PAPER

# Machine Learning Based English-to-Korean Transliteration Using Grapheme And Phoneme Information

**Jong-Hoon OH**[†a)] ***and*** **Key-Sun CHOI**[†b)]**,** ***Nonmembers***

**SUMMARY** Machine transliteration is an automatic method to generate characters or words in one alphabetical system for the corresponding characters in another alphabetical system. Machine transliteration can play an important role in natural language application such as information retrieval and machine translation, especially for handling proper nouns and technical terms. The previous works focus on either a grapheme-based or phoneme-based method. However, transliteration is an orthographical and phonetic converting process. Therefore, both grapheme and phoneme information should be considered in machine transliteration. In this paper, we propose a grapheme and phoneme-based transliteration model and compare it with previous grapheme-based and phoneme-based models using several machine learning techniques. Our method shows about 13∼78% performance improvement.

***key words:*** *Machine Transliteration, Machine learning, Information retrieval, Machine translation, Natural language processing*

## 1. Introduction

Machine transliteration is an automatic method to generate characters or words in one alphabetical system for the corresponding characters in another alphabetical system. Transliteration is used to translate proper names and technical terms especially from languages in Roman alphabets to languages in non-Roman alphabets such as from English to Korean, Japanese, Chinese and so on. One possible method to generate transliteration is based on the use of dictionaries, which contains English words and their possible transliterated forms. However, this is not a practical solution since proper nouns and technical terms, which are frequently transliterated, usually have rich productivity. Another method is machine transliteration. Some research has been done on machine transliteration from English to other languages including English to Japanese [1], English to Chinese [2] and English to Korean [3]–[8].

Machine transliteration plays an important role in natural language applications such as monolingual information retrieval [6], cross-lingual information retrieval [3] and machine translation [9]. From the viewpoint of monolingual information retrieval, machine transliteration bridges the gap between the transliterated localized form and its original form by generating all possible transliterated forms from each original form. In domain-specific literature, proper nouns and technical terms are either written in their original forms and or they are transliterated into local words. Because of the variations of localization, it is difficult to retrieve relevant documents when either a transliterated form or its corresponding original word is indexed without any relations established between them even though they indicate the same word. This is called "vocabulary mismatch", a problem caused by the non-standardized transliterations of a word from its original form. "Transliteration equivalence" refers to a set of the same words that include all possible transliterated forms and the original word. For example, Korean transliterations, 'de-i-ta', 'de-i-teo', and 'de-ta'*, are "transliteration equivalence" derived from the English word *data*. From the viewpoint of machine translation, machine transliteration generates translations of proper nouns and technical terms that are not registered in a translation dictionary.

In English-to-Korean transliteration, two machine transliteration methods have been studied: "grapheme-based transliteration method" and "phoneme-based transliteration method". "Graphemes" refer to the basic unit (or the smallest contrastive units) of written language: for example, English has 26 graphemes or letters, Korean has 24, and Japanese has 50. "Phonemes" are the simplest significant unit of sound (or the smallest contrastive units of the spoken language): for example, the /M/, /AE/, and /TH/** in *math*. In this paper, we will symbolize the grapheme-based transliteration method as $\psi_{GT}$, and the phoneme-based one as $\psi_{PT}$, for simplicity. Note that grapheme-based translit-

†The author is with Dept. of Computer Science, KAIST, 373-1 Guseong-Dong Yuseong-Gu Daejeon 305-701 Republic of Korea

a) E-mail: rovellia@world.kaist.ac.kr

b) E-mail: kschoi@world.kaist.ac.kr

*In this paper, Korean transliterated results or Korean transcriptions will be represented in a quotation mark (''). A symbol '-' will be used for indicating a syllable boundary.

**(www.cs.cmu.edu/∼laura/pages/arpabet.ps). ARPAbet symbol will be used for representing phonemes. ARPAbet is one of the methods used for coding phonemes into ASCII characters In this paper, we will denote phonemes and pronunciation with two slashes like so: /AH/. Pronunciation represented in this paper is based on *The CMU Pronunciation Dictionary* and *The American Heritage(r) Dictionary of the English Language*.

eration and phoneme-based transliteration will stand for the transliteration results generated by $\psi_{GT}$ and $\psi_{PT}$, respectively. $\psi_{GT}$ is a one-step converting process from an English grapheme to Korean graphemes; while $\psi_{PT}$ is a pipelined two-step converting process — from an English grapheme to phoneme and then again to a Korean grapheme. Because $\psi_{PT}$ needs phoneme information that cannot be directly acquired from the English word alone, one more converting step from the grapheme to the phoneme is necessary.

Transliterated results generated by the two methods are usually different because they generate transliterations based on different information. Though transliteration is a more phonetic process ($\psi_{PT}$) than orthographic one ($\psi_{GT}$) [9], we should consider both phoneme and grapheme information to obtain a more accurate transliterated result since many transliterations are generated by $\psi_{PT}$ as well as by $\psi_{GT}$[†] However, previous works choose either phoneme information or grapheme information as their focus, meaning they simplify a transliteration problem into either $\psi_{GT}$ or $\psi_{PT}$ and assume that one of them is able to cover transliterations generated by the other. Their approach is unnatural because there are transliterations either $\psi_{GT}$ or $\psi_{PT}$ fails to generate. For example, a standard Korean transliteration of *amylase*, 'a-mil-la-a-je', which is a grapheme-based transliteration, cannot be generated by $\psi_{PT}$. $\psi_{PT}$ generates 'a-mil-le-i-seu' based on /AE M AH L EY S/ rather than the standard, 'a-mil-la-a-je'. To overcome this problem, we should consider both grapheme and phoneme information for machine transliteration.

We propose a new machine transliteration method based on both grapheme and phoneme information symbolized as $\psi_{GPT}$. Our method has some advantages over $\psi_{GT}$ and $\psi_{PT}$ in that $\psi_{GPT}$ can handle a grapheme-based transliteration as well as a phoneme-based transliteration in the same framework since it considers phoneme information as well as grapheme information. $\psi_{GPT}$ can (1) reduce the ambiguity of transliterations, and (2) negotiate between the source language grapheme and the phoneme. First, $\psi_{GPT}$ can reduce ambiguities of transliterations which $\psi_{PT}$ or $\psi_{GT}$ cannot by using grapheme and its corresponding phonemes during the transliteration process. Transliteration is used to find the most relevant one among transliteration candidates derived from the grapheme or phoneme of source language words. Intuitively, if there are more transliteration candidates for certain graphemes or phonemes, it becomes more difficult to obtain relevant transliterations. As such, effectively reducing the ambiguities or the number of transliteration candidates is the key to making an effective machine transliteration system. From the viewpoint of reducing ambiguities in transliteration, $\psi_{GPT}$ has an advantage over $\psi_{GT}$ and $\psi_{PT}$ since the former uses the correspondence between grapheme and phoneme. For example, in $\psi_{PT}$, phoneme /AH/ produces high ambiguities since it can be mapped to almost every single vowel in the source and target languages; for example, underlined grapheme corresponds to /AH/: *cinem<u>a</u>*, *couns<u>e</u>l*, *<u>a</u>bil<u>i</u>ty* in English, and their Korean transliterations 'sine-m<u>a</u>', 'ka-un-s<u>e</u>l', '<u>eo</u>-bil-l<u>i</u>-ti'. If we know the correspondence between grapheme and phoneme in this context, we can more easily infer its transliteration, since a target language transliteration of /AH/ usually depends on the source language grapheme corresponding to /AH/. In an example of $\psi_{GT}$, a Korean transliteration result of grapheme *i* can be 'i', 'a-i' and so on. We can infer its correct transliteration by looking at its corresponding phoneme such as /IH/, /AY/ and so on. Second, $\psi_{GPT}$ can handle the negotiation of grapheme and phoneme information. Because the relevant target language grapheme can be derived from either source language grapheme or phoneme, it is important for a machine transliteration system to select one of them in a certain context. For example, the Korean grapheme 'e' can be derived only from English grapheme *e* in *n<u>e</u>o*, and the Korean grapheme 'a-i' can be derived only from the phoneme /AY/ in /M <u>AY</u> S IH N/. $\psi_{GPT}$ can determine the relevancy of the English grapheme or the phoneme by looking at their contexts.

## 2. Related Works

The previous works on English-to-Korean transliteration are classified into either grapheme-based or phoneme-based transliteration model and these will be described in Sect. 2.1 and 2.2, respectively.

### 2.1 Grapheme Based English-to-Korean Transliteration

Grapheme-based transliteration models are classified into statistical translation based model, decision tree based model, and transliteration network based model. The grapheme-based model is used simply to convert an English grapheme to Korean graphemes. In this section, we will describe the key points of each model.

#### 2.1.1 Statistical Translation Based Model

Lee [3], [5] proposed a grapheme-based English-to-Korean transliteration model based on the statistical translation model. Equation (1) attempts to generate a transliterated Korean word $K$ for a given source English word $E$. He defined "*pronunciation unit*" (PU) as graphemes that correspond to a phoneme. He then segments an English word into *pronunciation units*

[†]In our English-to-Korean transliteration test set [5], [10], we find that about 60% are a phoneme-based transliterations, while about 30% are grapheme-based ones. The others are transliterations generated by combining $\psi_{GT}$ and $\psi_{PT}$.

(PUs) and attempts to find the most relevant Korean graphemes corresponding to the *PUs*. For example, the English word '*board* (/B AO R D/)' was segmented into its pronunciation units like so, $b$(/B/): $oa$(/AO/): $r$(/R/): $d$(/D/); here *b, oa, r* and *d* are *PUs*. With them, an English word $E_i$ is represented as $E_i = epu_{i1}, epu_{i2}, \ldots, epu_{in}$ where $epu_{ij}$ is the $j^{th}$ *PU* of $E_i$. Sequences of Korean *PUs*, $kpu_{i1}, kpu_{i2}, , kpu_{in}$ are generated from $E_i$. Lee [3], [5] generated all possible English *PU* sequences and their corresponding Korean *PU* sequences. For example, *board* can be divided into *PU* sequences such as *b:oar:d, b:oa:r:d, b:o:a:r:d* †† and so on. All possible Korean *PUs* are then generated from them such as 'b:o:deu', 'b:o:reu:deu', 'b:o:a:reu:deu' and so on. The best result is then selected among them as a Korean transliteration word using Eq. (1).

$$argmax_K P(K|E) = argmax_K P(K)P(E|K) \quad (1)$$

$$P(K) \cong p(kpu_1) \prod_{i=2}^{n} p(kpu_i|kpu_{i-1})$$

$$P(E|K) \cong \prod_{i=1}^{n} p(epu_i|kpu_i)$$

Kim [4] expanded Lee's [3], [5] research by considering more information in estimating $P(E|K)$, as in Eq. (2) He used both additional information – Korean *PUs* $kpu_{i-1}$ and $kpu_{i+1}$ – and a neural network, to approximate $P(E|K)$.

$$P(E|K) \cong \prod_{i=1}^{n} p(epu_i|kpu_{i-1}, kpu_i, kpu_{i+1}) \quad (2)$$

This approach is an effective model of the transliteration process with *PU*, which represents a phonetic property of a source language word. But errors that occur in *PU* segmentation are hurdles in transliteration, and generating all possible *PUs* for each side is a time consuming process: If the total number of English *PUs* is *N* and the average number of Korean *PUs* generated by each English *PU* is *M*, the total number of generated Korean *PU* sequences will be about $N \times M$.

### 2.1.2 Decision Tree Based Model

Kang [6], [8] proposed an English grapheme-to-Korean grapheme conversion method based on decision trees. Seven contextual graphemes — the left three, right three, and current English grapheme — are used for determining Korean graphemes corresponding to English graphemes. For each English grapheme, its corresponding decision trees are constructed. Table 1 shows an example of a transliteration process for *data*. (*E*) represents the current English grapheme; (*L1, L2, L3*) represent left contexts; (*R1, R2, R3*) represent right contexts; *K* represents generated Korean graphemes by decision trees; and $ represents the start and end of a word.

**Table 1** An example of decision tree based English-to-Korean transliteration

| L3 | L2 | L1 | (E) | R1 | R2 | R3 | | K |
|---|---|---|---|---|---|---|---|---|
| $ | $ | $ | d | a | t | a | → | 'd' |
| $ | $ | d | a | t | a | $ | → | 'e-i' |
| $ | d | a | t | a | $ | $ | → | 't' |
| d | a | t | a | $ | $ | $ | → | 'a' |

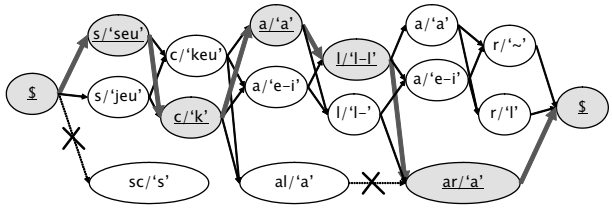


**Fig. 1** A Korean transliteration network for an English word *scalar*

The merit in this approach is its consideration of wider contextual information. Unlike other grapheme based methods, it takes into account the left three and right three contexts. However, one drawback of the method is that it does not consider any phonetic aspects of transliteration.

### 2.1.3 Transliteration Network Based Model

Kang & Kim [7] proposed an English-Korean transliteration model based on a finite transition network. They also focused on a direct grapheme-by-grapheme conversion method from English graphemes to Korean graphemes. All possible grapheme sequences are generated to make a transliteration network, as in Fig. 1. Each node in the network is composed of more than one English grapheme (including grapheme and chunk of graphemes) and its corresponding Korean graphemes. Each arc in the network is weighted with Eq. (3). "Context" refers to the historical information of English graphemes, where a specific Korean grapheme ("output") was generated. The best optimal path is found by the highest sum of weights, with a Viterbi algorithm [11] and a Tree-Trellis algorithm [12].

$$weight(context, output) = \frac{C(output)}{C(context)} \quad (3)$$

The advantage here are: (1) consideration of the phonetic aspect called 'chunks of graphemes' as well as graphemes; and (2), unlike [3], [5], it performs an one-step procedure using a transliteration network as opposed to a two-step pipelined procedure (grapheme

†† ':' will be used as a PU boundary

chunking then transliteration). However, some disadvantages include: (1) a somewhat limited context size (2), although it considers the phonetic aspect on a grapheme level, the information is not enough in that further information, such as phonemes corresponding to a grapheme (or a chunk of graphemes), is necessary to generate relevant target language transliterations.

### 2.2 Phoneme-Based English-to-Korean Transliteration Model

Lee [5] proposed a phoneme-based English-to-Korean transliteration model. His model generates Korean transliterations with two-step converting procedures; an English PU-to-phoneme converting procedure that uses a statistical translation based model described in Sect. 2.1.1; and a phoneme-to-Korean PU converting procedure that relies on English-to-Korean Standard Conversion Rules (EKSCR) [13] which describes transliteration rules from English phonemes to Korean graphemes using phoneme as a condition, and outputs Korean graphemes if the condition met.

This approach suffers two main disadvantages: (1) error propagation, and (2) limitations in EKSCR. First, an English PU-to-phoneme converting procedure usually makes errors and the errors which are then propagated to the next step. The propagated errors make it difficult for a transliteration system to generate relevant transliterations. Second, the EKSCR does not contain enough rules to generate relevant Korean transliterations since its main focus is on a methods of mapping from one English phoneme to one Korean grapheme without the context of graphemes and phonemes. For example, the English word *board* and its pronunciation /B AO R D/ are transliterated into 'bo-reu-deu' by EKSCR. However, the correct one, 'bo-deu', can be acquired when their contexts are considered. Due to these limitations, this phoneme-based model shows worse performance than the grapheme-based one described in Sect. 2.1.1.

### 2.3 Summary

Most previous works are focused on $\psi_{GT}$ rather than $\psi_{PT}$, because the former has more advantages than the latter. First, unlike $\psi_{PT}$, $\psi_{GT}$ does not require any knowledge about pronunciation, meaning while $\psi_{GT}$ requires only the [source language grapheme→target language grapheme] conversion patterns, $\psi_{PT}$ needs the [source language grapheme→phoneme] conversion patterns and the [pronunciation→target language grapheme] conversion patterns. Second, there is an error propagation phenomenon in $\psi_{PT}$ because it is composed of two steps. Errors in the first step usually make it difficult to generate correct transliterations in the second step. This is the main reason why the performance of $\psi_{PT}$ is lower than that of $\psi_{GT}$, though many transliterations are phoneme-based transliterations rather than grapheme-based transliterations.

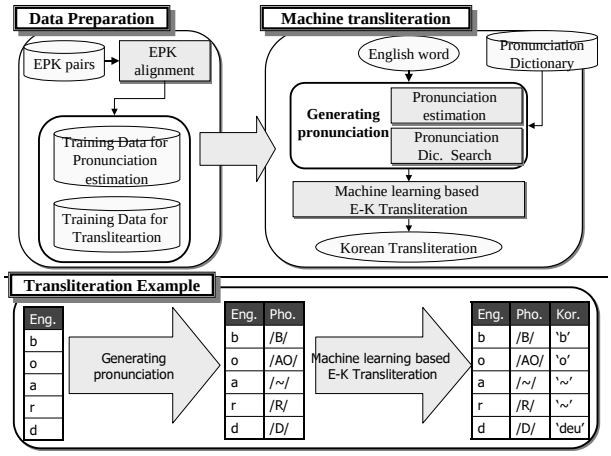


**Fig. 2** The overall system architecture

However, $\psi_{GT}$ also shows limitations. Though it is a relatively simple and effective model, it hardly considers phonetic features such as phonemes. This causes errors when phoneme rather than grapheme provide important clues for generating the right transliteration.

Because of the nature of the two models, they cannot consider both grapheme and phoneme simultaneously. This makes it difficult to generate relevant transliteration, particularly when one model encounters source language words that should be transliterated through negotiation between the source language grapheme and phoneme or by a different transliteration model. Our method, $\psi_{GPT}$, relieves the problems described above. Although, like $\psi_{PT}$, $\psi_{GPT}$ requires pronunciation knowledge, the latter can shield against some errors caused by error propagation using grapheme information corresponding to phoneme. Our model can handle transliterations that either $\psi_{GT}$ or $\psi_{PT}$ fail to generate.

## 3. Machine Learning Based English-to-Korean Transliteration Model

The proposed overall system architecture of our English-to-Korean machine transliteration system is shown in Fig. 2. Our system is composed of two main parts – data preparation and machine transliteration.

The goal of data preparation is to create training data for our machine transliteration model. To achieve the goal, we devise an EPK alignment algorithm. The EPK alignment (Sect. 3.1) algorithm recognizes the correspondence among the "English grapheme", "Phoneme" and the "Korean grapheme". Because we model a transliteration process as a converting process from English grapheme and its corresponding phoneme to Korean grapheme, the correspondences and their contexts acquired from the EPK alignment give an ev-

idence where certain English grapheme and phoneme are transliterated into Korean graphemes in a specific context. Data acquired from the data preparation part is used for training data of "estimating pronunciation" and "generating transliteration" in the machine transliteration part.

The machine transliteration part is a key part of our system. The main goal is to generate the most relevant Korean transliterated word. This part is composed of "generating pronunciation" step and "generating transliteration" step. The *generating pronunciation* step is to generate pronunciations using pronunciation dictionary and pronunciation estimation that is based on machine learning algorithms. The *generating transliteration* step is to generate Korean transliterations based on grapheme and phoneme information acquired from the previous step.

The bottom side of Fig. 2 shows our machine transliteration process with an example *board*. First, phonemes /B/, /AO/, /∼/, /R/ and /D/ that are corresponds to *b, o, a, r,* and *d*, respectively, are generated by the generating pronunciation step. Here, /∼/ means silence. Note that the result is a sequence of the [English grapheme→phoneme] relations, because our system utilizes the [English grapheme→phoneme] conversion patterns. Next, Korean transliterations are generated with the similar manner. A Korean transliteration result is a sequence of the [English grapheme, phoneme→Korean grapheme] relations. Here, '∼' is null grapheme. By concatenating the sequence of Korean graphemes, we can get the Korean transliteration 'bo-deu'. From Sect. 3.2 to Sect. 3.3, we will describe details of our machine transliteration part.

### 3.1 EPK Alignment: Making Training Data For English-to-Korean Machine Transliteration

For an English-to-Korean machine transliteration, we need training data in order to make conversion patterns. The training data should offer correspondence among English grapheme, phoneme, and Korean grapheme because we need conversion patterns, [English grapheme, phoneme → Korean grapheme] in $\psi_{GPT}$ and [English grapheme → phoneme] for estimating pronunciation.

Because constructing training data by hand is time consuming and it is difficult to maintain consistency, we devise an algorithm to automatically uncover correspondences. We devise three alignment algorithms called EP alignment, PK alignment, and EPK alignment, named according to the correspondences dealt with by each algorithm. The EP alignment algorithm finds the most phonetically probable correspondence between English grapheme and phoneme; while the PK alignment algorithm finds the most phonetically probable correspondence between phoneme and Korean grapheme; and the EPK alignment is a hybrid process of EP alignment and PK alignment that deals with correspondences among English grapheme, phoneme and Korean grapheme by combining results of EP and PK alignment using phoneme information as a pivot.

**Table 2** One possible alignment result generated by the EP alignment algorithm

| English grapheme | b | o | a | r | d |
|---|---|---|---|---|---|
| Pronunciation | /B/ | /AO/ | /∼/ | /R/ | /D/ |

**Table 3** One possible alignment generated by the PK alignment algorithm

| Pronunciation | /B/ | /AO/ | /R/ | /D/ |
|---|---|---|---|---|
| Korean grapheme | 'b' | 'o' | '∼' | 'deu' |

**Table 4** The EPK alignment result derived from Table 2 and 3

| English grapheme | b | o | a | r | d |
|---|---|---|---|---|---|
| Pronunciation | /B/ | /AO/ | /∼/ | /R/ | /D/ |
| Korean grapheme | 'b' | 'o' | '∼' | '∼' | 'deu' |

Consider an example when the English word is *board*, its corresponding pronunciation is /B AO R D/, and its corresponding Korean word is 'bo-deu'. We can obtain the alignment results described in Table 2, 3, and 4 using each alignment algorithm. The EPK alignment algorithm uses results generated by EP alignment and PK alignment algorithms. Phonemes are used as a pivot. In other words, it uses a mapping relation of the same phoneme in an EP alignment result and a PK alignment result. For example, in Table 2 and 3, the English grapheme *b* is linked to phoneme /B/, and phoneme /B/ is linked to Korean grapheme 'b', respectively. We can then find an EPK alignment result for English grapheme *b* as described in Table 4 by a transitive rule like (*b* → /B/ and /B/ → 'b' then *b*→ /B/ → 'b').

Because an EP alignment algorithm and a PK alignment algorithm share the same framework, in this paper we will limit our discussion to the EP alignment algorithm. The EP and PK alignment algorithm is based on Levenshtein distance (LD) algorithm [14]. LD is a measure of the similarity between two strings; source string *s* and target string *t*. The distance is the number of *deletions*, *insertions*, or *substitutions* required to transform *s* into *t*.

Similar to LD, EP alignment and PK alignment attempt to find a way to phonetically transform *s* into *t* with minimal cost. To model alignment algorithms, we introduce the new operations and cost schemes into LD. Instead of *deletion*, *insertion* and *substitution* operations used in LD, we introduce *match* (M), *source side skip* (SS), and *target side skip* (TS) operations. Alignment units in both sides aligned with a *match* operation mean that they have had correspondence. Those aligned with *SS* (*TS*) mean that the current alignment unit in the source side (the target side) is skipped and

**Table 5** Cost schemes for the EP and PK alignment

| Operation | Condition | Cost |
|---|---|---|
| Match | $s_i$ and $t_j$ are phonetically similar | 0 |
| | $s_i$ is a semi-vowel and $t_j$ is a vowel | 30 |
| | $s_i$ is a vowel and $t_j$ is a semi-vowel | 30 |
| | $s_i$ and $t_j$ are phonetically dissimilar consonants | 240 |
| | $s_i$ and $t_j$ are phonetically dissimilar vowels | 100 |
| | otherwise | 250 |
| Skip | All conditions | 40 |

that the target side (the source side) tends to be aligned with the next alignment unit in the source side (the target side). In other words, *SS* (or *TS*) ignores the current alignment unit in the source side (or the current one in the target side). Since the behavior of "match", "source side skip", and "target side skip" are similar to that of "substitution", "deletion" and "insertion", respectively, we can use the framework of LD for our alignment algorithm.

While LD assigns the same cost (say 1) to all operations (*deletion, insertion*, and *substitution*), we assign a different cost to each operation according to properties of alignment unit and operation type. Table 5 shows manually constructed cost scheme that we used. Our cost scheme is based on cost scheme of Kang's alignment algorithm [8]. The basic philosophies of our cost scheme are as follows:

- it assigns a low cost between phonetically similar alignment units
- it prefers a *match* operation to a *skip* operation when $s_i$ and $t_j$ are phonetically similar
- it prefers a *skip* operation to a *match* operation when $s_i$ and $t_j$ are phonetically dissimilar.

Manually constructed similarity tables are used for determining whether two alignment units ($s_i$ and $t_j$) are phonetically similar or not. For example, EP similarity table describes that English grapheme *b* and phoneme /B/ is phonetically similar. PK similarity table contains that phoneme /DH/ is phonetically similar to Korean grapheme 'd' and 'j'.

An alignment process is similar to the LD algorithm. First, a matrix for alignment ($D$) between input strings ($s$ and $t$) is constructed with the initial value (Eq. (4)).

$$\begin{aligned} d[i,0] &= i \times 300; \\ d[0,j] &= j \times 300; \end{aligned} \tag{4}$$

Second, the algorithm assigns the most relevant operation and its cost to each cell ($d[i,j]$) in $D$ (Eq. (5)).

$$\begin{aligned} d[i,j].cost &= min(a,b,c); \\ d[i,j].op &= op(argmin(a,b,c)); \\ a &= d[i-1,j].cost \\ &\quad +cost(S(s_t,t_j)); op = SS \end{aligned} \tag{5}$$

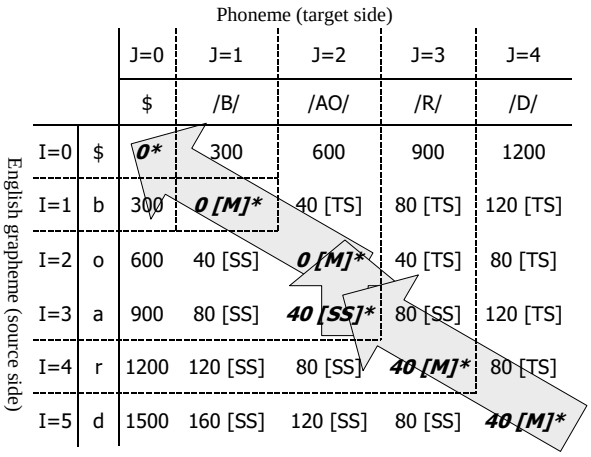


**Fig. 3** An example of an EP alignment

$$\begin{aligned} b &= d[i,j-1].cost \\ &\quad +cost(S(s_t,t_j)); op = TS \\ c &= d[i-1,j-1].cost \\ &\quad +cost(M(s_t,t_j)); op = M \end{aligned}$$

Finally the algorithm tries to retrieve the best alignment result using the cell's operation and its cost calculated in the previous steps.

Figure 3 shows an alignment example between the English word *board*, and its pronunciation, /B AO R D/, where $ is dummy letter; it represents the start of strings and it's index is 0. In Fig. 3, each cell contains its minimal cost derived from its operation. The best alignment result can be acquired by retrieving the cells represented with '$*$' from $d[n,m]$ to $d[0,0]$ where $n$ and $m$ are the length of $s$ and $t$, respectively. The next cell that should be retrieved is determined by the cell's operation. In the case of $M$, the algorithm tries to retrieve a cell from $d[i,j]$ to $d[i-1,j-1]$. For *SS* (or *TS*), the next cell is $d[i-1,j]$ (or $d[i,j-1]$). With these retrieval rules, we can obtain the most relevant path, ($d[5,4] \rightarrow d[4,3] \rightarrow d[3,2] \rightarrow d[2,2] \rightarrow d[1,1] \rightarrow d[0,0]$) in Fig. 3, that makes the lowest sum of costs.

Kang *et al.*[8] proposed an English-to-Korean alignment algorithm. Though the alignment algorithm shows high performance, it has high time complexity ($O(k^{(n+m)})$), where $k$ is a kind of operation, $n$ is the length of input string in one side and $m$ is the length of input string in the other side. In order to consider all possible correspondences, Kang *et al.*[8] constructs an $n-way$ branching search tree and then finds the best result using a depth first search. Similar to our alignment algorithm, manually constructed cost scheme and similarity table determine the best alignment result making lowest sum of costs. However, the search space increases exponentially when the length of input strings becomes longer. This is the reason why time complexity of the alignment algorithm is $O(k^{(n+m)})$. The time complexity makes it difficult to construct large-scale

training data. Our alignment algorithm reduces the time complexity without loss of performance because we adopt Kang's cost scheme and similarity table that describes phonetically similar alignment pairs. Because we only need to calculate operation and its penalty score in the $n \times m$ matrix to get an alignment result, the time complexity of our algorithm becomes $O(n \times m)$.

Fujii *et al.*[15] proposed an English-to-Japanese alignment algorithm similar to our alignment algorithm. Given the English and Japanese katakana words, the algorithm tries to find the correspondence between English grapheme and katakana character using manually constructed similarity table and cost scheme between English grapheme and grapheme in katakana characters. The main difference between our alignment algorithm and Fujii's alignment algorithm is the level of units in the alignment pairs that each algorithm tries to find. The Fujii's algorithm tries to find the "grapheme-to-syllable (or grapheme-to-katakana character) alignment pair" while our focus is "grapheme-to-grapheme (or grapheme-to-phoneme) alignment pair". Therefore, our alignment results are more fine-grained results than Fujii's. For example, the Fujii's algorithm aligns English-Korean transliteration pair, <text, 'tek-seu-teu'>,as <tex, 'tek seu'>, and <t, 'teu'>; while ours more finely aligns it as <t, 't'>, <e, 'e'>, <k, 'k seu'>, and <t, 'teu'>. With the same manner, the Fujii's aligns English-Japanese transliteration pair, <text, 'te-ki-su-to'>, as <te, 'te'>, <x, 'ki su'>, and <t, 'to'>; while ours more finely align it as <t, 't'>, <e, 'e'>, <k, 'ki su'>, and <t, 'to'>.

### 3.2 Grapheme and Phoneme Based English-to-Korean Transliteration Model ($\psi_{GPT}$)

In this subsection, we will describe the overall framework of grapheme and phoneme-based transliteration model ($\psi_{GPT}$). $\psi_{GPT}$ is composed of two functions; $\delta_p$ and $\delta_t$ ($\psi_{GPT}$: $\delta_p \times \delta_t$). $\delta_p$ refers to a "generating pronunciation function" and it outputs more than one phoneme corresponding to input English grapheme $s_i$. It can be represented as $\{\delta_p(s_i); S \rightarrow 2^P\}$ where $P$ is a set of phonemes, $S$ is a set of English graphemes. $\delta_t$ is a "generating transliteration function" and it outputs Korean graphemes for an English grapheme and phoneme pair: $\{\delta_t(s_i, \delta_p(s_i)); \delta_t : S \times 2^P \rightarrow 2^T\}$ where $T$ is a set of Korean graphemes. Figure 4 summarizes the overall framework of $\psi_{GPT}$.

Basically the two functions are trained by a supervised learning method. To train each component function, we should define the features that represent training instances. Table 6 shows four feature types, $f_S$, $f_P$, $f_{GS}$, and $f_{GP}$; $f_S$ , $f_P$ , $f_{GS}$ and $f_{GP}$ represent English grapheme feature, phoneme feature, generalized features of $f_S$, and generalized feature of $f_P$, respectively. According to component functions, we use different feature types. For example, $\delta_p(s_i)$ uses ($f_S$, $f_{GS}$); and $\delta_t(s_i, \delta_p(s_i))$ uses ($f_S$, $f_P$, $f_{GS}$, $f_{GP}$).

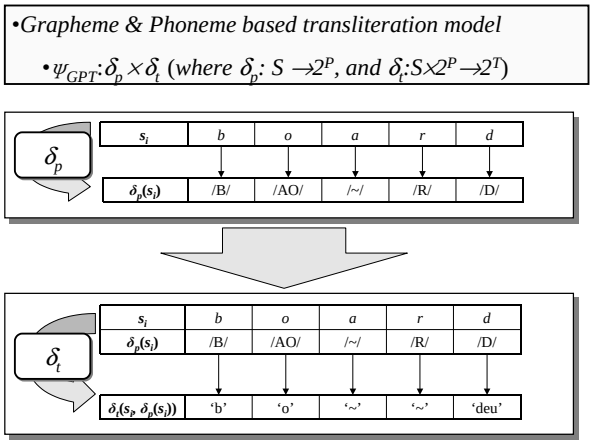


**Fig. 4** The overall framework of $\psi_{GPT}$

**Table 6** The feature types used for our machine transliteration model

| Feature type | Description | Possible feature values |
|---|---|---|
| $f_S$ | English Graphemes | 26 alphabets (a to z) |
| $f_P$ | Phonemes | Phonemes in $2^P$ (/AA/, /AE/, etc.) |
| $f_{GS}$ | Generalized $f_S$ | Consonant (C), Vowel (V) |
| $f_{GP}$ | Generalized $f_P$ | Consonant (C), Vowel(V) Semi-vowel (SV), silence (∼) |

From Sect. 3.2.1 to Sect. 3.2.2, we will describe some details of the component functions. The following machine learning algorithms — the maximum entropy model [16], decision tree [17], [18], and memory-based learning [19] — are used for training each component function ($\delta_p$ and $\delta_t$) and will be given in Sect. 3.3.

#### 3.2.1 Generating Pronunciation

Generating pronunciation is composed of two steps. The first step involves a search in a pronunciation dictionary which contains English words and their pronunciation. In this paper, we use *The CMU Pronouncing Dictionary* [20], which contains 120,000 English words and pronunciation pairs. The second step involves estimating pronunciation. If an English word is not registered in the dictionary, we must estimate its pronunciation.

Let $SW = s_1, s_2, \ldots, s_n$ be an English word, and $PSW = \delta_p(s_1), \delta_p(s_2), \ldots, \delta_p(s_n)$ be $SW$'s pronunciation, where $s_i$ represents the $i^{th}$ grapheme and $\delta_p(s_i)$ represents phonemes corresponding to $s_i$. Estimating pronunciation is a task that involves finding the most relevant phoneme among a set of all possible phoneme candidates, which can be generated from grapheme $s_i$. Training data is acquired from the EP alignment result of *The CMU Pronouncing Dictionary*. Features used

| Index | Feature type | L3 | L2 | L1 | C0 | R1 | R2 | R3 | | $\delta_p$(C0) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $f_S$ | \$ | \$ | \$ | *b* | o | a | r | → | */B/* |
| | $f_{GS}$ | \$ | \$ | \$ | *C* | V | V | C | | |
| 2 | $f_S$ | \$ | \$ | b | *o* | a | r | d | → | */AO/* |
| | $f_{GS}$ | \$ | \$ | C | *V* | V | C | C | | |
| 3 | $f_S$ | \$ | b | o | *a* | r | d | \$ | → | */~/* |
| | $f_{GS}$ | \$ | C | V | *V* | C | C | \$ | | |
| 4 | $f_S$ | b | o | a | *r* | d | \$ | \$ | → | */R/* |
| | $f_{GS}$ | C | V | V | *C* | C | \$ | \$ | | |
| 5 | $f_S$ | o | a | r | *d* | \$ | \$ | \$ | → | */D/* |
| | $f_{GS}$ | V | V | C | *C* | \$ | \$ | \$ | | |

**Fig. 5** An example of $\delta_p$ for *board*

for estimating pronunciation are the current grapheme $s_i$, its context in a window ($s_{i\pm k}$), and their generalized features, where $k$ is the context window size. We set $k$=*3* which shows the best result.

Figure 5 shows an example of estimating pronunciation for the English word *board*. In our pronunciation estimator, $SW$ is represented as follows:

$$\begin{aligned} X_P &= \{x_P(s_1), x_P(s_2), \ldots, x_P(s_n)\} \\ x_P(s_i) &= \{L3(F_P), L2(F_P), L1(F_P), C0(F_P), \\ &\quad R1(F_P), R2(F_P), R3(F_P)\} \\ F_P &= < f_S, f_{GS} > \end{aligned}$$

In Fig. 5, (*L1, L2, L3*) represents the left three contexts; and (*R1, R2, R3*) represents the right three contexts. *C0* means the current English grapheme which corresponding phoneme should be estimated, and $\delta_p(C0)$ means the estimated phoneme of *C0*. \$ is a symbol for representing the start or end of words. The result can be interpreted as follows. The most relevant phoneme that can be generated from C0={$f_S$=*b*, $f_{GS}$=C} is /B/ in the context of *L3, L2, L1, C0, R1, R2* and *R3*. Others are generated in the same manner. Thus, we can get the pronunciation of *board* as /B AO R D/ by concatenating generated phoneme sequences, $\delta_p(s_1 = b), \delta_p(s_2 = o), \ldots, \delta_p(s_n = d)$. Note that /~/ is ignored in concatenating phoneme sequences because it is silent.

### 3.2.2 Generating Korean Transliterations

Let $SW = s_1, s_2, \ldots, s_n$ be a source language word, $PSW = \delta_p(s_1), \delta_p(s_2), \ldots, \delta_p(s_n)$ be $SW$'s pronunciation, and $TW = \delta_t(gp_1), \delta_t(gp_2), \ldots, \delta_t(gp_n); gp_i = (s_i, \delta_p(s_i))$ be a target language word of $SW$, where $s_i$, $\delta_p(s_i)$, and $\delta_t(gp_i)$ represent the $i^{th}$ source language grapheme, $s_i$'s corresponding phoneme, and $gp_i$'s corresponding target language grapheme, respectively. Generating transliteration function, $\delta_t$, finds the most probable target language grapheme among a set of all possible target language grapheme candidates, which can be derived from $gp_i = (s_i, \delta_p(s_i))$. Training data is acquired from an EPK alignment result described in Sect. 3.1. We use four feature types: $f_S$, $f_P$, $f_{GS}$, and $f_{GP}$.

Figure 6 shows an example of $\delta_t$. *SW (board)* and $PSW$ are represented as follows.

$$\begin{aligned} X_T &= \{x_T(s_1), x_T(s_2), \ldots, x_T(s_n)\} \\ x_T(s_i) &= \{L3(F_T), L2(F_T), L1(F_T), C0(F_T), \\ &\quad R1(F_T), R2(F_T), R3(F_T)\} \\ F_T &= < f_S, f_P, f_{GS}, f_{GP} > \end{aligned}$$

In Fig. 6, (*L1, L2, L3*) represents the left three contexts and (*R1, R2, R3*) represents the right three contexts. *C0* means the current source language grapheme and phoneme to be transliterated, and $\delta_t(C0)$ means generated target language graphemes (Korean graphemes). The result in Fig. 6 can be interpreted as follows. The most probable Korean grapheme of *C0*={$f_S$=*b*, $f_P$=/B/, $f_{GS}$=C, $f_{GP}$=C} is $\delta_t(C0)$='b' in the context of *L1, L2, L3, C0, R1, R2* and *R3*. Other Korean graphemes can be generated by the same manner. We can get a Korean transliteration of *board* and /B AO R D/ as 'bo-deu'. Note that '~' is ignored because it is a null grapheme.

## 3.3 Machine Learning Methods For English-to-Korean Transliteration

In this subsection we will describe a way of modeling component functions using three machine learning methods (maximum entropy model, decision tree and memory-based learning).

### 3.3.1 Maximum Entropy Model

The maximum entropy model is a widely used probability model, which can incorporate heterogeneous information effectively. It does not require independence assumption to divide an event into sub-events and it enables flexible modeling with many overlapping features [21], [22]. An event $ev$ is usually composed of a target event ($t$) and a history event ($h$), say $ev = (t, h)$. In the maximum entropy model, event $ev$ is represented by a bundle of feature functions, $f_i(ev)$, which represent the existence of a certain characteristic in event $ev$. A feature function is a binary valued function. It is activated ($f_i(ev) = 1$) when it meets its activating condition otherwise it is deactivated ($f_i(ev) = 0$) [21], [22].

The maximum entropy model, $p_M$, is a log-linear model that gives a conditional probability of event $ev$ as described in Eq. (6), where $\tau(h)$ is a set of targets

| Index | Feature type | L3 | L2 | L1 | C0 | R1 | R2 | R3 | | $\delta_t$(C0) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $f_S$ | \$ | \$ | \$ | ***b*** | o | a | r | → | **'b'** |
| | $f_P$ | \$ | \$ | \$ | ***/B/*** | /AO/ | /~/ | /R/ | | |
| | $f_{GS}$ | \$ | \$ | \$ | ***C*** | V | V | C | | |
| | $f_{GP}$ | \$ | \$ | \$ | ***C*** | V | /~/ | C | | |
| 2 | $f_S$ | \$ | \$ | b | ***o*** | a | r | d | → | **'o'** |
| | $f_P$ | \$ | \$ | /B/ | ***/AO/*** | /~/ | /R/ | /D/ | | |
| | $f_{GS}$ | \$ | \$ | C | ***V*** | V | C | C | | |
| | $f_{GP}$ | \$ | \$ | C | ***V*** | /~/ | C | C | | |
| 3 | $f_S$ | \$ | b | o | ***a*** | r | d | \$ | → | **'~'** |
| | $f_P$ | \$ | /B/ | /AO/ | ***/~/*** | /R/ | /D/ | \$ | | |
| | $f_{GS}$ | \$ | C | V | ***V*** | C | C | \$ | | |
| | $f_{GP}$ | \$ | C | V | ***/~/*** | C | C | \$ | | |
| 4 | $f_S$ | b | o | a | ***r*** | d | \$ | \$ | → | **'~'** |
| | $f_P$ | /B/ | /AO/ | /~/ | ***/R/*** | /D/ | \$ | \$ | | |
| | $f_{GS}$ | C | V | V | ***C*** | C | \$ | \$ | | |
| | $f_{GP}$ | C | V | /~/ | ***C*** | C | \$ | \$ | | |
| 5 | $f_S$ | o | a | r | ***d*** | \$ | \$ | \$ | → | **'deu'** |
| | $f_P$ | /AO/ | /~/ | /R/ | ***/D/*** | \$ | \$ | \$ | | |
| | $f_{GS}$ | V | V | C | ***C*** | \$ | \$ | \$ | | |
| | $f_{GP}$ | V | /~/ | C | ***C*** | \$ | \$ | \$ | | |

**Fig. 6** An example of $\delta_t$ for *board*

observable with history $h$, $\alpha_i$ is a model parameter, and $Z_h$ is the normalization factor. $\alpha_i$ represents a weight of feature function $f_i(ev) = f_i(t,h)$.

$$p_M(t|h) = \frac{1}{Z_h}\prod_i \alpha_i^{f_i(t,h)} \tag{6}$$
$$Z_h = \sum_{t' \in \tau(h)} \prod_i \alpha_i^{f_i(t,h)}$$

The maximum entropy model yields a probability distribution that maximizes the likelihood of training data being given a set of feature functions. In other words, it yields model parameters that maximize the model's likelihood. GIS (Generalized Iterative Scaling), IIS (Improved Iterative Scaling), and L-BFGS (Limited-Memory Variable Metric) algorithm are well-known algorithms used to estimate model parameters in the maximum entropy model. Among them, L-BFGS has been found to be effective for parameter estimation [16].

$$t^* = argmax_{t \in 2^P} p_M(\delta_p)(t|h) = \frac{1}{Z_h}\prod_i \alpha_i^{f_i(t,h)} \tag{7}$$
$$t^* = argmax_{t \in 2^T} p_M(\delta_t)(t|h) = \frac{1}{Z_h}\prod_i \alpha_i^{f_i(t,h)}$$
$$Z_h = \sum_{t' \in \tau(h)} \prod_i \alpha_i^{f_i(t,h)}$$

$$f_i(t,h) = \begin{cases} 1, & if\ h(s_i) = e_j\ \& \\ & \quad t(\delta_p(s_i)) = p_k \\ 0, & otherwise \end{cases} \tag{8}$$
$$f_j(t,h) = \begin{cases} 1, & if\ h(\delta_p(s_i)) = p_j\ \& \\ & \quad t(\delta_t(s_i, \delta_p(s_i))) = t_k \\ 0, & otherwise \end{cases}$$

$\delta_p$ and $\delta_t$ based on the maximum entropy model can be represented as Eq. (7). Given the history event or activated feature function, $\delta_p$ finds the most probable phonemes, which maximize $p_M(\delta_p)$, and $\delta_t$ finds the most probable target language graphemes, which maximize $p_M(\delta_t)$. A target event of $\delta_p$ is a phoneme in a set of $2^P$ and that of $\delta_t$ is a target language grapheme in a set of $2^T$. A feature function in each module is constructed using defined features: ($f_S$, $f_{GS}$) for $\delta_p$ and ($f_S$, $f_P$, $f_{GS}$, $f_{GP}$) for $\delta_t$. Equation (8) shows some examples of the feature functions that we used. We use an L-BFGS algorithm for estimating a model parameter, and Zhang's maximum entropy modeling tool to implement $\delta_p$ and $\delta_t$ [16].

### 3.3.2 Decision Tree

Decision tree learning is one of the most widely used and well-known methods for inductive inference. It is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree [23]. Learned trees can also be represented as sets of if-then rules which can readily be expressed so that humans can easily understand them. A decision tree is composed of a leaf node and a decision node. A leaf node indicates a class of examples and a decision node specifies some test to be carried out on a single attribute, with one branch and a sub-tree for each possible outcome of the test. A decision tree can be used to classify an example by starting at the root of the tree and moving through it up to the leaf node, which provides a class of the example.

ID3 is a greedy algorithm which constructs decision trees in a top-down manner [17]. It searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively does the operation and partitions subsets to get their best attribute until it meets stop criteria. An important thing in constructing a decision tree is to select an attribute which is most useful for classifying examples at each node in the tree. ID3 adopts a statistical measure called *information gain* that measures how well a given attribute separates training examples according to their target classification. In other words, *information gain* is the expected reduction in entropy caused by partitioning examples according to selected attribute [18].

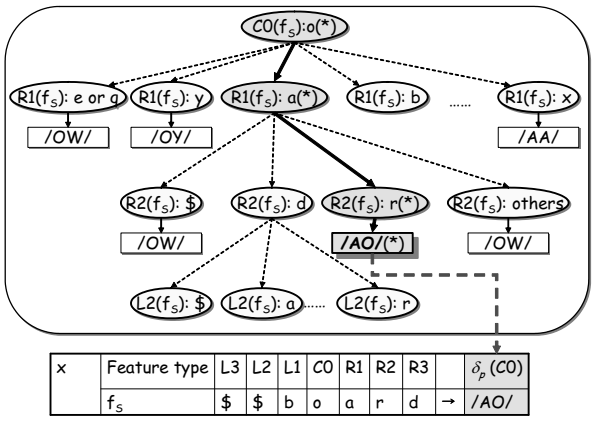


**Fig. 7** A fraction of decision tree for pronunciation estimation

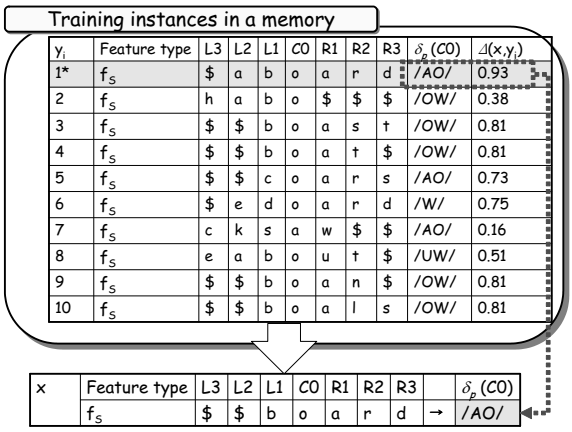


**Fig. 8** Memory-based learning for pronunciation estimation

We use C4.5 which is a well-known tool for decision tree learning and implementation of Quinlan's ID3 algorithm. Figure 7 shows a fraction of our decision tree for $\delta_p$. In the figure, rectangles indicate a leaf node and circles indicate a decision node. In order to simplify our examples, we do not use $f_{GS}$. In Fig. 7, we can infer that attribute $C0(f_S)$, which means the current English grapheme, has higher *information gain* than any other attribute because it is the first attribute to be tested. A decision tree for $\delta_p$ is constructed with an attribute sequence, C0($f_S$), R1($f_S$), R2($f_S$) and so on. Given input data, $x$, its class (phoneme corresponding to source language grapheme $o$) is determined as /AO/ by retrieving the decision tree from C0($f_S$)=$o$ to R2($f_S$)=$r$. In Fig. 7, $(*)$ indicates a path for the decision.

### 3.3.3 Memory-Based Learning

Memory-based learning (MBL) is an example based learning method. It is also called instance-based learning and case based learning method [24]–[26]. It is based on a $k$-nearest neighborhood algorithm [25], [27]. MBL algorithm represents a training object as a vector composed of feature, feature value and the object's class. In the training phase, MBL puts all training data as examples in memory and clusters some examples with a $k$-nearest neighborhood principle. It then outputs a result or a class using similarity-based reasoning between input data and examples in memory Let input data be $x$ and a set of examples in memory be $Y$, the similarities between $x$ and $Y$ are estimated by a distance function, $\triangle(x, Y)$. Modified value difference metric, overlap metric, Jeffrey divergence metric, and dot product metric are usually used as $\triangle(x, Y)$ [19]. MBL selects an example $y_i$ or a cluster of examples that are most similar to $x$, and then it assigns the $y_i$'s class or the class of the cluster to an $x$'s class.

We use a memory-based learning tool called TiMBL (Tilburg memory-based learner) version 5.0 [19]. We set the distance function as an overlap metric, which shows the best result for our task. Figure 8 shows examples of $\delta_p$ based on MBL. All training data are represented with their features and their class ($\delta_p(C0)$ in Fig. 8). They are stored in memory through a training phase. By comparing the similarities between $x$ and $Y$ using distance metric $\triangle(x, Y)$, we can output the $y_1$'s class /AO/ as an $x$'s class in Fig. 8.

## 4. Experiments

### 4.1 Experimental Setup

We perform experiments for English-to-Korean transliteration using two data sets. They contain an English word and its corresponding standard Korean transliteration. Test Set I [3], [5] is composed of 1,650 English-Korean transliteration pairs. Since the test set was used as a common test bed for previous English-to-Korean transliteration methods, we use it for comparison between our method and the previous ones. For comparison purposes, 1,500 pairs are used as training data and 150 pairs are used as test data. Test Set II [8], [10] consists of 7,185 E-K pairs; the number of training data is 6,185 and that of test data is 1,000. We use Test Set II for several tests.

Evaluation is performed by word accuracy (W. A.) and character accuracy (C.A.), which were used as the evaluation measure in the previous works (Eq. (9)).

$$W.A. = \frac{\#\ of\ correct\ transliterations}{\#\ of\ generatedtransliterations} \qquad (9)$$
$$C.A. = \frac{L-(i+d+s)}{L}$$

where $L$ represents the length of the original string, and $i, d$, and $s$ represent the number of *insertions, deletions* and *substitutions*, respectively. If $L < (i + d + s)$, we consider it as zero [28].

**Table 7** Results of Comparison Test for Test Set I: HST is evaluated with top-20 results generated by both his grapheme-based transliteration model and phoneme-based one. The top-20 results are ranked by their probability. Lee reported that the hybrid results were better than those generated by either grapheme-based method or phoneme-based one.

| | Method | C.A. | W.A. | PI |
|---|---|---|---|---|
| Previous works | HST | 69.3% | 40.7% | 53.98% |
| | GNN | 79.00% | 35.10% | 78.55% |
| | GDT | 78.10% | 37.60% | 66.68% |
| | GTN | 86.50% | 55.30% | 13.33% |
| Proposed | MEM | 86.15% | 51.72% | 21.17% |
| | DT | 90.35% | 58.33% | 7.44% |
| | MBL | 91.51% | 62.67% | 0% |

**Table 8** Results of Comparison Test for Test Set II.

| | Method | C.A | W.A | PI |
|---|---|---|---|---|
| Previous works | GDT | 81.80% | 51.40% | 24.32% |
| | GMEM | 85.36% | 52.40% | 21.95% |
| | GTN | 88.05% | 55.10% | 15.97% |
| Proposed | MEM | 87.54% | 53.90% | 18.55% |
| | DT | 90.26% | 61.60% | 3.73% |
| | MBL | 90.45% | 63.50% | 0% |

We perform three experiments as follows.

- Comparison Test: Comparison between our grapheme & phoneme-based transliteration ($\psi_{GPT}$) and the previous English-to-Korean transliteration methods.
- Dictionary Test: Evaluating the performance of $\psi_{GPT}$ by a way of "generating pronunciation" (dictionary search and estimating pronunciation) in $\delta_p$.
- Training Data Size Test: Evaluating the performance of $\psi_{GPT}$ when the training data size has been changed

## 4.2 Experimental Results

### 4.2.1 Comparison Test

Table 7 and 8 show results of Comparison Test for Test Set I and Test Set II, respectively. In this test, we use $\psi_{GPT}$ using the maximum entropy model (MEM), decision tree (DT), and memory-based learning (MBL). The previous works to be compared are HST [3], [5], GNN [4], GDT [6], [8], GTN [7], and GMEM [1]. In this result, we perform *external comparison* (between ours and the previous ones), and *internal comparison* (between the machine learning methods that we use).

From the viewpoint of *external comparison*, our method outperforms the previous ones both in W.A. and C.A. Our method shows significant performance improvement (PI) about 13%∼78% in Test Set I, and about 16%∼24% in Test Set II. From the result, we found phoneme information not previously considered that provides the key-point in performance improvement. From the viewpoint of a machine learning technique and the framework of transliteration, our DT (Decision Tree) method and the previous decision tree based method used by Kang [6], [8] are very similar. The only difference is whether phoneme information as well as grapheme information is used or not. This is the main reason why there is a performance gap between ours and the previous method.

From the viewpoint of *internal comparison*, the winner is MBL, followed by DT, and finally MEM. This result indicates that the vector-based method (MBL) is better than the rule-based (DT) and the probability-based methods (MEM) in our task. MEM shows the lowest position and the biggest performance gap in comparison with the others. We find two main reasons for this phenomenon. The first one is high ambiguity of vowel transliteration. There are many more candidates for vowel transliteration than there are for consonant transliteration. On average, more than 10 candidates are generated for a vowel, while only 1∼4 candidates are generated for a consonant. Though others also suffer from high ambiguity, MEM is more sensitive to it due to its probabilistic nature. The second reason is its sensitivity to data sparseness; MEM shows lower performance when there is smaller training data. Though others also suffer from the data sparseness problem, they have greater ability to control it with similarity reasoning and decision rules using fraction of features. Even though MEM gives high weights to discriminative features and adopts a *Gaussian smoothing method*, limitations on handling the data sparseness problem are due to its probabilistic nature. The probabilistic models for English-to-Korean transliteration, such as Lee's method [3], [5] and Kim's [4] in Table 7 fail to achieve good performance for the same two reasons - high ambiguity of vowel transliteration and sensitivity to the data sparseness problem.

### 4.2.2 Dictionary Test

For the Dictionary Test, we use Test Set II. First, we evaluate the performance of our method in terms of existence and nonexistence of pronunciation in the pronunciation dictionary (Dictionary Test I). Second, we investigate the performance of our method when pronunciation dictionary is not used (Dictionary Test II).

Table 9 shows results of Dictionary Test I. In Table 9, $R$ represents performance of $\psi_{GPT}$ when it uses a pronunciation dictionary search, and $NR$ represents when it uses estimating pronunciation in pronunciation generation as $\delta_p$. The number of words in $R$ is 687 and that in $NR$ is 313. From the viewpoint of comparison between $R$ and $NR$, $R$ shows higher performance than $NR$. This indicates that correct pronunciation is very helpful to generate correct Korean transliterations in $\psi_{GPT}$. We find that the performance of $NR$ is the key point to determine whether a certain machine learning method is better, because the performance gap between the best one and the worst one in $NR$ is much bigger

**Table 9** Results of Dictionary Test I

| | | R | NR |
|---|---|---|---|
| MBL | C.A. | 90.83% | 87.10% |
| | W.A. | 68.58% | 52.40% |
| DT | C.A. | 89.86% | 87.10% |
| | W.A. | 66.23% | 50.16% |
| MEM | C.A. | 89.24% | 83.31% |
| | W.A. | 60.70% | 38.34% |

**Table 10** Results of Dictionary Test II

| | | |
|---|---|---|
| MBL | C.A. | 88.46% |
| | W.A. | 56.90% |
| DT | C.A. | 87.43% |
| | W.A. | 56.30% |
| MEM | C.A. | 85.12% |
| | W.A. | 47.40% |

than that in $R$. The main reason for the bigger gap is errors in estimating pronunciation; the higher the performance in estimating pronunciation results, the higher the performance in generating Korean transliterations in the result. Note that estimating pronunciation based on MBL shows about 65% word accuracy, while that based on MEM shows about 48% word accuracy in our system.

Table 10 shows the results of Dictionary Test II. The W.A. of MBL and DT is about 56∼57%. Because [6], [8] in Table 8 and our DT adopt the same machine learning algorithm, say the decision tree, they can be directly and indirectly compared as $\psi_{GT}$ and $\psi_{GPT}$, respectively. In the comparison, our method shows about 9% performance improvement, though the performance gap between Kang's method and ours when the pronunciation dictionary used is much bigger. In the Dictionary Test II, we find that our method shows reasonable performance even though there is not pronunciation dictionary.

#### 4.2.3 Training Data Size Test

Table 11 shows results of the Training Data Size Test in which we use Test Set II. Table 11 shows the performance of $\psi_{GPT}$ when training data is changed from 20% to 100%. Obviously, the more training data we have, the higher the system performance. However, we wanted to test, here, whether our method shows reasonable performance even if there is small training data. Unfortunately, MEM fell short of our expectations because of its probabilistic nature as described in Sect. 4.2.1. But in this case, even if 20% of training data is used, $\psi_{GPT}$ based on MBL and DT shows relatively high performance compared with 100% training size performance.

### 4.3 Summary

The experimental results in this section can be summarized as follows:

**Table 11** Results of Training Data Size Test

| | | 20% | 40% | 60% | 80% | 100% |
|---|---|---|---|---|---|---|
| MBL | C.A. | 87.9% | 88.4% | 88.8% | 89.5% | 90.5% |
| | W.A. | 58.4% | 59.7% | 60.5% | 62.6% | 63.5% |
| DT | C.A. | 87.4% | 87.8% | 88.1% | 89.8% | 90.3% |
| | W.A. | 54.9% | 56.4% | 58.2% | 60.3% | 61.6% |
| MEM | C.A. | 85.7% | 86.9% | 87.2% | 87.0% | 87.5% |
| | W.A. | 47.9% | 50.6% | 51.6% | 51.4% | 53.9% |

- Our method outperforms the previous ones in that it achieves about 13%∼78% performance improvements (Comparison Test).
- MBL is the best machine learning method in our task (Comparison Test).
- Correct pronunciation is very helpful to generate a correct Korean transliteration (Dictionary Test).
- Our method shows reasonable performance even if there is small training data: with 20% training data our method achieves about 55%∼59% W.A (Training Data Size Test).

## 5. Conclusion

We propose a grapheme and phoneme-based machine transliteration model ($\psi_{GPT}$). Unlike the previous works, our method uses both grapheme and phoneme information in English-to-Korean transliteration. Through this combination, we achieve 13%∼78% performance improvements.

In this paper, we showed that grapheme information as well as phoneme information is useful for machine transliteration. In experiments, we showed that MBL is the best machine learning method in our task and correct pronunciation is very helpful to generate a correct Korean transliteration. Our method shows reasonable performance even if there is small training data: with 20% training data our method achieves about 55%∼59% W.A.

In future work, we will use other machine learning methods such as SVM [29]. Our method may be useful in various NLP applications such as automatic bilingual dictionary construction, information retrieval, machine translation, speech recognition and so on.

## Acknowledgments

This work was supported by the Korea Ministry of Science and Technology, the Korea Ministry of Commerce, Industry and Energy, and the Korea Science and Engineering Foundation (KOSEF).

#### References

[1] I. Goto, N. Kato, N. Uratani, and T. Ehara, "Transliteration considering context information based on the maximum entropy method," Proceedings of MT-Summit IX, 2003.

[2] H. Li, M. Zhang, and J. Su, "A joint source-channel model for machine transliteration," Proceedings. of ACL 2004, 2004.
[3] J.S. Lee and K.S. Choi, "English to korean statistical transliteration for information retrieval," Computer Processing of Oriental Languages, 1998.
[4] J.J. Kim, J.S. Lee, and K.S. Choi, "Pronunciation unit based automatic english-korean transliteration model using neural network," roceedings of Korea Cognitive Science Association, 1999.
[5] J.S. Lee, An English-Korean transliteration and Retransliteration model for Cross-lingual information retrieval, Ph.D. thesis, Computer Science Dept., KAIST, 1999.
[6] B.J. Kang and K.S. Choi, "Automatic transliteration and back-transliteration by decision tree learning," Proceedings of the 2nd International Conference on Language Resources and Evaluation, 2000.
[7] I.H. Kang and G.C. Kim, "English-to-korean transliteration using multiple unbounded overlapping phoneme chunks," Proceedings of the 18th International Conference on Computational Linguistics, 2000.
[8] B.J. Kang, A resolution of word mismatch problem caused by foreign word transliterations and English words in Korean information retrieval, Ph.D. thesis, Computer Science Dept., KAIST, 2001.
[9] K. Knight and J. Graehl, "Machine transliteration," Proceedings of the 35th Annual Meetings of the Association for Computational Linguistics, 1997.
[10] Y.S. Nam, Foreign dictionary, Sung An Dang, 1997.
[11] G.D. Forney, "The viterbi algorithm," Proceedings of the IEEE, pp.268–278, 1973.
[12] K.F. Soong and E.F. Huang, "A tree-trellis based fast search for finding the n best sentence hypotheses in continuous speech recognition," IEEE International Conference on Acoustic Speech and Signal Processing, pp.546–549, 1991.
[13] Korea Ministry of Culture & Tourism, "English to korean standard conversion rule," 1995.
[14] V.I. Levenshtein, "Binary codes capable of correcting, deletions, insertions and reversals," Soviet Phys. Dokl, vol.10, pp.707–710, 1966.
[15] A. Fujii and T. Ishikawa, "Japanese/english cross-language information retrieval: Exploration of query translation and transliteration," Computers and the Humanities, vol.35, no.4, pp.389–420, 2001.
[16] L. Zhang, "Maximum entropy modeling toolkit for python and c++," 2004.
[17] J.R. Quinlan, "Induction of decision trees," Machine Learning, vol.1, pp.81–106, 1986.
[18] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kauffman, 1993.
[19] W. Daelemans, J. Zavrel, K.V.D. Sloot, and A.V.D. Bosch, "Timbl: Tilburg memory-based learner - version 4.3 reference guide," 2003.
[20] CMU, "The carnegie mellon university CMU pronouncing dictionary v0.6," 1997.
[21] A.L. Berger, S.D. Pietra, and V.J.D. Pietra, "A maximum entropy approach to natural language processing," Computational Linguistics, vol.22, no.1, pp.39–71, 1996.
[22] Y. Miyao and J. Tsuji, "Maximum entropy estimation for feature forests," Proceedings of Human Language Technology Conference, 2002.
[23] T.M. Mitchell, Machine learning, New-York: McGraw-Hill, 1997.
[24] D.W. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," Machine Learning, vol.6, no.3766, 1991.
[25] D.W. Aha, "Lazy learning: Special issue editorial," Artificial Intelligence Review, vol.11:710, 1997.
[26] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features," Machine Learning, vol.10, pp.57–78, 1993.
[27] C. Stanfill and D. Waltz, "Toward memory-based reasoning," Communications of the ACM, vol.29, no.12, pp.1213–1238, 1986.
[28] P.A.V. Hall and G.R. Dowling, "Approximate string matching," ACM Computing Surveys, vol.12, pp.381–402, 1980.
[29] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, 1995.

**Jong-Hoon Oh** received the B.S. degree in Information Engineering from SungKyunKwan University, Korea, in 1998, the M.S. degree in 2000 in Computer Science from KAIST. Since 2000, he has been a Ph.D. student of the Dept. of Computer Science, KAIST. His research interests include machine transliteration, automatic term recognition, and terminology.

**Key-Sun Choi** received the B.S. degree in Mathematics from Seoul National University, Korea, in 1978, the M.S. degree in 1980 and the Ph.D. degree in 1986 in Computer Science from KAIST. Since 1988, he has been a faculty member of the Dept. of Computer Science, KAIST and also a director of KORTERM/BOLA. His research interests include natural language processing, machine translation, information retrieval, and terminology.