Homework 6 Language Modeling

Instructions

Answer the questions and upload your answers to courseville. Answers can be in Thai or English. Answers can be either typed or handwritten and scanned. No need to submit your code. Many parts of this homework is **optional**. **No points for any optional parts.**

Toy Example

Consider the following data. Each line is a separate sentence.

เช้า ฟาด ผัด ฟัก
เย็น ฟาด ฟัก ผัด
เที่ยง ผัด ฟัก
เย็น ฟาด ผัด
ผัด เช้า ฟาด เที่ยง

- Compute the unigram and bigram using the Training data. Don't forget the sentence start and end.
- Compute the probability of the following test utterance using the bigram model.

  เช้า ฟาด ผัด เช้า

- The following sentence gives a probability of 0 because some bigram sequences do not exist. Instead, we can compute the probability by using an interpolated model between a bigram and a unigram. Compute the probabiliy of the sentence. Use an interpolation weight of 0.5. Also, use the same interpolated model for the previous sentence.

  เที่ยง ผัด เช้า ฟาด

SRILM

Given a corpus of text training data in the target domain, the goal of statistical language modeling is to learn a model that can provide a probability for arbitrary word sequences that one might encounter in that domain. In other words, the model can provide a probability for a word sequence that did not appear explicitly in the training data. Note that special care must be taken, however, if a word does not appear in the training set. This is known as the out-of-vocabulary problem.

In the remainder of this asssignment, we will explore various approaches and techniques to language modeling using the SRI Language Modeling Toolkit (SRILM), an open source toolkit for building and applying statistical language models, developed at the SRI Speech Technology and Research Laboratory. You can see the documentation at `http://www.speech.sri.com/projects/srilm/manpages/`

Docker

To run SRILM, we will use Docker, an open source software container. Install Docker in your system by following the instructions at `https://www.docker.com/`

The benefit of Docker is that you can load in pre-built image files that contains the application you want to use or deploy, reducing the overhead in creating such systems over multiple machines.

Here, we will download an image that contains Kaldi and SRILM. Kaldi is a state-of-the-art open source ASR toolkit. We will use Kaldi as our main tool in the project.

Download the image using the command. Note, the image is large (4G).

```
docker pull dtjones/speech-recognition
```

We can create a container (an instance) of the image by

```
docker run -it --name <local unique container name> dtjones/speech-recognition
```

The -it starts an interactive session. You are now operating in the instance, which is running a Debian distribution. To re-attach to the same instance after you closed it, do

```
docker start -i <local unique container name>
```

To delete the instance

```
docker rm <local unique container name>
```

Download the dataset from courseville. Extract it to a directory. We can mount the dataset to the instance by

```
docker run -it -v <path to HW6>/HW6:/mnt/HW6 --name <name> dtjones/speech-recognition
```

Verify the mounting by running in the instance

```
ls /mnt/HW6
```

In the instance, Kaldi is installed in /kaldi, and SRILM is installed in /kaldi/tools/srilm . To add Kaldi and SRILM executables to the PATH environment variable so that it can be called easily, do

```
cd /kaldi/egs/wsj/s5
source ./path.sh
```

Verify that you can call an SRILM command

```
sh-4.3# ngram
need at least an -lm file specified
```

An n-gram language model predicts a word based on the previous $n - 1$ words. To estimate such a model from a training corpus using SRILM, run:

```
ngram-count -text <dataset> -order <n> -lm <lm>
```

By default, `ngram-count` applies Good-Turing discounting to smooth the probability estimates. The resulting language model file is stored in the ARPA LM format[1], which lists the log probability for each n-gram and an optional backoff weight. To evaluate the language model on a test corpus, run:

```
ngram -order <n> -lm <lm> -ppl <dataset>
```

`ngram` reports summary statistics on the dataset and computes the overall log probability log-prob and word perplexity ppl. The order argument specifies the maximal n-gram order to apply, by default 3. It allows a single model to be evaluated with various orders n. To evaluate models with order higher than 3, this parameter must be set explicitly.

In addition to summary statistics and overall perplexity, when computing the perplexity using `ngram`, we can request detailed analysis similar to by specifying the `-debug 2` argument.

```
ngram -order <n> -lm <lm> -ppl <dataset> -debug 2
```

Create a trigram language model called `text.lm` from `flights.train`. Evaluate it on `goodbye.txt` with `-debug 2`. You should see

```
good bye
p( good | <s> )  = [2gram] 0.0314114 [ -1.50291 ]
p( bye | good ...)  = [3gram] 0.986842 [ -0.00575233 ]
p( </s> | bye ...)  = [3gram] 0.99375 [ -0.00272286 ]
1 sentences, 2 words, 0 OOVs
0 zeroprobs, logprob= -1.51139 ppl= 3.19004 ppl1= 5.69762
```

The probability of the sentence "good bye" is the product of the probabilities of the two words and the end of sentence marker </s>. To avoid arithmetic underflow, we sum up the log probabilities in [ ] to obtain the overall sentence log probability of -1.51139. To compute the perplexity, we normalize the log probability[2] by the number of terms, or $ppl = 10^{logprob/N}$. In this example, $N = 3$ since we include the end of sentence marker in the computation of the overall probability.

[1] `http://www.speech.sri.com/projects/srilm/manpages/ngram-format.html`

[2] SRLIM uses base 10

- In this section we will work with orthographic transcriptions of utterances from the air travel domain. Train a trigram model `flights.lm` from the training corpus flights.train. Compute the perplexity of the model on both the training set and development set flights.dev. Plot the training and development set perplexities as a function of the n-gram order from 1 to 10. (It is sufficient to train a single 10-gram model and evaluate it multiple times with different orders n.) Explain the trends you observe. Which order achieves the best performance on the development set?

- You can turn off Good Turing smoothing by using the `-addsmooth 0` option. Compare the models with and without smoothing. Which has better perplexity? Note that SRILM excludes 0 probability words from the computation (or the number would be infinity!). You can see how many 0 probability words are there by looking at the zeroprobs number.

Vocabulary

Evaluate the development set using a trigram model and examine the detailed analysis (using -debug 2 on the penultimate sentence "`long island mcarthur airport`". Notice that both `island` and `mcarthur` are labeled as out-of-vocabulary (OOV) words. Thus, the probability for both words is 0. Since having a 0 probability for any word results in an overall probability of 0 for the entire corpus, SRILM skips over these words when computing the perplexity and instead reports the total number of encountered OOV words.

One approach to avoid OOV words is to train the language model using the combined vocabulary flights.vocab extracted from the training and development sets.

```
cat flights.train flights.dev > flights.all
ngram-count -write-vocab flights.vocab -text flights.all
```

- Build a trigram `flights.lm2` on the training corpus using vocabulary flights.vocab (use `-vocab`). Note that by explicitly specifying a vocabulary file, OOV words from the dataset will be replaced with the unknown word token <unk>. By default, n-grams containing <unk>are discarded. Compare the individual n-gram probabilities estimated by `flights.lm2` and `flights.lm` For what types of n-grams do the probabilities increase? Decrease? Why?

- Compute the development set perplexity using `flights.lm2`. How does it compare with the perplexity computed using `flights.lm` trained with only the training set vocabulary? Is this a meaningful comparison? Why or why not?

Kneser-Ney Discounting

- In this task, we will study the Kneser-Ney discounting method. To discount only the unigrams, run:

```
ngram-count -vocab <vocab> -text <dataset> -order <n> -ukndiscount1 -lm <lm>
```

Compute the change in development set perplexity of the trigram model when we discount only the unigrams, bigrams, or trigrams. Compute the change in perplexity when we discount all n-gram orders (`-ukndiscount1 -ukndiscount2 -ukndiscount3`). Are the perplexity reductions for discounting each order of a trigram model additive?

Class N-gram language Models

In situations where the training data is sparse for individual words, we may want to map sets of words down to equivalence classes to reduce the number of parameters that have to be estimated. Classes can be created manually or automatically.

For example, consider the following training data

```
fly to Thailand
fly from Australia
fly to Japan
Australia is hot
```

Without any smoothing, the trigram "`Fly to Australia`" would have a probability of 0. However, if we realize that Thailand, Australia, and Japan are countries. We can replace the countries with a class, COUNTRY.

```
fly to COUNTRY
fly from COUNTRY
fly to COUNTRY
COUNTRY is hot
```

Now the trigram "`Fly to Australia`" would have a non-zero probability,

$$P(Australia \mid Fly\ to) = P(COUNTRY \mid Fly\ to)P(Australia \mid COUNTRY)$$

where

$$P(Australia \mid COUNTRY) = \frac{c(Australia)}{c(COUNTRY)} = \frac{2}{4}$$

To train a class n-gram model, we need to first replace the words in training corpus with their respective class labels. We have provided the script build_class_ngram to simplify the training process.

- Look at `flights.manual.classes` for examples of human-generated classes. Build a class trigram `flights.mclass.lm` from the training corpus with the manual classes and `flights.vocab`. Compare the perplexity with other models. You can build the class n-gram by

```
./build_class_ngram flights.manual.classes flights.train 3 \
  flights.vocab flights.mclass.lm
```

Generating the classes can be a long process, SRILM provides a way to learn the classes automatically from data. Try

```
mkdir classes
ngram-class -text flights.train -numclasses 1 \
  -classes classes/flights.classes -full -save 10
```

Interpolation

Oftentimes, we can obtain improved performance by linearly interpolating class n-gram with other n-gram models trained on the same data. To compute the perplexity and optionally build the interpolated mixture model, run:

```
ngram -lm <lm0> -mix-lm <lm1> -lambda <lambda0> \
  -ppl <dataset> [-write-lm <mix-lm>] 2> /dev/null
```

- Interpolate the class n-gram with the model trained using KN discounting, and plot the perplexity of the mixture model as a function of the interpolation weight `lambda0`. What is the optimal choice of `lambda0`?

THE FOLLOWING SECTIONS ARE OPTIONAL. NO POINTS GIVEN.

GOWAJEE APP LM

- Look at `gowajee.train, gowajee.dev, gowajee.test`. These are data collected from all groups combined. Use the techniques you learned, try to minimize the perplexity.