** times in the data
  - This should result in 73 classes
- Czech
  - Use the first **15000** tokens, discard the rest
  - Compute classes for tokens seen least **20** times
  - Yields 77 classes
- In both cases
  - Print out every merge as it takes place
  - Stop after reaching **15** classes, then print out the members of those classes

Inspecting the Word Classes

Look at the resulting classes as well as the hierarchy as a whole

- Do you observe any interesting results?
  - For example, is it easy to assign “names” to some of the classes?

- How does the class structure compare among the languages?

2. Visualizing Word Embeddings

Visualizing Word Embeddings

Objectives:

- obtain pre-trained word embeddings from fastText
- visualize the embeddings of the class members from previous part
- re-cluster the words using their embeddings
- visualize and compare to the original classes

Visualizing Word Embeddings

FastText embeddings

- Word embeddings trained on the Common Crawl corpus https://fasttext.cc/docs/en/crawl-vectors.html
- Can be used with the `fasttext` python package
  - Remember to call `!pip install` from the colab so the notebook can be run
- FastText distribution offers embeddings with the dimension of 300
  - This is too large (about 7GB per language)
  - Instead, you will use reduced models with dimension 50

Visualizing Word Embeddings

FastText embeddings

- The following code can be used to download and instantiate a fasttext model with word embeddings dimension of 50:

```
!wget "https://ufallab.ms.mff.cuni.cz/~helcl/npfl147/cc.en.50.bin"
!wget "https://ufallab.ms.mff.cuni.cz/~helcl/npfl147/cc.cs.50.bin"

import fasttext
ft_en = fasttext.load_model('cc.en.50.bin')
```

You can play with larger models but use the 50 variant in your submission

Visualizing Word Embeddings (cont.)

Visualize the embeddings of the class members in 2D

- Use the `PCA` class from scikit-learn to reduce the dimensionality
- Plot the (73 for English, 77 for Czech) points in 15 colors
  - such that words from the same class have the same color…
- Add labels to the points so you can see which word they correspond to

Re-cluster

Instead of using the word classes algorithm, let's cluster the embeddings.

- Take the same groups of 73 and 77 words from each language
- Get their embeddings
  - you should already have this from the previous step
- Run a clustering algorithm on the vectors to get 15 clusters
  - you can choose an algorithm from scikit-learn, e.g. KMeans or AgglomerativeClustering
- Visualize again with the clusters obtained this way
- How do the plots compare?

Submission

Submission details (same as last time)

Each submission should include:

- Filled-out Google form checklist
  - use the checklist as a reference for what steps to take
  - it will also try to steer your in the correct direction
  - submission link on course website
- A self-contained Google colab notebook
  - must be runnable from start to end with no edits necessary
  - should not depend on your environment (e.g. access files on your Drive)
  - *in case you have an issue with Google you might hand in a single *.ipynb file, but:*
    - *we will open it in Google colab (so it needs to be compatible)*
    - *it still needs to be self-contained and runnable from start to end*
    - *you will still need to fill out the Google form*

Submission details (same as last time)

Note on the use of generative models

- You are allowed (and encouraged to try) to use AI to help you generating your code
  - if you use it, describe how did you do it and to what extent did it help
  - always check the model output -- it can get the math wrong, generate inefficient code, ...
  - regardless of how the code was produced, you must be able to explain every detail
- There is not much free form text required, but when there is, you should use your own words
  - models can help with discussion, but can also be quite confused or just wrong
  - if something is unclear, send us an email

Submission details (same as last time)

Tips and tricks

- installing packages from within the notebook:
  https://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-pip
- choose your data structures wisely
  `Counter`s and `defaultdict`s are your friends
- write language-independent code, avoid copy-pasting
- develop/debug on a small data sample