Entropy and LM Smoothing

due 25 November, 8:00 pm

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

Entropy of a Language

Entropy of language

Objectives:

- obtain data from the Hugging Face repository
- preprocess using a tokenizer
- get basic statistics of the datasets
- calculate conditional entropy
- compare conditional entropy across languages and tokenizers

Entropy of language

Getting data from Hugging Face

- using the `load_dataset` from Hugging Face `datasets` library
- use the `ufal/npfl147` dataset
  - https://huggingface.co/datasets/ufal/npfl147
  - this is a collection of random subsets of the HF's FineWiki dataset in multiple languages
- get data for Czech, English, and another language of your choosing
  - the set is limited to Slovak, Norwegian, Turkish, Basque, German, and Irish.
- textual data is stored in the `"text"` column of each example

Entropy of language

Preprocessing

- split the text into words with `MosesTokenizer`
    - from the Sacremoses library
    - note that the tokenizer constructor has a `lang` parameter -- make sure to set it to the correct language
- do not use any other form of pre-processing (such as lowercasing or normalization)
- ignore document boundaries
    - concatenate everything into a long list
    - keep languages separate

Entropy of language

Counting -- for each language, get the following stats:

- unigram (token) and bigram frequencies
    - get trigram frequencies as well for the 2nd part
- number of unique unigrams and bigrams (vocabulary size)
- data size (number of all tokens)

Entropy of language

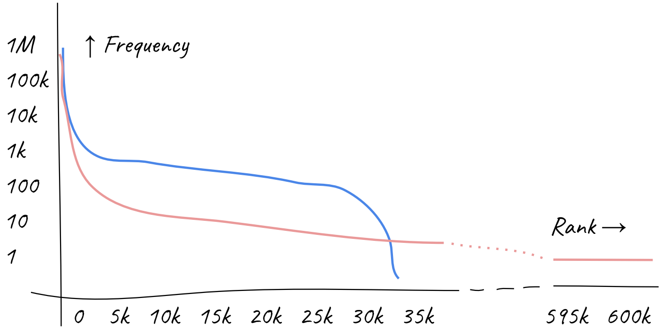
Measuring entropy

- use the formula for conditional entropy
- tabulate the following (ideally using the `tabulate` package)
  - language
  - number of unique tokens
  - number of unique bigrams
  - data size (total number of tokens)
  - entropy
- how does the entropy compare among the languages?
  - which language has the highest entropy? which one lowest?
  - explain the differences

Entropy of language

Experiment

- run the counting again, now using the XLM-R tokenizer
    - subword tokenizer (splits less frequent tokens to counteract the long tail distribution)
    - supports many languages (used to train the XLM-R model)
    - load with `AutoTokenizer.from_pretrained("xlm-roberta-base")`



- tabulate everything again -- how does entropy compare to the baseline tokenizer?
    - again, try to point out and explain the differences

LM Smoothing

N-gram LM Smoothing

Objectives:

- create training, heldout and test splits for your data
- estimate parameters of a 3-gram language model
- use EM algorithm to find weights for interpolated smoothing
- measure test data cross entropy

N-gram LM Smoothing

Cross-entropy

- report cross-entropy of **test data S**:
  - with initial lambdas
  - with the best lambdas
- plot how cross-entropy changes with lambda 3:
  - set lambda 3 to values from `np.linspace(0,0.95,20)`
  - keep best uniform, unigram and bigram lambdas
    - but re-normalize so that they sum up to 1 again
- explain the results
  - how does cross-entropy change with lambda 3 and why?
  - are there systematic differences across languages?

N-gram LM Smoothing

Split data

- create a train, heldout, and test splits
  - use the first 700 documents as training data T
  - then the next 200 documents as heldout data H
  - and the last 100 documents as test data S
  - work with the same languages as in part 1

Compute n-gram frequencies

- get counts of unigrams, bigrams, and trigrams
  - careful, the data is different from the previous part
- also make note of vocabulary size for uniform probability

N-gram LM Smoothing

Estimate n-gram probabilities from the **training** data T

$$\begin{aligned} p_0 &= 1/|V| \\ p_1(w_i) &= c_1(w_i)/|T| \\ p_2(w_i|w_{i-1}) &= c_2(w_{i-1}, w_i)/c(w_{i-1}) \\ p_3(w_i|w_{i-2}, w_{i-1}) &= c_3(w_{i-2}, w_{i-1}, w_i)/c(w_{i-2}, w_{i-1}) \end{aligned}$$

.. and the smoothed trigram probability

$$p(w_i|w_{i-2}, w_{i-1}) = \lambda_0 p_0 + \lambda_1 p_1(w_i) + \lambda_2 p_2(w_i|w_{i-1}) + \lambda_3 p_3(w_i|w_{i-2}, w_{i-1})$$

where $\sum_{i=0}^{3} \lambda_i = 1$

N-gram LM Smoothing

Finding the best lambdas

$$p(w_i|w_{i-2}, w_{i-1}) = \lambda_0 p_0 + \lambda_1 p_1(w_i) + \lambda_2 p_2(w_i|w_{i-1}) + \lambda_3 p_3(w_i|w_{i-2}, w_{i-1})$$

- use the EM algorithm to find the best set of smoothing params
    - start with uniform lambdas, 0.25 each
    - best = minimalizes the cross entropy
    - this should be run with the **heldout** data **H**
- be mindful about corner cases
    - probability of OOVs -- zero when history known, but uniform for OOV history

Submission

Submission details

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

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

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 sample (e.g. 10 documents from one language)
  … e.g. `load_dataset(“ufal/npfl147”, lang, split=”train[:10]”)`