King Fahd University of Petroleum & Minerals
Information and Computer Science Department

ICS 472– Natural Language Processing
Second Semester 2021/2022 (222)

Assignment #2:
N-gram Language Models

Overview

In this assignment, you will practice the creation and parameter setting of n-gram language models (LM) in a number of languages. You will compare different LM parameters including tokenization techniques and study how they affect the performance of language models on in-domain and out-of-domain texts. Finally, you will build an n-gram based language identifier that distinguishes among 22 European languages.

Resources

A. Contents of the zipped folder

- Data sets
  - The **English-Mix** subdirectory has a number of files from different genres in English only:
    - UNCorpus.train — United Nations Corpus text for **training** LMs
    - UNCorpus.test — United Nations Corpus text for **testing** LMs
    - wiki.test — Wikipedia Corpus portion for **testing** LMs
    - Fair.test — My Fair Lady transcript for **testing** LMs
  - The **Europarl** subdirectory has files from the 22 languages of the European Parliament organized in three directories
    - train/train.* — Training data

      The files all have a two letter extension signifying the **label** of the language.

      We organize them below by language family.

      Baltic Family*: Lithuanian (lt), Latvian (lv)*

      Finno-Ugric Family*: Hungarian (hu), Finnish (fi), Estonian (et)*

***Germanic Family**: English (en), German (de), Danish (da), Dutch (nl), Swedish (sv)*

***Greek Family**: Greek (el)*

***Romance Family**: Romanian (ro), Portuguese (pt), Italian (it), Spanish (es), French (fr)*

***Semitic Family**: Maltese (mt)*

***Slavic Family**: Slovak (sk), Czech (cs), Polish (pl), Bulgarian (bg), Slovene (sl)*

- dev/dev.{1-22} — Development data
- dev/dev.gold — A file indicating the language of the dev.{1-22} file
- test/test.{1-15} — Blind test data. The gold answer file is ***not*** provided.

- The **Tools** subdirectory
  - Porter Stemmer is provided in the script ***porter.pl***
  - A simple tokenizer and detokenizer are provided in the scripts ***tokenizer.pl*** and ***detokenizer.pl*** (from the Moses Project) -- They use the file ***nonbreaking_prefix.en.***

- Shell script for Task1 (task1.sh)
  - A bash shell that runs the commands for Task1.
  - This is provided to get you introduced to doing shell scripts.
  - Read up on how to write shell scripts. One resource is here: https://flaviocopes.com/bash-scripting/

- **answers.txt** is the template you should use to provide the answers to all the questions in the assignment.

B. Tools to install

- SRILM Toolkit
  - Download and install the SRILM toolkit from here: http://www.speech.sri.com/projects/srilm/ ○ Follow the directions in INSTALL.
  - SRILM is very powerful and versatile. Make sure to look through its manual pages: http://www.speech.sri.com/projects/srilm/manpages/
  - You will need to learn about using the commands ngram-count, ngram, and disambig. Some of these commands have 100+ parameters! The manual page for srilm-faq is helpful.

Note: KenLM is another alternative to SRILM, in case you find difficulties installing SRILM.

- BPE Tokenizer
  - We recommend you use Rico Sennrich's system https://github.com/rsennrich/subword-nmt
  - Follow the instructions in the link above on how to install and use.
  - You will primarily need to use the learn_bpe and apply_bpe modes.
  - Use "-h" to get more details on usage.

Submission Instructions (5 points)

Your answers to this assignment should follow these instructions (**each is 1 points**).

1. The zipped file name is <last_name>-<kfupmid>-nlp-assignment-2.zip
   - All letters in the filename should be lower case.
   - Example: ahmad-2XXXX-nlp-assignment-2.zip
2. Directory name should be the same as the file name without ".zip"
   - Example: ahmad-7XXXX-nlp-assignment-2/
3. In the abovementioned directory, provide all the answers to the questions in the Task 1, Task 2 and Task 3 in plain text file (not PDF, not RTF) with the name: **answers.txt**
4. Provide the code you created to do language identification for Task 3.
5. Provide the language identification prediction file for the test set with the name **test.pred.**

Upload the zipped file to the blackboard.

Task 1 : Let's LM! (10 points)

This task is a guided example of how to build a basic language model and evaluate it. You will create a LM for the UN Corpus and testing the perplexity of test text from the UN (in-domain) and the Wikipedia (out-of-domain).

First - Tokenize the texts

```
cd Tools/
perl tokenizer.pl -no-escape < ../English-Mix/UNCorpus.train >
../English-Mix/UNCorpus.train.tok
perl tokenizer.pl -no-escape < ../English-Mix/UNCorpus.test > ../English-Mix/UNCorpus.test.tok
perl tokenizer.pl -no-escape < ../English-Mix/Wiki.test > ../English-Mix/Wiki.test.tok

cd ..
```

Question 1.a (1 point) What is the effect of this tokenization step? Describe the changes in the text. Hint: check out *nonbreaking_prefix.en.*

Question 1.b (2 point) What is the difference between the raw and tokenized files in terms of the number of **tokens** and the number of **types** (unique tokens)? Explain the difference in the numbers given the used tokenization.

Second - Create the LM (*check out the SRILM manual page for ngram-count for parameters*)

```
ngram-count -text English-Mix/UNCorpus.train.tok -order 3 -lm
English-Mix/UNCorpus.train.tok.3.lm -addsmooth 0.01
```

For the next three questions, check out the SRILM manual page for ngram-format.

Question 1.c (1 point) What is the total number of 1-gram, 2-gram and 3-gram entries?

Question 1.d (1 point) What is the purpose of each column in the 1-grams data?

Question 1.e (1 point) Why is there no third column in the 3-grams portion?

Third - Compute Perplexity (checkout the SRILM manual page for ngram for parameters)

```
ngram -lm English-Mix/UNCorpus.train.tok.3.lm -order 3 -ppl English-Mix/UNCorpus.test.tok file
English-Mix/UNCorpus.test.tok: 500 sentences, 19169 words, 16 OOVs 0 zeroprobs, logprob= -
25308.24 ppl= 19.39789 ppl1= 20.95907
ngram -lm English-Mix/UNCorpus.train.tok.3.lm -order 3 -ppl English-Mix/Wiki.test.tok
file English-Mix/Wiki.test.tok: 500 sentences, 13999 words, 3053 OOVs 0 zeroprobs,
logprob= -36285.73 ppl= 1479.676 ppl1= 2065.266
```

From the SRILM FAQ (with some paraphrasing):

By default any word not observed in the training data is considered OOV and OOV words are silently ignored by ***ngram*** *during perplexity (ppl) calculation. In the example above (with Wiki.test.tok):*

```
file English-Mix/Wiki.test.tok: 500 sentences, 13999 words, 3053 OOVs 0
zeroprobs, logprob= -36285.73 ppl= 1479.676 ppl1= 2065.266
```

There are `3053` *out-of-vocabulary words. This is the number of unknown word tokens, i.e. tokens that appear in* `Wiki.test.tok` *but not in* `UNCorpus.train.tok` *from which* `UNCorpus.train.tok.3.lm` *was generated.*

logprob (log probability) *is* `-36285.73`*. This is the total logprob ignoring the 3053 unknown word tokens. The logprob does include the probabilities assigned to </s> tokens which are introduced by ngram-count. Thus the total number of tokens which this logprob is based on is*

words - OOVs + sentences = 13999 - 3053 + 500

ppl (perplexity) is `1479.676`*. This is the geometric average of 1/probability of each token, i.e., perplexity. The exact expression is:*

ppl = 10^(-logprob / (words - OOVs + sentences))

ppl1 *is the average perplexity per word excluding the </s> tokens.*

Question 1.f (1 point) Why is the UN LM perplexity different for the Wikipedia and UN?

Question 1.g (1 point) Compute the OOV rate for the UN and Wikipedia test files? (% of OOVs/words).

Question 1.h (1 point) Why is the UN LM OOV rate different for the Wikipedia and UN?

Fourth - For fun, generate some random UN resolution texts!

```
ngram -lm English-Mix/UNCorpus.train.tok.3.lm -gen 1
```

```
62 increasing need for adequate of the Rome Statute World Conference on Women and ratifying
traditional knowledge and experience mine-action participate in the field ,
```

Question 1.i (1 point) Provide the text you randomly generated. How is its fluency? How is it in terms of coherence?

Task 2: How Many Ways to LM? (40 points)

In this task, you will compare a number of ways of building a LM. You will try the following variables.

1. **Tokenization (applied to train and test files)**
    a. **Raw**: the original text.
    b. **Simple Tokenization**: apply the simple tokenizer you used in Task 1. Give the files the extension **.tok**
    c. **Lower Case**: lower case the simple tokenization. Give the files the extension **.tok.lc**
    d. **Porter Stem**: apply Porter stemmer to the Lower Case tokenization. Give the files the extension **.tok.lc.port**
    e. **BPE**: apply BPE to the Lower Case tokenization. Give the files the extension **.tok.lc.bpe.** ***IMPORTANT*** **:** *(a) Use with the default setting for min-frequency and symbol count. And (b) only learn BPE using the train corpus.*
2. **Training size (applied to train file only)**
    a. 100% : 70,000 lines
    b. 50% : 35,000 lines
    c. 25% : 17,500 lines
    d. 12.5% : 8,750 lines
    e. 6.25% : 4,375 lines
3. **LM order**
    a. 1 (unigram)
    b. 2 (bigram)
    c. 3 (trigram) (default)
    d. 4 (4-gram)
    e. 5 (5-gram)
4. **Smoothing model**
    a. Laplace (add 1 smoothing): use -addsmooth (see ngram-count documentation)
    b. Add-k smoothing (k=0.1): use -addsmooth (see ngram-count documentation)
    c. Good-Turing (*default; do not worry about warning messages*)
    d. Kneser-Ney: use -kndiscount

Metrics: You will evaluate the LMs produced using two metrics: OOV rate and Perplexity.

Domain Settings: You will compare performance of training data (UN Corpus) on one in-domain (UNCorpus.test) and two out-of-domain settings (Wiki.test and Fair.test).

These are 625 settings for a language model. And 1,875 results (3 x 625). We will only look at a subset of these to determine patterns.

First - For all of the training data (size 100% only), for order 3 and Kneser-Ney smoothing, compare all five tokenziations in terms of OOV and Perplexity on in-domain and out of domain data. Make sure to train and evaluate on the same tokenization.

Question 2.1.a (4 points). Fill in the below table.

	Words	OOV	OOV%	ppl
UNCorpus.test:				
UNCorpus.test.tok:				
UNCorpus.test.tok.lc:				
UNCorpus.test.tok.lc.port:				
UNCorpus.test.tok.lc.bpe:				
Wiki.test:				
Wiki.test.tok:				
Wiki.test.tok.lc:				
Wiki.test.tok.lc.port:				
Wiki.test.tok.lc.bpe:				
Fair.test:				
Fair.test.tok:				
Fair.test.tok.lc:				
Fair.test.tok.lc.port:				
Fair.test.tok.lc.bpe:				

Question 2.1.b (2 points). What can you say about the interaction between tokenizations and OOV rate for in-domain and out-of-domain cases?

Question 2.1.c (2 points). What can you say about the interaction between tokenizations and perplexity?

Question 2.1.d (2 points). What do the results above suggest about the similarity between the Wikipedia and UN, vs My Fair Lady and the UN?

Second - For **tok.lc** (lower case tokenization), order 3 and Kneser-Ney smoothing, compare all five training data sizes in terms of OOV and Perplexity on in-domain and out of domain data. Make sure to train and evaluate on the same tokenization.

Question 2.2.a (4 points). Fill in the below table.

	Train Size	Words	OOV	OOV%	ppl
UNCorpus.test	70000				
UNCorpus.test	35000				
UNCorpus.test	17500				
UNCorpus.test	8750				
UNCorpus.test	4375				
Wiki.test	70000				
Wiki.test	35000				
Wiki.test	17500				
Wiki.test	8750				
Wiki.test	4375				
Fair.test	70000				
Fair.test	35000				
Fair.test	17500				

Fair.test	8750				
Fair.test	4375				

Question 2.2.b (2 points). What can you say about the relationship between the training size and OOV rate?

Question 2.2.c (2 points). What can you say about the relationship between the training size and perplexity for in domain data?

Question 2.2.d (2 points). What can you say about the relationship between the training size and perplexity for out-of-domain data? Is it similar or different to in-domain data?

Third - For **tok.lc** (lower case tokenization), all the training data (100%) and Kneser-Ney smoothing, compare all five orders 1 to 5. Make sure to train and evaluate on the same tokenization and in the same order (use -order in the ngram command to override the default 3).

Question 2.3.a (4 points). Fill in the below table.

		Words	OOV	OOV%	ppl
UNCorpus.test	Order 1.lm				
UNCorpus.test	Order 2.lm				
UNCorpus.test	Order 3.lm				
UNCorpus.test	Order 4.lm				
UNCorpus.test	Order 5.lm				
##					
Wiki.test	Order 1.lm				
Wiki.test	Order 2.lm				
Wiki.test	Order 3.lm				
Wiki.test	Order 4.lm				
Wiki.test	Order 5.lm				
###					
Fair.test	Order 1.lm				
Fair.test	Order 2.lm				

Fair.test	Order 3.lm				
Fair.test	Order 4.lm				
Fair.test	Order 5.lm				

Question 2.3.b (2 points). What can you say about the relationship between the LM order and OOV rate?

Question 2.3.c (2 points). What can you say about the relationship between the LM order and perplexity for in domain data?

Question 2.3.d (2 points). What can you say about the relationship between the LM order and perplexity for out-of-domain data? Is it similar or different to in-domain data?

Fourth - For **tok.lc** (lower case tokenization), all the training data (100%) and order 5, compare the four different smoothing methods presented above. Make sure to train and evaluate on the same tokenization and in the same order (use -order in the ngram command to override the default 3).

Question 2.4.a (4 points). Fill in the below table.

	Smoothing	Words	OOV	OOV%	ppl
UNCorpus.test	Add 1				
UNCorpus.test	Add 0.1				
UNCorpus.test	Good-Turing				
UNCorpus.test	Kneser-Ney				
Wiki.test	Add 1				
Wiki.test	Add 0.1				
Wiki.test	Good-Turing				
Wiki.test	Kneser-Ney				
Fair.test	Add 1				
Fair.test	Add 0.1				

Fair.test	Good-Turing				
Fair.test	Kneser-Ney				

Question 2.4.b (2 points). What can you say about the relationship between the smoothing method and perplexity for in domain data?

Question 2.4.c (2 points). What can you say about the relationship between the smoothing method and perplexity for out-of-domain data? Is it similar or different to in-domain data?

Question 2.4.d (2 points). What is the best smoothing method overall?

Task 3: Guess The Language! (40 points)

Using the **Europarl/train** files, build multiple language models and use them to identify the language of the files in the **Europarl/dev** files. The correct labels for the **Europarl/dev** are in **Europarl/dev.gold.** A common method for doing this task is to use the perplexity of a text against all language models, and select the language of the model with lowest perplexity. *Hint: The modeling is done at the character level.*

Question 3.a (20 point) Write code (in python) that (a) trains, (b) identifies the languages in a test directory, and (c) evaluates its predictions against a gold reference file. There are three modes that take the following arguments:

```
Usage: identify-language.py TRAIN <train-dir>/<train-base> <lang-id-model-dir>
Usage: identify-language.py PREDICT <test-dir> <lang-id-model-dir> Usage:
identify-language.py EVALUATE <gold> <prediction>
```

- The code is expected to know the list of language id labels (it,lt,et,fr,hu,lv,cs,en,da,de,mt,nl,pl, fi,pt,ro,sk,sl,sv,bg,el,es).
- The training files all have the name **< train-base>.<label>**. (<label> is the two-character language id)
- The test files all have a digit extension (e.g., dev.12, test.3, whatever.66) ● The PREDICT mode will output the prediction file to STDIN.
- Provide the code you created and the accuracy result you produce.

You may find the following notes helpful to this task.

(a) To use srilm from within your code, you will need to make system commands. in python, use **os.system()** . Example:

```
>>> import os
>>> cmd = "ls"
>>> os.system(cmd)
Applications                Movies
Desktop                     Music
Documents                   Pictures
Downloads                   Public
Dropbox                     Untitled
Google Drive                nltk_data
Google Drive File Stream    perl5
Library                     wekafiles
```

(b) To use perl regex with a number of languages other than English, you will need to use the following instruction to work with unicode:

perl -pe 'use open \":std\",\":encoding(UTF-8)\"; **< YOUR REGEX GOES HERE>;**'

The three basic ways you will use the code are the following:

(1) To train on the train data and create a model directory that houses all the .lm files:

```
python identify-language.py TRAIN Europarl/train/train modelxid
```

(2) To predict the answers for the dev set and print them to STDIN/file

```
python identify-language.py PREDICT Europarl/dev modelxid > dev.predict
```

(3) To evaluate the predictions against the gold

```
python identify-language.py EVALUATE Europarl/dev/dev.gold dev.predict
```

Using all the data in training and predict using the full dev files, you should be able to reach 100% accuracy on the dev set.

Question 3.b (5 points) Modify your code to train with one line only from training, and to use one line only from the dev files, and to use order 1 for lm.

Run your system using this training data and report the results on the dev set. (1 point)

- It is reasonable to expect the accuracy to go down. Is there a pattern to the errors? (4 points)

Question 3.c (15 points) Using your best model you have, identify the languages of the provided test set (**Europarl/test**) . Provide your answer in a file named "**test.pred**". The file should consist of two tab separated columns marking for each file test.<n>, its two-character language id.

test.1<tab><lang>
test.2<tab><lang>
...
test.15<tab><lang>

The format should be comparable to **Europarl/dev.gold**.

The points for this question will be based on how many labels your system assigns correctly.