New Mathematics and Natural Computation
Vol. 4, No. 1 (2008) 87–106


World Scientific
www.worldscientific.com

# USING GRAPHEME $n$-GRAMS IN SPELLING CORRECTION AND AUGMENTATIVE TYPING SYSTEMS

ALKET MEMUSHAJ and TAREK M. SOBH*
*School of Engineering*
*University of Bridgeport, Connecticut, USA*
**sobh@bridgeport.edu*

Probabilistic language models have gained popularity in Natural Language Processing due to their ability to successfully capture language structures and constraints with computational efficiency. Probabilistic language models are flexible and easily adapted to language changes over time as well as to some new languages. Probabilistic language models can be trained and their accuracy strongly related to the availability of large text corpora.

In this paper, we investigate the usability of grapheme probabilistic models, specifically grapheme $n$-grams models in spellchecking as well as augmentative typing systems. Grapheme $n$-gram models require substantially smaller training corpora and that is one of the main drivers for this thesis in which we build grapheme $n$-gram language models for the Albanian language. There are presently no available Albanian language corpora to be used for probabilistic language modeling.

Our technique attempts to augment spellchecking and typing systems by utilizing grapheme $n$-gram language models in improving suggestion accuracy in spellchecking and augmentative typing systems. Our technique can be implemented in a standalone tool or incorporated in another tool to offer additional selection/scoring criteria.

*Keywords*: Natural language processing; language modeling; statistical language modeling; grapheme $n$-grams.

## 1. Background Research

### 1.1. *Introduction*

The field of Natural Language Processing (NLP) began in earnest in the 1950s and since then, it has grown into a major field of study with a large number of practical applications such as speech recognition, spelling correction, handwriting recognition, optical character recognition, etc.[7,8] NLP focuses on modeling natural languages in order to capture and to utilize the structures they exhibit. Early NLP efforts have focused on linguistic, rule-based approaches such as generative grammars that specify the well-formed expressions of a natural language. These rigorous linguistic models were complex, inflexible and indeed have suffered from their inability to adapt and evolve into the usage of the language, new domains of knowledge, as well as new languages. More current NLP efforts are making wide use of statistical

methods in language processing when probabilistic language models have proven to be easily constructed, adapted and commercially successful.

Probabilistic language models are constructed based on statistical processing of large text corpora. Collocations, expressions consisting of two or more words that are usually used together, are the simplest expression of knowledge that can be gained by statistical processing. From collocations, one can generalize and attempt to rephrase the problem as the probability with which words are placed next to each other or even beyond. One of the best known statistical models is the $n$-gram model which attempts to assign a probability to an unobserved event based on the $n-1$ previously observed events. Language models based on $n$-grams are created by assigning a probability to the occurrence of the next word, after $n-1$ words have already been observed. $N$-gram models are trained by processing large amounts of data so that we can account for every possible occurrence of the words in a vocabulary.

### 1.2. *Markov chains*

Markov chains underlie the field of statistical natural language processing. A Markov Chain is a special case of a Weighted Automata. In a Markov Chain, the input sequence uniquely determines the state change sequence. A Markov Chain assumes that the next state depends only on a finite number of previous states.

Let $X_0, X_1, X_2, X_3, \ldots, X_n$ be a sequence of random variables taking their values in the same finite alphabet $X = \{X = 0, 1, 2, \ldots, c\}$. In this case, the Bayes' formula applies:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, X_2, \ldots, X_{i-1})$$

The random variables are said to form a Markov Chain if for all values of $i$:

$$P(X_i | X_1, X_2, \ldots, X_{i-1}) = P(X_i | X_{i-1})$$

As a consequence, for Markov Chains

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_{i-1})$$

This means that the value at time $i$ depends only on the value at the preceding time and on nothing that went on before.

### 1.3. *Language modeling and n-grams*

In statistical NLP, determining the probability $P$ of a word sequence $W = w_1, w_2, \ldots, w_n$ is called language modeling. Given the large number of possible word combinations, we make a Markov assumption, that is: only the local context matters in determining the probability of the next word. These statistical language models

are called $n$-gram models. An $n$-gram model uses the previous $n-1$ observed words to predict the next one. An $n$-gram model is defined as the probability function, a conditional probability, of the current word given $n-1$ previous words. This can be expressed with this formula:

$$P(W_1, W_2, \ldots, W_{n-1}, W_n) = P(W_1)P(W_2|W_1)P(W_3|W_1^2)\cdots P(W_n|W_1^{n-1})$$
$$P(W_1^n) = \prod_{k=1}^{n} P(W^k|W_1^{k-1})$$

The assumption made in $n$-gram models is that only the local context (the prior $n-1$ words) is relevant, and so we modify the above general probability function as below:

$$P(W_n|W_1^{n-1}) \approx P(W_n|W_{n-N+1}^{n-1})$$

For the case when $N = 2$, we have a Bigram. When $N = 3$, we have a Trigram. When $N = 4$, we have a Fourgram. Substituting $N$ in each case, we have these equations:
Bigram Equation:

$$P(W_1^n) \approx P(W_n|W_{n-1}^{n-1})$$

Trigram Equation:

$$P(W_1^n) \approx P(W_n|W_{n-2}^{n-1})$$

Fourgram Equation:

$$P(W_1^n) \approx P(W_n|W_{n-3}^{n-1})$$

In this paper, we will use the above equations to calculate grapheme $n$-grams by substituting $W_n$ with $G_n$.

### 1.4. *Maximum likelihood estimate (MLE)*

$N$-gram models can be trained by counting words in a corpus and normalizing. We take a training corpus and from this corpus take the count of a particular $n$-gram and divide this count by the sum of all the $n$-grams that share the same first word. For example, for a bigram, the MLE is calculated as:

$$P(W_2|W_1) = f(W_2|W_1) = \frac{C(W_1, W_2)}{C(W_1)}$$

This process generates the MLE (Maximum Likelihood Estimate) because all the probability mass is given to the $n$-grams observed in the training corpus while a zero probability mass is given to all other $n$-grams, which could be observed if the training corpus were larger.

### 1.5. *Smoothing algorithms*

Smoothing is a technique used to better estimate probabilities when there is insufficient data to estimate probabilities accurately.[3,4] Simple $n$-gram models are trained from some corpus and because any particular training corpus is finite, naturally occurring $n$-grams are bound to be missing from it. Assigning a non-zero probability to an unseen $N$-gram is the focus of the smoothing techniques and algorithms and accounts for the fact that even though some $N$-grams have not been seen in the training corpus, they can possibly be encountered if our training corpus were larger or included data from a different knowledge domain.

#### 1.5.1. *Add one smoothing*

A simple way of doing smoothing is to add 1 to the count of each $N$-gram represented in our $N$-dimensional matrix of observed $N$-grams and then multiplying by a normalization factor $N/N + V$, where $V$ is the number of grapheme types (the alphabet size). The adjusted count would then be expressed as:[7]

$$c_i^* = (c_i + 1)\frac{N}{(N + V)}$$

In the case of a bigram, the smoothed probability function $p^*$ changes as shown below:
Normal Bigram Probability:

$$P(W_2|W_1) = \frac{C(W_1, W_2)}{C(W_1)}$$

Adjusted Bigram Probability:

$$p^*(W_n|W_{n-1}) = \frac{C(W_{n-1}W_n) + 1}{C(W_{n-1}) + V}$$

where $V$ is the alphabet size. Add-One smoothing performs worse than other smoothing techniques unless there is a very large amount of data in the training corpus.[3,4]

#### 1.5.2. *Witten-Bell discounting*

Smoothing can also be seen as discounting, lowering the probability mass spread over the observed $n$-grams in the existing corpus and spreading that over the non-observed $n$-grams. Witten-Bell discounting is based on the concept of estimating the probability of seeing a zero-frequency $n$-gram. This probability can be "modeled" by the probability of seeing an $n$-gram for the first time. This probability is simple to compute since the count of "first-time" $n$-grams is equal to the number of $n$-gram types seen in the training data. Witten-Bell discounting is then formulated as:[7]

$$C_i^* = \begin{cases} \frac{T}{Z} \times \frac{N}{N+T} & \text{if } C_i = 0 \\ C_i \frac{N}{N+T} & \text{if } C_i > 0 \end{cases}$$

where $N$ is the number of $n$-gram tokens, $T$ is the number of observed $n$-grams and $Z$ is the total number of zero-count $n$-grams.

### 1.5.3. *Good-Turing estimation*

Good-Turing Estimation reallocates the probability mass of $n$-grams that occur $r+1$ times in the training data to the $n$-grams that occur $r$ times:[8]

$$r^* = (r+1)\frac{n_{r+1}}{n_-}$$

where $n_r$ is the number of $n$-grams seen exactly $r$ times. This can be converted to a probability for the $n$-gram $\alpha$ with $r$ count: $P(\alpha) + \frac{r^*}{N}$

### 1.5.4. *Back-off*

The discounting seen above can help us with the zero-count $n$-grams, but we can also utilize a lower-level $n$-gram to compute the probability of a higher-level $n$-gram. There are two methods to do just that — back-off and deleted interpolation. In both models, an $N$-gram model is built based on an $N-1$ model. The difference between the two is that in back-off, once an $n$-gram has a non-zero count then lower level $n$-grams are not utilized.

For a trigram model, the back-off algorithm is expressed as:[7]

$$P(W_i|W_{i-2}W_{i-1}) = \begin{cases} P(W_i|W_{i-2}W_{i-1}) & \text{if } C(W_{i-2}W_{i-1}W_i) > 0 \\ \alpha_1 P(W_i|W_{i-1}) & \text{if } C(W_{i-2}W_{i-1}W_i) = 0 \text{ and } C(W_{i-1}W_i) > 0 \\ \alpha_2 P(W_i) & \text{otherwise} \end{cases}$$

where the weighing coefficients $\alpha_1$ and $\alpha_2$ are functions of the preceding word string, $\alpha_1$ is a function of $W_{n-2}^{n-1}$ and $\alpha_2$ is a function of $W_{n-1}$ for the trigram model.

### 1.5.5. *Interpolation*

Interpolation is a method that combines bigram probabilities with unigram probabilities to estimate the probability of an unseen bigram by utilizing the probability of the unigram contained within the bigram.[7] The deleted interpolation algorithm combines the probabilities of all lower order $n$-grams with the highest order $n$-gram by attaching linear weights to all language models. The Linear Interpolation for an $n$-gram model can be expressed as:

$$P_{LI}(W_i|W_{i-n+1}^{i-1}) = \lambda_1 P(W_i|W_{i-n+1}^{i-1}) + \cdots + \lambda_n P(W_i)$$

where the sum of all $\lambda$ is equal to 1: $\sum \lambda_i = 1$

Recursively, this equation can be expressed as:

$$P_{LI}(W_i|W_{i-n+1}^{i-1}) = (1 - \lambda_{W_{i-n+1}^{i-1}})P(W_i|W_{i-n+1}^{i-1}) + \lambda_{W_{i-n+1}^{i-1}} P_{LI}(W_i|W_{i-n+2}^{i-1})$$

The $\lambda$ values are trained in order to maximize the likelihood of the held-out corpus (a portion of the corpus that is separated from the training data and used to test out the language model trained on the training data).

### 1.6. *Entropy and perplexity*

Entropy is one of the most common metrics used to evaluate $n$-gram models. Entropy is the minimum number of bits it would take to encode a piece of information.[2] Entropy is calculated by establishing a variable $X$ that ranges over words or letters and that has a probability function $p(x)$. Entropy is then defined as:[2]

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Perplexity is defined as $2^H$ and gives us the weighted average number of choices a variable has to make.[6,7] Thus, if the entropy of our grapheme language model is 2, then on average, there are 4 possible choices for the next grapheme.

### 1.7. *Minimum edit distance*

The minimum edit distance between two strings is the smallest number of changes (insertions, deletions, substitutions) needed to convert one string into the other where each of the above operations has a cost of 1. A more general requirement is that each of the operations can have a different cost, i.e. insertions and deletions would have a cost of 1 and substitution would have a cost of 2 since a substitution is comprised of one deletion and one insertion. The minimum edit distance is calculated by utilizing dynamic programming. The general case minimum edit distance algorithm,[7] where the cost of each operation is a variable, is shown below:

*function MIN-EDIT-DISTANCE* (*target,source*) *returns min-distance*
    $n \leftarrow LENGTH(target)$
    $m \leftarrow LENGTH(source)$
    *Create a distance matrix* $distance[n+1, m+1]$
    $distance\ [0,0] \leftarrow 0$
    *for each column* $i$ *from* 0 *to* $n$ *do*
        *for each row* $j$ *from* 0 *to* $m$ *do*
            $distance[i,j] \leftarrow MIN(distance[i-1,j] + ins\text{-}cost) target_i,$
            $distance[i-1,j-1] + subst\text{-}cost(source_j, target_i),$
$distance[i,j-1] + del\text{-}cost(source_j)$

## 2. Proposed Approach

### 2.1. *Overview*

Probabilistic language modeling is widely used in spelling correction and handwriting recognition systems among other widespread usage of such models. As mentioned before, probabilistic language models are built via processing written or

recorded spoken language in order to infer structural knowledge about the language. The most common probabilistic models are word $n$-gram models. Word $n$-gram models suffer from data sparseness because of the uneven distribution of lexical constructs in the text corpus used to train the model. Building even the simplest word $n$-gram model, a bigram, which attempts to assign a probability to the event where we encounter word $W_n$ after we have seen word $W_{n-1}$, requires that we attempt to assign probabilities to all the possible $\{W_nW_{n-1}\}$ word pairs from their frequency in the given training corpus. Even for a modest dictionary $N$ of 10,000 words, it would require that we see at least one event for each of the $N^2$ possible pairs, in this case 100 million possible pairs. The possible number of word sequences increases to $N^3$ and $N^4$ for trigram and fourgram models respectively. Training such models requires immense amounts of text and since even large corpora such as those for the English language suffer from data sparseness, a language model based on word $n$-grams for the Albanian language is even more of a challenge since there are no available corpora for the Albanian language.

Our technique attempts to overcome this challenge by utilizing grapheme $n$-gram models which can infer morphological knowledge of the language from a smaller corpus. The intuition that led us to attempting to utilize such models is based on the fact that graphemes are the smallest sub-word unit in a language.

Spellcheckers and handwriting recognition systems are judged on their coverage/recall, flagging/precision, and suggestion adequacy characteristics. Our technique attempts to improve suggestion adequacy in either a spelling correction system or a typing system by utilizing grapheme $n$-gram probabilistic models. In a spelling correction system, after a word has been flagged as misspelled, a Minimum Edit Distance algorithm can be used to compare a given word to each word from a dictionary and then generate a list of suggested replacement words that differ from the original word with a minimal number of insertions, deletions, or replacements. Simply listing these suggestions does not provide satisfactory suggestion adequacy so we need to add additional criteria to the ranking so that a correct suggestion from the list is always at or near the front of the list of suggestions. In an augmentative typing system, suggestions can be generated while words are being typed in order to minimize the amount of typing needed for each word.

In the sections below, we describe the steps and decisions in the process as well as the results generated by our technique.

### 2.2. *Obtaining and preparing the text source*

The first step in building an $n$-gram language model is the creation and preparation of the training corpus. A fiction novel by Fatos Kongoli was chosen as our corpus as it is a somewhat representative example of the commonly written and spoken language today in Albania. The text is short, containing 83,141 word tokens and 25,405 word types. The large number of word types, as compared to English, reflects the highly inflectional nature of Albanian. The text was in raw form with no markups or

any pre-processing done. We used this text to build the dictionary as well as the grapheme bigram, trigram, and fourgram probabilities. The text was converted to lowercase assuming that any proper names will have a very low frequency and will not bias our probabilities. Word tokens were assumed to be the text within white spaces. All non-alphabetic characters were removed, leaving only words and white spaces in the text.

### 2.3. *Building the language model*

The next step in building the language model is the generation of Maximum Likelihood Estimates for our $n$-grams. In this paper, we have generated Maximum Likelihood Estimates for the Bigram, Trigram, and Fourgram models. The formula for generating the MLE for a bigram is:

$$P(W_2|W_1) = f(W_2|W_1) = \frac{C(W_1, W2)}{C(W_1)}$$

Some of the bigram MLE values are shown in the table below:

Table 1. Bigram maximum likelihood estimates.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| a | 0.000116 | 0.007512 | 0.005125 | 0.020906 | 0.00099 | 0.015024 | 0.008269 |
| b | 0.168818 | 0 | 0 | 0.000253 | 0.098203 | 0 | 0 |
| c | 0.210843 | 0 | 0 | 0 | 0.189157 | 0 | 0 |
| d | 0.050644 | 0 | 0 | 0 | 0.092934 | 0 | 8.70E−05 |
| e | 0.004328 | 0.002255 | 0.005181 | 0.032671 | 6.10E−05 | 0.00896 | 0.025905 |
| f | 0.174653 | 0 | 0 | 0 | 0.084805 | 0 | 0 |
| g | 0.245828 | 0 | 0 | 0.00194 | 0.022119 | 0 | 0 |

The formula for generating MLE for a trigram is:

$$P(W_3|W_1, W_2) = f(W_3|W_1, W_2) = \frac{C(W_1, W_2, W_3)}{C(W_1, W_2)}$$

Some of the trigram MLE values are shown in the table below:

Table 2. Trigram maximum likelihood estimates.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| aa | 0 | 0 | 0 | 0 | 0 | 0.5 | 0 |
| ab | 0.364341 | 0 | 0 | 0 | 0.062016 | 0 | 0 |
| ac | 0.090909 | 0 | 0 | 0 | 0.034091 | 0 | 0 |
| ad | 0.08078 | 0 | 0 | 0 | 0.027855 | 0 | 0 |
| ae | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ba | 0 | 0.065967 | 0 | 0.007496 | 0 | 0 | 0.002999 |
| bb | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The formula for generating MLE for a fourgram is:

$$P(W_4|W_1,W_2,W_3) = f(W_4|W_1,W_2,W_3) = \frac{C(W_1,W_2,W_3,W_4)}{C(W_1,W_2,W_3)}$$

Some of the MLE values for the fourgram model are shown below:

Table 3. Fourgram maximum likelihood estimates.

| | a | b | c | d | e | f | g | h | i | j | k | l |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| aaa | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| aab | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| aac | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| aad | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| aae | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| aba | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.744681 | 0 | 0.148936 | 0.021277 |
| abb | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

As is clear from the generated MLE values, MLE allocates all of the probability mass on the observed events and zero probability mass on unseen events. In building our language model, we have not only calculated the MLE values, but we have also applied two smoothing algorithms. The tables below show samples of values for both Add-One Smoothing and Witten-Bell Discounting for the bigram, trigram, and fourgram models.

Table 4. Bigram probabilities after add-one smoothing.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| a | 0.000175 | 0.00757 | 0.005183 | 0.020964 | 0.001048 | 0.015083 | 0.008328 |
| b | 0.169071 | 0.000253 | 0.000253 | 0.000506 | 0.098456 | 0.000253 | 0.000253 |
| c | 0.212048 | 0.001205 | 0.001205 | 0.001205 | 0.190361 | 0.001205 | 0.001205 |
| d | 0.050731 | 8.70E−05 | 8.70E−05 | 8.70E−05 | 0.093021 | 8.70E−05 | 0.000174 |
| e | 0.004389 | 0.002316 | 0.005242 | 0.032732 | 0.000122 | 0.009021 | 0.025966 |
| f | 0.174968 | 0.000315 | 0.000315 | 0.000315 | 0.08512 | 0.000315 | 0.000315 |
| g | 0.246023 | 0.000194 | 0.000194 | 0.002134 | 0.022313 | 0.000194 | 0.000194 |

Table 5. Bigram probabilities after Witten-Bell discounting.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| a | 0.000116 | 0.00751 | 0.005123 | 0.020899 | 0.00099 | 0.015019 | 0.008266 |
| b | 0.167607 | 0.000512 | 0.000512 | 0.000251 | 0.097499 | 0.000512 | 0.000512 |
| c | 0.203351 | 0.002369 | 0.002369 | 0.002369 | 0.182435 | 0.002369 | 0.002369 |
| d | 0.050536 | 0.000177 | 0.000177 | 0.000177 | 0.092737 | 0.000177 | 8.70E−05 |
| e | 0.004326 | 0.002254 | 0.005179 | 0.032659 | 6.10E−05 | 0.008957 | 0.025896 |
| f | 0.173206 | 0.000637 | 0.000637 | 0.000637 | 0.084102 | 0.000637 | 0.000637 |
| g | 0.244475 | 0.000393 | 0.000393 | 0.00193 | 0.021997 | 0.000393 | 0.000393 |

Table 6. Trigram probabilities after add-one smoothing.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| aa | 0.03125 | 0.03125 | 0.03125 | 0.03125 | 0.03125 | 0.0625 | 0.03125 |
| ab | 0.301887 | 0.006289 | 0.006289 | 0.006289 | 0.056604 | 0.006289 | 0.006289 |
| ac | 0.076271 | 0.008475 | 0.008475 | 0.008475 | 0.033898 | 0.008475 | 0.008475 |
| ad | 0.077121 | 0.002571 | 0.002571 | 0.002571 | 0.028278 | 0.002571 | 0.002571 |
| ae | 0.021277 | 0.021277 | 0.021277 | 0.021277 | 0.021277 | 0.021277 | 0.021277 |
| ba | 0.001435 | 0.064562 | 0.001435 | 0.008608 | 0.001435 | 0.001435 | 0.004304 |
| bb | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |

Table 7. Trigram probabilities after Witten-Bell discounting.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| aa | 0.025911 | 0.025911 | 0.025911 | 0.025911 | 0.025911 | 0.137243 | 0.025911 |
| ab | 0.352976 | 0.001418 | 0.001418 | 0.001418 | 0.060081 | 0.001418 | 0.001418 |
| ac | 0.086634 | 0.002045 | 0.002045 | 0.002045 | 0.032488 | 0.002045 | 0.002045 |
| ad | 0.080022 | 0.000521 | 0.000521 | 0.000521 | 0.027594 | 0.000521 | 0.000521 |
| ae | 0.0084 | 0.0084 | 0.0084 | 0.0084 | 0.0084 | 0.0084 | 0.0084 |
| ba | 0.000282 | 0.065707 | 0.000282 | 0.007467 | 0.000282 | 0.000282 | 0.002987 |
| bb | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |

Table 8. Fourgram probabilities after add-one smoothing.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| aaa | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aab | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aac | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aad | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aae | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aba | 0.012987 | 0.012987 | 0.012987 | 0.012987 | 0.012987 | 0.012987 | 0.012987 |
| abb | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |

Table 9. Fourgram probabilities after Witten-Bell discounting.

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| aaa | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aab | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aac | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aad | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aae | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |
| aba | 0.00036 | 0.00036 | 0.00036 | 0.00036 | 0.00036 | 0.00036 | 0.00036 |
| abb | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 | 0.033333 |

After the MLE, Add-One and Witten-Bell language models were created, and we calculated the entropy for the Bigram, Trigram, and Fourgram models as below:

| | |
|---|---|
| Bigram MLE Entropy | 3.01867990768389 |
| Bigram Add-One Entropy | 3.15301080452977 |
| Bigram Witten-Bell Entropy | 3.18008073400134 |
| | |
| Trigram MLE Entropy | 2.01839172229128 |
| Trigram Add-One Entropy | 3.97363007007406 |
| Trigram Witten-Bell Entropy | 3.5851641154334 |
| | |
| Fourgram MLE Entropy | 1.43200113013433 |
| Fourgram Add-One Entropy | 4.77572387001751 |
| Fourgram Witten-Bell Entropy | 4.42011483996596 |

We notice that the entropy of any of our smoothed $n$-gram models is higher than its MLE entropy. The smoothed models spread the probability mass from the observed events onto unobserved events. While this is required in word $n$-gram models since it is possible that any sequence of $n$ words could be observed in a text, it is a bad choice with grapheme $n$-grams as it is morphologically impossible for many grapheme sequences to be observed in a language. Thus, the smoothed models allocate probability mass on impossible events. While it can be argued that many words are not in the corpus, words in a language are formed of syllables which follow finite construction rules.[5] The entropy for our grapheme fourgram MLE model for the Albanian language is 1.43. The English language entropy has been estimated to be 1.3 by Shannon[9] and 1.75 by Brown *et al.*[1] We will use the MLE model in our spellchecking system.

### 2.4. *Minimum edit distance implementation*

Minimum Edit Distance calculates the difference in characters between two strings when a character insertion, removal, or replacement is each given a weight of one. Upon encountering a misspelled word during a text scan, we can calculate the minimum edit distance for the misspelled word vs all the words in the dictionary. If the dictionary is already loaded in the main memory (in a system with adequate memory), this scan does not tax the computer and is quick to find the minimum edit distance in each comparison. The minimum edit distance can take any values for the given dictionary, but we will only accept suggestions with a maximum distance of three since words that differ by more than 3 characters are quite different and misspelling is probably not the reason for the input word not being in the dictionary. During our work on the system, it became apparent that the minimum edit distance is useful only when it is smaller or equal than 1/3 of the length of the words being compared. If we have a word with 3 characters, and a total of 1,000 words in the dictionary with a length of 3, then the input word will have a minimum edit

distance of 3 with all of these words. Thus, our suggestion list would contain at least all the 1,000 3-character words + all the 2-character words, etc. So it is apparent that the minimum edit distance needs to be used carefully in determining possible suggestions for a misspelled word. We take this into account in our system and, for all words with a length smaller or equal to 3, we allow only suggestions with a minimum edit distance of 1. For all words with length between 4 and 7, we allow only suggestions with a minimum edit distance of 1 or 2. For words having more than 7 characters, we will allow a minimum edit distance of up to 3. Given the relative infrequency of words with more than 10 letters, we will not consider the case when the minimum edit distance of two words is higher than 3.

### 2.5. *Ranking suggestions created by minimum edit distance*

Once we create a list of all the suggestions, grouped according to the minimum distance, we will most probably have more than one suggestion for each group of suggestions. If we assume that all the suggestions with a smaller minimum edit distance are better suggestions, then that is one criteria of ranking the suggestions. But as we said above, if more than one suggestion exists for each group then we need to rank those suggestions as well. We now have to use this data to create a scoring system to rank our suggestions. For suggestions that have a minimum edit distance of 1 with the original word, we find the position of the differing character, if both the word and the suggestion have the same length. For suggestions that have a minimum edit distance of 2 with the original word, we find the positions of the differing characters, if both the word and the suggestion have the same length. When the length of the misspelled word and the suggestion being scored is the same, it means that we have had character replacement. If the minimum edit distance is 1, then we have one character replacement. If the minimum edit distance is 2, then we have 2 characters that have been replaced and so on for a minimum edit distance of 3. When the replaced character is in the middle of the word, it is neither the first nor the last character in the word; we analyze the trigram formed by the character in question and the surrounding 2 characters, one on each side.

- p**e**rdor
- p**ë**rdor

Thus, if the word we analyze is ‘perdor’ and the suggestion is ‘përdor’, we see that ‘e’ has been replaced by ‘ë’. We analyze the trigram {p, ë, r} and find the probability/occurrence count assigned to it when the corpus is processed. When the minimum edit distance between the words is 2, we will have to analyze 2 trigrams for each suggestion.

- perdor
- përçor
- mendor

If the misspelled word is 'perdor' and a suggestion of minimum edit distance 2 is 'përçor', the differing characters are 'ë' at position {1} and 'ç' at position {3} so we will have to analyze the trigrams {p, ë, r} and {r, ç, o}. Another suggestion of minimum edit distance 2 is the word 'mendor' and in this case the differing characters are 'm' at position {0} and 'n' at position {2}.

### 2.6. *Working with special cases*

While we can easily find a trigram where the differing character is in the middle, for 'n'–{e, n, d} we won't be able to do so for the first differing character as it is the first character in the word. We encounter the same problem when the differing character is the last in the word. In both these cases, we could use bigram probabilities. For our example, we would calculate the bigram probability {m, e}. We observe though that this probability is orders of magnitude larger than any trigram probabilities we have calculated and can skewer our results. We need to modify this probability so that it can be included in our trigram probability distribution. For this reason, I have included two meta-characters, ^-to denote the beginning of the word and $ - to denote the end of the word. In this case, the bigram {m, e} would become the trigram {^ , m, e}. We now have to modify our corpus processing program to calculate the probabilities for the trigrams {^ , y, z} and {x, y, $}.

### 2.7. *Comparing words with different lengths*

Comparing words with different lengths is more complicated than comparing words with the same length. We need to decide how we will compare the two words in a way that fits our overall approach and that probabilities assigned reflect the correct probability mass.

#### 2.7.1. *Comparing words with a minimum edit distance of 1*

When the words being compared (which have different lengths) have a minimum edit distance of 1, then we have two cases to consider:

(1) When the suggested word is shorter than the misspelled word

- ma**s**kinash (misspelled word)
- ma_kinash (suggestion)

The differing character here is 's' at position 2 which has been removed from the misspelled word to produce the suggestion. We are assigning the probability of the trigram {a, k, i} to the suggestion, thus assigning the probability of the trigram {c, x, c} where x is the character from the suggestion, found at the same position as the "extra" character in the misspelled word.

(2) When the suggested word is longer than the misspelled word

- p_rdor (misspelled word)
- p**ë**rdor (suggestion)

The differing character here is ë which is inserted at position 1 in the misspelled word, between p and r, to produce the suggestion. At this point, we will then find the trigram probability for {p, ë, r} and assign it to the suggested word.

2.7.2. *Comparing words with a minimum edit distance of 2*

There are 4 cases to be considered when the words being compared have different lengths and minimum edit distance of 2:

(1) When the suggested word is 2 characters longer than the misspelled word.
In this case the misspelled word is simply missing two characters (if it were to be a misspelled version of the suggestion).

At this point, we somewhat repeat the arguments from the analysis done in the comparison for words with minimum edit distance of 1. We locate the two additional characters x1 and x2 in the suggested word and then calculate the probability of the trigrams {c, x1, c} and {c, x2, c} where the c-s are the surrounding characters (whether c is an alphabetical character or one of the meta characters introduced in our solution makes no difference).

- m_sonj_tore (misspelled word)
- m**ë**sonj**ë**tore (suggestion)

In this case, we need the trigram probabilities for {m, ë, s} and {j, ë, t} so that we can assign a score/probability to the word.

(2) When the suggested word is 2 characters shorter than the misspelled word.
In this case, the misspelled word is two characters longer than the suggested words, meaning that the misspelled word was just the wrongly-spelled suggestion.

- bor**ë**bar**ë**dha (misspelled word)
- bor_bar_dha (suggestion)

In this case, we locate the two additional characters at positions p1 and p2 in the misspelled word, then we calculate the probabilities (on the trigrams formed in the suggested word {c, x(p1), c} and {c, x(p2), c} where c again represents the surrounding characters in the word and x is assumed to be a function that returns the character at position p in a given string. For this example, the trigrams would be: {r, b, a} and {r, d, h} since when the first 'ë' is removed, 'b' takes its place, and when the second 'ë' is removed, 'd' takes its place.

(3) When the suggested word is 1 character longer than the misspelled word.

When the length of the suggested word is only one character longer and the minimum edit distance is 2, then we have an additional character and a replaced character.

- mund_s**a**j (misspelled word)
- mund**ë**s**o**j (suggestion)

In this example, we will calculate probabilities for {d, ë, s} (to account for the added character 'ë') and {s, o, j} (to account for the replacement of 'a' with 'o').

(4) When the suggested word is 1 character shorter than the misspelled word.

- trashagimt**o**ri (misspelled word)
- trash_gimt**a**ri (suggestion)

In this example, the suggested word has one less character, the second 'a' from the misspelled word has been dropped, and 'o' in the misspelled word has been replaced with 'a' in the suggested word. The probability assigned to this word will be calculated from the trigram probabilities {h, g, i} and {t, o, r}.

2.7.3. *Comparing words with a minimum edit distance of 3*

There are 6 cases to be considered when the words being compared have different lengths and minimum edit distance of 3:

(1) When the suggested word is 3 characters longer than the misspelled word.

We locate the three additional characters x1, x2, and x3 in the suggested word and then calculate the probability of the trigrams {c, x1, c}, {c, x2, c}, and {c, x3, c} where the c-s are the surrounding characters.

- inxh_n_er_ (misspelled word)
- inxh**ini**er**e** (suggestion)

The probability assigned to this suggestion is calculated from the probabilities of these three trigrams: {h, i, n}, {n, i, e}, and {r, e, $}

(2) When the suggested word is 2 characters longer than the misspelled word.

When the length of the suggested word is only 2 characters longer and the minimum edit distance is 3, then we have 2 additional characters and a replaced character.

- manicion (misspelled word)
- m**u**nicion**et** (suggestion)

We can see that the second character has been replaced with a 'u' and two characters have been inserted at the end of the word. In this example, the suggestion will be assigned a probability by getting these 3 trigram probabilities: {m, u, n}, {n, e, t}, and {e, t, $}

(3) When the suggested word is 1 character longer than the misspelled word.

When the length of the suggested word is only 1 character longer and the minimum edit distance is 3, then we have 1 additional character and 2 replaced characters.

- aut**a**bus**i**_ (misspelled word)
- aut**o**bu**z**it (suggestion)

In this example, we can see that the misspelled word is one character shorter than the suggested word although it has a minimum edit distance of 3 with it. We have two characters that have been replaced and one character being added. The probability assigned to the shown suggestion would be calculated by the probabilities of these 3 trigrams: {t, o, b}, {u, z, i}, {i, t, $}.

(4) When the suggested word is 3 characters shorter than the misspelled word.

In this case, we locate the three additional characters at positions p1, p2, and p3 in the misspelled word, then we calculate the probabilities {c, x(p1), c}, {c, x(p2), c}, and {c, x(p3), c} where c again represents the surrounding characters in the word and x(p) is assumed to be a function that returns the character at position p in a given string (we are only dealing with strings comprised of only one word plus the word-start and word-end characters).

(5) When the suggested word is 2 characters shorter than the misspelled word.

When the length of the misspelled word is 2 characters longer and the minimum edit distance is 3, then we dropped only 1 character and replaced only 1 character.

- kalkulim (misspelled word)
- kal_u_am (suggestion)

The example above shows this case clearly. Two characters have been dropped (second 'k' and the second 'l') and we have a character replacement. The probability assigned to this suggestion will be calculated from these 3 trigram probabilities: {l, u, a}, {u, a, m}, and {a, m, $}.

(6) When the suggested word is 1 character shorter than the misspelled word.

When the length of the suggested word is only one character shorter and the minimum edit distance is 3, then we have dropped one character and replaced two characters.

- aut**a**bus**i** (misspelled word)
- aut**o**bu**z**_ (suggestion)

We can see that the suggested word is one character shorter than the misspelled word, but the minimum edit distance between the two is 3. We have two character replacements and the deletion of a character. The suggested word will then be assigned a probability by getting these two trigram probabilities: {t, o, b} and {u, z, $}. Given that the dropped character is the last one, it has then removed the third trigram possibility which was to be expected for a minimum edit distance of 3.

### 2.8. *Combining probabilities and minimum edit distance*

We now have two methods to rank (score) the list of generated results. One way is through the minimum edit distance results, where suggestions with a lower minimum edit distance are ranked higher. But, what if we have more than one suggestion with the same minimum edit distance value? In fact, this is the case more often than not. We could sort the suggestions alphabetically, but we have no other way of ranking suggestions. To overcome the lack of an objective ranking method, we augment the minimum edit distance results with the combined trigram probabilities of the replaced, inserted, or removed characters. This is the second method of ranking the suggestions. As the basic building block of a word, graphemes are combined and grouped only in certain ways, to each of which we can assign a probability.

As shown above, depending on the minimum edit distance and the length of the input word as compared to each suggestion, we can assign a value to each of these suggestions to be used as a ranking criterion. This value will be used to judge whether a word is more likely (based on observed usage of graphemes in the corpus) to be the replacement for the misspelled word. Each suggestion is now ranked by two criterions — minimum edit distance and the probability assigned by our algorithm. One step needs further research: whether the assigned probability is able to cause a suggestion with a higher minimum edit distance value than another suggestion to be ranked as a better suggestion if the probability assigned to it is higher as well.

### 2.9. *Augmentative typing systems*

Another area where a grapheme language model can be very helpful is on augmenting typing systems such as the ubiquitous cell phones and their useful short messaging capability. When typing an SMS message on a restricted pad such as a phone pad, with only 9 buttons which represent all letters of the alphabet, the selection between letters associated with the same number (from 1 to 9) are done via multiple taps for each letter. For example, typing the word “punoj” requires that we tap the 7 button once to produce “p”, then tap the 8 button twice to produce “u”, button 6 twice to produce “n”, button 6 three times to produce “o” and finally button 5 once to produce “j”. For the word “punoj”, the tapping sequence is thus “788666665”. Our augmentative system would reduce the typing of this word in a phone pad via rearranging the order of letters (so that the selection of a letter via multiple taps does not depend on an arbitrary order, but is arranged dynamically according to the language model we have built above) and produce a tapping sequence of “78665” due to these bigram probabilities:

| | | |
|---|---|---|
| $P(u\|p) = 0.03322$ | $P(t\|p) = 0.024478$ | $P(v\|p) = 0.000546$ |
| $P(n\|u) = 0.145076$ | $P(m\|u) = 0.040785$ | $P(o\|u) = 0$ |
| $P(o\|n) = 0.011946$ | $P(m\|n) = 0$ | $P(n\|n) = 0$ |
| $P(j\|o) = 0.081279$ | $P(k\|o) = 0.035342$ | $P(l\|o) = 0.046758$ |

## 3. Results

### Input word with a length of 3

The first table shows the list of suggestions as it is generated by the minimum edit distance algorithm.

| Input word/Suggestions | m.e.d | Probability assigned |
|---|---|---|
| **due** | | |
| de | 1 | 0.000250509739755261 |
| dhe | 1 | 0.00608199596013407 |
| dje | 1 | 0.000764213256721747 |
| dua | 1 | 0.00130011383923617 |
| duk | 1 | 0.00403352391099611 |
| duke | 1 | 0.000853001518913485 |

The second table shows the list of suggestions after the suggestions have been ranked by the assigned probabilities.

| Input word/Suggestions | m.e.d | Probability assigned |
|---|---|---|
| **due** | | |
| dhe | 1 | 0.00608199596013407 |
| duk | 1 | 0.00403352391099611 |
| dua | 1 | 0.00130011383923617 |
| duke | 1 | 0.000853001518913485 |
| dje | 1 | 0.000764213256721747 |
| de | 1 | 0.000250509739755261 |

m.e.d (due, dhe) = 1 and both strings have the same length: there is a character substitution (u replaced by h) and P{d, h, e} = 0.00608199596013407
m.e.d (due, duk) = 1 and both strings have the same length: there is a character substitution (e replaced by k) and P(u, k, $) = 0.00403352391099611
m.e.d (due, dua) = 1 and both strings have the same length: there is a character substitution (e replaced by a) and P{u, a, $} = 0.00130011383923617
m.e.d (due, duke) = 1 but the suggestion is one character longer: there is character addition (du{k}e) and P {u, k, e} = 0.000853001518913485
m.e.d (due, de) = 1 but the suggestion is one character shorter: there is a character removal. The 'u' has been dropped. P{d, e, $) = 0.000250509739755261

### Input word with a length of 5

The first table shows the list of suggestions as it is generated by the minimum edit distance algorithm.

| Input word/Suggestions | m.e.d | Probability assigned |
|---|---|---|
| **eshte** | | |
| ishte | 1 | 0.00262876676274825 |
| qeshte | 1 | 0.00103374905266095 |
| ashtu | | 6.1900429179395E−07 |
| deshe | | 1.01505240811767E−06 |
| ecte | | 1.00553003865164E−11 |

(*Continued*)

(*Continued*)

| Input word/Suggestions | m.e.d | Probability assigned |
|---|---|---|
| **eshte** | | |
| edhe | | 1.31768678385065E−06 |
| elite | | 1.45872242707193E−07 |
| ese | | 2.07255829446721E−06 |
| është | | 2.86260324583581E−06 |
| ethe | | 8.32679425007423E−08 |
| heshta | | 1.80167855735485E−06 |
| heshti | | 1.26515789463149E−06 |
| heshtje | | 1.64470526302094E−06 |
| ishe | | 6.91041469292992E−06 |
| kashte | | 1.36219154336138E−07 |
| kishte | | 6.63794621835648E−06 |
| kushte | | 1.35340321082356E−06 |
| reshti | | 9.77375197569394E−07 |
| shteg | | 6.78642680598394E−05 |
| shtet | | 6.78642680598394E−05 |
| shti | | 1.29447915055858E−05 |
| veshje | | 2.8998179125663E−06 |
| vishte | | 2.66063248227224E−07 |
| yshti | | 1.08597244174377E−08 |

The second table shows the list of suggestions after the suggestions have been ranked by the assigned probabilities.

| Input word/Suggestions | m.e.d | Probability assigned |
|---|---|---|
| **eshte** | | |
| ishte | 1 | 0.00262876676274825 |
| qeshte | 1 | 0.00103374905266095 |
| shtet | 2 | 6.78642680598394E−05 |
| shteg | 2 | 6.78642680598394E−05 |
| shti | 2 | 1.29447915055858E−05 |
| ishe | 2 | 6.91041469292992E−06 |
| kishte | 2 | 6.63794621835648E−06 |
| veshje | 2 | 2.8998179125663E−06 |
| është | 2 | 2.86260324583581E−06 |
| ese | 2 | 2.07255829446721E−06 |
| heshta | 2 | 1.80167855735485E−06 |
| heshtje | 2 | 1.64470526302094E−06 |
| kushte | 2 | 1.35340321082356E−06 |
| edhe | 2 | 1.31768678385065E−06 |
| heshti | 2 | 1.26515789463149E−06 |
| deshe | 2 | 1.01505240811767E−06 |
| reshti | 2 | 9.77375197569394E−07 |
| ashtu | 2 | 6.1900429179395E−07 |
| vishte | 2 | 2.66063248227224E−07 |
| elite | 2 | 1.45872242707193E−07 |
| kashte | 2 | 1.36219154336138E−07 |
| ethe | 2 | 8.32679425007423E−08 |
| yshti | 2 | 1.08597244174377E−08 |

A web-based system based on the above approach was developed, allowing several users to write text as they normally would and then spell-check the paragraph with our system. For each word that was not encountered in the dictionary, the system would generate a list of suggestions, ranked according to the probability score system as above. Every time should a user select one of the suggestions, the suggestion position in the list was recorded in a database.

## 4. Conclusion

The MLE grapheme $n$-gram model is preferable to the smoothed grapheme $n$-gram models since the latter allocate probability mass to morphologically impossible grapheme $n$-gram sequences. MLE grapheme models have thus a lower entropy and higher accuracy for grapheme sequence prediction. We chose to implement the MLE trigram model for augmenting our minimum edit distance results and a simple typing system based on the ubiquitous phone pad.

The suggestion accuracy rate was better overall when the minimum edit distance results were augmented with the trigram probabilities, but they were not as good as expected. In the meantime, a larger corpus of the language would provide for more accurate grapheme $n$-gram probabilities.

An augmented typing system based on grapheme $n$-gram language models on the other hand, is largely an effective system which is capable of shortening the input sequence in a phone pad, on average, by 30%.

## References

1. P. F. Brown, V. J. Dell Pietra, R. L. Mercer, S. A. Della Pietra and J. C. Lai, An estimate of an upper bound for the entropy of English, *Computational Linguistics* **18**(1) (March 1992) 31–40.
2. E. Charniak, Statistical language learning, *Language, Speech, and Communication* (MIT Press, Cambridge, MA, 1993).
3. S. F. Chen, Building probabilistic models for natural language, PhD Thesis, Harvard University (1996).
4. S. F. Chen and J. Goodman, An empirical study of smoothing techniques for language modeling, Harvard University Technical Report TR-10-98, MA 1998.
5. R. Jacobson and M. Halle, Fundamentals of Language, *Walter de Gruyter* (Amsterdam, 2002).
6. F. Jelinek, *Statistical Methods for Speech Recognition* (The MIT Press, Cambridge, MA 1998).
7. D. S. Jurafsky and J. H. Marting, *Speech and Language Processing* (Prentice Hall, Upper Saddle River, NJ, 2000).
8. C. D. Manning and H. Schütze, *Foundations of Statistical Language Processing* (The MIT Press, Cambridge, MA 2003).
9. C. E. Shannon, Prediction and entropy of printed English, *The Bell System Technical Journal* **30** (January 1951) 50–64.