Word Generator

●●● Explanatory Note ●●●

Word generator is written by Andreas Shcherbakov as part of a project run by Nick Thieberger with funding from ARC Future Fellowship (FT140100214)

Melbourne, 2015-2017

1. Intent & Motivation

Word Generator is an application that serves to model candidate words that might belong to a given language based on phonotactic rules learnt from an existing sample dictionary. Its intended usage is in creating questionnaires for exploration in field linguistics. However, it also may be used in a wide scope of language modeling. As well, it should be helpful for hunting authentic brand names or fancy words.

2. Phonemes vs. Letters

Although the Generator accepts words written in Unicode alphabetic letters, the internal modeling is done in phonemes. By convention, a phoneme may be represented either as one letter or as a fixed sequence of letters. User should provide a list of all phonemes known to be present in the language (or at least some approximation). To pass such information to the Word Generator, special headers would precede the training dictionary file content (see *Appendix I* for the format description). Phonemes should be classified into vowels and consonants.

One may represent words in phonetic symbols if it is convenient, but it is not necessary. It’s quite ok to use normal orthographic spelling of words. If single phoneme may be represented by combination of characters, you may want to place proper phoneme classification line(s) into the training dictionary file.

3. Algorithm overview & parameters

The underlying algorithm [1] employs a probabilistic approach based on phoneme collocation statistics collected from the sample dictionary. It considers phoneme sequences of fixed length commonly referred as *N-Grams*. In contrast to approaches based on Machine Learning, this one generally uses collected N-grams in a rather straightforward way powered by a bunch of simple and universal heuristics. This allows the application to effectively operate with a low number of sample words which is essential for modeling of small languages.

Also it should be noted that, in contrast to a ‘classical’ smoothing where full size N-Grams are backed up with shorter N-Grams to cover cases where concrete N-Grams are missing in the training dictionary, in our algorithm we backup concrete N-Grams with (the same in length) abstracted N-Grams. This method was proposed because the ‘traditional’ usage of short N-Grams for modeling appears to cause some deterioration in naturalness of generated words.

The application may work in the following modes.

1) peek some collection of C random candidate words or
2) find top *C* most probable words, where *C* is a user defined number.

Every candidate word is built incrementally, letter by letter. (In this model, the end-of-word is treated as a special letter which, when appended, causes the word to be completed and output as a candidate word). The probability of appending a given letter to an already generated prefix p is calculated upon N-Gram occurrence counts taken from the training dictionary. The length of a N-Gram (that includes the length of prefix plus one for the phoneme under selection) is kept to be the value of “**Max N-Gram**” parameter which by default equals five. However, for the left MaxNg-1 positions in a generated word we use shorter N-Grams starting at the beginning of a word.

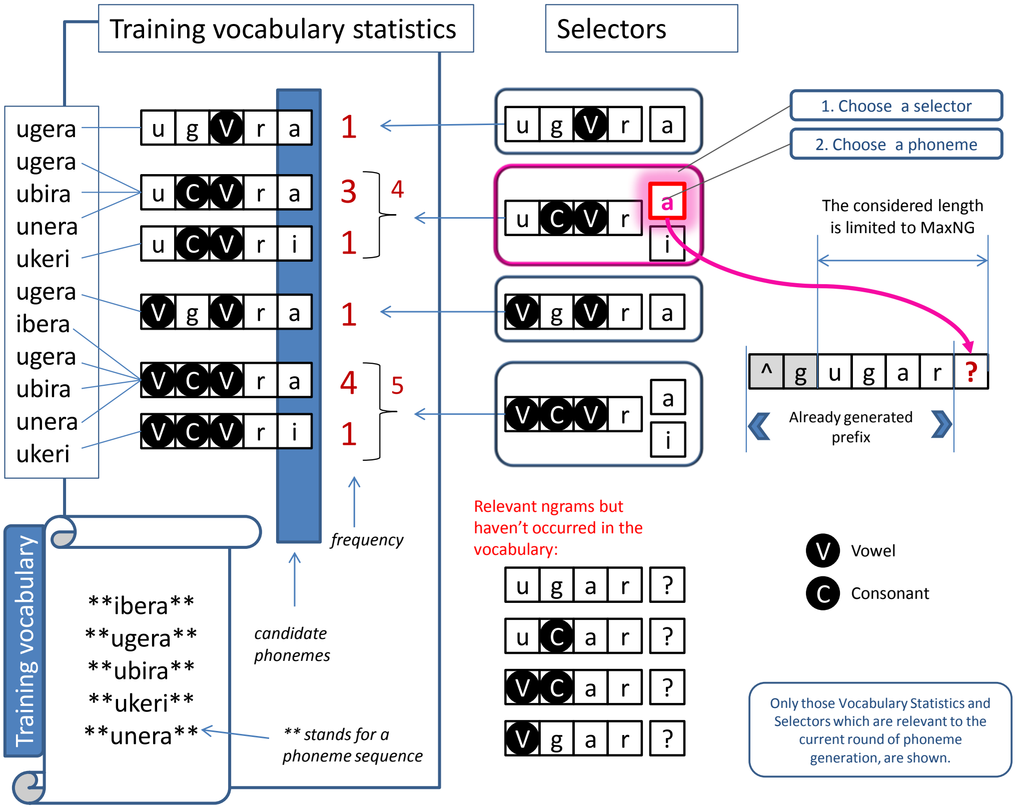

Figure 1. *A phoneme selection step*

Every generated word is produced according to the probability score of prefix continuation. To determine this score it considers not just concrete prefixes found in the training dictionary but also abstracted ones (where one or more phonemes are substituted with abstracted symbols like the consonant, the vowel and the frequent vowel.) As we don't expect most of generated N-Grams to explicitly occur in the training dictionary, we use phoneme abstraction to extend the search scope. This means that we consider variations of a concrete N-Gram where some letters are substituted by corresponding abstracted symbols. We try to abstract all vowel and consonant phonemes in a N-Gram except for two rightmost positions. This exception warrants that at least every pair of adjacent phonemes exactly coincides to some in the training dictionary.

Figure 1 illustrates a phoneme selection step. In the example we select a possible phoneme to extend a previously generated prefix of *'gagar'*. It's assumed we operate at **Max N-Gram** = 5; thus, we consider possible 5-Grams matching *'agar?'* pattern, where *'?'* stands for the phoneme to be selected. We consider optional legal abstraction of constituting vowel and consonants, except the last one, in all possible combinations. Then, in our example we have 2^3 = 8 options shown in the middle column. For every combination we try to find all extensions that are present in some training dictionary words. Once we find such extensions, we use their numbers of occurrences in the dictionary as the basis for the resulting phoneme candidate chances calculation.

The probability of given N-Gram choice at a phoneme selection step is a normalized multiplication of the following factors.

- N-Gram occurrence count in the training dictionary
- Abstraction Factor which is applied if some phoneme in an N-Gram is an abstracted symbol. This is done in order to encourage (or discourage if user prefers to do this) concrete N-Gram coincidence with respect to abstracted match. The higher the abstraction factor is, the richer variety of words we obtain but their phonotactics may become less similar to that of the original dictionary. It’s also possible to set so called Alpha-factor applied upon every additional abstracted phoneme.
- A Confidence factor which is a (normally increasing) user-tunable function of the Confidence Level (see below).

Confidence Level

The Confidence Level is a measure of distribution uniformity for phoneme abstractions. The idea is that if many different phonemes could occupy a position in N-Gram (assuming that the content of other positions remains the same) than it’s more expectable that any consonant may successfully (from a phonotactic point of view) occupy this focus position. Vise versa, if we only see one or two phoneme occurring at the focus position, we may hardly expect that other consonants are suitable here. If we see N-Grams ‘cabak’, ‘tabak’, ‘sabak’, ’zabak’ and 3 more like these, we are rather confident that any ‘*abak’ would work, where * is a consonant; however, if only ‘cabak’ was seen, but no other ‘*abak’, then probably we shouldn’t use an abstracted pattern.

To get such metrics while considering an abstraction of phonemes in a given (focus) position (assuming that content of other positions is the same), we use the following formula:

$$C = \frac{1}{M}\sum_i \left[C_i \ln(1 + \frac{n_i C_i M}{T})\right]$$

Where

- C is the abstraction confidence level;
- n_i is the *i*th distinct N-Gram occurrence count;
- C_i is the confidence level of *i*th N-Gram. If this N-Gram consists solely of concrete phonemes, C_i = 1. Otherwise, it is taken recursively from the previous abstraction resulted in the *i*th N-Gram.
- M is abstraction cardinality (i.e. the number of concrete phonemes known to match the abstracted symbol)

Note that the confidence level for a concrete phoneme is always assumed to be 1.

For the performance optimization, the logarithmic function is roughly simulated by a value table lookup.

4. Pruning

Whenever at some stage of word generation a maximum observable confidence level for matching (N-1)-Grams is below an adjustable threshold, that a word generation process is terminated and the word is rejected. Such a technique (called Pruning) helps one to eliminate multiple weird words those otherwise tend to be produced at high abstraction level settings. However, please note that in many cases you may really want some unusual words to be produced. So, high Confidence threshold values should be used with care.

5. Auto Detection of Vowel

Word Generator employs Sukhotin's algorithm to distinguish vowels of consonants. The idea is that vowels and consonants are much more frequently cross-collocated than located adjacent to the same class representatives ('VC' combination is more frequent than either of 'CC' or 'VV'). The algorithm works pretty well for all languages explored by the author. If a language has special characters that are beyond the Vowel/Consonant duality, like special signs, it's recommended to claim those phoneme or/and quasi-phonemes as unique ones explicitly by adding an 'Extra: phoneme [,phoneme..]' line to the vocabulary file used for Generator training. Still it isn't much critical as misprediction of class for a couple of phonemes shouldn't significantly disturb the overall look and feel of generated words.

6. Precision Self Evaluation.

An important quality metric (especially when one plays with Word Generator settings and parameters) is probability of true language lexeme prediction (beyond lexemes enumerated in a training vocabulary). To provide an answer to that question, we employ a cross-validation precision estimation procedure over a training vocabulary supplied. For each word, we modify internal statistics as if that word wasn't observed in the training vocabulary. Then we simulate generation of that particular word measuring the total probability of all variants ending in that one word production. The mean value of such a probability over all vocabulary words is displayed as Word Generator's evaluated precision. This mode may be activated by an option setting. To give an user a detailed picture of language word prediction power both for 'high' and 'low hanging fruits', Word Generator displays evaluated precision figures for various quantiles of most challenging vocabulary words (5..100%). Please note that a Monte Carlo method is used for probability normalizing, so the results aren't exactly the same for two repeated runs, with a dispersion being in a reverse relation to the number of generated words.

7. Technical optimizations

Graph Construction

For the sake of word statistics search optimization, the algorithm builds a suffix tree where every node represents a N-Gram except the last phoneme (let's call them (N-1)-Grams). Every node points directly to the following nodes.

1) to nodes representing each (N-1)-Gram (seen in the training dictionary) obtained by adding a phoneme to the end of current (N1)-Gram and then restricting its length to **Max N-Gram** minus 1 value by optional deleting of the beginning phoneme.

2) to nodes representing versions of given (N-1)-Gram where one concrete character becomes abstracted (for the positions in (N-1)-Gram where the abstraction is allowed, i.e. for any position except the last one).

Figure 2 illustrates such a graph (an example created upon reading a single word *'laris'*). Solid lines stand for pointers mentioned as (1) in the list above while dotted lines stand for (2). If **Max N-Gram** = 5, up to three links to abstracted nodes may exist per node. Please note that '^' character stands for the beginning of a word, and nodes drawn in brown color allow a word end to immediately follow.

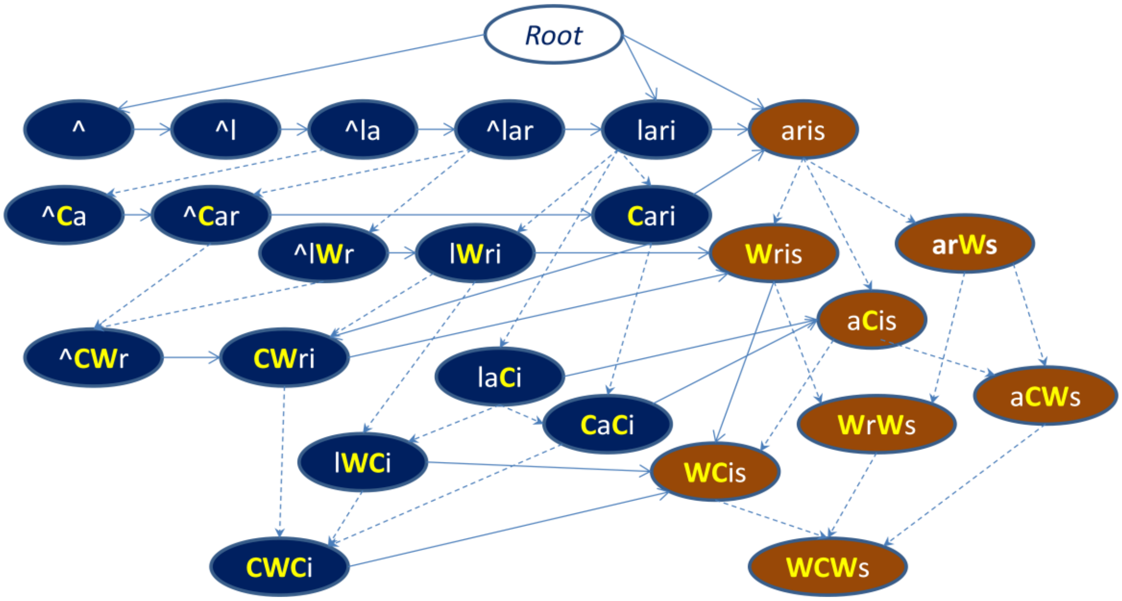

Figure 2. *An example of graph created for the training dictionary N-Gram statistics storage*

Probability scoring

We use a two-stage process to generate every phoneme. At the first stage we choose (randomly or by score, depending on execution mode) an abstracted (N-1)-Gram pattern based on its total probability (which is proportional to collective occurrence counts shown near enclosing braces at *Figure 1*). At the second stage we select a concrete letter among possible extensions of the (N-1)-Gram to a N-Gram regarding their relative frequencies divided by the (N-1)-Gram total frequency as a probability metric. This method allows us to abandon calculation of summed probability for all competing concrete N-Grams that may be reached in many ways via abstractions. However, while doing so we then have no true digits on the final probability score of words we generate. To address this issue in cases where we need this data (namely, when we have an option set to display the score and/or to sort the result by the score) we calculate exact multi-way probability post factum for the generated words.

8. Availability

The source code is available at http://github.com/andreas-softwareengineer-pro/word-generator

Web service is available on the PARADISEC site, http://paradisec.org.au/wordgen/wg.php

Author's Web version: http://regexus.com/wordgen/wg.php

References

[1] Shcherbakov, A., Vylomova, E. and Thieberger, N., 2016. Phonotactic Modeling of Extremely Low Resource Languages. In *Proceedings of the Australasian Language Technology Association Workshop 2016* (pp. 84-93).

Appendix I.

Training dictionary format

The dictionary used to learn information about phonotactics of the language which we are about to generate words in.

- This should be a plain text file (UTF8 encoding) that contains a plain list of words, one per line.
- It may also contain special headers to denote and/or classify phonemes and/or other kinds of stuff (quasi-phonemes) found in lexemes. It's ok for a phoneme or quasi-phoneme to be represented by a combination of characters. The format is as follows.

  keyword: comma_separated_list_of_phonemes

  ➢ Example: **Vowels:** a, aa, e, ee, i, ii, o, oo, u, uu if

Letter case in dictionary files doesn’t matter. Additional info (comment) enclosed in ‘()’, ‘{}’, ‘[]’ or ‘<>’ parentheses is allowed at word lines and ignored by the application.

Multiple instances of the same kind of header are allowed and their lists are concatenated.

Here is the full list of recognized header keywords.

- **Vowels:**

  Specifies vowel phonemes
- **Consonants:**

  Specifies consonant phonemes
- **Extra:**

  Specifies phonemes that should not be considered as a consonant nor a vowel; they may be signs (not really phonemes) or very specific phonemes. The idea is to exclude such a phoneme from statistical consideration, never allowing it to be abstracted and then substituted by another phoneme.
- **Ignore:**

  Enumerates a character (and/or character sequences) to ignore.
- **Stop:**

  Enumerates characters and/or sequences that disable the reading of the rest of a dictionary line (being themselves ignored) when they occur. This feature may, for example, be used to define a character that begins a comment.
- **Reject:**

  Enumerates characters or sequences that invalidate a whole dictionary line once they occur in it.
- **Break:**

  Enumerates characters or sequences that break a dictionary line into two separate lines, as if they were new line characters (CR). This allows one to place multiple words in a single line.

Note. It's possible to specify leading and/or trailing context around character(s) constituting a phoneme in header lists. The form is as follows:

*leading_characters***|***phoneme_characters***|***trailing_characters*

For ex., *'a|ch|o'* means that a given phoneme is recognized in words like *'xxxachoyyyy'* but not recognized in, say, *'xxxucho'*. Leading or trailing characters field may be left empty; still both vertical bar delimiters should be present in this form. Please note that using leading or trailing contexts may easily lead one 'real world' phoneme to be proliferated into multiple records in a dictionary header. It's absolutely ok for Word Generator operation.

Appendix II.

Running application in a Shell

To run the Word Generator in a Shell mode, the following command may be used.

```
wg [options] file [file ..]
```

where *file* is a training dictionary file compliant to the format described in *Appendix I*.

Supported options are as follows.

-a *real_number*
Alpha factor. The probability score of a prospective word is multiplied by it each time we select a N-Gram with two or more abstracted phonemes (and, repeatedly, per each extra abstracted phoneme beyond two).

-b *minimum_confidence*
Confidence level threshold required for a word generation to be proceeded (see *Pruning*).

-c *int_number*
The desired number of generated words. Note that the number may be less if too many words are filtered out due to a minimum score option or other conditions.

-C
Show a mark (#) near every generated word that hits a training dictionary word. Only makes sense in conjunction to –H option.

-D *int_number*

Length of grams that should be observed exactly as is in the training vocabulary in order to occur in generated words. Default = 2; minimum = 1; maximum = considered N-Gram size (see **–n**).

-f *real_number*[**,***real_number***:***real_number* …]
Abstraction factor. If a basic value is followed by '*,level*:*factor'* record(s), each one defines a value applied once the corresponding Confidence level thresholds is reached. Such a form allows one to define a Confidence aware Abstraction factor as a tabular function of Confidence level.

-h
Produce an HTML formatted output.

-H

Allows a generated word to hit a word from the training dictionary.

-m real_number
Minimum probability score for a generated word to be accepted.

-n *int_number*
Maximum considered N-Gram size. The default value is 5.

-p
Display self evaluated presision

-P
Requests printing some statistics extracted from the training dictionary

-r int_numer
Random seed. If set, executions with the same seeds will produce exactly the same words (provided that all generation controlling options are also the same).

-s *int_number*
Maximum number of quasi-syllables in a generated word. A quasi-syllable is a vowel group optionally accompanied by consonants , where a vowel group is either a standalone vowel or a continuous sequence of vowels.

-t
Turns on top score words mode (generating N most probable word). Unless this option is set, N random words are generated.

-T
Sort the result by probability in descending order.

-v *int_number*
Verbosity level.

-V *real_number*
Vowel Frequency Confidence (see ‘Vowels Clustering’ above).

-w

One lexeme mode: Use the whole vocabulary file as a single lexeme.

-%
Display probability score near every generated word. Please note that, in contrast to Precision displayed with **–p** option, (1) those scores aren’t well normalized; (2) no cross-validation technique is applied if even a generated word is present in the training vocabulary (when **–H** is also set).