A Context Free Spell Correction Method using Supervised Machine Learning Algorithms

Ahmed Yunus
Software Engineer, AlgoMatrix, Sylhet
B.Sc. (Engg.), Shahjalal University of Science and Technology,
Department of Computer Science and Engineering,
Sylhet-3114, Bangladesh

Md Masum
Associate Professor,
Shahjalal University of Science and Technology,
Department of Computer Science and Engineering,
Sylhet-3114, Bangladesh

ABSTRACT
Spell correction is a modern day necessity for a system that lets a user extract the proper result while searching different things. Misspelled words are highly likely to occur while typing in queries to these systems and when users misspell query, the users may get inconclusive or false information returned by the system. Spell correction can be context-free or context-sensitive based on the usage. This paper traverses a spell correction method using supervised machine learning algorithms in which the wrong word does not rely on any context. Also this paper includes the comparison between different supervised machine learning algorithms for this case and additionally provides the best case and limitation of this spell correction method.

General Terms
Spell Correction, Machine Learning, Context Free, Dictionary

Keywords
Supervised Machine Learning, Tf-idf, Tokenization, KNeighbour Classifier, Multinomial Naive Bayes, Decision Tree Classifier, Random Forest Classifier, Logistic Regression, F1-score, Accuracy, Precision, stop words, QWERTY keyboard etc.

1. INTRODUCTION
A basic spell checking system starts with scanning a text, compares it with a known list of correct words and then pushes out the closely matched word against the input text. A spell checker can be either context-free or context-sensitive. A context-free spell correction system is a system where the wrong word does not rely on anything but itself. In this system, the wrong word does not care about the previous words, the word after it or neither about the total meaning of the sentence. This condition makes the system a little bit complex as the less data is available to use for algorithms to predict the wrong word outcome. This paper proposes a new method in context-free spell correction that uses supervised machine learning algorithms which follows the below structure:

- **Related Works:** This section provides the necessary background study on some works relevant to the spell correction system. Although much work has been done on context-sensitive spell correction, only few works have been conducted on context-free spell correction using supervised machine learning.
- **Data Collection and Processing:** This section insights briefly about the data processing system on wrong words that would be used to feed into machine learning algorithms.
- **Methodology:**This section provides a brief description on the architecture that would be used to predict the wrong word using supervised machine learning algorithms.
- **Experiments and Results:**Describes the experimental setup, along with some models used for comparative evaluations. Analysis of the results along with possible reasons are discussed.
- **Best Use Case and Limitation:**This section describes what could be the best use case for using this methodology in order to predict unknown wrong words and it’s limitations also.
- **Future Works and Conclusion:** The paper concludes with some recommendations and provides scope for future research on this field.

2. RELATED WORKS
In the history of computer science, algorithmic techniques for detecting and correcting spelling errors in text have a long and robust history **[1].** Different methodologies like edit distance **[2]**, rule-based techniques [3], n-grams [4], probabilistic techniques [5] etc. have been proposed. All of these are based on the idea of calculating the similarity between the misspelled word and the words contained in a dictionary. [7]

There has been research on developing algorithms that are capable of recognizing a misspelled word, even if the word itself is in the vocabulary, based on the context of the surrounding words. The most successful algorithm to date is Andrew Golding and Dan Roth's "Winnow-based spelling correction algorithm" published in 1999, which has accuracy of 96% in detection and correcting of context-sensitive spelling errors, in addition to ordinary non-word spelling errors. [6]

There is also research on correction of misspelling words which uses revised n-gram models by selecting the most promising candidates from a ranked list of correction candidates that is derived based on n-gram statistics and lexical resources. The proposed algorithm, in the following, is a language independent spell-checker that is based on an enhancement of the n-gram model. It is able to detect the correction suggestions by assigning weights to a list of possible correction candidates, based on n-gram statistics and lexical resources, in order to detect the non-word errors and to derive correction candidates. [7]

Also there is a neural network approach like the PENN system, introduced to correct misspelled words. The PENN System does not flag a word misspelled initially. Words that are misspelled and corrected enough times are characterized as possible errors in the system. These corrections are used to train a feed-forward neural network so that if the same error is remade, the network can flag the offending word as a possible error.**[8]**

There are numerous projects available online on spelling correction using neural nets. These are also context-sensitive. Kunal Bhashkar deployed his spell checker system using Tensorflow, Sequence to Sequence Model, Bi-directional LSTM etc. [10] This is just one of many systems available online.

So a very little has been done on context-free spell correction using supervised machine learning algorithms. But a notable work has been done using supervised learning on Arabic spelling correction. The system used Naive-Bayes classifier with python NLTK's implementation to find out which one is the most likely to be the correction for the incorrect word. [9]

This paper introduces a new method that will convert a wrong word to a differently customized wrong word and feed these data to traditional supervised machine learning algorithms to predict the wrong data. For achieving this task different algorithms like Multinomial Naive Bayes, DecisionTreeClassifier, RandomForestClassifier and LogisticRegression will be used and we will compare the final outcomes using different measurement scales.

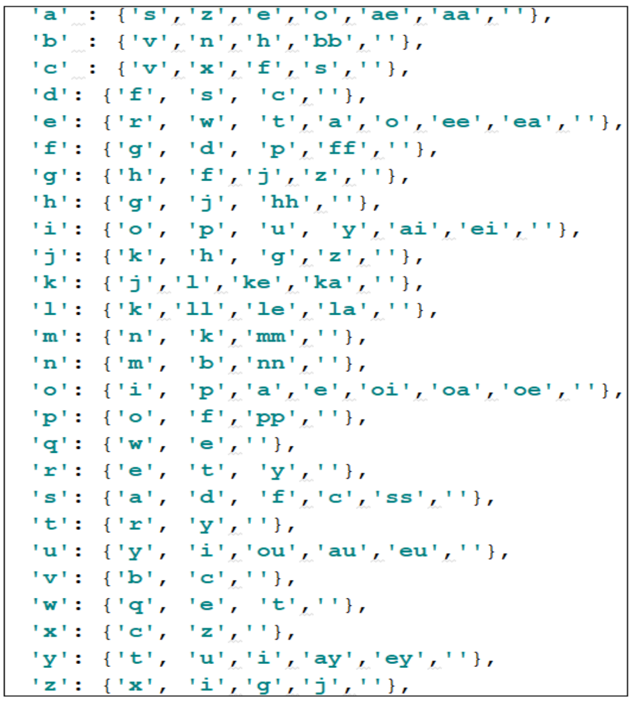
3. DATA COLLECTION AND PROCESSING:

The proposed methodology that will be discussed through this paper can be applied to any language based on the wrong and correct words. But we will focus on English language as a convenient solution. English comprises more than a million words, 171,476 words that are in current use and 20,000-30,000 words used by each individual person **[11].** There are available sources to collect English words but the problem relies with the wrong words list against a correct word. The wrong word here means that this is occurred due to spelling mistakes.

To start with, the first main task is to gather correct words from different sources. The first source is to take 1000 most common words in English [12]. Then we focused on most searched items on Amazon [13]. We also collected random English words for example brand names, product names and different vegetable, fruit names as our data. It is to be said, these selections are completely random and one can choose any English words they tend to use for the proposed model. Upon collecting this data, we have filtered and made a custom data set of 1000 English words that comprises different verbs, nouns, adjectives etc. The words that do not exceed the length of 3 are ignored in this dataset as those are not highly like to be misspelled.

The next task starts with fetching the wrong words that tend to be occurred by a user against a correct word. As we have selected random English words, it is quite difficult to get a list of wrong words against a single correct word. So we have taken another approach where we make wrong words programmatically. These methods are followed to prepare the wrong word dataset:

- **Swap letter:** Any two letters are being swapped in a word to make a wrong word of that correct word.
- **Add a letter:** Adds a new random letter in a correct word to convert it to a wrong word.
- **Keyboard character mapping:** As we are working with English language, to input any queries to a system, users use a QWERTY keyboard where the layout of the characters in the QWERTY keyboard leads a user to misspell a word. Based on this idea, a correct word can generate multiple wrong words. For example, while typing the letter 'e', a user may mistakenly type 'r' or 'w' as these are adjacent characters of 'e' in the QWERTY keyboard or the user might completely forget to type this letter. Figure 1 shows a keyboard mapping where there is possibility of changing the left side letter to right side values. These are selected mostly based on the adjacency of characters in the QWERTY keyboard. So to generate wrong words, the program will traverse the words from start to the length of the correct word, pick a single letter, and replace it with the right side values one by one and thus generate a wrong word. An example is that a word is 'come'. So 'm' is the third letter. 'm' letter can be mistyped with 'n', 'k', 'mm' or blanks. So from 'come' word's third letter 'm', 4 wrong words can be generated and they are: 'cone', 'coke', 'comme' and 'coe'. So this is how the wrong words list can be generated programmatically.



```
'a' : {'s','z','e','o','ae','aa',''},
'b' : {'v','n','h','bb',''},
'c' : {'v','x','f','s',''},
'd': {'f', 's', 'c',''},
'e': {'r', 'w', 't','a','o','ee','ea',''},
'f': {'g', 'd', 'p','ff',''},
'g': {'h', 'f','j','z',''},
'h': {'g', 'j', 'hh',''},
'i': {'o', 'p', 'u', 'y','ai','ei',''},
'j': {'k', 'h', 'g','z',''},
'k': {'j','l','ke','ka',''},
'l': {'k','ll','le','la',''},
'm': {'n', 'k','mm',''},
'n': {'m', 'b','nn',''},
'o': {'i', 'p','a','e','oi','oa','oe',''},
'p': {'o', 'f','pp',''},
'q': {'w', 'e',''},
'r': {'e', 't', 'y',''},
's': {'a', 'd', 'f','c','ss',''},
't': {'r', 'y',''},
'u': {'y', 'i','ou','au','eu',''},
'v': {'b', 'c',''},
'w': {'q', 'e', 't',''},
'x': {'c', 'z',''},
'y': {'t', 'u','i','ay','ey',''},
'z': {'x', 'i','g','j',''},
```

Figure 1: Keyboard mapping for possible wrong typo for a correct letter in a word.

Remove two letters: Sometimes a user can miss two letters in a word. So this can be a candidate for generating wrong words against a correct word.

It is to be noted, the first letter of a correct word is unchanged in every wrong letter that is being generated from a correct word. Following the rules mentioned above, we could generate more than 50 wrong words from one single correct word which is enough to feed into supervised machine learning algorithms to verify our methodology. In our case, from 1000 correct words, we have generated about 44000 wrong words as train and test data. For example a word **'grapefruit'**, we could have generated these wrong words:

Table 1: Wrong words of word 'grapefruit'

grpaefruit	grapefrut
grapefrfuit	grapefrupt
gtapefruit	grapefruot
geapefruit	grapefrueit
gyapefruit	grapefruyt

gapefruit	grapefruait
grpefruit	grapefruut
grzpefruit	gapefrui
grepefruit	gpefrui
grspefruit	gzpefrui
gropefruit	gepefrui
graepefruit	gspefrui
graapefruit	gopefrui
graefruit	gaepefrui
grappefruit	gaapefrui
graoefruit	gaefrui
grafefruit	gappefrui
graptfruit	gaoefrui
grapfruit	gafefrui
grapwfruit	gaptfrui
grapofruit	gapfrui
grapeefruit	gapwfrui
grapafruit	gapofrui
graprfruit	gapeefrui
grapeafruit	gapafrui
graperuit	gaprfrui
grapepruit	gapeafrui
grapegruit	gaperui
grapeffruit	gapeprui
grapedruit	gapegrui
grapeftuit	gapeffrui
grapefeuit	gapedrui
grapefyuit	gapeftui
grapefuit	gapefeui
grapefrit	gapefyui
grapefriit	gapefui
grapefrauit	gapefri
grapefrouit	gapefrii
grapefryit	gapefraui
grapefreuit	gapefroui
gapefryi	gapefreui

4. METHODOLOGY

Text data requires special preparation before starting to use it for predictive modeling. The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization). The scikit-learn library in Python offers easy-to-use tools to perform both tokenization and feature extraction text data. Word counts are a good starting point, but are very basic. One issue with simple counts is that some words like "the" will appear many times and their large counts will not be very meaningful in the encoded vectors. An alternative is to calculate word frequencies, and by far the most popular method is called Tf-idf. This is an acronym that stands for "Term Frequency – Inverse Document" Frequency which are the components of the resulting scores assigned to each word. Without going into the math, Tf-idf are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents **[14]**.

Supervised machine learning algorithms use text classification to predict the wrong words. As this is a supervised system, the train data must be labelled first. In our case, we have already selected 1000 correct words for this system. To be noted, correct words can be any dictionary of words selected by a human interpreter and this system will work according to that dictionary. In section 3, we have discussed how to generate wrong words from one single correct word and to conduct our research, we have generated about 44000 wrong words for 1000 correct words.

The next phase is to prepare the data to be fed for supervised machine learning algorithms. This is the most crucial part of this paper as the success of this proposed method heavily depends on the structure of input data rather than the algorithms. We have introduced three types of input as wrong words against a correct word that are to be fed into algorithms to predict an unknown wrong word. We have introduced three special terms here: Word Based Tokenization (WBT), Character Based Tokenization (CBT) and Advance Character Based Tokenization (ACBT).

In WBT, the whole wrong word is identified as the wrong word. For example: the wrong word of 'grapefruit' is 'gapefruit' and this 'gapefruit' will be directly identified as the wrong word. In our case, the Table-1 shows all the wrong words for the correct word 'grapefruit'. Supervised Algorithms here learn the mapping of input to output, and in this case wrong words to a correct word. So every wrong word is a feature and algorithms try to find patterns to make classification based on the inputs. The problem is that the inputs are all different and it causes a great number of features, leading to no match with one feature to another that makes the classification very hard.

In CBT, the whole word is divided into characters and all these characters are fed as a wrong word against a correct word. Table-2 depicts this idea.

Table-2: CBT transformation

Correct word	WBT	CBT
grapefruit	gapefruit	g a p e f r u i i t
grapefruit	grapefriit	g r a p e f r i i t
. . .	. . .	. . .

In the case of WBT, there is actually no pattern among wrong words as inputs. But in CBT, we can clearly find out that there is a pattern between ‘g a p e f r u i i t’ and ‘g r a p e f r i i t’. So what actually CBT does is, it breaks the characters and makes a sentence consisting of characters. So it helps the algorithms to find the patterns among the wrong words to make a classification. The wrong words must be vectored and for this reason Tf-idf has been chosen. But the above mentioned method also creates a problem in Tf-idf. As these are broken into characters, Tf-idf recognizes them as stop_words, which means these are commonly used words (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore [15]. As CBT breaks the word into individual letters, Tf-idf recognizes them as stop_words for which it removes entire data. So ‘g a p e f r u i i t’ cannot be fed into the algorithms. One way to overcome this is to append the same letter with itself so that the letters would become a word and Tf-idf would not recognize those words as stop_words. This appending would not affect the results as data dimensions will be the same across all train, test and new unknown wrong words. According to this rule, CBT will have a new shape that is shown in the next table.

Table-3: Final CBT transformation

Correct word	WBT	CBT
grapefruit	gapefruit	gg aa pp ee ff rr uu ii ii tt
grapefruit	grapefriit	gg rr aa pp ee ff rr ii ii tt
. . .	. . .	. . .

In this method, if words consist of English letters only, features are reduced dramatically which is only 26 and they are ‘aa’, ‘bb’, ‘cc’, ‘dd’, ‘ee’, ‘ff’, ‘gg’, ‘hh’, ‘ii’, ‘jj’, ‘kk’, ‘ll’, ‘mm’, ‘nn’, ‘oo’, ‘pp’, ‘qq’, ‘rr’, ‘ss’, ‘tt’, ‘uu’, ‘vv’, ‘ww’, ‘xx’, ‘yy’ and ‘zz’. As all wrong words are now made of these features, supervised machine learning will find a pattern among different wrong words for one class and according to that, the algorithms will make a prediction on new unknown wrong words to make a prediction from the dictionary.

To increase the number of features, ACBT has been introduced. ACBT adds up character positions to make this more unique among the wrong words of a correct word. Here the wrong word is ‘gapefruit’ where g’s position is 0, a’s position is 1 and so on (assuming this ‘gapefruit’ is an array of characters). So if we add these positions, this might help classifiers to train, test and predict data with better accuracy. Table-4 shows ACBT formation.

Table-4: ACBT transformation

Correct word	WBT	CBT	ACBT
grapefruit	gapefruit	gg aa pp ee ff rr uu ii ii tt	gg g0 aa a1 pp p2 ee e3 ff f4 rr r5 uu u6 ii i7 ii i8 tt t9
grapefruit	grapefriit	gg rr aa pp ee ff rr ii ii tt	gg g0 rr r1 aa a2 pp p3 ee e4 ff f5 rr r6 ii i7 ii i8 tt t9
. . .	. . .	. . .	. . .

5. EXPERIMENTS AND RESULTS

So we have prepared three kinds of inputs to test if supervised machine learning algorithms can predict unknown wrong words that happen due to spelling mistakes. We have selected multiple algorithms which are given below:

KNeighborsClassifier

- Multinomial Naive Bayes
- DecisionTreeClassifier
- RandomForestClassifier
- LogisticRegression

We have selected 1000 correct words that are defined by human interpreters and generated 44277 wrong words using different methods stated in Section 3. We splitted data into 30% as our test data and 70% as our train data. We fed our data into algorithms and the results we have found are mentioned below in the tables.

Table-5: Feature Information

Method Name	Total Features
WBT	32837
CBT	26
ACBT	604

Table-6: Mean and Standard Deviation (STD) Information

Model \ Method	WBT	CBT	ACBT
KNeighbors Classifier	Mean : 0.043138 STD : 0.004887	Mean : 0.763191 STD : 0.006133	Mean : 0.892916 STD : 0.007697
Multinomial Naive Bayes	Mean : 0.141836 STD : 0.007306	Mean : 0.507213 STD : 0.009165	Mean : 0.667203 STD : 0.006328

DecisionTreeClassifier	Mean : 0.328743 STD : 0.011555	Mean : 0.682109 STD : 0.010430	Mean : 0.808221 STD : 0.007296
RandomForestClassifier	Mean : 0.204394 STD : 0.007309	Mean : 0.719149 STD : 0.007349	Mean : 0.826064 STD : 0.004893
LogisticRegression	Mean : 0.185135 STD : 0.007029	Mean : 0.657886 STD : 0.007435	Mean : 0.829931 STD : 0.005252

Table-7: Accuracy, F1-Score,

Model \ Method	WBT	CBT	ACBT
KNeighbors Classifier	Accuracy: 12.78 % F1-Score: 0.07 Support: 8856	Accuracy: 77.30 % F1-Score: 0.64 Support: 8856	Accuracy: 90.14 % F1-Score: 0.80 Support: 8856
Multinomial Naive Bayes	Accuracy: 15.50 % F1-Score: 0.06 Support: 8856	Accuracy: 51.88 % F1-Score: 0.26 Support: 8856	Accuracy: 67.56 % F1-Score: 0.37 Support: 8856
DecisionTreeClassifier	Accuracy: 19.73 % F1-Score: 0.17 Support: 8845	Accuracy: 59.26 % F1-Score: 0.56 Support: 8856	Accuracy: 82.46 % F1-Score: 0.72 Support: 8856
RandomForestClassifier	Accuracy: 20.71 % F1-Score: 0.17 Support: 8854	Accuracy: 72.73 % F1-Score: 0.59 Support: 8856	Accuracy: 83.51% F1-Score: 0.72 Support: 8856
LogisticRegression	Accuracy: 19.58 % F1-Score: 0.10 Support: 8854	Accuracy: 66.82 % F1-Score: 0.40 Support: 8856	Accuracy: 84.74% F1-Score: 0.67 Support: 8856

We have used a completely unknown wrong word to predict the correct word. The unknown word is ‘gapeefruutttt’. The result is stated below:

Table-8: Prediction on unknown word ‘gapeefruutttt’

Model \ Method	WBT	CBT	ACBT
KNeighbors Classifier	broth	grapefruit	grapefruit
Multinomial Naive Bayes	energizer-alkaline-batteries	grapefruit	grapefruit
DecisionTreeClassifier	broth	figure	grapefruit
RandomForestClassifier	fund	grapefruit	agreement, 20% probability contribute, 20% probability grapefruit, 20% probability
LogisticRegression	classification	grapefruit	grapefruit

From Table-6, Table-7 and Table-8, we can clearly understand how CBT and ACBT outperforms the WBT method. It is expected as the features extracted via WBT method does not help the classifiers to predict unknown words. Also, it can be clearly understood that adding position feature in ACBT increases Accuracy, F1-Score and also predicts much better than CBT and WBT.

6. BEST USE CASE AND LIMITATION

The mentioned method works greatly if the dictionary is custom defined by a human interpreter. For example if this system is deployed behind an ecommerce system, then there would be words consisting of different brand names, product names etc. which will not have the same pattern of characters among themselves. If the words have the same patterns e.g. ‘research’ and ‘search’ in the dictionary, this system might not predict the wrong word to the correct word properly. This would work fine in a system where the letters are from English alphabets but the words might be different from English language dictionaries for example in a medical or pharmaceutical company.

7. FUTURE WORKS AND CONCLUSION

There are many scopes to improve the research. The research was conducted on 1000 correct words only which can be extended using all the English language's vocabulary to find out the accuracy. Also this method has the opportunity to be used other than English and that is a huge area of research. Also there is an opportunity to use Artificial Neural Network for this method and look at how that system performs. And lastly we hope to see the implementation of this method in various areas like in searching systems of websites in the future.

8. REFERENCES

[1] K. Kukich, "Techniques for automatically correcting words in text," ACM Computing Surveys, 24(4), 377–439, 1992.

[2] R. A. Wagner and M. J. Fisher, "The string to string correction problem," Journal of Assoc. Comp. Mach., 21(1):168-173, 1974

[3] E. J. Yannakoudakis and D. Fawthrop, "An intelligent spelling error corrector," Information Processing and Management, 19:1, 101-108,1983.

[4] Jin-ming Zhan, Xiaolong Mou, Shuqing Li, Ditang Fang, "A Language Model in a Large-Vocabulary Speech Recognition System," in Proc. Of Int. Conf. ICSLP98, Sydney, Australia, 1998.

[5] K. Church and W. A. Gale, "Probability scoring for spelling correction,"Statistics and Computing, Vol. 1, No. 1, pp. 93–103, 1991.

[6] Golding, Andrew R.; Roth, Dan (1999). "Journal Article". *Machine Learning*. SpringerLink. 34: 107–130. doi:10.1023/A:1007545901558

[7] Revised N-Gram based Automatic Spelling Correction Tool to Improve Retrieval Effectiveness,December 2009, DOI: 10.17562/PB-40-6

[8] Personalized Spell Checking using Neural Networks by Tyler Garaas, Mei Xiao, and Marc Pomplun

[9] Arabic Spelling Correction using Supervised Learning, September 2014, DOI: 10.3115/v1/W14-3615

[10] https://medium.com/@BhashkarKunal/spelling-correction-using-deep-learning-how-bi-directional-lstm-with-attention-flow-works-in-366fabcc7a2f

[11] https://englishlive.ef.com/blog/language-lab/many-words-english-language/

[12] https://www.ef.com/wwen/english-resources/english-vocabulary/top-1000-words/

[13] https://ahrefs.com/blog/top-amazon-searches/

[14] https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

[15] https://www.geeksforgeeks.org/removing-stop-words-nltk-python/