NLTK and Language Models

Lab session

In this session we will learn to use NLTK to build n-gram Language Models and apply them for language generation. We will also exploit KL Divergence for authorship identification.

1. Preliminary steps

Make sure you have the latest NLTK version (3.4 or higher).
You can install NLTK using pip: `pip install NLTK`

The following examples will be using Python 3 syntax and conventions.
Once you have installed NLTK, you need to download the required corpora. Launch the Python interpreter and type:

```
>>> import nltk
>>> nltk.download()
```

A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory to change the location. Next, select *all-corpora* to download.

NLTK provides many corpora covering different types of texts. We'll work with the C-span corpus of state of the union and inaugural speeches by US presidents.

These corpora can be accessed by typing:

```
>>> from nltk.corpus import state_union
>>> from nltk.corpus import inaugural
```

To list all documents in a corpus, you can use the fileids() method:

```
>>> inaugural.fileids()
```

To access the content of a given document, you can use the raw(), words() and sents() methods as follows:

```
>>> inaugural.raw('2009-Obama.txt')
>>> inaugural.words('2009-Obama.txt')
>>> inaugural.sents('2009-Obama.txt')
```

2. The lm module: basics

Since NLTK 3.4, the **lm** module allows to build language models.
First thing we need is to count types (words, bigrams, trigrams, etc.): we can do this in different ways, using Counter from the collections package or, more properly, by using **NgramCounter** from the lm module:

```
>>> from nltk.util import ngrams
>>> from nltk.lm import NgramCounter
```

Let's count all unigrams in the state_union corpus:

```
>>> text_unigrams = [ngrams(sent, 1) for sent in state_union.sents()]
>>> ngram_counts=NgramCounter(text_unigrams)
>>> ngram_counts.N()
```

Be careful since ngrams produces a generator: once the ngrams are used, they are "lost".

You can look at the frequencies of a type in a very simple way:
```
>>> ngram_counts['the']
```
For bigrams:
```
>>> ngram_counts[['the']]['new']
```

The "unigrams" member is a FreqDist object:
```
>>> ngram_counts.unigrams.most_common(20)
```
Shows the 20 most frequent types. If you have matplotlib installed, it is possible to display a rank/frequency diagram by typing:
```
>>> ngram_counts.unigrams.plot(20)
```

Vocabulary objects allow to create a vocabulary from a set of types and a frequency threshold:
```
>>> vocab = Vocabulary(state_union.words(), unk_cutoff=2)
```
The vocabulary will include all words that appear at least 2 times in the corpus.

Exercise 1. Count the number of unigrams for each of the presidents in the *inaugural* dataset. Which one held the longest discourse? Which one the shortest one?
1.b) Count the number of different types. Which president used the richest vocabulary for his speech?
Repeat the experiments in the *state_union* dataset and verify if these results hold (at least for the presidents after 1945).

3. Building a Language Model

Once we have the data we need to introduce padding to tell the model what the boundaries of the sentence are, and to calculate probabilities for the starting and the end of a sentence. Luckily, NLTK has a function that allows us to do it easily:

```
>>> from nltk.lm.preprocessing import pad_both_ends
>>> list(pad_both_ends(sentence, n=2)
```

Extracting the bigrams:
```
>>>list(bigrams(pad_both_ends(sentence, n=2)))
```

[('<s>', 'May'), ('May', 'God'), ('God', 'bless'), ('bless', 'America'), ('America', '.'), ('.', '</s>')]

The module **lm** will need also a flattened list of symbols to build the vocabulary. We can do it with the following function:
```
>>> from nltk.lm.preprocessing import flatten
>>> list(flatten(pad_both_ends(sent, n=2) for sent in corpus))
```

There is also a convenience method called everygram_pipeline that produces the two at the same time – from the manual:

```
padded_everygram_pipeline(order, text):
    """Default preprocessing for a sequence of sentences.

    Creates two iterators:
    - sentences padded and turned into sequences of `nltk.util.everygrams`
    - sentences padded as above and chained together for a flat stream of words

    :param order: Largest ngram length produced by `everygrams`.
    :param text: Text to iterate over. Expected to be an iterable of sentences:
    Iterable[Iterable[str]]
    :return: iterator over text as ngrams, iterator over text as vocabulary
data
    """
```

For example:

```
train, vocab = padded_everygram_pipeline(2, inaugural.sents())
```

Having prepared our data we are ready to start training a model. As a simple example, let us train a Maximum Likelihood Estimator (MLE).

```
>>> from nltk.lm import MLE
>>> lm = MLE(2)
```

This model is empty until we fill it with the data:

```
>>> lm.fit(train, vocab)
```

We can verify frequencies in the same way as we did with the NgramCounter:

```
>>> lm.counts['America']
384

>>> lm.counts[['bless']]['America']
4
```

The function score returns the probability of the given type

```
>>> lm.score('the')
0.057883609103212566
>>> lm.score('America')
0.0011974628755324656
>>> lm.score("America", ["bless"])
0.125
```

Note that these probabilities are not smoothed since we used a MLE model. For better results, models with smoothing are available:

- nltk.lm.**Lidstone** (requires the gamma parameter to increase scores)
- nltk.lm.**Laplace** (add 1)
- nltk.lm.**KneserNeyInterpolated**

We can also evaluate our model's cross-entropy and perplexity with respect to sequences of ngrams with the methods *entropy* and *perplexity* (they take as input a sequence of ngram tuples).

Exercise 2. Build a language model from the *state_union* dataset. Verify the probabilities for some words (example: “America”, “bless”) with different smoothing models.

4. Generation

Finally, we are using the model to generate language. The key function is *generate*, always in the lm interface:

```
def generate(self, num_words=1, text_seed=None, random_seed=None):
        """Generate words from the model.

        :param int num_words: How many words to generate. By default 1.
        :param text_seed: Generation can be conditioned on preceding context.
        :param random_seed: If provided, makes the random sampling part of
        generation reproducible.
        :return: One (str) word or a list of words generated from model.
```

Exercise 3. Generate some sentences from the full dataset. Build a separate model for each president and try to generate some sentences for each one. Test with different sizes of n-grams. Verify if the model is able to “reproduce” the style of some president better than another one.

Exercise 4. Neural Language Generation: download the Python script neuralLG.py from the moodle. This script contains a demonstration of how to create a neural language generator using Keras (spoiler: it’s extremely easy). Adapt the script to work on the presidential speech dataset and generate 10 sentences using the whole data to build the model and 10 with a model built using only the data for a president of your choice. Draw your conclusions in comparison with the classic language models.

5. Authorship attribution

We are now goint to use the KL Divergence to determine who is the author of a given speech. A method has been proposed in (Zhao et al., 2006) available in the moodle (airs06.pdf). We will replicate their method.

Exercise 5. We will use the *inaugural* speeches as test data and the *state_union* ones as training data. We will build a model for each president (since Truman) using unigrams only, then unigrams + bigrams, then including also trigrams.

Use Laplace smoothing at the beginning, if you have time you can implement the smoothing proposed in the AIRS paper.

An implementation of KLD between two probability distributions is available in SciPy: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html

5.a) Using the KL divergence we will find the “most dissimilar” US presidents and those who have the most similarities.

5.b) We will then use these models to try to assign each inaugural speech to the right president.

5.c) Test with inaugural speeches of older presidents, who are more similar to in the recent history?