

**EXPLORATION OF HEURISTICS OF INFORMATION
RETRIEVAL FUNCTIONS IN THE APPLICATIONS
OF APPROXIMATE STRING SIMILARITY
PROBLEMS**

A PROJECT REPORT

Submitted By

PRANAV AGRAWAL 13BCE0184

in partial fulfillment of the award

of the

B.Tech

degree in

Computer Science and Engineering

School of Computer Science and Engineering



VIT[®]
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)

VELLORE ■ CHENNAI

www.vit.ac.in

ABSTRACT

Ranking functions used in information retrieval are primarily used in the search engines and they are often adopted for various language processing applications. This project introduces some novel heuristics combined with probabilistic retrieval functions and are employed in the domain of approximate string similarity problem. A lot of algorithms have been proposed in the literature to solve approximate string similarity problems, however none of them makes use of probabilistic retrieval functions. We are the first to explore the intersection between these two areas and propose heuristic designs to resolve this problem. First we propose chunking heuristic function, BREAK. We show the variants BREAK-1, 2, OFF which splits up the terms with the sequential notion. Then we propose BREAK-n which generalizes these variants and scales to larger datasets. In order to relate these split ups, we propose a graphical error modelling heuristics MAKE over the BREAK variants. Finally, we propose TAKE curve, a novel feature engineering probabilistic distribution which replaces the prevalent normalization heuristics. Taking the advantage of flexibility over the choice of heuristics, we assess the variants on the cognate detection, mutant identification, and isolated spelling correction based problems. In the extensive evaluation methods, we found that our designs perform better than prevalent heuristics and are robust against database characteristics.

CONTENTS

1	INTRODUCTION	1
1.1	Related Work	3
1.2	System Architecture	5
2	BREAK: EXPANSION SETS OF WORDS	6
2.1	BREAK-1: Sequencing from 1 End	6
2.2	BREAK-2: Sequencing from 2 Ends	7
2.3	BREAK-Off: Sequencing with Offsets	8
2.4	BREAK-n: Sequential Splits for Longer Terms . .	10
2.4.1	Anchoring	11
2.4.2	Redefining Intersection	11
2.5	Combining BREAK with Ranking Functions . . .	13
3	MAKE : GRAPHICAL ERROR MODELLING	15
4	HEURISTICS ON WORD LENGTH SIMILARITY	23
4.1	Guidelines for Feature Engineering in Ranking Functions	23
4.1.1	Nature of the Function Curve	24
4.1.2	Nature of the Limits of the Curve	25
4.2	Design of the TAKE curve	26
4.3	Description of TAKE	29
4.4	Power penalization	30
4.5	Bucket-shaped sigmoidal penalization	31
5	EXPERIMENTAL SETUP	35
5.1	Spelling Correction	35
5.2	Cognate Detection	37
5.3	SNP Detection	38
6	ANALYSIS OF RESULTS	40

6.1	Analysis of Dirichlet Prior	41
6.2	Analysis of Hyperparameters	41
7	CONCLUSION	43
	BIBLIOGRAPHY	44

LIST OF FIGURES

Figure 1	The process of BREAK-2. On the left, algorithm slices pizza into pieces. The output set would be $\{1p, 2pi, 3iz, zz3, za2, a1\}$. On the right, algorithm breaks hearts into pieces. The output set would be $\{1h, 2he, 3ea, 4ar, rt3, ts2, s1\}$. 8
Figure 2	In this example, the words <i>piza</i> and <i>pizza</i> are given as <i>q</i> and <i>d</i> respectively, split into 2-grams. If we chose 2 anchor points, the position according to equation 1 would be at the first and the last (which are shown in arrows). Now we fill every $t_{e,ij}$ entry by calculating relative distance given in equation 2. 12
Figure 3	Graphical error model construction from <i>panc</i> to <i>pant</i> 18
Figure 4	Graphical error model construction from <i>sieze</i> to <i>seize</i> 19
Figure 5	Directed graph guiding from the incorrect <i>pth</i> to the correct <i>path</i> 20
Figure 6	Graphical error model construction 21
Figure 7	The nature of power penalization $h(d , q)$ curve. The value of $h(d , q)$ dips when $ d $ approaches the $ q $. As the γ increases, penalization would become more harsher. 30

Figure 8	The nature of sigmoidal curve $h(d , q)$ with $B_1 = 1$, $B_2 = 1$ and $c = 0.5$	32
Figure 9	Effect of Dirichlet prior against MAP with BREAK-2 (μ_1) and with MAKE (μ_2)	40
Figure 10	The effect of hyperparameter sensitivity in isolated spelling correction λ_1 and with cognate detection in λ_2	42

LIST OF TABLES

Table 1	A brief overview of related topics and prevalent methods used in approximate string similarity. Some of these methods are used as baselines in the evaluation section.	3
Table 2	Test dataset MRR results for spelling correction evaluation	36
Table 3	Test dataset results for cognate detection experiment	37
Table 4	Results for gene mutant detection simulation experiment	38

INTRODUCTION

Approximate string similarity problems deal with returning of a ranked list of closest possible strings against the given query string. Some representatives of this problem are,

1. **isolated spelling correction** which involves returning a list of suggestions for a misspelled word [1],
2. **genetic mutant identification** which is the process of returning a set of possible closest mutations of the given genome sequence [2],
3. **cognate detection** which deals with identification of the words which have same linguistic derivation [3].

On the other hand, information retrieval models portray the idea of relevance, so that one can score a document with a given respective query. There are prevailing models like BM25 [4], Dirichlet-prior smoothing [5], PL2 [6] which are commonly employed mainly in search engine application.

This paper deals with the intersection between these two areas which is largely under-explored in the literature. We show how the notion of retrieval can be incorporated in the approximate string similarity problem by breaking a word into small units. Furthermore, Nguyen et al. [2] have stated that broken words are more practical to query large databases of sequences as compared to conventional methods. Additionally, retrieval models provide a variety of alternative heuristics which can be

chosen for the desired application area [7]. Taking these advantages of flexibility of these models, the combination of approximate string similarity operations with information retrieval systems could be beneficial in many cases.

Hence the objectives of this paper are:

1. To explain the chunking methods and give the sequential notion. This paper demonstrates the BREAK algorithm which breaks a word into smaller pieces. First, we show a trivial chunking method BREAK-o. However, just separating a word is insufficient. So we propose some variants which contain sequential information, like BREAK-1, BREAK-2 and BREAK-Off. Then we propose BREAK-n which would not only generalize these variants but also would operate seamlessly for really long terms like biological sequences (Section 2).
2. To model a graphical error identification algorithm which probabilistically relates the split sets between query and the document. For this purpose, we proposed the algorithm MAKE over the expansion sets generated. (Section 3).
3. To design a generalized feature engineering curve which maximizes the proximity between query and document. We proposed TAKE curve, an inverse Gaussian function which models between the extremities of the limits generated. (Section 4).
4. To show how retrieval models and approximate string similarity based algorithms would come together. The variants of proposed heuristics are tested elaborately in applications of approximate string similarity problem like cog-

nate detection, mutant identification and isolated spelling correction (Section 5). We show that our algorithms are robust against database characteristics.

1.1 RELATED WORK

Table 1: A brief overview of related topics and prevalent methods used in approximate string similarity. Some of these methods are used as baselines in the evaluation section.

Problem Domain	Prevalent Methods	Source
Isolated Spelling Correction	Brute-force candidate generation method	[8]
	Weighted longest common subsequence search	[9]
	Usage of subsequences (s-grams)	[10] [11]
	Finite state automata based corrections	[12]
Cognate Detection	Using Naive Bayes	[13] [14]
	Using SVM	[15] [3]
Gene Mutant Identification	Dot matrix alignment	[16]
	Needleman-Wunsch alignment	[17]
	Smith-Waterman alignment	[18]
	FASTA tool	[19]
	BLAST tool	[20]
	T-COFFEE	[21]

Isolated spelling correction involves returning a list of suggestions for a misspelled word. Some of the prevalent techniques include brute-force candidate generation method [8], weighted longest common subsequence search [9], usage of subsequences [10, 11], and finite state automata based corrections [12]. Genetic mutant identification is the process of returning a set of possible closest mutations of the given genome sequence. Many biological sequence alignment algorithms like dot-matrix [16], Needleman-Wunsch [17], Smith-Waterman [18] and tools like FASTA [19], BLAST [20], T-COFFEE [21] are commonly employed for this problem. Cognates are the words existing in dif-

ferent languages but they have a common origin. For example, the word *night* in English and *nuit* in French are a pair of cognates which have a Proto-Germanic origin. Cognate detection is the problem of identification of such pairs within a cross-language corpus. Current machine learning based approaches include SVM and Naive-Bayes classifiers which distinguishes whether a pair is cognate or not. [3, 13–15]

Probabilistic retrieval functions are frequently used in search engines for returning a ranked list of relevant documents [22]. These functions provide a variety of heuristics which can be chosen for the desired application area [7]. Taking this advantage of flexibility of these functions, we propose modified versions of these functions for the applications in approximate string similarity problems.

Zhai et al. [22] lay down the in-depth analysis of variety of ranking functions used in information retrieval. Common features used in the information retrieval are term frequency, inverse document frequency and length normalization. TF-IDF (Term Frequency and Inverse Document Frequency) is a prevalent vector-space information retrieval technique. It balances the similarity of the query and the document, and penalizes the common terms [23]. Pivoted document length normalization with TF-IDF helps to reward shorter documents [24]. Okapi BM25 is a complex version of pivoted length normalization and it is the prevailing state-of-the-art retrieval method [4]. PL2 is a retrieval model based on divergence from randomness of the query term frequency [6]. Dirichlet Prior based retrieval is based on language modeling where the smoothing function is derived from Dirichlet distribution [5]. MPtf2ln and MDtf2ln are improvements on previous methods and more elaborate

ranking functions which balances the extremities of normalization effects [7]. They also lay down certain guidelines to evaluate the behavior of the ranking functions.

1.2 SYSTEM ARCHITECTURE

The setup of the project is similar to the information retrieval system. Here, the given strings are processed with our proposed chunking methods. The given chunks are stored in the inverted index in the form of the linked lists. This inverted index is processed with the double-barrel based cache which is developed according to the term frequency of the chunks. The query is given and with the help of the ranking function's heap-sort, the top relevant documents are collected and displayed to the user.

A lot of algorithms have been proposed in the literature to solve approximate string similarity problem, however none of them makes use of probabilistic retrieval functions. Nyugen et al. [2] have stated that word split-ups are more practical to query large databases of sequences as compared to conventional methods. Thus, combination of approximate string similarity operations with information retrieval systems could be beneficial in these cases.

BREAK: EXPANSION SETS OF WORDS

A typical text-based search engine system tokenizes queries and documents into words and uses heuristics to return a ranked list of relevant documents to the query. This analogy can be extended towards string-related operations, where instead of dividing a sentence into words, one could divide a string into small chunks.

The *k*-grams splitting technique illustrates the trivial chunking. The word *pizza* is split with $k = 2$ as: $\{\langle s \rangle p, pi, iz, zz, za, a \langle /s \rangle\}$. Here, $\langle s \rangle$ is the start token and $\langle /s \rangle$ is the stop token. For the sake of the simplicity, we have ignored terminal tokens in the *k*-gram splits. Hence, the split set looks like: $\{p, pi, iz, zz, za, a\}$. We would label this approach as BREAK-o because it splits the words into smaller *k*-grams without any sequential information.

2.1 BREAK-1: SEQUENCING FROM 1 END

We argue that BREAK-o could lead to an extremely generalized matching of tokens since an expansion set could be visualized a bag-of-words method. Thus, we propose a positional *k*-gram splitting technique, BREAK-1 which introduces position number in the splits to incorporate the notion of the sequence of the tokens in the word. For example, the word *pizza* could be position-wise split with $k = 2$ as: $\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$. Thus,

the member $4zz$ simply means that it is the fourth member of the set. The motivation behind this tweak is that it gives us a rough amount of sequential-insight for spelling splits when they are used in probabilistic retrieval ranking functions.

2.2 BREAK-2: SEQUENCING FROM 2 ENDS

The main disadvantage of the BREAK-1 is that some misspellings can easily disturb the order of the set which leads to low similarity. For example, if the misspelling (query) is *ppizza*, the split set would be $\{1p, 2pp, 3pi, 4iz, 5zz, 6za, 7a\}$. The order of the members after $2pp$ is misplaced, thus this would lead to low similarity with the correct spelling (document) $\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$. Only $\{1p\}$ is common between correct and incorrect spell-splits. Hence, we propose BREAK-2, a split algorithm which is robust against such displacements.

We attach position number to the left if the numbering begins from the start, and to the right if the numbering begins from the end. Then the smallest position number would be selected between the two position numbers. If the position numbers are equal, then we select the left position number as a convention. Figure 1 gives an exemplification of this algorithm illustrated with splits of *pizza* and *hearts*.

If the misspelling is *ppizza*, the double-end split set would be $\{1p, 2pp, 3pi, 4iz, zz3, za2, a1\}$. Thus it gives higher similarity with the correct spelling (document) as compared to positional split, as the set members $\{1p, zz3, za2, a1\}$ would be matched.

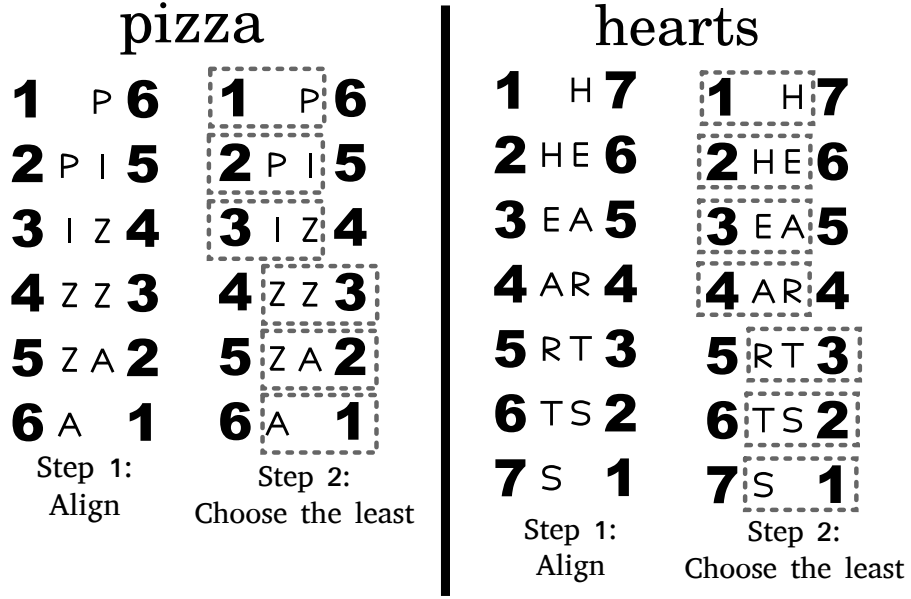


Figure 1: The process of BREAK-2. On the left, algorithm slices pizza into pieces. The output set would be $\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$. On the right, algorithm breaks hearts into pieces. The output set would be $\{1h, 2he, 3ea, 4ar, 5rt, 6ts, 7s\}$.

2.3 BREAK-OFF: SEQUENCING WITH OFFSETS

We propose another variant, BREAK-X-Off, which introduces offsets in the split-sets. (Here X stands for the BREAK variant number used). Let the BREAK-1 of pizza be

$$S_0 = \{1p, 2pi, 3iz, 4zz, 5za, 6a\}$$

which has no offsets. An offset of 1 would mean the counts would be displaced by 1, which is, $S_1 = \{2p, 3pi, 4iz, 5zz, 6za, 7a\}$. An offset of -1 would mean the counts would be displaced by -1, which is, $S_{-1} = \{op, 1pi, 2iz, 3zz, 4za, 5a\}$. Now the complete split-up would be, thus, $S = S_{-1} \cup S_0 \cup S_1$. Similarly, offsets can be applied to the BREAK-2 splits too.

The motivation behind BREAK-X-Off is that inclusion of offsets would be robust against the incorrect query provided. For

example, a spelling typographical error could have been caused by the insertion or deletion of the letter. Such insertions and deletions would displace the position sequences for the chunks. So the key here is to already index these possible errors, like for insertion, it would have an offset of 1 and for deletion, it would have an offset of -1.

For example, the split set for *pizza* which was originally

$$\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$$

after adding positional offset of 1 looks like:

$$\{2p, 3pi, 4iz, 5zz, 6za, 7a\}$$

Offset with double-ended split-set now looks like

$$\{2p, 3pi, 4iz, zz4, za3, a2\}$$

For deletion errors, positional offset of -1 is included. For example, the split set for *pizza* after adding positional offset of -1 looks like:

$$\{op, 1pi, 2iz, 3zz, 4za, 5a\}$$

Offset with double-ended split-set now looks like

$$\{op, 1pi, 2iz, zz2, za1, ao\}$$

The complete split-set would include offsets of -1, 0 and 1. Thus, complete split-set of *pizza* now looks like,

$$\{op, 1pi, 2iz, zz2, za1, ao, 1p, 2pi, 3iz, zz3, za2, a1, 2p, 3pi, 4iz, zz4, za3, a2\}$$

However, set members $\{op, ao, 2p, a2\}$ are unlikely to happen as the numbering always starts from 1. Hence we can ignore these members. The final set is now,

$$C = \{1pi, 2iz, zz2, za1, 1p, 2pi, 3iz, \\ zz3, za2, a1, 3pi, 4iz, zz4, za3\}$$

where C is the correct split-set.

Let the misspelling (query) be entered as *piza*. The I , incorrect split-set by this method would be:

$$I = \{1pi, 2iz, za1, 1p, 2pi, \\ 3iz, za2, a1, 3pi, 4iz, za3\}$$

Then the match $C \cap I$ is,

$$C \cap I = \{1pi, 2iz, za1, 1p, 2pi, \\ 3iz, za2, a1, 3pi, 4iz, za3\}$$

Thus this method received more number of term matches, $|C \cap I|$ than any other methods. Similarly this method can be visualized for other types of spelling errors like insertion of letters.

2.4 BREAK-N: SEQUENTIAL SPLITS FOR LONGER TERMS

The main drawback of BREAK-2 algorithm is that the double-ended counts would be inefficient for the longer words. With elongation, the advantage of having two extremities would fade away, causing a similar drawback like BREAK-1. This is undesirable for the non-natural language processing tasks like genome sequence analysis where a typical genetic sequence can be of thousands of base-pairs long. In this section, we tackle this

problem by proposing BREAK-n algorithm, which would generalize BREAK-2 and workaround for longer words.

2.4.1 Anchoring

Here, we introduce the notion of the anchor points. Notice that in BREAK-2 algorithm, we numbered the terms from two extremities. Now, instead of 2 extremities here, we number the terms relative to the n extremities or *anchor points*. For a sequence with a length l , divided into k -grams and n anchor points, the position of the i^{th} anchor point (pos_i) is given by,

$$\text{pos}_i = 1 + \frac{(i-1)(l+k-2)}{n-1} \quad (1)$$

Each anchor point i would have a corresponding vector $t_{e,i}$ having j entries. Every $t_{e,ij}$ entry will have information about the distance between i^{th} anchor point and their position j . Or simply,



$$t_{e,ij} = |\text{pos}_i - j| \quad (2)$$

Here e stands for entity which can be either a document d , or query q . We have illustrated a basic example in figure 2 to exemplify this process.

2.4.2 Redefining Intersection



Comparing the analogy from BREAK-2, we impose conditional intersection here. A term t_q in query and t_d in document is said to be *completely intersected* if any of their relative positions from

Query, $q = \text{piza}$



	p	pi	iz	za	a
From pos_1	$t_{q,11} = 0$	$t_{q,12} = 1$	$t_{q,13} = 2$	$t_{q,14} = 3$	$t_{q,15} = 4$
From pos_2	$t_{q,21} = 4$	$t_{q,22} = 3$	$t_{q,23} = 2$	$t_{q,24} = 1$	$t_{q,25} = 0$

Document, $d = \text{pizza}$



	p	pi	iz	zz	za	a
From pos_1	$t_{d,11} = 0$	$t_{d,12} = 1$	$t_{d,13} = 2$	$t_{d,14} = 3$	$t_{d,15} = 4$	$t_{d,16} = 5$
From pos_2	$t_{d,21} = 5$	$t_{d,22} = 4$	$t_{d,23} = 3$	$t_{d,24} = 2$	$t_{d,25} = 1$	$t_{d,26} = 0$

Figure 2: In this example, the words *piza* and *pizza* are given as q and d respectively, split into 2-grams. If we chose 2 anchor points, the position according to equation 1 would be at the first and the last (which are shown in arrows). Now we fill every $t_{e,ij}$ entry by calculating relative distance given in equation 2.

anchor points are same. To achieve this, we would investigate their column vectors at the matched positions.

Let the matching position for t_q and t_d be p_1 and p_2 , respectively. We would extract p_1 and p_2 column vectors and check their corresponding indices for every anchor point. Afterwards, we extract p_1 and p_2 column vectors, t_{q,p_1} and t_{d,p_2} respectively. Then we would see if any of their corresponding indices are same. If this happens, we say that they are *completely intersected*. In other words, any of the cells in the absolute difference between t_{q,p_1} and t_{d,p_2} should be 0. Mathematically,

$$\prod_{i=1}^n |t_{q,ip_1} - t_{d,ip_2}| = 0 \quad (3)$$

where n is the length of the column vector (number of the anchor points). If the resulting product around 0 is achieved, we would say that the term intersects completely between query and document.

Continuing the example from figure 2, let's assess the term za which is common between t_q ($p_1 = 3$) and t_d ($p_2 = 4$). The t_q column vector at $p_1 = 3$ is $t_{q,3} = [3 \ 1]^T$ and for t_d at $p_2 = 4$ is $t_{d,4} = [4 \ 1]^T$. Applying the equation 3, we get $(|3 - 4|)(|1 - 1|) = 0$, which means the term za is completely intersected. Analytically, this makes sense because position of za from last is same in the two entities, and hence it should be completely intersected. Although here we have shown the example of BREAK-n for a small word with two anchor points but BREAK-n can be logically extended to longer words with more anchor points.

2.5 COMBINING BREAK WITH RANKING FUNCTIONS

Before indexing the documents, the dataset must be split according to the desired BREAK variant. For the usage of BREAK-n, note that one can store the positional vectors while indexing the corresponding term, or generate the vector directly during evaluation. This choice highly depends on the time-space trade-off. However, in our experiments we have stored the vectors during indexing, prioritizing faster evaluation.

We have also combined length penalization heuristics (TAKE) with the probabilistic retrieval functions. The role of these heuristics is to reward the documents which are in closer to the length of the query. This can be useful in the spell correction and cognate detection experiments as misspellings do not much absolute length difference with the correct spelled ones. These length penalization heuristics can replace the conventional length normalization heuristics which are used in BM25 and Dirichlet. In addition, we have also introduced graphical error models

(MAKE) to analyze the confusion matrix generated during experimentation. The motivation behind this creation of graphical error model is to aid the identification and suggestion of solutions to the most probabilistic errors.

MAKE : GRAPHICAL ERROR MODELLING

Here we extend our proposed split-up heuristics to graphical error models or the algorithm MAKE. The motivation behind this creation of MAKE is to aid the identification and suggestion of solutions to the most probabilistic errors. For example, there are many common errors in the spelling typographical errors, like: *ie* is commonly mistaken with *ei*, adding or deleting of an extra *l* in *lly* and so on.

The algorithm 1 process of MAKE. Let Q be the query and D be the document. Thus $Q \cap D$ shows number of terms common between them. We are interested in the leftover terms in the sets. That means, we need to infer a certain pattern from leftover sets, which are $Q - \{Q \cap D\}$ and $D - \{Q \cap D\}$. Thus we can draw mappings to gather information of the corrections.

Let *first* and *second* be the **ordered sets** referring to $Q - \{Q \cap D\}$ and $D - \{Q \cap D\}$ respectively. If *first* or *second* are empty, then we insert empty token ϕ in them (Lines 2-5 of the algorithm).

If size of *first* is less than size of the *second*, then we keep inserting empty tokens ϕ in the middle of *first*, unless the size of *first* becomes equal to the size of *second* (Lines 6-8). Similarly, if the size of *second* is less than size of the *first*, then we keep inserting empty tokens ϕ in the middle of *second*, unless the size of *second* becomes equal to the size of *first* (Lines 9-11).

We now initialize a hash map *graph* (Line 12). We make pairs from the members of the *first* which corresponds to the *second* and insert in the *graph*. We insert pairs of the corresponding

Algorithm 1 MAKE algorithm

```
1: procedure MAKE(first, second)
2:   if first.size = 0 then
3:     first.insert( $\phi$ )
4:   if second.size = 0 then
5:     second.insert( $\phi$ )
6:   while first.size < second.size do
7:     position  $\leftarrow \lceil \frac{\text{first.size}}{2} \rceil$ 
8:     first.insert( $\phi$ , position)
9:   while first.size > second.size do
10:    position  $\leftarrow \lfloor \frac{\text{second.size}}{2} \rfloor$ 
11:    second.insert( $\phi$ , position)
12:  graph = {}
13:  for i  $\leftarrow$  0, first.size do
14:    map  $\leftarrow$  pair(first[i], second[i])
15:    graph.insert(map)
16:  for i  $\leftarrow$  0, first.size - 1 do
17:    map  $\leftarrow$  pair(first[i], second[i + 1])
18:    graph.insert(map)
19:  for i  $\leftarrow$  1, first.size do
20:    map  $\leftarrow$  pair(first[i], second[i - 1])
21:    graph.insert(map)
22:  graph.remove_duplicates()
23:  return graph
```

same index of *first* and *second* in the *graph* (Lines 13-15). We then insert pairs of the corresponding indexes of *first* and *second* in the *graph*, with one index ahead of others (Lines 16-18). In the same way, we insert pairs of the corresponding indexes of *first* and *second* in the *graph*, with one index before others (Lines 19-21).

After removing the duplicates, we can return the graph as a result. Following example shows the procedure of the algorithm.

Example 1, Deletion error: We will visualize *pizza* again. Let the correct split-set of *pizza* (ignoring the offsets) be D , which is indexed in the document. So, $D = \{1p, 2pi, 3iz, zz3, za2, a1\}$. Let the incorrect split-set of *piza* (ignoring the offsets) be Q given as the query. So, $Q = \{1p, 2pi, 3iz, za2, a1\}$. Thus the term matches are, $Q \cap D = \{1p, 2pi, 3iz, za2, a1\}$.

Now we can analyze the leftover *pizza* sets, which are $D - Q \cap D$ (*first*) and $Q - Q \cap D$ (*second*). Hence, $Q - Q \cap D = \{zz3\}$. Similarly, $D - Q \cap D = \{\phi\}$.

Graph is now constructed from the members of $D - Q \cap D$ to $Q - Q \cap D$. The graph constructed is $\{\phi \rightarrow zz3\}$.

Intuitively, $\phi \rightarrow zz3$ means that the letter *z* should have been inserted around position 3 from the last.

The query set Q , is mentioned here as I , which means incorrect split-set and the document set D , is mentioned here as C , which means the correct split set.

Example 2, Insertion error: For the misspelled word *pizzza*, we have:

$$C = \{1p, 2pi, 3iz, zz3, za2, a1\}$$

$$I = \{1p, 2pi, 3iz, 4zz, zz3, za2, a1\}$$

$$C \cap I = \{1p, 2pi, 3iz, zz3, za2, a1\}$$

$$C - C \cap I = \{\phi\}$$

$$I - C \cap I = \{4zz\}$$

Thus the graph constructed is, $\{4zz \rightarrow \phi\}$. Intuitively, $4zz \rightarrow \phi$ means that the letter *z* starting around 4th position should have been deleted.

Example 3, Substitution error: For the misspelled word *panc* and the correct word *pant*, we have:

$$C = \{1p, 2pa, 3an, nt2, t1\}$$

$$I = \{1p, 2pa, 3an, nc2, c1\}$$

$$C \cap I = \{1p, 2pa, 3an\}$$

$$C - C \cap I = \{nt2, t1\}$$

$$I - C \cap I = \{nc2, c1\}$$

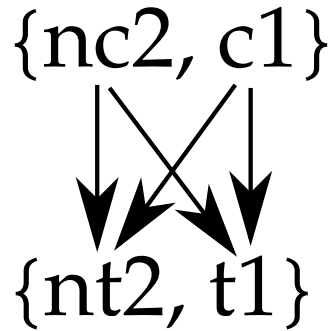


Figure 3: Graphical error model construction from *panc* to *pant*

The figure 3 clearly shows the set mappings done by the for loop. They correspond to the same index, one index ahead and one index behind from the first set to the second set.

Thus the graph constructed is, $\{nc2 \rightarrow nt2, nc2 \rightarrow t1, c1 \rightarrow nt2, c1 \rightarrow t1\}$. Intuitively, this graph depicts that the last letter should have been *t*.

Example 4, Swapping error: For the misspelled word *sieze* and the correct word *seize*, we have:

$$C = \{1s, 2se, 3ei, iz3, ze2, e1\}$$

$$I = \{1s, 2si, 3ie, ez3, ze2, e1\}$$

$$C \cap I = \{1s, ze2, e1\}$$

$$C - C \cap I = \{2se, 3ei, iz3\}$$

$$I - C \cap I = \{2si, 3ie, ez3\}$$



Figure 4: Graphical error model construction from *sieze* to *seize*

Thus the graph constructed is, $\{2si \rightarrow 2se, 2si \rightarrow 3ei, 3ie \rightarrow 2se, 3ie \rightarrow 3ei, 3ie \rightarrow iz3, ez3 \rightarrow 3ei, ez3 \rightarrow iz3\}$. Thus the graph neatly captures the idea that the letters *i* and *e* should have been swapped.

Example 5, Unequal set length: For the misspelled word *pth* and the correct word *path*, we have:

$$\begin{aligned} C &= \{1p, 2pa, 3at, th2, h1\} \\ I &= \{1p, 2pt, th2, h1\} \\ C \cap I &= \{1p, th2, h1\} \\ C - C \cap I &= \{2pa, 3at, th2\} \\ I - C \cap I &= \{2pt\} \end{aligned}$$

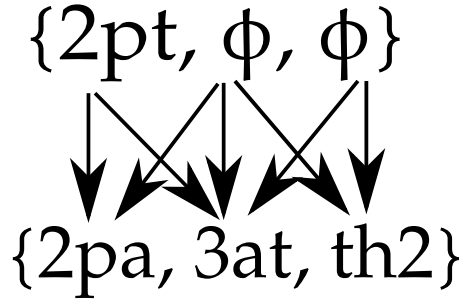


Figure 5: Directed graph guiding from the incorrect *pth* to the correct *path*

Since the size of topmost set was not equal to the bottom one, we inserted empty tokens in the top one unless they become of equal sizes. Thus the graph constructed is, $\{2pt \rightarrow 2pa, 2pt \rightarrow 3at, \phi \rightarrow 2pa, \phi \rightarrow 3at\}$. Thus the graph roughly conveys the information that the letter *t* should be second last.

Example 6, Unequal set length: For the misspelled word *patthhs* and the correct word *paths*, we have:

$$\begin{aligned} C &= \{1p, 2pa, 3at, th3, hs2, s1\} \\ I &= \{1p, 2pa, 3at, 4tt, th4, hh3, hs2, s1\} \\ C \cap I &= \{1p, 2pa, 3at, hs2, s1\} \\ C - C \cap I &= \{th3\} \\ I - C \cap I &= \{4tt, th4, hh3\} \end{aligned}$$



Figure 6: Graphical error model construction

Since the size of bottom set was not equal to the top one, we inserted empty tokens in the bottom set at the middle, unless they become of equal sizes. Thus the graph constructed is, $\{4tt \rightarrow \phi, th4 \rightarrow \phi, th4 \rightarrow th3, hh3 \rightarrow \phi, hh3 \rightarrow th3\}$. Thus the graph roughly conveys the information that the letter t and letter h should have been deleted.

In order to evaluate the efficiency of the graphical error model, we will add the probability of each edge of the graph to the score of the ranking function. In other words, this would be:

$$score(q, d) = \lambda \times rank(q, d) + (1 - \lambda) \times \pi(q, d)$$

where, λ is the weight ranging from $[0, 1]$, $score(q, d)$ is the similarity scoring function like BM25 [4] or Dirichlet [5] and $\pi(q, d)$ is the probabilistic estimate of given error occurring in the training data.

$$\pi(q, d) = \sum_{g \in G} count(g)^s$$

Here, G is the constructed graph and s is the strength parameter. $count(g)$ is the frequency of the error in whole training set of spelling errors. We have used s here, to avoid the overfitting

during the training, since $count(p)$ could heavily depend on the training data.

This function can also be normalized with respect to the cardinality of the pointer set. This could be useful since some of the extremely irrelevant incorrect matches would contain a lot of zeros. Thus normalization version, could avoid the skewness of the frequencies.

The normalized version or MAKE is:

$$\pi(q, d) = \frac{1}{|G|} \sum_{g \in G} count(g)^s$$

HEURISTICS ON WORD LENGTH SIMILARITY

4.1 GUIDELINES FOR FEATURE ENGINEERING IN RANKING FUNCTIONS

The main idea of the information retrieval systems is to include a proximity score in conjunction to the ranking function. A proximity score is an application-dependent score, like page-rank metrics, contextual similarity and so on. The function should have well engineered feature embedded in itself and it should be able to generalize other results. The objectives of the ranking function are:

- The score should be higher if the constituents of proximity score (real numbers or vectors) are similar.
- The proximity score should be below certain cut-off scores. These cut-off scores or limiting parameters can be determined through machine learning.
- The function should behave similar and able to generalize the behavior and working of the existing ranking functions in IR.

The assumption is taken that modeled function is bi-modal (as one has to consider extremities of the cut-off scores). Let $f(\mathbf{x}, \mathbf{y})$ be the modeled function, where \mathbf{x} and \mathbf{y} be the constituent vectors. In other words, this function determines sim-

ilarity between \mathbf{x} and \mathbf{y} and it is a function of $d(\mathbf{x})$ and $d(\mathbf{y})$ where $d(\cdot)$ is the distance metric.

Length normalization functions, especially in the BM25 or pivoted length function are inversely proportional to the TF-IDF score. Thus, the model would be $f(\mathbf{x}, \mathbf{y})$ inversely proportional to the TF-IDF score.

$$\text{TF-IDF}(\mathbf{q}, \mathbf{d}) \propto \frac{1}{f(\mathbf{x}, \mathbf{y})}$$

4.1.1 Nature of the Function Curve

Since the function has to be bi-modal distribution, distribution function would be modeled in the form asymmetrical inverted bell curve. Thus the trough of the curve should occur when distances of \mathbf{x} and \mathbf{y} are similar. In other words,

$$\frac{\partial f(\mathbf{x}, \mathbf{y})}{\partial (d(\mathbf{x}))} = 0 \text{ if } d(\mathbf{x}) \approx d(\mathbf{y}) \quad (4)$$

Similarly it can be shown that,

$$\frac{\partial f(\mathbf{x}, \mathbf{y})}{\partial (d(\mathbf{y}))} = 0 \text{ if } d(\mathbf{x}) \approx d(\mathbf{y})$$

From here onwards, arguments would be taken from the $d(\mathbf{x})$ perspective, as $d(\mathbf{y})$ would have similar arguments.

The function should decrease monotonically if $d(\mathbf{x}) < d(\mathbf{y})$.
In other words,

$$\frac{\partial f(\mathbf{x}, \mathbf{y})}{\partial (d(\mathbf{x}))} < 0 \text{ if } d(\mathbf{x}) < d(\mathbf{y}) \quad (5)$$

The function should increase monotonically if $d(\mathbf{x}) > d(\mathbf{y})$.
In other words,

$$\frac{\partial f(\mathbf{x}, \mathbf{y})}{\partial (d(\mathbf{x}))} > 0 \text{ if } d(\mathbf{x}) > d(\mathbf{y}) \quad (6)$$

4.1.2 *Nature of the Limits of the Curve*

Here the left extremity of the curve could be bounded to the parameter b_1 .

$$\lim_{d(\mathbf{x}) \rightarrow 0} f(\mathbf{x}, \mathbf{y}) = b_1 \quad (7)$$

Similarly, the right extremity of the curve could be bounded to the parameter b_2 .

$$\lim_{d(\mathbf{x}) \rightarrow \infty} f(\mathbf{x}, \mathbf{y}) = b_2 \quad (8)$$

The trough of the curve exists when $\frac{\partial f(\mathbf{x}, \mathbf{y})}{\partial (d(\mathbf{x}))} = 0$. Thus the limit is set to 1. It can not be set it to 0, because $f(\mathbf{x}, \mathbf{y})$ lies in the denominator of the scoring function.

$$\lim_{d(\mathbf{x}) \rightarrow d(\mathbf{y})} f(\mathbf{x}, \mathbf{y}) = 1 \quad (9)$$

The specific case of these guidelines can be applied to the recommendation systems.

4.2 DESIGN OF THE TAKE CURVE

Here, the function would be modeled using Richard's curve. The curve is defined by,

$$g(x) = l + \frac{u - l}{(A + e^{-B(x-M)})^{1/\nu}} \quad (10)$$

Here, l is lower limit, u is upper limit and M, ν, A, B are free parameters

The distance metrics are taken to be just real numbers. Thus, $d(x) = x$ and $d(y) = y$. The trough of the curve should occur when, $d(x) = d(y)$ or $x = y$. The function can be partitioned into three parts:

- monotonically decreasing Richard's curve, $g(x)$ when $x < y$
- monotonically increasing Richard's curve, $g(x)$ when $x > y$
- Value of heuristic function, $h(x, y) = 1$ when $x = y$

The definition of the function now looks like:

$$h(x, y) = \begin{cases} g(x_1) & \text{if } x < y \\ 1 & \text{if } x = y \\ g(x_2) & \text{if } x > y \end{cases} \quad (11)$$

where $x_1 \in [0, y]$, $x_2 \in [y, \infty]$ and $x_1, x_2 \subset x$

Applying equation (4), the constraint obtained is:

$$\left. \frac{\partial h(x, y)}{\partial x} \right|_{x=y} = 0 \quad (12)$$

From equation (5), the constraint obtained is:

$$\frac{\partial g(x_1)}{\partial x_1} < 0$$

From equation (6), the constraint obtained is:

$$\frac{\partial g(x_2)}{\partial x_2} > 0$$

From limiting condition (7), the constraint obtained is:

$$\lim_{x_1 \rightarrow 0} g(x_1) = b_1 \quad (13)$$

$$\left. \frac{\partial g(x_1)}{\partial x_1} \right|_{x_1=0} \approx 0 \quad (14)$$

From the limiting condition (8), the constraint obtained is:

$$\lim_{x_2 \rightarrow \infty} g(x_2) = b_2 \quad (15)$$

$$\left. \frac{\partial g(x_2)}{\partial x_2} \right|_{x_2=\infty} \approx 0 \quad (16)$$

From the limiting condition (9), the constraint obtained are:

$$\left. \frac{\partial g(x_1)}{\partial x_1} \right|_{x_1=y} \approx \left. \frac{\partial g(x_2)}{\partial x_2} \right|_{x_2=y} \approx 0 \quad (17)$$

$$\lim_{x_1 \rightarrow y} g(x_1) = \lim_{x_2 \rightarrow y} g(x_2) = 1 \quad (18)$$

After solving the constraints, and using the binomial approximation for small v , the solutions obtained are:

$$h(x, y) = \begin{cases} 1 + \frac{b_1 - 1}{1 + e^{B_1(x - cy)}} & \text{if } x < y \\ 1 & \text{if } x = y \\ 1 + \frac{b_2 - 1}{1 + e^{-B_2(x - (1+c)y)}} & \text{if } x > y \end{cases} \quad (19)$$

where,

- B_1 and B_2 are growth parameters, can be typically set to 1
- c is the trough curvature, where $c \in (0, 1)$

According to the requirements, length similarity would be rewarded between query and document lengths. In that case, $x = |d|$ and $y = |q|$. Intuitively, that means the value of the scoring function will be higher if $|d| \approx |y|$.

Substituting the values, we get:

$$h(|d|, |q|) = \begin{cases} 1 + \frac{b_1 - 1}{1 + e^{B_1(|d| - c|q|)}} & \text{if } |d| < |q| \\ 1 & \text{if } |d| = |q| \\ 1 + \frac{b_2 - 1}{1 + e^{-B_2(|d| - (1+c)|q|)}} & \text{if } |d| > |q| \end{cases} \quad (20)$$

This feature can now be substituted in the ranking function in place of length normalization function. For example, the original BM25 function is:

$$\text{score}(q, d) = \sum_{t \in d \cap q} f(t, q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t, d)}{f(t, d) + k \left(1 - b + b \frac{|d|}{\text{avgdl}}\right)}$$

The normalization feature $\left(1 - b + b \frac{|d|}{\text{avgdl}}\right)$ in the BM25 formula can be replaced with $h(|d|, |q|)$ to get our desired ranking function:

$$\text{score}(q, d) = \sum_{t \in d \cap q} f(t, q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t, d)}{f(t, d) + k (h(|d|, |q|))} \quad (21)$$

4.3 DESCRIPTION OF TAKE

The lengths of the misspelled word and correctly spelled word should almost be similar. Usually edit distance between misspelled and correctly spelled word are around 1 or 2. The idea here is to reward the documents more in which document length and query length are closer to each other, and similarly penalize the extremely dissimilar lengths.

Let $|q|$ be the query length and $|d|$ be the document length. Let $h(|q|, |d|)$ be the heuristic function to calculate length similarity.

The objective of the heuristic function should be:

1. $h(|q|, |d|)$ should increase if $|q|$ and $|d|$ are dissimilar

2. Similarly, $h(|q|, |d|)$ should decrease if $|q|$ and $|d|$ are similar

Here we have used two variants.

4.4 POWER PENALIZATION

This uses hyperparameter γ which varies from $[0, 1]$.

$$h(|q|, |d|) = (||q| - |d|| + 1)^\gamma \quad (22)$$

The nature of the curve is plotted in Figure 7.

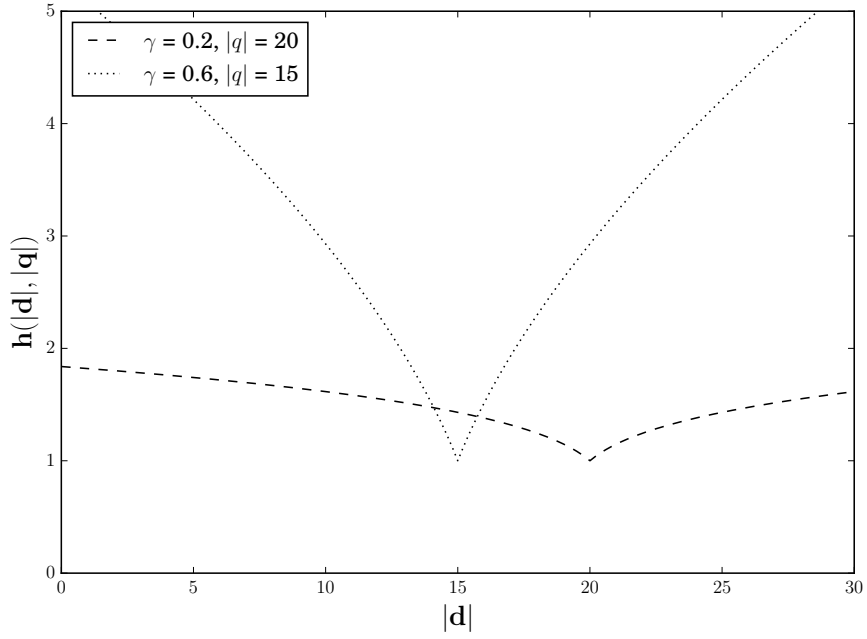


Figure 7: The nature of power penalization $h(|d|, |q|)$ curve. The value of $h(|d|, |q|)$ dips when $|d|$ approaches the $|q|$. As the γ increases, penalization would become more harsher.

4.5 BUCKET-SHAPED SIGMOIDAL PENALIZATION

Here, the function would be modeled using Richard's curve, also known as generalized sigmoidal curve [25]. The curve is defined by,

$$g(x) = l + \frac{u - l}{(A + e^{-B(x-M)})^{1/\nu}} \quad (23)$$

Here, l is lower limit, u is upper limit and M, ν, A, B are free parameters

As compared to normal sigmoid function, this function allows more flexibility in the choice of the parameters. Since, we have to model a penalization function, the aim would be to model a "trough" in the curve as $|q|$ approaches towards $|d|$. We divide this function into three parts:

- monotonically decreasing Richard's curve, $g(x_1)$ when $|d| < |q|$, or $\frac{\partial g(x_1)}{\partial x_1} < 0$
- monotonically increasing Richard's curve, $g(x_2)$ when $|d| > |q|$, or $\frac{\partial g(x_2)}{\partial x_2} > 0$
- Value of heuristic function, $h(|d|, |q|) = 1$ when $|d| = |q|$

where $x_1 \in [0, |d|]$, $x_2 \in [|d|, \infty]$ and $x_1, x_2 \subset |q|$. If we solve the limiting derivatives, use binomial approximations and remodel our parameters, we get the following penalization function:

$$h(|d|, |q|) = \begin{cases} 1 + \frac{b_1 - 1}{1 + e^{B_1(|d| - c|q|)}} & \text{if } |d| < |q| \\ 1 & \text{if } |d| = |q| \\ 1 + \frac{b_2 - 1}{1 + e^{-B_2(|d| - (1+c)|q|)}} & \text{if } |d| > |q| \end{cases} \quad (24)$$

where,

- b_1 is the upper limit on the left side
- b_2 is the upper limit on the right side
- B_1 and B_2 are growth parameters, can be typically set to 1
- c is the trough curvature, where $c \in (0, 1)$

The nature of this feature can be plotted as shown against various parameters (Figure 8).

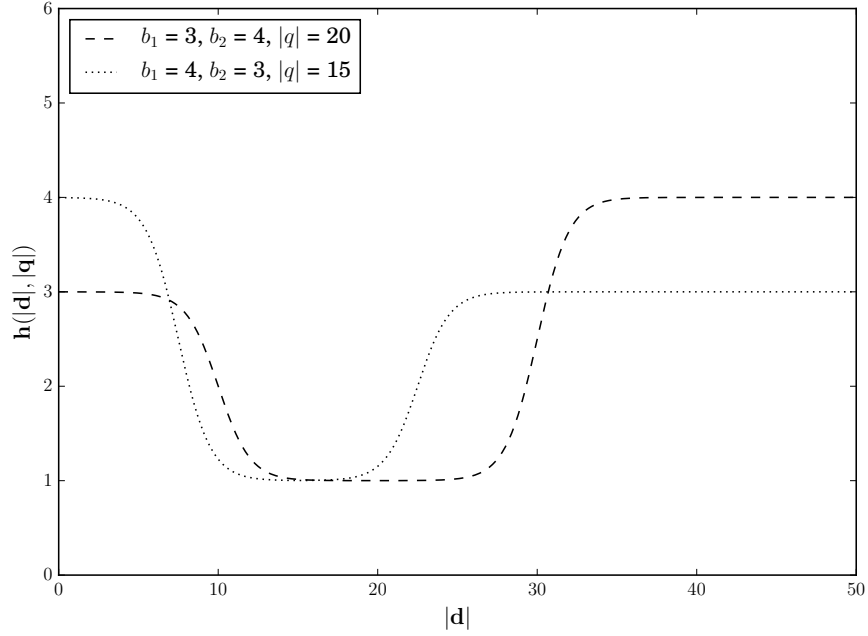


Figure 8: The nature of sigmoidal curve $h(|d|, |q|)$ with $B_1 = 1, B_2 = 1$ and $c = 0.5$

If we observe the graph we see a neat bucket-shaped trough when $|d|$ approaches $|q|$. It does not penalizes when the lengths are almost similar but starts penalizing when they are extremely dissimilar. This is feasible as a misspelling (query length) would be around the length of the correct spelling (document length).

Using harsh penalization when lengths are dissimilar can prune unnecessary spelling matches with this heuristic.

Clearly, these two penalization heuristics could do behave and achieve the stated objectives. We now demonstrate how we can use this penalization heuristics in our ranking functions.

We know that BM25 ranking function is given by:

$$\text{score}(q, d) = \sum_{t \in d \cap q} f(t, q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t, d)}{f(t, d) + k \left(1 - b + b \frac{|d|}{\text{avgdl}} \right)}$$

where $f(t, q)$ is the frequency of terms in queries, M is the number of documents, k is a hyperparameter which sets the bound, $f(t, d)$ is the frequency of terms in documents, $df(t)$ is the document frequency, b is the normalization parameter, $|d|$ is the document length and avgdl is the average document length.

Here $\left(1 - b + b \frac{|d|}{\text{avgdl}} \right)$ is the normalization heuristic. If we replace that heuristic with $h(|d|, |q|)$ we get:

$$\text{score}(q, d) = \sum_{t \in d \cap q} f(t, q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t, d)}{f(t, d) + k (h(|d|, |q|))} \quad (25)$$

If the value of $h(|d|, |q|)$ decreases, that is if $|d|$ and $|q|$ are similar, then the value of the $\text{score}(q, d)$ increases. This means similar document and query word lengths are rewarded. Vice versa can be said if the value of $h(|d|, |q|)$ increases.

We setup an experiment similar to section ??, to analyze the behavior of these heuristics on the spelling dataset. Here we

vary $h(|d|, |q|)$ heuristic variants to evaluate their performances. The results of the experiment are summarized in Table ??.

Clearly, we see that sigmoidal penalty outperforms the other variants. However, sigmoidal penalization function is difficult to train due to large number of parameters. Hence, we would only consider power penalization for the other experiments.

EXPERIMENTAL SETUP

We will evaluate our heuristics over three problems: isolated spelling correction, cognate detection and SNP detection. All of the hyperparameters of baselines presented in this section are tuned in the mentioned datasets according to their best performances. MeTA toolkit [26] is employed to build the search engine.

5.1 SPELLING CORRECTION

This problem would be tested using four different datasets:

1. **Conventional English Spelling Evaluation (CESE):** A collection of 4000 misspellings and their corrections were organized from Fawthrop's contribution in Birkbeck and Wikipedia spelling error corpus [27, 28]. These datasets are considered as the gold standard and are commonly tested by other researchers in the spelling correction. The dataset is divided into two parts: training set (comprises of 3000 words) and test set (comprises of 1000 words).
2. **Low Training Set Evaluation (LTSE):** We simulated 50000 English misspellings probabilistically. This dataset was then divided into two parts: training set (comprises of 5 words) and test set (comprises of 49995 words). This dataset is constructed to assess how algorithm behaves with less training data (LT).

3. **Corrupted Label Evaluation (CLE):** We simulated 50000 English misspellings probabilistically. This dataset was then divided into two parts: training set (comprises of 40000 words) and test set (comprises of 10000 words). From the 40000 words, we mislabeled 20000 of them randomly. This dataset is constructed to check the robustness of the algorithms.
4. **Hindi Spelling Evaluation (HSE):** A collection of 5000 Hindi errors were taken (error corpus constructed by ourselves). The dataset is divided into two parts: training set (comprises of 4000 words) and test set (comprises of 1000 words). We used the technique called *varn-viched* to tokenize the split the Hindi words into individual k-grams which is analogous to letter splitting technique in Roman languages.

We have chose five baselines for the evaluation purposes : a brute-force looped method [8], weighted longest common sub-sequence [9], finite state transducer [12], basic split-up (BREAK-o) with Jaccard similarity and skip-grams [10].

Table 2: Test dataset MRR results for spelling correction evaluation

Algorithm Applied	CESE	LTSE	CLE	HSE
Brute-force [8]	68%	70%	71%	38%
LCS [9]	61%	59%	52%	23%
FST [12]	81%	22%	19%	31%
BREAK-o + Jaccard	63%	68%	67%	35%
Skip-grams [10]	65%	70%	68%	29%
BREAK-o + Dirichlet	70%	65%	69%	66%
BREAK-o + BM25	69%	70%	68%	64%
BREAK-2 + Dirichlet	72%	72%	69%	68%
BREAK-2-Off + Dirichlet	75%	76%	71%	70%
BREAK-2 + TAKE + BM25	78%	79%	74%	75%
BREAK-2 + TAKE + MAKE + BM25	86%	38%	35%	84%

We used MRR (Mean Reciprocal Rank) here as our evaluation metric, since each misspelling had only one solution in our

dataset. Aspell [29] with the dictionary size of 60 has been used to create the spell-check dictionary for the English evaluation.

The results in Table 2 show that BREAK-2 with BM25 based length penalty (TAKE) and graphical error models (MAKE) performed the best in conventional English and Hindi spelling evaluation. BREAK-2 combined with the BM25 based length penalty (MAKE) performed the best in the corrupted label and low training dataset size evaluation, as these techniques are robust against data abnormalities.

5.2 COGNATE DETECTION

This experiment was performed on the dataset and evaluation scheme proposed by Taraka [3]. In this dataset, the word pairs are organized into cognate class numbers. Positive and negative labels are assigned to the same and different class numbers respectively. For comparisons, we chose SVM classifier for cognates detection as proposed by Houndrak et al. [13] and Naive Bayes based features as proposed by Alina et al. [15] The dataset was divided in the ratio of 3:1 for the training and testing purposes.

Table 3: Test dataset results for cognate detection experiment

Algorithm Applied	P@1	F1
Naive Bayes [15]	0.75	0.4
SVM [13]	0.78	0.46
BREAK-2-Off + Dirichlet	0.76	0.42
BREAK-2 + MAKE + TAKE + BM25	0.80	0.48

Precision at $k = 1$ and F1 scores are used to evaluate the algorithms used in this experiment. The results in Table 3 show

that BREAK-2 with MAKE and TAKE performed slightly better than the other results.

5.3 SNP DETECTION

This is a new problem for detecting SNPs (single-nucleotide polymorphism) which corresponds to genetic mutant identification suggested by Nguyen et al [2]. For this experiment, we simulated two datasets. The dataset D1 contained 10000 genomes. Each genome sequence comprises length of 500-600 base pairs for the artificial DNA sequences. 10 distinct mutations were created for each genome sequence. Then the dataset was populated with 99% noise, resulting in total size of D1 as 1 million genome sequences. The dataset D2 had 50000 protein sequences with length around 1000-1500 base pairs, which was populated with 99.9% noise, resulting in total size of D2 as 1.5 million protein sequences. The D2 dataset was artificially created with simulating the random patterns and mutations caused by the 21 proteins. The BREAK-o with Jaccard similarity is chosen as a baseline and we used MinDist proximity heuristic clubbed with BM25 and BREAK-o [30], to compare our results.

Table 4: Results for gene mutant detection simulation experiment

Algorithm	D1 (MRR)	D1 (NDCG)	D2 (MRR)	D2 (NDCG)
BREAK-o + Jaccard	0.33	0.16	0.31	0.12
MinDist + BM25 + BREAK-o	0.55	0.40	0.49	0.38
BREAK-n + Dirichlet	0.99	0.98	0.95	0.94

MRR and NDCG (Normalized Discounted Cumulative Gain) are used as metrics for our experiment. Table 4 shows that BREAK-n with Dirichlet outperformed others. We used 41 and

83 anchor points for D_1 and D_2 respectively which were obtained by experimentation on cross-validation dataset. This shows that our heuristics are scalable and works well with larger datasets.

ANALYSIS OF RESULTS

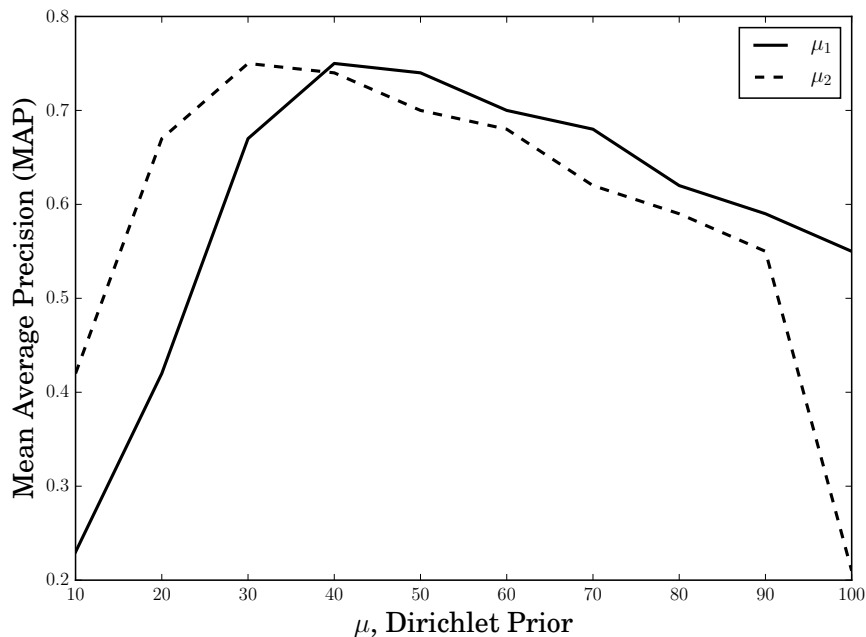


Figure 9: Effect of Dirichlet prior against MAP with BREAK-2 (μ_1) and with MAKE (μ_2)

Before indexing the documents, the dataset must be split according to the desired BREAK variant. For the usage of BREAK- n , note that one can store the positional vectors while indexing the corresponding term, or generate the vector directly during evaluation. This choice highly depends on the time-space trade-off. However, in our experiments we have stored the vectors during indexing, prioritizing faster evaluation.

6.1 ANALYSIS OF DIRICHLET PRIOR

The common trend between the experimental setup is that Dirichlet tends to perform better than Okapi BM25. This is due to the fact that Dirichlet prior smoothing factor, takes account the language modelling, which in turn, makes out to be more successful than vanilla BM25. We investigate the effect of Dirichlet prior with BREAK-2 μ_1 and with MAKE in μ_2 . Figure 9 shows that both reach the maximum MAP equally, however μ_1 reaches earlier.

6.2 ANALYSIS OF HYPERPARAMETERS

Now we analyze the effect of weighted average equation of MAKE, the hyperparameter tuning of the factor λ . We investigate the effect of hyperparameter sensitivity in isolated spelling correction λ_1 and with cognate detection in λ_2 .

Figure 10 describes the optimal conditions of the suitable hyperparameters. The late reach of λ_2 suggests that cognate detection experiments depend more on graphical error modelling, MAKE algorithm than the BREAK heuristic.

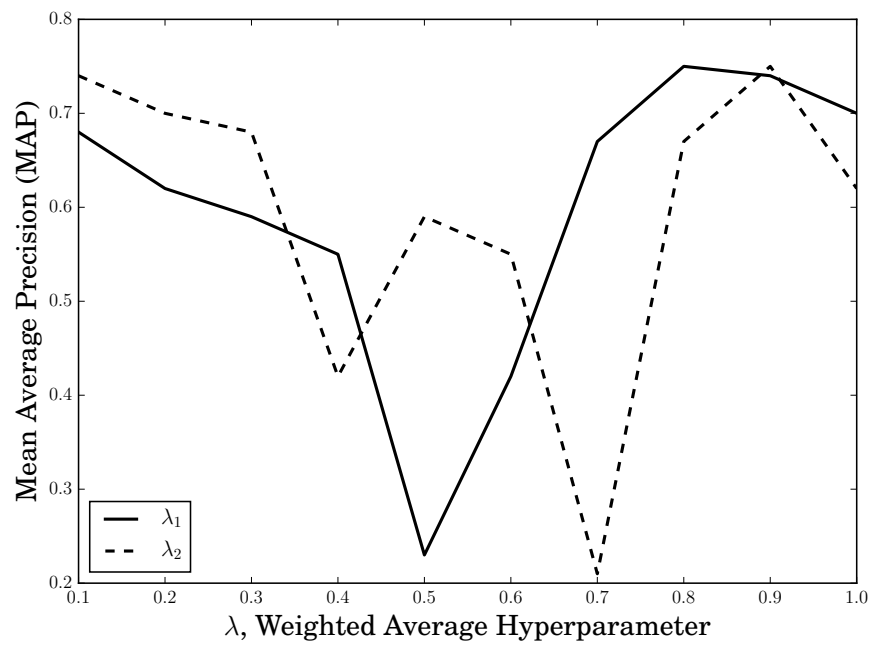


Figure 10: The effect of hyperparameter sensitivity in isolated spelling correction λ_1 and with cognate detection in λ_2 .

CONCLUSION

The central theme of this paper is to bring retrieval models and approximate string similarity algorithms together. We proposed a sequential splitting variants like BREAK-1, 2, Off, n. To relate these sequential splits, we proposed MAKE algorithm, which constructs the probabilistic distribution between the query and document split sets. Finally we designed the TAKE curve, to fit these heuristics in the normalization function of the ranking functions. In the experiments, we saw that our heuristics were robust against the datasets characteristics like mislabeling, low training dataset size and scales well in larger datasets. We plan to improve BREAK-n and apply SNP detection problem to real datasets and complex bioinformatics applications in the future.

BIBLIOGRAPHY

- [1] Karen Kukich. “Techniques for automatically correcting words in text.” In: *ACM Computing Surveys (CSUR)* 24.4 (1992), pp. 377–439.
- [2] Ken Nguyen, Xuan Guo, and Yi Pan. *Multiple Biological Sequence Alignment: Scoring Functions, Algorithms and Evaluation*. John Wiley & Sons, 2016.
- [3] Taraka Rama. “Automatic cognate identification with gap-weighted string subsequences.” In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 1227–1231.
- [4] S. E. Robertson and S. Walker. “Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval.” In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '94. Dublin, Ireland: Springer-Verlag New York, Inc., 1994, pp. 232–241. ISBN: 0-387-19889-X. URL: <http://dl.acm.org/citation.cfm?id=188490.188561>.
- [5] Chengxiang Zhai and John Lafferty. “A Study of Smoothing Methods for Language Models Applied to Information Retrieval.” In: *ACM Trans. Inf. Syst.* 22.2 (Apr. 2004), pp. 179–214. ISSN: 1046-8188. DOI: [10.1145/984321.984322](https://doi.org/10.1145/984321.984322). URL: <http://doi.acm.org/10.1145/984321.984322>.

- [6] Gianni Amati and Cornelis Joost Van Rijsbergen. "Probabilistic Models of Information Retrieval Based on Measuring the Divergence from Randomness." In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), pp. 357–389. ISSN: 1046-8188. DOI: [10.1145/582415.582416](https://doi.org/10.1145/582415.582416). URL: <http://doi.acm.org/10.1145/582415.582416>.
- [7] Hui Fang, Tao Tao, and Chengxiang Zhai. "Diagnostic Evaluation of Information Retrieval Models." In: *ACM Trans. Inf. Syst.* 29.2 (Apr. 2011), 7:1–7:42. ISSN: 1046-8188. DOI: [10.1145/1961209.1961210](https://doi.org/10.1145/1961209.1961210). URL: <http://doi.acm.org/10.1145/1961209.1961210>.
- [8] Peter Norvig. "How to write a spelling corrector." In: *De: http://norvig.com/spell-correct.html* (2007).
- [9] Aminul Islam and Diana Inkpen. "Real-word spelling correction using Google Web IT 3-grams." In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*. Association for Computational Linguistics. 2009, pp. 1241–1249.
- [10] Anni Järvelin, Antti Järvelin, and Kalervo Järvelin. "s-grams: Defining generalized n-grams for information retrieval." In: *Information Processing & Management* 43.4 (2007), pp. 1005–1019.
- [11] Heikki Keskustalo, Ari Pirkola, Kari Visala, Erkka Lepänen, and Kalervo Järvelin. "Non-adjacent digrams improve matching of cross-lingual spelling variants." In: *International Symposium on String Processing and Information Retrieval*. Springer. 2003, pp. 252–265.
- [12] Tommi Pirinen, Krister Lindén, et al. "Finite-state spell-checking with weighted language and error models." In:

Proceedings of LREC 2010 Workshop on creation and use of basic lexical resources for less-resourced languages. 2010.

- [13] Grzegorz Kondrak and Tarek Sherif. "Evaluation of several phonetic similarity algorithms on the task of cognate identification." In: *Proceedings of the Workshop on Linguistic Distances*. Association for Computational Linguistics. 2006, pp. 43–50.
- [14] Diana Inkpen, Oana Frunza, and Grzegorz Kondrak. "Automatic identification of cognates and false friends in French and English." In: *Proceedings of the International Conference Recent Advances in Natural Language Processing*. 2005, pp. 251–257.
- [15] Alina Maria Ciobanu and Liviu P Dinu. "Automatic Detection of Cognates Using Orthographic Alignment." In: *ACL* (2). 2014, pp. 99–105.
- [16] Yue Huang and Ling Zhang. "Rapid and sensitive dot-matrix methods for genome analysis." In: *Bioinformatics* 20.4 (2004), pp. 460–466.
- [17] Saul B Needleman and Christian D Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.
- [18] Temple F Smith and Michael S Waterman. "Identification of common molecular subsequences." In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.
- [19] William R Pearson. "[5] Rapid and sensitive sequence comparison with FASTP and FASTA." In: *Methods in enzymology* 183 (1990), pp. 63–98.

- [20] W James Kent. "BLAT—the BLAST-like alignment tool." In: *Genome research* 12.4 (2002), pp. 656–664.
- [21] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. "T-Coffee: A novel method for fast and accurate multiple sequence alignment." In: *Journal of molecular biology* 302.1 (2000), pp. 205–217.
- [22] ChengXiang Zhai and Sean Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Morgan & Claypool, 2016.
- [23] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. "Interpreting TF-IDF Term Weights As Making Relevance Decisions." In: *ACM Trans. Inf. Syst.* 26.3 (June 2008), 13:1–13:37. ISSN: 1046-8188. DOI: [10.1145/1361684.1361686](https://doi.org/10.1145/1361684.1361686). URL: <http://doi.acm.org/10.1145/1361684.1361686>.
- [24] Amit Singhal, Chris Buckley, and Mandar Mitra. "Pivoted Document Length Normalization." In: *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '96. Zurich, Switzerland: ACM, 1996, pp. 21–29. ISBN: 0-89791-792-8. DOI: [10.1145/243199.243206](https://doi.org/10.1145/243199.243206). URL: <http://doi.acm.org/10.1145/243199.243206>.
- [25] Colin PD Birch. "A new generalized logistic sigmoid growth equation compared with the Richards growth equation." In: *Annals of Botany* 83.6 (1999), pp. 713–723.
- [26] Sean Massung, Chase Geigle, and ChengXiang Zhai. "META: A Unified Toolkit for Text Retrieval and Analysis." In: *ACL 2016* (2016), p. 91.

- [27] Roger Mitton. "Birkbeck spelling error corpus." In: *Retrieved 2009 from <http://ota.ahds.ac.uk>* (1985).
- [28] Wikipedia. "Commonly misspelled English words." In: *Wikipedia* (2017).
- [29] Kevin Atkinson. "Spell Checking Oriented Word Lists." In: *GNU Aspell* (2016).
- [30] Tao Tao and ChengXiang Zhai. "An exploration of proximity measures in information retrieval." In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2007, pp. 295–302.