# Multi-Candidate Ranking Algorithm Based Spell Correction

Chao Wang
The Home Depot
Atlanta, GA
chao_wang1@homedepot.com

Rongkai Zhao
The Home Depot
Atlanta, GA
rongkai_zhao@homedepot.com

## ABSTRACT

Spell correction is an important component in Natural Language Processing (NLP). In the context of a product search engine, an effective spell correction system can improve the accuracy of the search results and reduce the occurrence of No Results Found (NRF). Conversely, a sub-optimal spell correction has negative effects, e.g., failing to correct misspelled queries, modifying correct queries into wrong ones. In this paper, three novel components / algorithms currently used in The Home Depot (THD) spell correction service is presented: (1) word embedding based dictionary construction; (2) multi-path candidates generation; (3) high dimensional cluster analysis based ranking model. The dictionary provides data about the inner relationships among the words for a given corpus. With the dictionary, the candidate generation algorithm recommends a set of correction candidates for several misspelling hypothesis, e.g., word editing error, word breaking error, word concatenation error, fat finger typing error, and so on. Then the ranking model projects the candidates into a high dimensional space and sorts them based on cluster density analysis. In the experiment, the new THD spell correction is compared with the old version (without these features), Lucene spell correction and Grammarly spell correction. The evaluation results indicated the THD spell correction has higher correction accuracy than the other widely used implementations.

## KEYWORDS

Spell correction, Spell checker, Spell corrector, Word embedding, Dictionary Building, Multi-candidate generation, Ranking model, Similarity context

**ACM Reference format:**
Chao Wang and Rongkai Zhao. 2019. Multi-Candidate Ranking Algorithm Based Spell Correction. *In Proceedings of the SIGIR 2019 Workshop on eCommerce (SIGIR 2019 eCom)*, 8 pages.

## 1 INTRODUCTION

The spell correction [1; 2] has been widely used in search engines, works as a "gate keeper" for query parsing [3]. Generally, spell correction has two components: spell checker and spell corrector. The spell checker is used to check the validity of the queries at word level as well as phrase level. If a query is valid, no action is needed from the spell corrector, otherwise, the query is passed onto spell corrector for revision. The spell corrector has a set of common error hypothesis, these hypothesis are based on the observation of common user mistakes. Unlike deep learning based approaches, we found the deterministic approach has much higher accuracy in a close domain. Common user mistakes including wrong spelling of a word, fat finger typing error, failing to break a composite word, unnecessarily creating a composite word, aggressive device native word level spell corrector introduced error, phonetic error, foreign language input, keyboard malfunction, etc. For each hypothesis, correction candidates will be generated and ranked by the cluster density in a high dimensional word embedding space. The final result is the most probable set ordered by the rank.

Categorized by the correction context, most standalone spell corrections (e.g., Jazzy spell correction [4], Aspell spell correction [5] and Hunspell spell correction [6]) are word-level approaches, which correct the misspelled words without considering the context information. The other spell corrections (Lucene spell correction [7], Grammarly spell correction [8], Microsoft Bing spell correction [9] and Google spell correction [10]) adopt context information. In other words, they are context based solution.

Categorized by the correction methods, most of the spell correction algorithms [1; 4; 6; 7] use Edit Distance (e.g., insertion, deletion, substitution) [11] and Phonetics Matching [12] as objective function to find closely related words. Some other spell corrections [13; 14] construct noisy channel models which recover the intended correction $c$ in word set $C$ from a misspelled word $w$ to maximize $Pr(c)Pr(w|c)$ where $c \in C$; $Pr(c)$ is a prior model of word probabilities; $Pr(w|c)$ is a model of the noisy channel for word transformations from $c$ to $w$ due to Edit Distance. Another spell corrections [15; 16] adopt Deep Neural Networks [17] based language models. Xie et al. [15] proposed to use char-level encoder-decoder recurrent neural network [18] to learn the character relationships within the words and phrases, which can avoid the problem of out-of-vocabulary words. Chollampatt and Ng [16] presented a multi-layer convolutional encoder-decoder neural network [19] for this char-level correction. Based on their analysis, the proposed network can cover more grammatical errors than recurrent neural network.

Before continuing onto further detail, here are some examples of common spelling problems and corrections:

(1) Single word error correction, e.g. "garage dor opener" -> "garage door opener";

(2) Multi-word errors correction, e.g. "garge dor opener" -> "garage door opener";

(3) Word breaking issue, e.g. "kohlertoilet" -> "kohler toilet";

(4) Word breaking issue with spelling errors, e.g. "kholertiolet" -> "kohler toilet";

(5) Word concatenation issue, e.g. "tom cat mouse trap" -> "tomcat mouse trap";


*In: J. Degenhardt, S. Kallumadi, U. Porwal, A. Trotman (eds.):*
*Proceedings of the SIGIR 2019 eCom workshop, July 2019, Paris, France, published at http://ceur-ws.org*

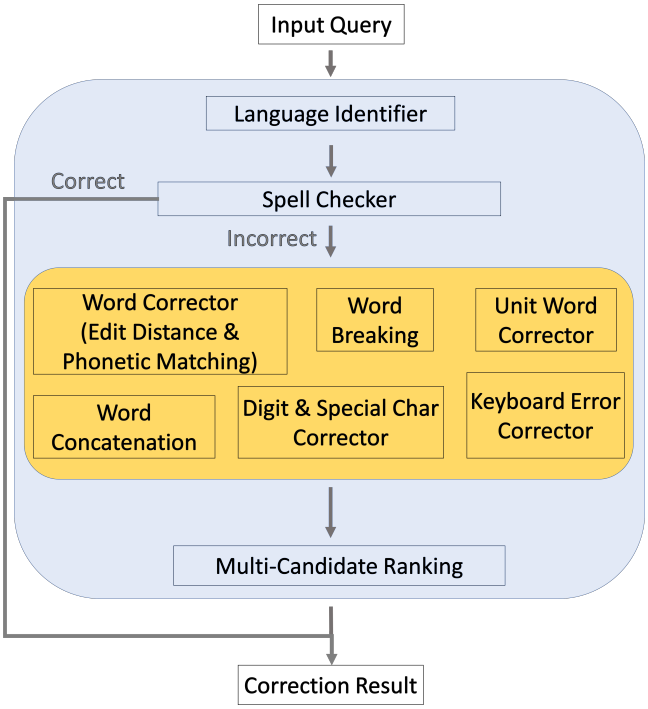


**Figure 1: Spell Correction Architecture**

(6) Word concatenation issue with spelling errors, e.g. "replace ment light bulb" -> "replacement light bulb";

(7) Word correction containing digits and special characters, e.g. "door ;ocks" -> "door locks";

(8) Real word errors correction, e.g. "mug knife" -> "mud knife".

Where the real word error is that the individual words ("mug", "knife") of the query are valid but the phrase ("mug knife") does not make sense.

## 2 ARCHITECTURE

In this paper, three main components / algorithms are presented: (1) word embedding based dictionary construction; (2) multi-path candidates generation; (3) high dimensional cluster analysis based ranking model.

As shown in the architecture (see Figure 1), the input query is first passed into Language Identifier to check which language the query belongs to, e.g., Spanish or English (English spell correction is mainly focused in this paper). After that, the query is put into Spell Checker to verify the validity. If valid, directly return the original query as final result; if not, pass the query into Spell Corrector. In Spell Corrector, the invalid query will be corrected by all of means synchronously, e.g., word correction, word breaking, word concatenation, fat finger typing error, and so on. This procedure may generate multiple candidates based on different correction components. Then these candidates are ranked based on the closeness of their word vectors clustering. Finally, the best candidate is returned as the correction result. In addition, the proposed dictionary works through the whole architecture.

## 3 WORD EMBEDDING BASED DICTIONARY CONSTRUCTION

### 3.1 Traditional Dictionary Structure

Most of the spell correction dictionaries [6; 7] are just composed of unique English words. However, it lacks of the word relationship information for the context-based spell correction algorithms [8–10; 13–16]. In order to extract the context information, the following methods can be adopted: Given a specific corpus, e.g., user query log, calculate the occurrence of every unique valid bigram words to construct a bigram co-occurrence model; feed the corpus into some char-level network models [15; 16] to learn the inner relationships among the characters. However, these methods still have some drawbacks. The co-occurrence model is only limited to bigram coverage, which is hard to be extended to n-gram (n > 2) words due to the limited memory space. In addition, the construction rules of n-gram (n > 2) words model are much restricted, which can easily lead to an overfitting issue. Conversely, the char-level model is much flexible to generate some unreasonable correction results.

### 3.2 Multi-Source Dictionary Construction Using Word Embedding

For our spell correction, in addition to bigram co-occurrence model, word embedding model based dictionary construction is proposed to extract the context information from the corpuses. We use user query log, product catalog and selected Wikipedia documents for context extraction. Wikipedia documents that are not related to our product catalog are pruned.

In order to better capture and utilize the word relationships in the context sensitive environment for our spell correction purpose, Word2Vec [20] is chosen as our word embedding model [21], it converts words into vectors and project them into a n-dimensional vector space, which could reveal word relationships in terms of geometric orientations.

The three datasets (product catalog dataset, user query log dataset and related Wikipedia dataset [22]) are integrated based on word embedding model as the following diagram (see Figure 2). Initially, the dictionary model is built with product catalog dataset as the kernel part. The catalog dataset has stronger correlations than the query log dataset and Wikipedia dataset. In addition, it is much more accurate (containing few misspellings) than the query log dataset. Therefore, it can be used to build the initial model and be helpful for the fast and accurate training convergence. However, it also has some drawbacks, e.g., limited contents (small coverage) and finite semantic expressions (low flexibility), because they are mainly provided by the product vendors. The query log dataset can complement these missed contents. Therefore, the query log dataset is incrementally feed into the initial model so that the model can capture more information. Along with the increase of embedding model coverage, additional noise (misspelled words) is also introduced. The two datasets, especially the user query log, contain sizable misspelling errors. The occurrences of some error bigram words are very high (very popular), which is hard to be removed based on threshold mechanisms. Wikipedia dataset based embedding model is adopted to trim the incorrect contents from the already trained model due to its higher coverage and diluted percentage of those

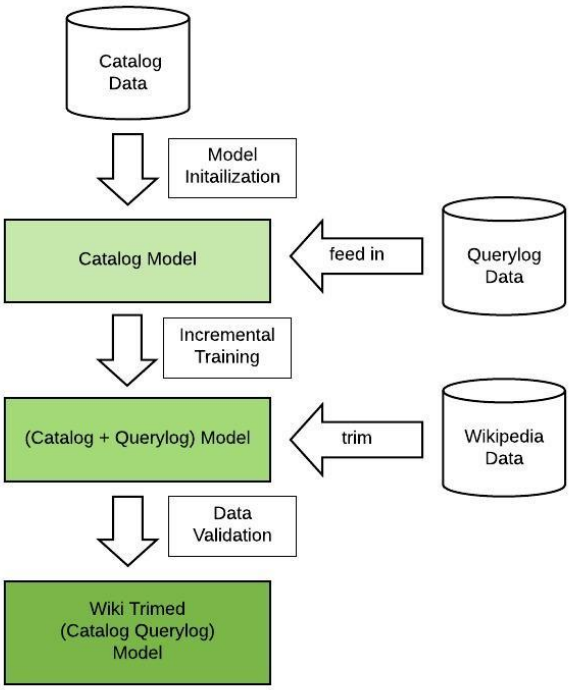


**Figure 2: Integration Diagram of Different Datasets Based Embedding Models**

errors. In addition, the Wikipedia model can also be used to validate whether a given bigram is a proper phrase or not in the bigram co-occurrence model.

Based on the constructed model, all of feature vectors of the existing words can be generated by it. So the similarity score of the bigram words can be calculated via Equation (1).

$$S(w_a, w_b) = \cos\theta = \frac{\vec{v}_a \cdot \vec{v}_b}{\mathbf{v_a v_b}} \tag{1}$$

where $S(w_a, w_b)$ represents the similarity score of the bigram words $w_a$ and $w_b$; $\theta$ represents the angle between the words $w_a$ and $w_b$; $v_a$ represents the feature vector of word $w_a$; $v_b$ represents the feature vector of word $w_b$.

# 4 MULTI-CANDIDATE GENERATION

## 4.1 Introduction of Different Spell Corrector Components

In the proposed spell correction, multiple correction candidates can be generated by means of different correction components listed as follows:

1. Word Corrector: It is mainly referred to single misspelled word correction. Given a misspelled word, all of the pronunciation similar words are retrieved based on phonetics matching. Among the retrieved words, the one with the smallest edit distance is chosen as the best correction result. For example, "garadge" -> "garage".

2. Word Breaking: It is to break a misspelled word into multiple valid words. In addition, the combination of the words should make sense. For example, "cordlessdrill" -> "cordless drill".

3. Word Concatenation: It is to concatenate multiple valid or invalid words into one valid word. Maybe the original words are all valid, but their combination does not make sense. The invalid combination is called real word error. For example, "dish washer" -> "dishwasher".

4. Fat Finger Typing Error: Similar to Word Corrector, it adopts keyboard layout to retrieve all of the possible correction words rather than using phonetics matching. For example, "gloor" -> "floor" where "g" is near to "f" on the keyboard.

5. Digit & Special Character Error Corrector: Similar to alphabetic characters word corrector, it is to identify if the digits or special characters of the query are useless or not. Then go for different correction methods based on the identification. For example, "drill1" -> "drill".

6. Unit Word Corrector: It is used to correct misspelled unit words, most of the unit words in the query are followed by a numeric token. For example, "18 voult drill" -> "18 volt drill".

## 4.2 Structures of Different Spell Corrector Components

Generally, these spell corrector components can be organized as two kinds of structures: Cascade Structure and Parallel Structure.

*4.2.1 Cascade Structure.* As shown in Figure 3, all of the corrector components are cascaded one by one. Every component has a user determined threshold (passing occurrence). For any specific component, if the occurrence of the correction result is higher than the threshold, it will be returned as the final result; if not, go to the next corrector component. The cascade structure has lower CPU runtime complexity. However, it may be stuck with some sub-optimal correction result. For example, suppose the best correction result of a query should be gotten from Word Concatenation (Component NO. 3). But the occurrence of the correction result from Word Corrector (Component NO. 1) has been higher than the given threshold. Then the final result is not the best one. So it depends on the cascade order of the correction components and the threshold of every component. However, it is hard to change or tune them to get the best performance, which varies for different queries.

*4.2.2 Parallel Structure.* As shown in Figure 4, all of the correction components can also be organized in a parallel structure. Different from the cascade structure, all of the possible correction results of the corrector components can be obtained. Then these candidates can be put into a ranking system (Described in Section 5) to get the best one. The advantage of the parallel structure is the global optimal feature of the correction results. In addition, the defined thresholds are not required. However, it needs more CPU power than the cascade structure. In the proposed spell correction, the parallel structure is adopted to get the best correction result.

# 5 RANKING MODEL

In this paper, the ranking model of spell correction acts as a sorter and selector of the generated correction candidates. In this section, two kinds of ranking models are presented: unigram word & bigram

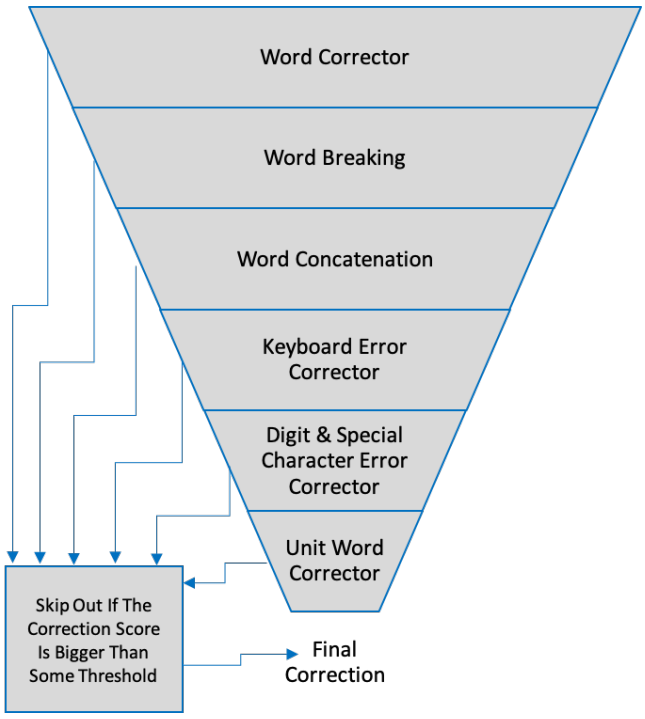


**Figure 3: Cascade Structure**

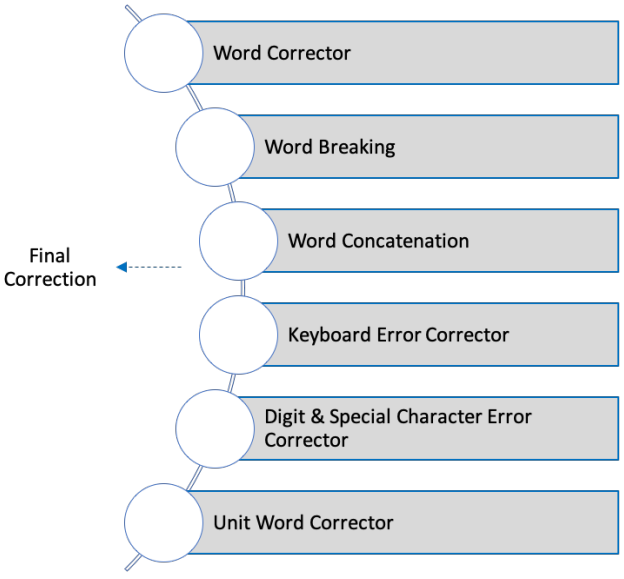


**Figure 4: Parallel Structure**

words occurrence based ranking model, and word embedding based ranking model.

## 5.1 Unigram Word & Bigram Words Occurrence Based Ranking Model and The Problems

For a given corpus, the occurrence of every existing unigram word & bigram words can be gotten through occurrence accumulation. The occurrence can reflect the popularity of the unigram word and the bigram words. For single word candidate, its ranking score can be defined as the occurrence of the unigram word. Similarly, for two-word candidate, its ranking score can be defined as the occurrence of the bigram words. However, not all of the candidates are only composed of single word or two words. As the number of words increasing from 2 to more, it is impossible to record every n-gram occurrence for computation. As for the candidates containing more than three words, the following equation is used to calculate the occurrence of the candidate.

$$O_q = \sum_{i=0}^{n-1} O(w_i, w_{i+1}) \tag{2}$$

where $O_q$ represents the occurrence of the candidate; $n$ represents the number of the words in the candidate; $w_i$ represents the $i$th word; $O(w_i, w_{i+1})$ represents the occurrence of the bigram words $w_i$ and $w_{i+1}$.

Based on the above method, all of the candidates have the occurrence value. Therefore, for multiple correction candidates of a given query, these candidates can be ranked based on their occurrences.

However, the method also has three drawbacks:

1. The occurrence depends on the quality of the source data. If the source data contains much noises, the occurrences of some wrong bigram words are also very big.

2. Suppose there are more than one kind of data sources (e.g., user query log and product catalog), for a given candidate, every data source may have an occurrence. However, it is hard to combine the occurrences together for the same candidate because different data sources have different noise level and different signal coverage.

3. In Equation (2), the occurrence of the candidate is defined as the mean value of the bigram words occurrences. But the variance of the bigram words occurrences is not considered. Suppose there are two candidates, one of them has big mean and big variance; the other one has small mean and small variance. It is hard to conclude which one is better.

Therefore, a novel ranking model is proposed to solve the above problems.

## 5.2 Word Embedding Based Ranking Model

As described in Section 3, a novel word embedding based dictionary is generated. The dictionary is combined with the product catalog dataset, the query log dataset and the related Wikipedia dataset based on word embedding mechanism. They works as different roles in the integration. The product catalog dataset acts as an initial model builder; the user query log dataset acts as an incremental data feeder; the related Wikipedia dataset acts as a data validator. In other words, the first two datasets are used to build a Core Embedding Model(CEM) and the Wikipedia dataset is used to build an Auxiliary Embedding Model(AEM), where all of the noises and incorrectly paired words in CEM are removed by cross validation with the information contained in the AEM. This integration mechanism

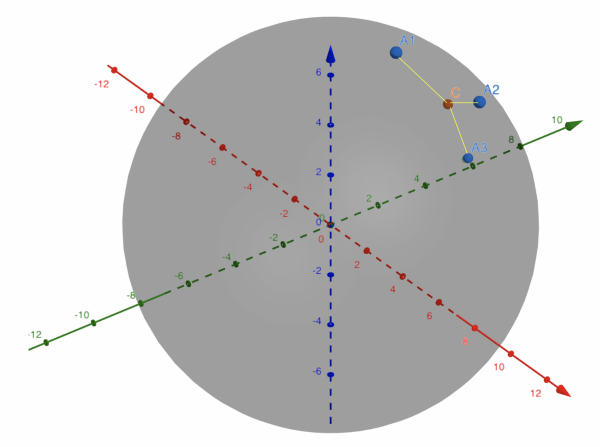


**Figure 5: The Generation of The Centroid Vector**

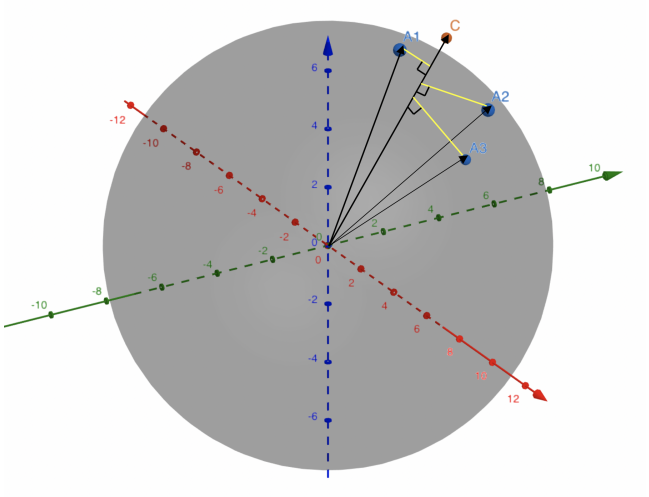


**Figure 6: The Centroid Vector Based On The Perpendicular Distance**

solves Problem 1 and Problem 2 in Section 5.1. In order to solve Problem 3, different ranking metrics are proposed.

In order to best describe the validity of a given phrase containing multiple words, a good ranking metric should be robust to the number of words and represent the closeness of the word vectors. Based on the Word2Vec models, the valid n-grams words tends to cluster closely in the embedding space. Therefore, for any given multi-word candidate, the spreadness of the set of embedded vectors corresponding to those words could be a good indicator of how likely this candidate is valid. The less spread the word vectors, the more likely the candidate is valid. Then, by calculating the average distance of all the vectors to their centroid vector, a ranking score can be generated to define the validity of the candidate. Mathematically, it is a Least Squares optimization problem in n-dimensional vector space. As shown in Figure 5, the centroid vector $C$ needs to be fitted by a set of word vectors $A_1$, $A_2$ and $A_3$ in a given phrase.

Goal and Metric:

1. For a given phrase, the centroid vector should satisfy that its distances to all the word vectors could be calculated and aggregated to describe the spreadness/closeness of word vectors.

2. The metric need to capture the outliers such that if any words are more distant to the center than the others, then all the words in the phrase as a whole should not be considered coherent enough than the case where all the word vectors have similar distances to the center. Squared distance is a good choice here.

Objective Function Choices:

Based on the above requirements, an objective function is proposed, which is in terms of the Euclidean distances of the center to all the tips of the polytope formed by the word vectors ($d$ dimensional space):

$$f(v, C) = Argmin_{C \in R^d} \sum_{i=1}^{n} \left\| \vec{v}_i - \vec{C} \right\|^2, \tag{3}$$

where the center $C$ should have the property such that the sum of the squared distances from all the words in a candidate should be minimized. Since this is a quadratic equation, by taking the first order derivative and setting it to 0, the solution can be obtained:

$$\vec{C} = \frac{\sum_{i=1}^{n} \vec{v}_i}{n}. \tag{4}$$

This centroid vector is pointing to the center of the polygon or polytope formed by the given set of word vector projections on the surface of the unit n-sphere centered at the origin. Furthermore, the distances from all the word vectors for the given phrase to it could describe the spreadness of the words. Intuitively, less spread means better. The solution vector $C$ usually lies in the interior of the polygon or polytope as shown in Figure 5, which is not a unit vector, thus not on the surface of the n-sphere. The solution is unique.

Another choice could be the centroid obtained from the spherical K-means method, where $K = 1$ in this case. The centroid vector $C$ needs to minimize the sum of squared perpendicular distances from all other vectors to itself, as shown in Figure 6. The objective function is

$$f(v, C) = Argmin_{C \in R^d} \sum_{i=1}^{n} [sin(\vec{v}_i, C) * \|\vec{v}_i\|]^2. \tag{5}$$

$v_i$ is an unit vector, it can be transformed as

$$\begin{aligned} fv, C) &= Argmin_{C \in R^d} \sum_{i=1}^{n} [1 - \cos^2(\vec{v}_i, C)] \\ &= Argmin_{C \in R^d} \sum_{i=1}^{n} [1 - < \vec{v}_i, \vec{C} >^2]. \end{aligned} \tag{6}$$

The solution to the equation is not unique and the iterative approximation method is not guaranteed to converge. This method is more closely related to the plane fitting method in high dimensional spaces. One other method closely related to the spherical K-means has the following objective function:

$$f(v, C) = Argmin_{C \in R^d} \sum_{i=1}^{n} [1 - \cos(\vec{v}_i, C)], \quad (7)$$

which seeks to minimize the loss of the similarities and their maximum value 1. This method has the same drawbacks as the previous one does. Therefore, Equation (3) is chosen as the objective function of the ranking metric in this paper.

After comparisons, the ranking metric is proposed as the mean squared distance of all the tips of the polytope formed by word vectors to the center $C$ :

$$\hat{D} = \frac{\sum_{i=1}^{n} \|\vec{v}_i - \vec{\hat{C}}\|^2}{n}, \quad (8)$$

where $\hat{D}$ is the spreadness score of the ranking metric; the center vector $C$ is obtained from the Euclidean distance based objective functions as shown in Equation (3). It could correctly capture the spreadness of the word vectors and has the range $[0, n]$. Smaller value $\hat{D}$ indicates better result (more closeness of the word vectors in the phrase).

# 6 EXPERIMENT

In this experiment, different kinds of evaluations are implemented on a collected testing dataset. It involves the comparisons between THD spell correction with the presented algorithms (word embedding based dictionary generation, multi-candidate generation, word embedding based ranking model) and that without them. In addition, the proposed THD spell correction is also compared with Lucene spell correction [7]. All of the testing are specified in different correction types.

## 6.1 Evaluation Dataset

The testing dataset contains correct queries and 9 error types of queries, collecting from THD query log. It has 2,095,028 terms totally.

(1) Correct queries (1,559,534 terms), occupies about 74.44% of total queries.

(2) Brand name related errors (4,678 terms), e.g., "ryoby drill" for correct spelling "ryobi drill", occupies about 0.22% of total queries.

(3) Non-word errors (10,083 terms), e.g., "garage door openr" for correct spelling "garage door opener", occupies about 0.48% of total queries.

(4) Real word errors (1,528 terms), e.g., "mug knife" for correct one "mud knife", occupies about 0.07% of total queries.

(5) Word breaking errors (10,594 terms), e.g., "firepit" for correct one "fire pit", occupies about 0.51% of total queries.

(6) Word concatenation errors (89,762 terms), e.g., "night light replace ment bulbs" for correct one "night light replacement bulbs", occupies about 4.28% of total queries.

(7) Phonetics related errors (345,323 terms), e.g., "foto frame" for correct one "photo frame", occupies about 16.48% of total queries.

(8) Product types related errors (9,490 terms), e.g., "hamer drill" for correct one "hammer drill", occupies about 0.45% of total queries.

(9) Unit word related errors (50,538 terms), e.g., "12 vot cordless drill" for correct one "12 volt cordless drill", occupies about 2.41% of total queries.

(10) Dimension word related queries (13,498 terms), e.g., "4in.x 4in. wall tile" for correct one "4in. x 4in. wall tile", occupies about 0.64% of total queries.

All of the queries are labelled by Microsoft Bing spell correction API [23] as ground truth. Since Google does not provide with API service for spell correction, it cannot be used for the evaluation.

## 6.2 Evaluation Method

The testing dataset is divided into correct queries and different error types. The correction accuracy (see Equation (9)) can be used to describe the evaluation performance.

$$P_{accuracy} = \frac{N_{correction}}{N_{total}} \quad (9)$$

where $P_{accuracy}$ represents the correction accuracy; $N_{correction}$ represents the number of algorithm-modified (e.g., THD spell correction) results which are the same with ground truth; $N_{total}$ represents the number of correct queries or the number of queries in some specific error type.

For the correct queries, the correction accuracy can also be called true positive rate or recall. For different error types, true negative rate can be derived from the correction accuracy.

## 6.3 Result

The 2 million Bing-labelled queries dataset has totally 10 different correction types. As shown in Table 1, (1) correct queries are represented by "Correct"; (2) brand name errors are represented by "Brand"; (3) non-word errors are represented by "NWE"; (4) real word errors are represented by "RWE"; (5) word breaking errors are represented by "Break"; (6) word concatenation errors are represented by "Concatenate"; (7) phonetics errors are represented by "Phonetic"; (8) product type errors are represented by "Product"; (9) unit word errors are represented by "Unit"; (10) dimension errors are represented by "Dim".

In addition, the following correction algorithms are compared based on the dataset:

(1) THD spell correction with bigram co-occurrence model (described in Section 3.1), and cascade structure of corrector components (described in Section 4.2.1), is the first version of THD spell correction and is represented as "THD SC v1".

(2) THD spell correction with word embedding based context dictionary (described in Section 3), parallel structure based multi-candidate generation (4), and word embedding based ranking model (described in Section 5), is the second version of THD spell correction and is represented as "THD SC v2".

(3) Lucene spell correction [7], is represented as "Lucene SC".

As shown in Table 1, THD spell correction v1 performs better than Lucene spell correction on overall correction accuracy. For different types of queries, THD spell correction v1 also has higher accuracy than Lucene spell correction except real word error type. The biggest difference between THD spell correction v1 and Lucene spell correction is that the first method adopts bigram co-occurrence model to get the context information among the words of a given phrase. The experiment results prove that adopting the context

**Table 1: Spell Correction Comparison on 2 Million Bing-Labelled Queries Dataset**

| | Correct | Brand | NWE | RWE | Break | Concatenate | Phonetic | Product | Unit | Dim | Overall |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Count** | 1,559,534 | 4678 | 10,083 | 1,528 | 10,594 | 89,762 | 345,323 | 9,490 | 50,538 | 13,498 | 2,095,028 |
| **THD SC v1** | 91.57% | 64.71% | 42.94% | 15.05% | 22.71% | **25.25%** | **40.96%** | 62.17% | **33.19%** | **23.76%** | 77.71% |
| **THD SC v2** | **95.36%** | **67.36%** | **43.18%** | 17.24% | **23.07%** | 24.29% | 40.03% | **64.98%** | 32.25% | 23.37% | **80.32%** |
| **Lucene SC** | 81.24% | 43.50% | 26.10% | **21.92%** | 4.18% | 13.97% | 20.72% | 48.23% | 10.76% | 5.72% | 65.26% |

information is a huge benefit to increase correction accuracy. However, due to some noises issues existing in the bigram co-occurrence model, some wrong bigram words also have high occurrences, e.g., popular error "dish washer". It results in a low performance on the correction of the real word errors.

The highest accuracy values of different correction types are marked with Bold in the tables. As shown in Table 1, the proposed THD spell correction v2 has the highest overall accuracy among the three algorithms. For the comparisons on most of the correction types ("Correct", "Brand", "NWE", "Break" and "Product"), THD spell correction v2 is also the best one. It proves that word embedding based dictionary, multi-candidate based ranking model and their integration can improve spell correction. The correction accuracy of real word errors is still lower than that of Lucene spell correction. But it is much better than that of THD spell correction v1. As for the correction types "Concatenate", "Phonetic", "Unit" and "Dim", THD spell correction v1 still gets the highest accuracy but has very minimal differences compared with THD spell correction v2.

## 7 CONCLUSION

In summary, the proposed algorithms of word embedding based multi-source (product catalog, query log and Wikipedia) dictionary construction and multi-candidate ranking model can improve spell correction much more than the other context based algorithms, e.g., just using bigram co-occurrence model, especially which performs much better than that only considering edit distance and phonetics matching. Adopting word embedding, multiple data sources can be integrated together without considering the weight mechanism to balance them. In addition, the merits of these data sources are also effectively exerted, e.g., using Wikipedia data to trim the invalid terms due to the low noises performance. With parallel structure based multi-candidate generation, all of the possible candidates can be gotten synchronously. Then the proposed word embedding based ranking model can be used to select the best one.

## 8 FUTURE WORK

Through all of the comparisons, real word error is regarded as the weakest point for the proposed THD spell correction algorithms. Most of the real word errors are blocked by spell checker component of spell correction, especially for some popular errors, e.g., "coffee mud" for correct one "coffee mug". Even searching with Google and Microsoft Bing, they cannot do anything with them as well. Observed from some search engines behaviors, when encountering these search queries, they would present some results only searching "coffee" or only searching "mud". Among them, the users should click the really correct ones they want. The clicked content can be used as a user feedback to help identify the error "coffee mud" and correct it into "coffee mug". The user behavior data can be feed into a deep neural network for the real word errors checking and correcting.

## REFERENCES


[1] James L Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980.

[2] Neha Gupta and Pratistha Mathur. Spell checking techniques in nlp: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(12), 2012.

[3] James F Allen. Natural language processing. 2003.

[4] Fei Liu, Fuliang Weng, Bingqing Wang, and Yang Liu. Insertion, deletion, or substitution?: normalizing text messages without pre-categorization nor supervision. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 71–76. Association for Computational Linguistics, 2011.

[5] Daniel Naber et al. *A rule-based style and grammar checker*. Citeseer, 2003.

[6] Tommi Pirinen and Krister Lindén. Creating and weighting hunspell dictionaries as finite-state automata. *Investigationes Linguisticae*, 21:1–16, 2010.

[7] Manu Konchady. *Building Search Applications: Lucene, LingPipe, and Gate*. Lulu. com, 2008.

[8] Genaro V Japos. Effectiveness of coaching interventions using grammarly software and plagiarism detection software in reducing grammatical errors and plagiarism of undergraduate researches. *JPAIR Institutional Research*, 1(1):97–109, 2013.

[9] Youssef Bassil and Mohammad Alwani. Post-editing error correction algorithm for speech recognition using bing spelling suggestion. *arXiv preprint arXiv:1203.5255*, 2012.

[10] Youssef Bassil and Mohammad Alwani. Ocr post-processing error correction algorithm using google online spelling suggestion. *arXiv preprint arXiv:1204.0191*, 2012.

[11] Eric Sven Ristad and Peter N Yianilos. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.

[12] Justin Zobel and Philip Dart. Phonetic string matching: Lessons from information retrieval. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172. ACM, 1996.

[13] Eric Brill and Robert C Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics, 2000.

[14] Mark D Kernighan, Kenneth W Church, and William A Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*, pages 205–210. Association for Computational Linguistics, 1990.

[15] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y Ng. Neural language correction with character-based attention. *arXiv preprint arXiv:1603.09727*, 2016.

[16] Shamil Chollampatt and Hwee Tou Ng. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[18] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[19] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[20] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[21] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 302–308, 2014.

[22] David Milne and Ian H Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518. ACM, 2008.
[23] Srikanth Machiraju and Ritesh Modi. Azure cognitive services. In *Developing Bots with Microsoft Bots Framework*, pages 233–260. Springer, 2018.

## ACKNOWLEDGMENTS

We thank our colleagues, Yuemeng Li, Mingzi Cao, Felipe Castrillon, Nagaraj Palanichamy for assistance with the software development and experiments, and Surya Kallumadi for comments that greatly improved this paper.