Week 4 lab

Markov generation and language modelling

Download a new **Astro-ph abstracts corpus (https://canvas.sydney.edu.au/courses/2437/files/1802760/download?wrap=1)** **(https://canvas.sydney.edu.au/courses/2437/files/1802760/download?wrap=1)** (extracted from **Astro-ph arxiv (http://arxiv.org/archive/astro-ph)** with **get-arxiv-abstracts.py (https://github.sydney.edu.au/COMP5046-Natural-Language-Processing/comp5046-labs-2018/blob/master/lab04/get-arxiv-abstracts.py)**). It contains one document per line.

Generation

The following task was presented as Extension in Lab 1, but we now frame it in terms of language models.

We are going to use a markov language model to generate text based on astrophysics abstracts.

Read: **Nature: *Publisher Withdraws more than 120 Gibberish Papers* (http://www.nature.com/news/publishers-withdraw-more-than-120-gibberish-papers-1.14763)**

Work in pairs or groups of three

1. Tokenise the text and collect sufficient statistics for a bigram model.
2. Calculate the empirical probability distributions for each word given the preceding word (where the preceding word may be "<START-DOC>"; and a subsequent word may be "<END-DOC>").
3. Generate new abstracts by starting with "<START-DOC>" and generating each word by drawing from the empirical probability distribution of words which follow the previous word (or two words). I.e. draw each word x according to the empirical distribution $P(w_t \mid w_{t-1} = y)$ where y is the previous word. See **Sampling from a multinomial distribution** below. Stop generating when your next selected word is "<END-DOC>".
4. Do the generated texts look realistic? What kinds of consistent problems do you see that you would not expect in real abstracts?
5. Either extend your model to trigrams, or use the **example solution (https://github.sydney.edu.au/COMP5046-Natural-Language-Processing/comp5046-labs-2018/blob/master/lab04/markov.py)**. How much more realistic are abstracts generated by a trigram model? How much larger is a trigram model than a bigram model?
6. What do you expect to happen as you increase the n-gram length?

Probability estimates

Work in pairs or groups of three

1. Use the model built above the estimate the probabilities of astrophysics abstracts. It is often useful to report the *logarithm* of probabilities for readability and numerical stability.

2. Calculate the log-likelihood of the training data under the bigram model, i.e. the sum of the log-likelihood of each abstract.

3. Apply a simple smoothing technique, i.e. Laplace/Lidstone, so that we can apply this model to unseen text. How does this affect overall model likelihood?

4. Estimate the probability of the first abstract. Retrain your model on all abstracts but the first and estimate its probability again. How has it changed?

5. Extension: Measure the perplexity of the model on collections of text from different domains (e.g. other scientific domains, wikipedia text, news media). Perplexity is defined in **SLP3 chapter 4 (https://web.stanford.edu/~jurafsky/slp3/4.pdf)**, section 4.2.1, as $P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$, and is the standard evaluation metric for language modelling.

Algorithm: Sampling from a multinomial distribution

Once you know the (n-1)-gram of history, generation is about selecting the next token from a conditional probability distribution, $P(x|w_{t-1} = y)$. We do not want to select the most probable word; we want to sample one word in accordance with the distribution. How can we do this?

Let's say the words following some history have this count:

```
     counts
bat       8
cat       2
eat       2
fat       6
hat       9
mat       6
oat       3
pat       6
rat       7
sat       2
```

One way to sample is to simply create a list like ['bat', 'bat', 'bat', 'bat', 'bat', 'bat', 'bat', 'bat', 'cat', 'cat', 'eat', 'eat', 'fat', 'fat', 'fat', 'fat', 'fat', 'fat', 'hat', 'hat', 'hat', 'hat', 'hat', 'hat', 'hat', 'hat', 'hat', 'mat', 'mat', 'mat', 'mat', 'mat', 'mat', 'oat', 'oat', 'oat', 'pat', 'pat', 'pat', 'pat', 'pat', 'pat', 'rat', 'rat', 'rat', 'rat', 'rat', 'rat', 'rat', 'sat', 'sat'] where each word is repeated the number of times you saw it. Now you can just choose a random number up to the total, 51, and select the corresponding word.

More efficiently, we can produce a probability distribution by dividing by the total:

```
     counts  probability
bat       8     0.156863
cat       2     0.039216
eat       2     0.039216
fat       6     0.117647
hat       9     0.176471
mat       6     0.117647
```

```
oat       3     0.058824
pat       6     0.117647
rat       7     0.137255
sat       2     0.039216
```

To select from this probability distribution, we need to generate a random number from 0 to 1, such as r := 0.48. We can select the corresponding word by going through our list of candidates and subtracting from r until r is less than 0. The first word where our adjusted r is less than or equal to 0 is the word we pick:

```
     counts  probability  subtracting from r
bat       8     0.156863            0.323137
cat       2     0.039216            0.283922
eat       2     0.039216            0.244706
fat       6     0.117647            0.127059
hat       9     0.176471           -0.049412
```

Here we pick "hat". Perhaps another example will help with the intuition behind it: if "bat" had probability 0.5, then we would pick it, because subtracting 0.5 from 0.48 would give us -0.02.