Identifying and Comparing Languages with Language Modeling and Bytepair Encoding

1st Simon du Toit
Applied Mathematics Department
Stellenbosch University

***Abstract*—In this paper I identify and compare five different languages. This is done using two simple natural language tools: N-gram modeling and byte-pair encoding. A functional language identification model is constructed using a set of character-level trigram language models. Both the language models and byte-pair encoding imply a strong similarity between some of the languages. The results indicate that even these basic tools are able to extract insights from different languages.**

I. Introduction

I implement a character-level trigram language model and a byte-pair encoding algorithm for the task of analysing and identifying text from five different languages. I first describe the data I use and how it is preprocessed. I then give an explanation of how each tool is designed and implemented for this problem. I evaluate the language models by generating sample text and measuring their perplexity on validation texts. Language identification is evaluated on a test set and byte-pair encoding is used to compare each pair of languages. I show the results and discuss their implications.

II. Design

A. Data Processing

The training data consists of one text file for each of the five languages: English, Afrikaans, Dutch, isiXhosa and isiZulu. Each line in a file contains a paragraph. I obtain a list of individual sentences by splitting the paragraphs on full stops, exclamation marks and question marks which are followed by a space.

First it is necessary to normalize the text. I turn the remaining full stops, which appear in acronyms and abbreviations, into spaces. I remove diacritics by performing canonical decomposition on the text, which converts each character to a standard canonically equivalent form. Every character is made lowercase and every digit is converted to zero. I remove all characters that are not a letter, space or zero digit. Finally, I strip away leading and trailing spaces from each sentence.

The model uses start-of-sentence tokens and end-of-sentence tokens to be able to start and finish generating a sentence. Since this is a trigram model, I append two start-of-sentence tokens to the beginning of each sentence and one end-of-sentence token to the end of each sentence.

The validation data is already split into sentences and mostly normalized. All that is needed is to remove newline characters at the end of each sentence. The language identification test data is similar, though each sentence is of a different language and starts with a corresponding label. The labels are separated from the text in processing.

B. Language Modeling

The language model is a character-level trigram model. The model is stored as two lookup tables, one indexed by every possible trigram and the other indexed by every possible bigram. The values in the tables correspond to the frequency with which each trigram or bigram appears in the training set.

To account for sparsity in the training data, absolute discounting is used to smooth the model. I first construct the trigram table and subtract α from the value of each non-zero entry, where α is a control parameter of the model. The accumulated α's are then uniformly distributed to the zero entries. The total sum of values in the table thus remains constant. Finally, I construct the bigram table from the smoothed trigram table by summing the table values for every trigram that begins with a given bigram.

The model can be used to generate new text and to analyse the perplexity on given text. These functionalities are implemented to evaluate the model and derive insights from the languages. Text is generated one character at a time. To generate a character, the model samples from all the trigrams which start with the two preceding generated characters. The sampling probabilities are the trigram table values divided by the bigram table value for the two preceding characters. The third character of the sampled trigram is generated. This gives a probability distribution for $P(c_t|c_{t-1}c_{t-2})$, where c_t is the character generated at time step t. The model always begins by generating two start-of-sentence tokens and stops once it generates an end-of-sentence token.

Perplexity is used to measure how likely the model is to generate a specific piece of text. Perplexity is calculated as:

$$\mathrm{PP} = P_\theta(c_{1;T})^{-\frac{1}{T}}$$

Here θ are the model parameters (in our case, the table values) and T is the number of characters in the sentence. To avoid numerical instability from multiplying many probabilities, the log form is implemented:

$$\mathrm{PP} = 2^{-\frac{1}{T}\log(P_\theta(c_{1;T}))}$$

The log probability of the model generating a given sentence is calculated by summing the log probabilities of every character:

$$\log(P_\theta(c_{1;T})) = \sum_{t=3}^{T} \log(P_\theta(c_t|c_{t-1}c_{t-2}))$$

Here it is assumed that c_1 and c_2 are start-of-sentence tokens and c_T is the end-of-sentence token. For multiple sentences, the log probabilities and Ts are accumulated before computing the overall perplexity.

C. Language Identification

The language model can be used to perform language identification. A language identification model is constructed by training a separate language model on each language. The language of a sentence can be predicted by selecting the language of the model which has the lowest perplexity on the sentence. This corresponds to the modelled language which is most likely to produce the given sentence.

D. Byte-pair Encoding

Byte-pair encoding is used to create subword units out of a given text. I perform byte-pair encoding by first dividing sentences into individual characters as the smallest unit. Subword units are then iteratively constructed by selecting the most frequent pair of adjacent units and merging them into a single unit. I create and store the first 100 merged subword units in this way.

Byte-pair encoding is used to compare the similarity between languages. Two languages can be compared by creating the subword units for each and measuring how many subword units are common across both languages. Languages with more subword units in common are more similar.

III. Implementation

The language models and byte-pair encoding are implemented in Python. Text normalization is performed using regular expressions and the `normalize` function from the default `unicodedata` library. Splitting the paragraphs into sentences produces some empty sentences, which are removed. The language model lookup tables are stored in dictionaries. The character '[' is used to represent the start-of-sentence token and ']' represents the end-of-sentence token.

The α smoothing parameter for a model is tuned on the validation data of the corresponding language. A search is performed over the values $0.1, 0.2, ...0.9$ and $alpha$ is chosen as the value which gives the minimum perplexity on the validation text. This results in $\alpha = 0.4$ for Afrikaans, Dutch and isiXhosa and $\alpha = 0.5$ for English and isiZulu.

IV. Results and Insights

A. Language Modeling

First I analyse my language model created from the English training data. Figure 1 shows the conditional probability of each trigram given the first two letters are "t" and "h". The trigram "the" is predictably the most common trigram by far, as this a common word in English. This is followed by a few common subwords. The lowest probabilities are for trigrams that do not appear in the training data, none of them would not make sense in standard English.

A sample sentence generated by each model is shown in Figure 2. Each sentence is gibberish, but gibberish which vaguely resembles text from the modelled language. The generated sentences of each model vary in length.

Trigram	Probability of third character given first two
the	0.68113
th	0.09821
tha	0.08313
thi	0.04468
tho	0.03815
thr	0.03384
thn	0.00455
ths	0.00407
th]	0.00293
thu	0.00232
thy	0.00192
thc	0.00192
thl	0.00091
thw	0.00064
thd	0.00057
thm	0.00051
tht	0.00024
thp	0.0001
thf	3e-05
thb	3e-05
thh	3e-05
thg	1e-05
thx	1e-05
thq	1e-05
thj	1e-05
th0	1e-05
thk	1e-05
thz	1e-05
thv	1e-05

Fig. 1. Probability of each character given the two preceding characters. Calculated by dividing trigram table values by bigram table value.

Figure 3 shows the perplexity of each language model on the validation data of each language. Each model has the lowest perplexity on the validation text of the language it is trained on, which indicates that the model is working. The perplexities allow us to compare the languages. A low perplexity between a model and text of a different language indicates that those two languages are similar. The table implies that Afrikaans is similar to Dutch and that isiXhosa is similar to isiZulu, which is expected given the common ancestries of these language pairs. English is also more similar to its Germanic cousins than to the African languages.

B. Language Identification

The language identification model achieves an accuracy of 91.6% on the test set. Most of the mistakes are due to classifying Afrikaans sentences as Dutch sentences and vice versa, or similarly being unable to distinguish between isiXhosa and isiZulu. This is likely due to the similarity between these

Language	Generated Sentence
Afrikaans	hul dikaarin die thet m met in digterterne eesie van die n das welingen int enhe geele uit diensetwee nolkenie vankeen duital.
English	hern tom thicanin samplect ton es isinity and therefir of bratelin 00 rations ano
Dutch	he het van werijn hiet algdelistotse lan wer heilagelfstaathosfeeslicest ammistreautect
isiXhosa	intshaka uphondiante nga kupha kwisela eziseka ke ukunga waba ngeegokufalanzifuth-elokuso okhathemnte ngasitshama
isiZulu	buso sealznzibhena kukukublivedisifuthala undlu yiswe kaka e ze

Fig. 2. Sample sentences generated by the model when trained on each language.

Model	Validation Language				
	Afrikaans	English	Dutch	isiXhosa	isiZulu
Afrikaans	7.84	17.55	10.55	36.48	39.29
English	19.49	7.57	19.35	41.11	44.45
Dutch	10.14	14.94	7.62	42.29	46.56
isiXhosa	26.13	16.11	26.04	8.1	8.84
isiZulu	27.62	18.64	27.14	9.34	8.37

Fig. 3. Perplexity of each language model on different language validation sets.

pairs of languages as observed in the perplexity measurements. Many of the missclassified sentences are very short, some consisting of a single word, which makes them more difficult to classify. This is exasperated by the presence of English loanwords in non-English languages.

C. Byte-pair Encoding

Languages compared	Number of common subword units
isiXhosa-isiZulu	81
Afrikaans-Dutch	71
English-Afrikaans	43
English-Dutch	39
English-isiXhosa	30
English-isiZulu	30
Afrikaans-isiXhosa	23
Afrikaans-isiZulu	23
Dutch-isiXhosa	18
Dutch-isiZulu	18

Fig. 4. Comparison of languages by measuring the number of common subword units. Language pairs are sorted in descending order of similarity. Byte-pair encoding is used to produce 100 subword units for each language.

Figure 4 shows the number of common subword units for each language pair, sorted in descending order. This metric implies that isiXhosa and isiZulu is the most similar language pair. This is followed by Dutch and Afrikaans, which is the only other pair where most of the subword units are common. This is the same conclusion as the one drawn from the language model perplexities.

V. Conclusion

N-gram language modeling and byte-pair encoding are successfully used to generate, identify and compare five different languages. The character-level trigram model functions as expected. The language model is successfully applied to the task of language identification. Analysing the perplexities of the models on the validation data of each language shows which languages are similar to each other: Afrikaans and Dutch, and isiXhosa and isiZulu. This is corroborated by the results of applying byte-pair encoding to the training sets and measuring the number of common subword units between languages. It is clear that these simple natural language processing tools are able to derive insights from different languages.