ELEC-E5510 — Exercise 3 part 1: N-gram language models

In speech recognition, a language model is used for defining probabilities for sentences. For example, if the recognizer has the following hypotheses that are equally probable according to the acoustic phoneme models, the language model can be used to choose the correct hypothesis:

- Wright a letter
- right a letter
- write a letter

The purpose of the exercise is to:

- understand what n-gram language models are
- learn to train n-gram models using the SRILM toolkit
- learn to evaluate the quality of an n-gram model using an evaluation text corpus

To understand the theoretical part behind this exercise in more depth, you can read this chapter.

Use the submission instructions for returning your answers. **Deadline is Wednesday 15.11.2023 at 23:59**.

Preparations

SRILM is a very good open source toolkit for training n-gram models, and it has been installed under the course directory. In order to use SRILM tools, you need to add the course path to your PATH environment variable. If your shell is `/bin/bash` or `/bin/zsh` (find out by typing: `echo $SHELL`), type

```
PATH="$PATH:/work/courses/T/S/89/5150/general/bin/srilm"
```

If your shell is `/bin/tcsh`, type

```
set path = ($path /work/courses/T/S/89/5150/general/bin/srilm)
```

The manual pages of the SRILM toolkit explains the options of the tools in more detail, if you need more help.

N-gram models

Recall from the lecture notes what the n-gram models are and how the maximum likelihood (ML) estimates are computed. For a reference, 1-gram ML estimate can be computed as

$$P(w_i) = \frac{C(w_i)}{\sum_j C(w_j)}$$

and 2-gram ML estimate

$$P(w_i \mid w_j) = \frac{C(w_j, w_i)}{C(w_j)}$$

where $C(w_i)$ is the count of word w_i and $C(w_j, w_i)$ is the count of sequence "$w_j w_i$".

Let's create a toy text corpus using the following words: it, is, on, in, a, the, box, bag. Additionally SRILM toolkit always uses <s> for denoting start of the sentence, and </s> for denoting end of the sentence.

Create a text file `train.txt` that contains the following new sentences (one sentence per line). Use only lower-case letters. You can also omit the <s> and </s> tags, because the toolkit adds them automatically if they are missing.

```
<s> the box is in the bag </s>
<s> it is in a bag on the box </s>
<s> is the box in it </s>
```

The `ngram-count` command can be used for computing how many times different n-grams appear in the training data. You can obtain 1-gram counts with the following command:

```
ngram-count -order 1 -text train.txt
```

Try computing also 2-gram counts and 3-gram counts by changing the `-order` parameter. To explore more possible options see SRILM manual.

Question 1

Fetch the 1-gram counts again and compute the maximum likelihood estimates for the following 1-gram probabilities by hand (you can use a calculator):

- *P(in)*
- *P(a)*
- *P(</s>)*

Note: SRILM reports also the count of <s>. You need to ignore it (don't include it in the counts) since <s> is always assumed in the beginning of the sentence and is never generated later.

Question 2

Use `ngram-count` to get necessary counts and compute the following 2-gram and 3-gram estimates (maximum likelihood) below by hand. Note that the notation P(bag | in the) means the probability that word "bag" appears after "in the" (for example in the sentence "in the bag"). The variable X in the equations below may represent any word in the vocabulary (there are 9 words excluding <s>). This means you need to report even n-grams that do not appear in our corpus.

- *P(X | is) for all X*
- *P(X | in the) for all X*

Hint: you can use "grep" to filter the output. For example, to get all n-grams that start in "is in" and have non-zero probability according to our corpus, you can type:

```
ngram-count -order 3 -text train.txt | grep "^is in"
```

We see that maximum likelihood estimation gives zero probabilities for word sequences that have not occurred in the training data. In speech recognition this may be a problem, because sentences with zero probability can not be recognized. That is the reason why smoothing is used.

Smoothing

The idea of smoothing is to give some probability mass to n-grams that have not occurred in the training data. Thus, probability mass has to be taken from n-grams that occur in the training data. There are several smoothing methods, but here we consider absolute discounting with interpolation. The equation for the 2-gram probability then becomes

$$P(w_i \mid w_j) = \frac{\max(0, C(w_j, w_i) - D)}{C(w_j)} + P(w_i) b(w_j)$$

where

$$b(w_j) = \frac{\text{\# of different words appearing after } w_j}{C(w_j)} D$$

Question 3

Get the 2-gram counts again.

***a)** Using interpolated absolute discounting (D=0.5) compute P(in | is) and P (</s> | is) by hand.*

***b)** Compare to results you got in Question 2. What has changed? Why?*

Instead of computing estimates by hand, we can use ngram-count tool to compute the language model.

```
ngram-count -order 2 -interpolate -cdiscount1 0 -cdiscount2 0.5 \
  -text train.txt -lm 2gram.lm
```

The above command trains a 2-gram model `2gram.lm` using interpolated absolute discounting (D=0.5). The discounting term (D) must be specified for each n-gram order separately. For order 1, we use zero, because we do not want to smooth 1-gram estimates. The generated model file is in a text format, so you can take a look at it. It is a standard ARPA format for n-gram models.

Now we have generated an n-gram model and we can use the `ngram` tool to compute probabilities for new sentences. Create a file `test.txt` that contains the following sentences:

```
<s> it is in the bag </s>
<s> it is in the box </s>
<s> it is </s>
```

The probabilities can be computed as follows. Note that the tool outputs log-probabilities (base 10) since the probabilities are often very small. To transform log probabilities back to the regular probabilities you need to raise 10 to the logprob power.

```
ngram -lm 2gram.lm -ppl test.txt -debug 1
```

If you use the flag `-debug 2`, you see the probabilities for each n-gram separately. You can check that P(in | is) and P(</s> | is) match with what you computed in Question 3.

Question 4

***a)** What are the log-probabilities of the above sentences?*

***b)** Which sentence is the most probable one according to the model?*

***c)** Give an example of a sentence (non-empty, no out-of-vocabulary words) whose probability is even higher than any of the above.*

N-gram models can also be used to generate random sentences according to probabilities defined by the model. Try generating 10 random sentences using the above model. Use again `ngram` tool with options `-gen` and `-lm`. The `-gen` option specifies the number of sentences to be generated.

Real data preparation

The language model files we are going to create in the last part of the exercise can be quite large (around 100-200 megabytes). If you don't have enough space in your home directory (check with `quota`) you can also run the experiments in a temporary directory under `/tmp`. Note, however, that the `/tmp` is a local directory, you can access the files only on the same workstation. Also, the files are not backuped and the system may remove old files automatically during night. If you need to use the temporary directory, write the following:

```
mkdir -p /tmp/$USER
cd /tmp/$USER
```

Real data

Next we create real Finnish n-gram models using a large text corpus of 1.5 million sentences. The training data is in a compressed file `/work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz`. Take a look at the training data (press q to quit zless):

```
zless /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz
```

We are going to make a language model that allows the most common 60000 words from the corpus. Use `ngram-count` to compute the 1-gram counts from the corpus, and write the counts in a file using `-write countsfile` option.

Take a look at the counts file you created. Let's sort the words by the counts and select only the 60000 most common words (the corpus actually contains over 800000 different word forms).

```
sort -n -r -k 2 countsfile | head -n 60000 | cut -f 1 > 60000.words
```

Create a 2-gram model using Kneser-Ney smoothing, including only the most common 60000 words:

```
ngram-count -order 2 -vocab 60000.words -kndiscount1 -kndiscount2 \
  -interpolate -text /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz \
  -lm 2gram.lm.gz
```

Create also a 1-gram model and a 3-gram model similarly (use different filenames for the models). When training a 1-gram model, omit the `-kndiscountX` flags altogether, since we can not smooth 1-gram models. When training a 3-gram, you need to use all `-kndiscount1 -kndiscount2 -kndiscount3` flags to enable Kneser-Ney smoothing for all orders. You can ignore the "warning: no singleton counts" for 1-gram training.

Question 5

The file `/work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz` contains 10000 sentences that are not included in the training data. This test data can be used for evaluating the models that you just created. Use the `ngram` tool to compute the log-probability of the test data for each model (omit the -debug flag to avoid excess output).

***a)** Which of the models gave the best probability for the test data?*

***b)** What is the proportion of out-of-vocabulary (OOV) words in the test data (the ngram tool prints the relevant information for this)?*

Morph n-gram model

A vocabulary of 60000 most common words is usually enough for languages such as English, but for Finnish it does not cover the language very well. Remember that there were over 800000 distinct words in the training data. On the other hand, increasing the vocabulary size radically can be cumbersome for speech recognition algorithms.

One solution is to split the words in the training data into shorter units and build an n-gram model over those units. One can for example use the Morfessor algorithm, which splits words into morpheme-like units (morphs) that can be efficiently modeled. The file `/work/courses/T/S/89/5150/general/data/stt/stt.train.mrf.utf8.gz` contains the training data split into morphs by using Morfessor. Take a look at the file using `zless`. Note that the end of each word is marked with an underscore character '_').

Train 1-gram, 2-gram and 3-gram morph models similarly as we trained word models. Just use the morphed training data, and omit the `-vocab 60000.words` flag. You can ignore the "warning: discount coeff 1 is out of range: -0" for 1-gram training.

Question 6

*Compute the log-probabilities for the morphed test data as above using the morph 1-gram, 2-gram and 3-gram models. Note that you must use the **morphed** version of the test data: `/work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz`. Otherwise the units of the model and the data do not match and you get lots of OOV words.*

***a)** Which of the models is the best one?*

***b)** What was now the number of OOV morphs (the tool talks about words since it knows nothing about morphs)?*

Note about comparing word and morph models

Note that you got much lower (i.e. worse) log-probability for the test data when you used a morph model instead of a word model. Strictly speaking, the word model should give zero probability (log-probability minus infinity) for the test data, since the word model can not generate the OOV words. However, the `ngram` tool skips the OOV words and computes the probability only for the words that are in the vocabulary. That corresponds to predicting each OOV with a probability of one. The morph model, on the other hand, is able to give positive probability for all the words and thus shows lower probability for the test data. But the morph model generally works better with (Finnish) speech recognition, since it can generate much larger set of words.

If you want, you can try generating random sentences using the word and morph models that you have created. You may note that the morph model is able to generate even words that are not in the training data, while the word model can only create words that are in the vocabulary.