First-Year Project 3: Natural Language Processing
Exercise 3: N-gram Language Modelling
Christian Hardmeier

In this exercise, you will get familiar with n-gram language models using the `nltk` library.

You can work with the corpora from the two tasks you selected for your project. Additionally, you can download a dataset of news editorials collected and distributed by the organisers of the Conference on Machine Translation from here: `http://data.statmt.org/news-commentary/v16/training-monolingual/news-commentary-v16.en.gz`

This is a sizeable data set (albeit still quite small by the standards of the NLP community), and if you find that your computer becomes intolerably slow or runs into memory problems, you can just use a subset of the corpus for the exercises.

1. Remove a subset of about 5000 sentences from the news commentary dataset to be used for evaluation. For the TweetEval datasets, you can use the standard training/validation split.
2. Load and tokenise your datasets so that, for each of the corpora, you get a list of sentences, each sentence represented by a list of tokens. Use the same tokeniser for all datasets.
3. Follow the instructions at

   `http://www.nltk.org/api/nltk.lm.html#module-nltk.lm`

   to train maximum-likelihood language models of varying orders (e.g., $n = 2, \ldots, 6$) for each of your corpora. Make a note of the size of the n-gram lists for each n-gram order. You might also plot them.

   IMPORTANT: We will want to compare perplexities across different corpora. This only produces meaningful results if all the models use exactly the same vocabulary. To ensure that, create a vocabulary from the largest of your datasets and use it for *all* the corpora.
4. Use the `lm.generate` function to generate some example text from each of your models and compare.
5. Use `lm.score` and `lm.logscore` to calculate the scores of a couple of n-grams you find in the texts with the different models. Try 1 or 2 n-grams with all function words and 1 or 2 n-grams that contain specific content words or names.
6. Use the `lm.perplexity` function to compute the perplexity of the validation sets from each dataset with each of the language models and compare. Keep in mind that LOWER perplexity is BETTER.
7. Repeat the steps above with language model class implementing another smoothing algorithm, e. g., `WittenBellInterpolated`, `KneserNeyInterpolated` or `Laplace`.