



Project 1 reviews

Print Options:

- ☒ Include Questions & Answers ☒ Include Comments ☒ Include All Reviews ☒ Include File Info

[Print](#)

HAONAN LI'S PEERMARK REVIEW OF ANONYMOUS'S PAPER (100% COMPLETED)

ASSIGNED QUESTIONS

1. Briefly summarise what the author has done

The author applies two approximate matching methods to a spelling correction task.

2. Indicate what you think the author has done well, and why

The author use pictures to assist exhibition of the experiments.

3. Indicate what you think could have been improved, and why

There are several points you could have been improved.

1. You should write it carefully and seriously. The report should be well formatted, for example: center the title, use more high quality picture rather than screenshot, eliminate misspelled words. These are the most basic requests of an academic paper.
2. Reference are not enough and inaccurate. You should cite the paper that publish the dataset you used and the source of the methods you used.
3. Systematic evaluation is needed. Using a table to show your results seems more professional. And you can use precision and recall to evaluate it or just use accuracy if you like, but not "correct rate" .
4. Do not put your source code in the report, do not put the screenshot of dataset in the report. Just describe it by a few words.
5. Focus more on the performance of the methods. Use more words to demonstrate it. You can also do more experiments. For example, apply n equals 3 and 4 for n-gram algorithm and compare the performance among them.
6. Conclusions of your work is essential.

COMMENTS LIST

No comments added

SUBMITTED FILE INFO

file name	comp90049_p1.pdf
file size	469.83K

"COMP90049 PROJECT1" BY ANONYMOUS

COMP90049 Project 1 Report

1.Introduction

When the user enter English words, an error often occurs and we need to correct it. Research on Spelling correction has a long history. Edit distance. Initially proposed by Damerau and Levenshtein, has been widely used in generic spelling correction. More recent work on offline spelling correction tends to dovus on search engine queries[1].

2.Dataset

There are a misspell list, one dictionary list which is a list of about 400K tokens from the English language, and a correct list ,which help us to judge whether the spelling are corrected correctly. We need to apply two spelling correct method and evaluate the behaviour of the method.

I am going to use two methods, Levenshtein Distance and n-Gram Distance to correct the misspelled words, and evaluate the behaviour of each methods.

3.Levenshtein Distance

Levenshtein Distance refers to the minimum number of edit operations required to transform one word to another word. If the distance between them is larger, it means the more they are different. Permissive editing operations include replacing one character with another, inserting one character, and deleting one character. This concept was proposed by Russian scientist Vladimir Levenshtein is 1965. It can be used for DNA analysis, spell detection, plagiarism identification, and so on. We can find that the key points are that there are only three operations. Insert, delete, replace. So how could we use Levershtein distance to correct misspelled words? We can use dictionary to predict the word . Just calculate the Levershtein edit distance with all the words in the misspelled.txt with the word in the dictionary.txt. The world with the smallest Levenshtein distance is likely to be the correct world, which is the result of the correction.

```
public class LevenshteinDistance {

    public double distance(String w1,String w2){
        double[][] m = new double[w1.length()+1][w2.length()+1];
        for(int i=0;i<m.length;i++){
            m[i][0]=i;
        }
        for(int i=0;i<m[0].length;i++){
            m[0][i]=i;
        }
        for(int i=1;i<m.length;i++){
            for(int j=1;j<m[0].length;j++){
                m[i][j] = min(m[i][j-1]+1,m[i-1][j]+1,m[i-1][j-1]+cost(w1.charAt(i-1),w2.charAt(j-1)));
            }
        }
        return m[w1.length()][w2.length()];
    }

    protected double cost(char c1,char c2) {
        return c1==c2?0:1;
    }

    protected double min(double i, double j, double k) {
        double t = i<j?i:j;
        return t<k?t:k;
    }

}
```

According to the above code, we can calculate the Levenshtein distance between the words in misspelled text and the word in the dictionary text. The main part of this algorithm are two nested for loops which can calculate the value of $m[j][i]$ according to $m[i][j-1]$, $m[i-1][j]$ and $m[i-1][j-1]$.

It is easily to find that the edit distance is consistent with the number of letters in the string. The longer the string is, the larger the Levenshtein can be. So we should find a way to normalize.

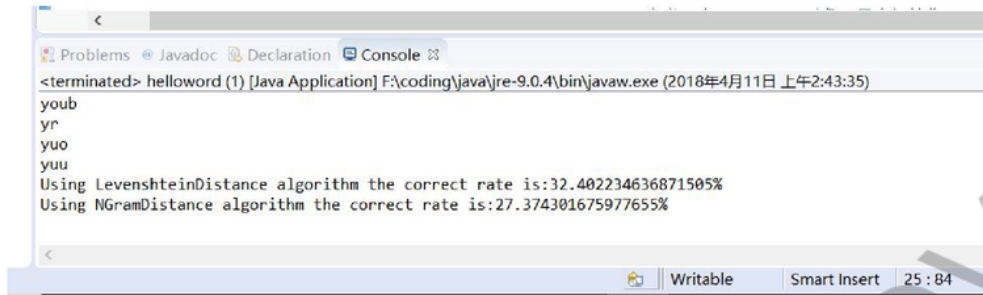
```
double d1 = new LevenshteinDistance().distance(s1, s2);
Lmax = s1.length() > s2.length() ? s1.length() : s2.length();
similarity1 = 1 - d1 / Lmax;
// MGcom
```

As the above code show, we can put d and Lmax(the letters of the longer string) in numerator and denominator. When two long string has a small d, it means they are not similar.



The Screenshot above show the predicted string which calculated from Levenshtein distance algorithm. We can find addn is indeed more similar to adn than and. It just need insert a "d" from addn to adn.

The accuracy of this method is as shown in the picture, which is 32.4% approximately.



```
<terminated> helloworld (1) [Java Application] F:\coding\java\jre-9.0.4\bin\javaw.exe (2018年4月11日 上午2:43:35)
youb
yr
yuo
yuu
Using LevenshteinDistance algorithm the correct rate is:32.402234636871505%
Using NGramDistance algorithm the correct rate is:27.374301675977655%
```

But in our daily life, spelling misspelled are always caused by incorrect keyboard touch. For example, the letter A is next to Q, S and W. So when the user want to input A, the probability to touch Q, S and E in more greater than the probability to mistype other letters. Because P is too far apart from A, user input P is basically a real idea. Another reason is spelling mistake, for example, pronunciation of letter e is similar to the pronunciation of a in some words. It is easy to spell letter to latter.

But after I observe the dataset carefully, it seems that the misspelled are random, not similar to the real life misspelling. So I did not use Keyboard optimization algorithm is this dataset.

N-GRAM Distance

O-Gram is a very important concept in natural language processing. People can use N-gram to predict or evaluate whether a sentence is reasonable. On the other hand ,another role of N-Gram is to evaluate the degree of difference between two strings. We can not only define edit distance between two strings, but also we can define the N-Gram distance between them.

N-gram-based models are Markov models over sequences of tuples that are generated monotonically. Tuples are minimal translation units (MTUs) composed of source and target cepts.² The N-gram-based model has the following drawbacks: (i) only pre-calculated orderings are hypothesized during decoding, (ii) it cannot memorize and use lexical reordering triggers, (iii) it cannot perform long distance reorderings, and (iv) using tuples presents a more difficult search problem than in phrase-based SMT[2].

Suppose there is a string s , then the N-gram of string represents the substring of s of which length are N . If there are two string ,and find their N-Gram, you can define the N-Gram distance between two strings from the perspective of the number of their total substrings.


```

51 char[] t_j = new char[n]; // jth n-gram of t
52
53 // 初始化第一横排的编辑距离
54 for (i = 0; i<=s1; i++) {
55     p[i] = i;
56 }
57
58 for (j = 1; j<=t1; j++) { // 开始处理第二个横排, ...到t1最后一个横排
59     //construct t_j n-gram. 构建n-gram
60     if (j < n) { // 补充前缀
61         for (int ti=0; ti<n-j; ti++) {
62             t_j[ti]=0; //add prefix
63         }
64         for (int ti=n-j; ti<n; ti++) {
65             t_j[ti]=w2.charAt(ti-(n-j));
66         }
67     }
68     else { // 直接取n-gram
69         t_j = w2.substring(j-n, j).toCharArray();
70     }
71     d[0] = j;
72     for (i=1; i<=s1; i++) {
73         cost = 0;
74         int tn=n;
75         //compare sa to t_j. 计算f(i,j)
76         for (int ni=0; ni<n; ni++) {
77             if (sa[i-1+ni] != t_j[ni]) {
78                 cost++;
79             }
80             else if (sa[i-1+ni] == 0) { //discount matches on prefix
81                 tn--;
82             }
83         }
84         float ec = (float) cost/tn;
85         // minimum of cell to the left+1, to the top+1, diagonally left and up +cost
86         d[i] = Math.min(Math.min(d[i-1]+1, p[i]+1), p[i-1]+ec);
87     }
88     // copy current distance counts to 'previous row' distance counts

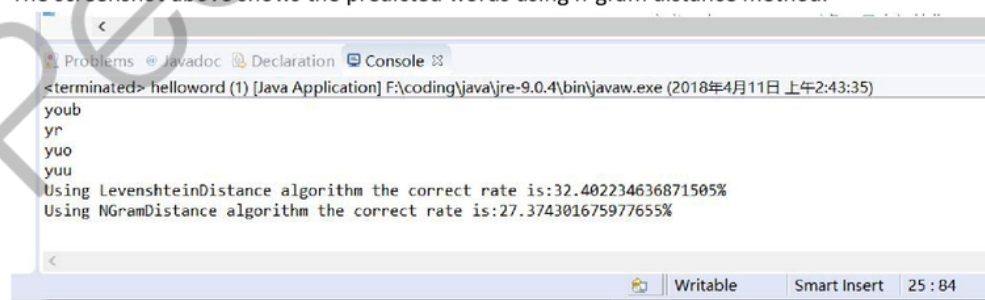
```

According to the above code, we can calculate the n-gram distance between the words in misspelled text and the word in the dictionary text. We let $n=2$ in this algorithm.

The number of "n" of selection dependent letters is mainly related to the calculation conditional probability. In theory, as long as there is a large enough material, n is bigger and better, because the information considered is more. The conditional probability is very good, and just statistics on the number of occurrences of each tuple. However, the actual situation is limited training corpus, it is easy to produce sparse data, and does not meet the law of large numbers. On the other hand, when n is too large, the probability of one long sub-string equals to the sub-string of another word is too small. When the $n=1$, the n-gram distance of two different words which made up of the same letters will be nearly 0. It make no sense.



The screenshot above shows the predicted words using n-gram distance method.



The accuracy of this method is as shown in the picture, which is 27.34% approximately. The accuracy of n-gram is in some kind of lower than LevenshteinDistance algorithm (32.4%) using this kind of dataset. Using LevenshteinDistance algorithm need 190s approximately while using n-gram algorithm need 240s. But we can not say which method is better. Because it just using this limited dataset. There two algorithms still have a lot of optimization space.

References

1. Duan, H., & Hsu, B. J. P. (2011, March). Online spelling correction for query completion. In Proceedings of the 20th international conference on World wide web (pp. 117-126). ACM.
2. Durrani, N., Schmid, H., Fraser, A., Koehn, P., & Schütze, H. (2015). The operation sequence model—combining N-gram-based and phrase-based statistical machine translation. *Computational Linguistics*, 41(2), 185-214.

