# CMPUT 650 Assignment 3: Language Identification

**Saeed Najafi**
Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
snajafi@ualberta.ca

**Aaron Derakhshan**
Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
aderakhs@ualberta.ca

## 1 Introduction

### 1.1 Task Description

In this assignment, we aim at training a character-level language model for different languages and use it to identify language of an unknown text. The intuition behind this approach is that a language model trained on the language X tends to give higher probabilities (lower perplexity) for a text written in X than any other languages. The training data that we use is part of the Universal Declaration of Human Rights (UDHR) which are in a variety of languages.

### 1.2 Implementation

We implement[1] a character-level n-gram language model using python programming language (without available libraries such as NLTK[2]). N is the size of n-gram that is the only parameter in our system. Since we are at character level, we treat each word as a sentence that we used to have in word-level language models. We start by detecting n-grams of each word and creating a dictionary of them for each N. For example if N=3, in the word 'Word', we detect the following trigrams: '< < W', '< W o', 'W o r', 'o r d' and 'r d >' (< and > represent start and end symbols respectively).

To handle unseen n-grams, we implement two smoothing methods: Laplace (Add-One) and Good-Turing. In the un-smoothed method, when we encounter an unseen n-gram, which is not in our dictionary, instead of returning log(zero), we return log(0.00000000000000000001). To calculate probabilities of a seen n-gram, we use Maximum Likelihood Estimation method. For example to calculate P(o|W), we count occurrences of seeing 'Wo' and divide it by by the number of times we see 'W' with any other characters.

$$P(o|W) = \frac{Count(`Wo')}{Count(`W')}$$

[1]https://github.com/SaeedNajafi/Language-Identification
[2]http://www.nltk.org/

## 2 Smoothing methods

### 2.1 Add-one

In add-one smoothing, we assume that we have seen every possible n-gram at least once. Therefor to calculate P(o|W), we use the following formula.

$$P(o|W) = \frac{Count(`Wo') \quad + \quad 1}{Count(`W') \quad + \quad character\_set\_size}$$

### 2.2 Good-Turing

In Good-Turing smoothing approach (Gale, 1995), we start by finding the maximum number of times (max_r) that an n-gram (with size N) has been seen in the training corpus. For example when max_r is 10, we have at least one n-gram that has been occurred 10 times and there aren't any n-grams occurred more than 10 times. We define N(r) as the number of different n-grams that have occurred r times in the training corpus. We have R={0, 1, ... , max_r} and N(0), N(1), ... , N(max_r). T is the total number of n-grams and N(0) is the number of all unseen n-grams. T is calculated from R using the following formula:

$$T = \sum_{r=0}^{max\_r} N(r) * r$$

Good-Turing estimates probability of an n-gram which has occurred r times in the corpus with the following formula:

$$E(r) = \frac{(r+1)*N(r+1)}{T*N(r)}$$

For example to calculate P(o|W), we have,

$$P(o|W) = \frac{P(`Wo')}{P(`W')} = \frac{E(Count(`Wo'))}{E(Count(`W'))}$$

It is very important to smooth N(r) in Good-Turing method, especially for large values of r. N(r) can even be 0 so it cannot be used in the estimation. One of the suggested approaches for smoothing N(r) in Good-Turing is to fit a power law function ($a * c^b$) to the N(r) – r diagram. We implement this approach and fit a power law function using Scipy[3] and Numpy[4] libraries.

## 3 Results

To find the best N value for each smoothing method and language, we run our system for every language in the training data, and calculate the perplexity over development set languages. We select values of N for a language and a smoothing method which can result in the lowest perplexity. In the test step, to identify language of an unknown text using each smoothing method, we run our language model with its best N value for every language. To calculate perplexity of a text, we find log perplexity of each word separately and then, we sum log perplexities and after negating this sum, we compute the average. So, the perplexity of a text is the negative average of log perplexities of its words. The best N value for all languages and different smoothing methods are Add-one 3 and Good-Turing 3. We only tested up to 7-gram language models. The results of language identification over development set is presented in table 1

Table 1: Accuracy over Dev set

| Method | #Correct | #Incorrect | Acc |
|---|---|---|---|
| No-Smoothing | 53 | 2 | 96.36% |
| Add-one | 55 | 0 | 100% |
| Good-Turing | 53 | 2 | 96.36% |

Good-Turing made mistakes on two languages and selected their other families “udhr-hat-popular” instead of “udhr-hat-kreyol” and “udhr-deu-1996” instead of “udhr-deu-1901”. However, No-Smoothing method mislabeled two different languages (“udhr-quy” for “udhr-qxu” and “udhr-zul” for “udhr-nbl”).

The main reason for getting high accuracy on development set is that the length of text files that we have for each language is long enough for a character language model to detect the language of the text perfectly. If we shorten development set files to only one sentence, then the accuracy of language identification will decrease as depicted in table 2 (we only shortened test files but training and tuning were done on all sentences).

[3]https://www.scipy.org/

[4]http://www.numpy.org/

Table 2: Accuracy over shorter Dev set

| Method | #Correct | #Incorrect | Acc |
|---|---|---|---|
| No-Smoothing | 52 | 3 | 94.54% |
| Add-one | 52 | 3 | 94.54% |
| Good-Turing | 49 | 6 | 89.09% |

## References

William A. Gale. 1995. Good-turing smoothing without tears. *Journal of Quantitative Linguistics*, 2.