PART - II

STATE OF THE ART: PHONETIC AND SEMANTIC MATCHING APPROACHES
CHAPTER 3

PHONETIC MATCHING APPROACHES

3.1 INTRODUCTION

In multilingual environment, there are many languages involved in processing the text or keywords. We can maintain the data in all the languages or data can be maintained in global English language, but for the first approach we need translator or transliteration. In such environment, there are many issues to be resolved. They can be broadly categories into two main areas like phonetic and semantic matching. In both areas, if a user doesn’t enter exact keywords then the system may fail to provide the information. The keywords should be matched phonetically or semantically in order to retrieve the exact information.

In phonetic matching, the main problem is to tackle various pronunciations by people from various communities for the same keyword, especially in rural areas. In many cases or scenarios the conversation may have ambiguity. For example, conversation with doctor especially in rural areas, conversation with shop keepers etc. In these scenarios, either the words used does not have proper pronunciation or may not have correct spellings or may be written in different ways. In all the cases, we need a system to tackle mentioned problems.

3.2 CLASSIFICATION OF MATCHING APPROACHES

3.2.1 TRANSLATOR-BASED SYSTEM

In this category, each string of a language is translated into a uniform representation by using a text-to-phonetic (TTP) system. This system will translate each text string into phonetic form. This phonetic form is a set of an encoding standard IPA, in which all the alphabet characters are represented in phonetic form. For some of the Indian languages, either TTP systems may not be available or they need to be developed.
Using this system, we cannot have the phonetic form of all the characters, especially for Hindi or Marathi. In order to match, edit distance can be calculated with some threshold value. Figure 3.1 shows the general architecture for text-to-phonetic based system.

### 3.2.2 CODE-BASED SYSTEMS

Using some code in the numeral form or other form, entire string is translated into a code format. This code always starts with first character of a string followed by at least four characters. We need to compare the codes of both the strings to match two strings. If the codes are same then we may say that both strings are phonetically matching. The codes can be generated by grouping the alphabets according to their phonemes. Each group will have the same code value. Some system starts the code with 0, some starts with 1. But sometimes, we may get same code for different strings. Examples of these systems are soundex, phonix etc. Some systems may group the alphabet characters and assign the code for each group. If two strings are having maximum groups those are having same codes then we may say that both strings matches phonetically, example is Q-gram method.
Figure 3.2 shows the general architecture for code-based system, where the codes will be generated by using rules. The matcher will match the codes for equivalence.

### 3.2.3 PHONETIC RULE-BASED SYSTEMS

These systems work on the phonetic rules designed for a particular language. These rules are used to group the alphabet characters according to phonemes. After applying these rules, each string is converted into its phonetic form either in text form or in some code form. In order to match, these forms are compared with some threshold value. These systems are easy to use, but difficult to build phonetic rules for a language.

![General Architecture of Phonetic Rule-based System](image)

**Figure 3.3** General Architecture of Phonetic Rule-based System

Figure 3.3 shows the general architecture for phonetic rule-based system, where the rules for each language apply to convert the string into its phonetic form. The matcher will use threshold value in order to match.

### 3.3 MATCHING STRATEGIES

The classifications mentioned in above section leads to the some building blocks to be used for matching system. In particular, the following aspects of building a working matching system have to be taken into account as well:

- How to use the standard phonemics system to convert from text to phonemics which is representing phonemics alphabet characters? Also for matching, how to calculate the edit distance between two phonemics strings for a particular language?
- How to involve the user in the matching process? For a particular language, a user may decide how to assign a code for a particular group of alphabet of characters.
Later the entire process will be carried out by the proposed algorithm to form a code for an entire string.

- How to form phonemics rules for a language? According to language phonemic properties of alphabet characters, their structure, their alignment and different way of use of modifiers will decide the phonemic form of a string.
- How to extract the final alignment? This can be done by using various thresholds, e.g. threshold as tolerance value, number of digits from a code of a string, number of grams that matches and so on.

### 3.4 ON HAND APPROACHES

#### 3.4.1 TEXT-TO-PHONETIC (TTP) BASED PHONETIC MATCHING

Multilingual names matching and retrieving is defined as matching of same names across multiple languages and retrieving all names that match aurally in the specified set of target languages. Hence matching is restricted to attributes that contain proper name [3].

A. Kumaran in his research work has suggested the implementation details by using MLNameJoin operator. For a set of languages, the strings are composed of characters from an alphabet of each language. To generate cross-lingual query in relational database using relational operator like MLNameJoin text-to-phonetic (TTP) methodology is being used, which matches strings phonetically [3].

In this approach, for an entered string all the names are retrieved that matched aurally irrespective of the language. The database is maintained in multilingual environment in one of the database system. They are encoded in Unicode format. Any two strings are being converted to its phonemic form by using Text-to-Phonetic (TTP) system. They are encoded in the IPA alphabet. An edit distance is calculated between two strings from TTP representation. Edit distance is defined as the minimum number of edit operations required to transform a string from one form to other. The operations might be character insertion, deletion and substitution. If distance is within range of the tolerance value, they are phonetically close
depending on the user specified value of tolerance. Tolerance value may depend on a specific domain or application. But this approach is difficult for Indian languages since TTPs are not available for all the Indian languages [3].

3.4.2 USING MLLike and MLLexEqual OPERATORS

Rupesh Bajaj in his project report titled ‘an Integrating Multilingual Database operator in PostgreSQL’ has suggested three multilingual operators used for phonetic operations. The operators are MLLike and MLLexEqual.

The MLLike operator can be used for queries where the condition for matching has been given with same pronunciation irrespective of the language. This operator provides the phonemic regular expression matching in user specified set of languages. The SQL LIKE operator can also be used to join two tables of different language sets, but joining is not possible without phoneme regular expression matching [4].

For example, for retrieving all the records with author name starts with ‘न’, the SQL query with MLLike operator will be as follows-

```
SELECT Title, Author, Category
FROM MLBooks
WHERE Author InLanguages {English, Hindi, Marathi}
      MLLike 'न%';
```

Figure 3.4: Multilingual Name Query with MLLike Operator

Similar to MLLike operator, MLLexEqual operator which provides a phonemic matching functionality that is used in cases where one has to determine if two words are phonetically equivalent. This operator takes an input name in one language and returns all records that
have the same name in all or in a user-specified set of languages. The threshold parameter specified in the query determines the quality of matches [4].

For example, SQL query using MLLexEqual operator will be as –

```
SELECT Title, Author, Category
FROM MLBooks
WHERE Author InLanguages {English, Hindi, Marathi}
     MLLexEqual 'न%' WithThreshold 0.25;
```

Figure 3.5: Multilingual Name Query with MLLexEqual Operator

The approach is based on global dictionary and local column as phonetic column. In global dictionary approach, the system maintains one relation as global dictionary that gives the relationships of the strings and its equivalent phoneme. When the user gives the query, the query is converted into a query with MLLexEqual operator. The global dictionary is being searched for equivalent strings with respect to its phoneme. If found then the corresponding tuples are retrieved as IR. This approach saves a lot of space, but time taken to execute will be much more as compared to MLLike approach. In local column approach, for each column an equivalent phoneme column is maintained to find match for each column. When a user gives the query, existence of phoneme column is checked for the column involved in the query every time. If it does not exist, it is created first and populated with their phoneme strings. Whenever the user inserts or updates the tuples in the relation, corresponding value in the phoneme column has to be updated. In both approaches, TTP (Text-to-Phoneme) converter is used to generate the phoneme. The author also proposed hybrid approach, which combine both the approaches [4].
3.4.3 PHONETIC MATCHING USING SOUNDEX AND Q-GRAM

In this section, we describe the various approaches for phonetic matching. Existing approaches were developed to match the strings approximately, whereas our approach matches the strings exactly. These approaches used for phonetic matching.

3.4.3.1 Soundex for English

In this approach, each string is converted into a code, which consists of a first letter of a string and three numbers. The numbers are assigned to each letter as per guidelines described by the algorithm [5]. Zeros are added at the end if necessary to produce a four-character code, additional letters are discarded. The soundex codes and algorithm are given in Table 3.1 and Algorithm 3.1 respectively. The pitfall of this algorithm is that it produces the same code for phonetically two different strings or produces different codes even if they are same.

Table 3.1: Soundex Phonetic Codes

<table>
<thead>
<tr>
<th>Code:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters:</td>
<td>a, e, i, o, u, y, h, w</td>
<td>b, p, f, v</td>
<td>c, g, j, k, q, s, x, z</td>
<td>d, t</td>
<td>l, m, n</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm 3.1

Objective: Phonetic Matching using Soundex Approach

Input: Two string to match

Output: Soundex code

1. Retain the first letter of the string.
2. Remove all occurrences of the following letters, unless it is the first letter: a, e, i, o, u, y, h, w.
3. Assign numbers as given in code table.
4. If two or more letters with same number were adjacent then omit all except first.
5. Return the first four bytes padded with 0 if necessary.

EXAMPLE

Consider the two strings in English, ‘*sandy*’ and ‘*sandhya*’. After applying the rules from table 3.1 and steps from algorithm 3.1, we are getting the same code as ‘s530’ for both the strings. So, by looking at the code we have to say that both strings are phonetically matched but both the strings are phonetically different.

3.4.3.2 Soundex for Hindi (Proposed Approach)

If we consider the steps of algorithm 3.1 for Hindi language and assign the code as per their phonetic group, we have codes for letters as shown in table 3.2. If we match the strings with this algorithm, we get different codes for the same string or we cannot generate the code. From this methodology, the code can be generated by taking ‘halant’ for each consonants concatenated together with vowel. This changes the code and takes long time to parse the string and assign the code to it. We have found out that there is no code for many alphabets or letters for Hindi language, such as ण, झ, छ, घ etc.

Table 3.2: Soundex Code for Hindi as Per Algorithm

<table>
<thead>
<tr>
<th>Code:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters:</td>
<td>अ, इ, औ, उ, व, ह, घ</td>
<td>ब, प, फ</td>
<td>च, ग, ज, क, स, झ</td>
<td>ड, ट, ल</td>
<td>न, म</td>
<td>र</td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLE

If we consider the same rules and algorithm for Hindi as shown in table 3.2, the codes for ‘संदी’ and ‘संध्या’ strings are same as ‘s530’. By using soundex method, we acquired the
same code for both the strings entered in Hindi. As codes are same, we have to say that both strings matched phonetically, but they are not matching. There is an ambiguity to match. So, we can conclude that the soundex algorithm or approach will work only for English language not for either Hindi or Marathi.

3.4.3.3 Q-Gram for English

This approach measures the distances between two strings in the form of q-gram counts, where q-gram of a string s is any substring of s of some fixed length q. A simple such measure is to count number of q-grams for the two strings with a higher count yielding a stronger match [5]. But, this algorithm is not exactly phonetic since they do not operate based on comparison of the phonetic characteristics of words. The steps for the approach are given in algorithm 3.2. Since phonetically similar words often have similar spellings, this technique can provide favorable results. It also successfully matches misspelled or otherwise muted words even if they are rendered phonetically disparate. For example, for the strings, ‘sandy’ and ‘sandhya’, with q = 2, the substrings are as shown in table 3.3. With q-gram algorithm, we acquired 3-grams matches out of 4-grams and others are not matched from the string. Depending on number of q-grams, we have to find out similarity of strings that matches phonetically.

<table>
<thead>
<tr>
<th>String 1: ‘Sandy’</th>
<th>sa</th>
<th>an</th>
<th>nd</th>
<th>dy</th>
<th>----</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>String 2: ‘Sandhya’</td>
<td>sa</td>
<td>an</td>
<td>nd</td>
<td>dh</td>
<td>hy</td>
<td>ya</td>
</tr>
</tbody>
</table>

**Algorithm 3.2**

Objective: Phonetic Matching using Q-Gram Approach

Input: Two strings to match

Output: Number of q-grams

1. Select the value for q.
2. Make substrings of the strings with number of letters equal to q.
3. for q value 
   i. First string = first q letters.
   ii. Next substring = second letter of previous string + next letter.
   iii. Repeat.
4. Count the number of substrings in common.
5. Maximum count results in most matches.

### 3.4.3.4 Q-gram for Hindi (Proposed Approach)

If we apply the same rules as in algorithm 3.2 in order to form q-grams for Hindi strings then the entire string changes or none of the q-gram matches with other q-gram. One problem with this method is that if parsing of a string goes wrong then it affects the generated code.

**EXAMPLE**

Consider the same strings as, ‘सँडी’ and ‘संड्या’. After applying Q-gram method for Hindi, we acquired result as only two 2-grams that matched and two does not matched. There is also an ambiguity for phonetically match. Table 3.4 shows the various grams for the same strings. If we formed the probable q-grams by considering q = 2 for each string then from table 3.4 we can conclude that none of the q-gram matches. It means both string does not matches.

<table>
<thead>
<tr>
<th>String1: सँडी</th>
<th>न</th>
<th>ल्द</th>
<th>बई</th>
<th>--</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>String 2: संड्या</td>
<td>स अ</td>
<td>अन</td>
<td>न अ</td>
<td>अंध्य</td>
<td>ध आ</td>
</tr>
</tbody>
</table>
3.5 DRAWBACKS OF EXISTING PHONETIC MATCHING APPROACHES

The following are some of the drawbacks from existing phonetic approaches.

- In one of the approach, we need to find IPA code for each string for phonetic matching which is difficult and may be available for Indian languages. Some of the letters are having approximate same IPA code in Hindi.
- Also we need to use text-to-phonetic (TTP) system for each language. Use of TTP makes the system complex.
- The algorithm depends on the user’s predefined threshold value which is based on length of the strings. So, there may be an ambiguity in matching.
- The soundex and Q-gram methods use code for each alphabet. These methods are either generating wrong results or may not accommodate the code for all the alphabets for Hindi and Marathi languages.
- Entire code of a string is not considered to match in soundex approach, this can lead to ambiguity in matching.
- Different interpretation of a string to make q-grams may lead to ambiguity in Q-gram approach.
- Phonetic rule-based system is not available for Hindi and Marathi.

3.6 SUMMARY

Many various phonetic matching approaches, methods, algorithms have been proposed. But all these need lot of parameters, number of external resources and so on. Basically all these methods are dependent on either international phonemic alphabet or translation system for each language. Some approaches rely on code for each alphabet or rules based on pronunciation for matching.

In this chapter, we classified the general approaches for phonetic matching. In proposed approaches, these classifications have been applied and evaluated. We also made a survey on the proposed approaches like soundex, Q-gram which may work for English but may give wrong result for Hindi and Marathi languages. Some drawbacks of existing approaches are also mentioned.