Comp90049 Knowledge Technology
Project1: Approximate Matching

Name: Ao Li ID: 798657

1. Introduction

In this report, some different approximate matching approaches are mentioned to predict a Latin name for all the Persian names on the given data set, which is so-called back-transliteration. It's divided into 4 parts.

Firstly, introduce some basic methods, but most of the basic methods' didn't work well. Secondly, for the sake of improving the performance, some existed matrixes and some matrixes that are trained from the data set as a replacement score matrix have been used to improve the classical algorithms. Thirdly, critically comparing and analyze each approaches' results with the given data set by calculating the precision and recall. Lastly, conclude the entire work and discuss possible future works.

Two main data sets used in this project:

1) "train.txt"

   It is a large list of names in the Persian script, with their equivalent in the Latin script. It's used for comparing the predicted Latin name(s) with the actually intended name and for training the replacement score matrix.

2) "names.txt"

   It is a list of Latin names. Like a dictionary for the algorithm to find which Latin name(s) is(are) the predicted name(s) for each Persian name.

2. Basic Methods

In this section, I will give a short overview for the entire basic approximate matching algorithm used in the project. Besides, giving some critical analyzing for these methods and provide some optimized direction for improving the algorithms' performance.

2.1. Edit Distance

It is a method for quantifying how different two strings are to one another by calculating the minimum number of operations required to transform one string into the other. It can be classified into two category---Global Edit Distance (GED) and Local Edit Distance (LED). LED is like GED, but for searching the best substring match between two strings, particularly suitable when comparing two strings of different lengths.

2.2. N-Gram Distance

N-Gram comes from a concept for a contiguous sequence of n items from a given string. Besides, use formula calculating each two words $N-Gram\ Distance$[2] and the most similar pairs of names will have the smallest N-Gram Distance..

2.3. Soundex

$Soundex$[3] is a phonetic algorithm for indexing names by sound with a translation table. Since the translation table is classified by sound, not suitable for the name translation. Besides, the code-space is small so that many different strings will map into same code. So, it's not implemented in this report, but the concept of "The first letter in a word is very important" in Soundex is used to improve the method in the next section.

2.4. Critical Analysis

Although all the algorithms mention above are popular and classical for approximate matching. But there are still some direction for improve:

For example：

1) To *Edit Distance*, not all the replacements are equivalent, so the cost of different pairs of letters' Match, Replacement should be under considering. Besides, Insertion and Deletion cost among letters is another way for improving. Also, Replacement between letter and space is remarkably.

2) To *Soundex*, try to find some relationship to the first letter between Persian name and Latin name and make the relationship become a translating table.

3. Improved Methods

There are many ways for improving the name translation correctness. In this section, it introduces some of the different methods that based on one or more basic method(s) above.

3.1 Use BLOSUM62 matrix

$BLOSUM$[4] (Blocks Substitution Matrix) is an existed replacement matrix used for sequence alignment of proteins in bioinformatics and BLOSUM62 is one kind of BLOSUM matrix with the midrange of the proteins. However, the letters are not ordered lack some letters in an original BLOSUM62 matrix. In order to easy calculating, the BLOSUM62 matrix is modified as Figure.1 (order the letters and set all value ≥ 0).

```
   A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
A [ 8, 4, 4, 2, 3, 2, 4, 2, 3, 4, 3, 3, 3, 3, 4, 3, 3, 3, 5, 3, 4, 2, 1, 4, 2, 4 ],
B [ 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ],
C [ 4, 4, 13, 1, 0, 2, 1, 1, 3, 4, 1, 3, 3, 1, 4, 1, 1, 1, 3, 3, 4, 3, 2, 4, 2, 4 ],
D [ 2, 4, 1, 10, 6, 1, 3, 3, 1, 4, 3, 0, 1, 5, 4, 3, 4, 2, 4, 5, 4, 1, 0, 4, 1, 4 ],
E [ 3, 4, 0, 6, 9, 1, 2, 4, 1, 4, 5, 1, 2, 4, 4, 3, 6, 4, 4, 4, 4, 1, 1, 4, 2, 4 ],
F [ 2, 4, 2, 1, 1, 10, 1, 3, 4, 4, 1, 4, 4, 1, 4, 0, 1, 1, 2, 2, 4, 3, 5, 4, 7, 4 ],
G [ 4, 4, 1, 3, 2, 1, 10, 2, 0, 4, 2, 0, 1, 2, 4, 2, 2, 2, 4, 5, 4, 4, 2, 4, 1, 4 ],
H [ 2, 4, 1, 5, 4, 3, 2, 12, 1, 4, 3, 1, 2, 5, 4, 2, 4, 4, 3, 4, 4, 2, 2, 4, 6, 4 ],
I [ 3, 4, 3, 1, 1, 4, 0, 1, 8, 4, 1, 6, 5, 1, 4, 1, 1, 1, 2, 2, 4, 5, 1, 4, 3, 4 ],
J [ 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ],
K [ 3, 4, 1, 3, 5, 1, 2, 3, 1, 4, 9, 2, 3, 4, 4, 3, 5, 6, 4, 4, 4, 1, 1, 4, 2, 4 ],
L [ 3, 4, 3, 0, 1, 4, 0, 1, 6, 4, 2, 8, 6, 1, 4, 1, 2, 2, 2, 2, 4, 7, 2, 4, 3, 4 ],
M [ 3, 4, 3, 1, 2, 4, 1, 2, 5, 4, 3, 6, 9, 2, 4, 2, 4, 3, 3, 3, 4, 2, 3, 4, 3, 4 ],
N [ 2, 4, 1, 5, 4, 1, 4, 3, 1, 4, 4, 1, 2, 10, 4, 2, 4, 4, 5, 4, 4, 1, 0, 4, 2, 4 ],
O [ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ],
P [ 3, 4, 1, 3, 3, 0, 2, 2, 1, 4, 3, 1, 2, 3, 4, 11, 3, 2, 3, 5, 4, 2, 0, 4, 1, 4 ],
Q [ 3, 4, 1, 4, 6, 1, 2, 4, 1, 4, 5, 2, 4, 4, 4, 3, 9, 5, 4, 4, 4, 2, 2, 4, 3, 4 ],
R [ 3, 4, 1, 0, 4, 1, 2, 4, 1, 4, 6, 2, 3, 4, 4, 2, 5, 9, 3, 3, 4, 1, 1, 4, 2, 4 ],
S [ 5, 4, 3, 4, 4, 2, 4, 3, 2, 4, 4, 2, 3, 5, 4, 3, 4, 3, 8, 5, 4, 2, 1, 4, 2, 4 ],
T [ 3, 4, 3, 5, 4, 2, 5, 4, 2, 4, 4, 2, 3, 4, 4, 5, 4, 3, 5, 8, 4, 2, 1, 4, 2, 4 ],
U [ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4, 4, 4, 4 ],
V [ 4, 4, 3, 1, 2, 3, 1, 1, 7, 4, 2, 5, 5, 1, 4, 2, 2, 1, 2, 2, 4, 8, 1, 4, 3, 4 ],
W [ 1, 4, 2, 0, 1, 5, 2, 2, 1, 4, 1, 2, 3, 0, 4, 0, 2, 1, 1, 1, 4, 1, 15, 4, 6, 4 ],
X [ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9, 4, 4 ],
Y [ 2, 4, 2, 1, 2, 7, 1, 6, 3, 4, 2, 3, 3, 2, 4, 1, 3, 2, 2, 2, 4, 3, 6, 4, 11, 4 ],
Z [ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 9 ]
```

Figure 1 Modified BLOSUM62 Matrix

However, some of the downsides for BLOSUM62 matrix are

1) It's a kind of substitution matrix trained from the relationship of proteins, but from Persian to Latin.
2) Proteins just use 20 Latin letters not 26. Thus, lack of some of the relationship between letters and hard to decide the value without other technique supports.

3.2 Use matrix trained from data set

An ideal thought is that if we can find one matrix that is trained from the relationship between Persian and Latin names. In the report, three matrixes are trained by using the same method, but respectively chosen various numbers of random lines from data set. The training method is as below:

1) Randomly choose 100\500\1000 lines 10 times from "train.txt" as shown in Figure.2. But for each line the Persian and Latin name must be in the same length depicted in Figure.3. Because it's difficult to calculate the score when two strings are various length, need to consider which letter replaced by the blankspace.

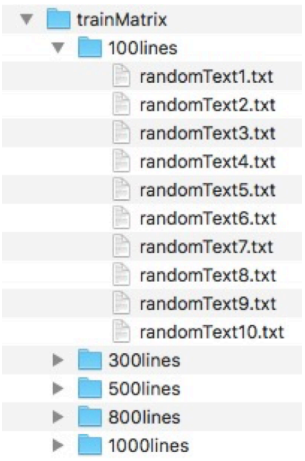


Figure 2. Three different matrix trained files

Figure 3. Example random lines in the randomText.txt files

Note: Using different numbers of lines for training the matrix is for critical analysis the performance in next part.

2) For each capacity, using the total numbers of lines to train a replacement matrix shown in Figure.4. For example, if Persian name "abc" → Latin names "abs", that will be [a][a] -> plus 1, [b][b] -> plus 1, [c][s] -> plus 1.

3) Due to the values are uneven in an original trained matrix, use mathematical formula (as Formula 1) turning all the values in [1,26]. An example matrix after modified is shown in Figure.5.

$$N_{x,y} = \frac{N_{max} - N_{min}}{O_{max} - O_{min}} \times (O_{x,y} - O_{min}) + N_{min} \quad \forall x, y \in 1, N$$

Formula.1 Even Values in Trained Matrix

```
    A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z
A [763,   4,  3,  3, 31,  0,  2,  3, 43,  0,  1,  3,  1,  4, 82,  0,  0,  7, 47,  2, 65,  0,  2,  0, 16,  1 ],
B [  3, 147,  0,  0,  3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0 ],
C [  0,   0,  2,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
D [  0,   0,  0,226,  4,  0,  1,  0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0 ],
E [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
F [  0,   0,  0,  0,  0, 68,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0 ],
G [  3,   0,  0,  0,  3,  0,142,  0,  2,  0,  0,  4,  0,  2,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
H [  3,   0,  0,  0, 22,  0,  0, 70,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
I [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
J [  1,   0,  0,  0,  5,  0,  7,  0,  0, 44,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
K [  2,   0, 79,  0,  8,  0,  0,  0,  3,  0,173,  0,  0,  1,  2,  0,  8,  1,  0,  0,  3,  0,  2,  3,  0,  0 ],
L [  5,   0,  0,  0, 12,  0,  0,  0,  1,  0,  0,278,  0,  0,  1,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0 ],
M [  3,   0,  0,  0,  3,  0,  0,  0,  2,  0,  0,  0,219,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
N [  2,   0,  1,  1, 14,  0,  8,  0,  2,  0,  0,  1,  0,466,  2,  0,  0,  1,  1,  4,  0,  0,  0,  0,  0,  0 ],
O [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
P [  1,   0,  0,  0,  4,  0,  0,  0,  1,  0,  0,  1,  0,  0,  0, 84,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
Q [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
R [  4,   1,  0,  0,  8,  0,  1,  0, 10,  0,  1,  0,  0,  3,  2,  0,  0,365,  1,  0,  0,  0,  0,  0,  0,  1 ],
S [  2,   1,  9,  0,  6,  0,  1,  0,  1,  0,  5,  5,  3,  6,  2,  4,  1,  0,250, 26,  1,  0,  1,  4,  0,  0 ],
T [  5,   0,  0,  0, 10,  0,  0,  4,  3,  3,  0,  0,  0,  0,  5,  0,  0,  4,  2,202,  2,  0,  0,  0,  0,  1 ],
U [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
V [  6,   2,  0,  1, 16,  2,  0,  0,  2,  0,  0,  6,  4,  2,310,  1,  2,  8,  3,  2,134, 49, 60,  2,  0,  0 ],
W [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
X [  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
Y [ 22,   1,  3,  4, 80,  1,  3,  0,447,  0,  4,  9,  1, 21,  2,  2,  0, 11,  5, 11, 18,  3,  1,  0, 98,  4 ],
Z [  4,   1,  0,  0,  2,  0,  0,  0,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0, 29,  0,  0,  0,  0,  0,  0, 44 ]
```

Figure 4. Original Trained Matrix

```
    A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z
A [ 26,  1,  1,  1,  2,  1,  1,  1,  2,  1,  1,  1,  1,  1,  4,  1,  1,  1,  3,  1,  3,  1,  1,  1,  2,  1 ],
B [  2, 26,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
C [  1,  1, 26,  1,  1,  1,  1, 14,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
D [  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
E [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
F [  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
G [  2,  1,  1,  1,  2,  1, 26,  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
H [  2,  1,  1,  1,  9,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
I [  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
J [  2,  1,  1,  1,  4,  1,  5,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
K [  1,  1, 12,  1,  2,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
L [  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
M [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
N [  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
O [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
P [  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
Q [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1,  1 ],
R [  1,  1,  1,  1,  2,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1,  1,  1 ],
S [  1,  1,  2,  1,  2,  1,  1,  1,  1,  1,  2,  2,  1,  2,  1,  1,  1,  1, 26,  4,  1,  1,  1,  1,  1,  1 ],
T [  2,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1 ],
U [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1 ],
V [  1,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  2,  1,  1, 12,  5,  6,  1,  1,  1 ],
W [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1 ],
X [  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, 26,  1,  1,  1,  1,  1,  1, 26,  1,  1 ],
Y [  2,  1,  1,  1,  5,  1,  1,  1, 26,  1,  1,  2,  1,  2,  1,  1,  1, 26,  2,  2,  1,  1,  1,  1,  6,  1 ],
Z [  3,  2,  1,  1,  2,  1,  1,  1,  1,  3,  1,  1,  1,  1,  1,  1,  1,  1, 17,  1,  1,  1,  1,  1,  1, 26 ]
```

Figure 5 Modified Trained Matrix

4) Set the trained matrix into the GED algorithm. Part of important codes is shown in Figure.6. But there is a small tricky here,

```
if pName[ i ] == lName[ j ]:
    distanceG[ i ][ j ] = max(
    distanceG[ i-1 ][ j-1 ] + trainedMatrix[ y ][ x ],
    distanceG[ i-1 ][ j ] - trainedMatrix[ y ][ x ],
    distanceG[ i ][ j-1 ] - trainedMatrix[ y ][ x ]
)
else:
    distanceG[ i ][ j ] = max(
    distanceG[ i-1 ][ j-1 ] + trainedMatrix[ y ][ x ] - trainedMatrix[ y ][ y ]
    distanceG[ i-1 ][ j ] - trainedMatrix[ y ][ x ],
    distanceG[ i ][ j-1 ] - trainedMatrix[ y ][ x ]
)
```

(1)

```
if pName[ i ] == lName[ j ]:
    distanceG[ i ][ j ] = max(
    distanceG[ i-1 ][ j-1 ] + trainedMatrix[ y ][ x ],
    distanceG[ i-1 ][ j ] - letter_empty_average[x],
    distanceG[ i ][ j-1 ] - letter_empty_average[y]
)
else:
    distanceG[ i ][ j ] = max(
    distanceG[ i-1 ][ j-1 ] + trainedMatrix[ y ][ x ] - trainedMatrix[ y ][ y ],
    distanceG[ i-1 ][ j ] - letter_empty_average[x],
    distanceG[ i ][ j-1 ] - letter_empty_average[y]
```

(2)

Figure 6 Tricky in implemented

It's hard to find the score for translating between letter and space, the algorithm firstly guesses as in Figure6 (1), if substitution score for two different letters is relatively high, the cost for insertion and deletion will higher than replacement. However, the performance is not really well so that modify the code as Figure6 (2), using an average cost for insertion and deletion.

4. Evaluation

After evaluate all the implement algorithms, some problems and improved ideas come up.

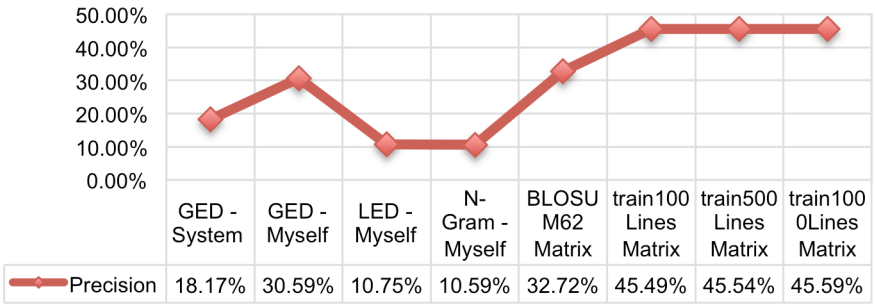
4.1. Evaluation between Basic and Trained Matrix Methods

Table.1 and Figure.7 show the number of correct prediction and precision of different methods. It's obviously find that the last three trained matrix approaches get better performance among those methods. Since the trained matrixes algorithm is based on the self-write GED method, so the precision is from approximately 30% up to 45%. But the differ between three trained matrix is small. Some of the reasons are as below:

1) The matrixes are trained and test on the same data set and it's not big enough, so some of the lines that randomly chosen are overlap.
2) To some extent, even the matrix in 3.2 step3) can reduce the difference between matrix.
3) The major problem for the trained matrix method mentioned in this report is that they don't train a substitution score between letters and space (insertion/deletion cost) so that limits to figure out translating between two various length strings situation.

Table 1 Compare Different Methods

Method Name	Correct Prediction	Total Prediction
Global System	2442	13437
Global Myself	4111	13437
Local Myself	1445	13437
N-Gram	1423	13437
Global Myself + BLOSUM62	4397	13437
Global Myself + train100Lines	6113	13437
Global Myself + train500Lines	6119	13437
Global Myself + train1000Lines	6126	13437



	GED - System	GED - Myself	LED - Myself	N-Gram - Myself	BLOSUM62 Matrix	train100 Lines Matrix	train500 Lines Matrix	train1000Lines Matrix
Precision	18.17%	30.59%	10.75%	10.59%	32.72%	45.49%	45.54%	45.59%

Figure 7 Correct Precision for different methods

4.2. Evaluate and Improve Trained Matrix Method

After observing some statistic results, come up some new ideas for improved the trained matrix method.

1) Apply Soundex algorithm idea----Set a translating table for the first letter. Figure.8 shows the first letter change percentage during the name translation.

```
FL-->FirstLetter
TN-->TotalNumber
NC-->NumberOfChanged
FL    TN      NC Percentage
a   2021    1126    55.71%
b   1238       0    0.00%
c    156       6    3.85%
d    764      12    1.57%
e      0       0    0
f    476      19    3.99%
g    713       3    0.42%
h    855       3    0.35%
i      0       0    0
j    371      67    18.06%
k   1339     650    48.54%
l    674       3    0.45%
m   1013       0    0.00%
n    336      33    9.82%
o      0       0    0
p    563       0    0.00%
q     33      19    57.58%
r    652      16    2.45%
s    900      47    5.22%
t    481       3    0.62%
u      0       0    0
v    644     403    62.58%
w      0       0    0
x      8       8    100.00%
y     87      17    19.54%
z    113      21    18.58%
```

Figure 8 Percentage for changing the first letter

```
yvhans johannes
yvkylv ukelo
yvmvra uemura
yvnal uenal
yvn iunn
yvnsvn jonsson
yvnyza unaiza
yvranym uranium
yvrbanyk urbanik
yvryna urena
yvzr user
zabz jobje
zakvb jacobje
zakvb jacques
zakvmyntay jacomyntie
zakyvs jacques
zak yzaak
zan john
zans jansje
zant jaantje
zarvs xaros
zfry jeffrey
zhra jahra
```

Figure 9 Part of example for the name change the first letter

So letters can be distinguished into two parts ---- First letter changed or not.

a) First part → [a,c,d,f,g,h,j,k,n,q,r,s,v,x,y,z]

b) Second part → [b,e,i,l,m,o,p,t,u,w]

Besides, conclude some first name changed situation (example shown in Figure.9), most translate relations are

clearly. Thus, the translating table for the first part is as Figure.10. It can reduce the situation like Figure. 11.

```
a ----> a/e/i/o/s/u        n ----> n/k
c ----> c/s/g              q ----> q/g
d ----> d/b/t/z            r ----> r/k/v
f ----> f/p                s ----> s/c/t
g ----> g/q                v ----> v/w
h ----> h/d/r              x ----> k
j ----> j/g/t/d            y ----> y/u/i
k ----> k/c/q              z ----> z/j
..
```

Figure.10 Translating Table

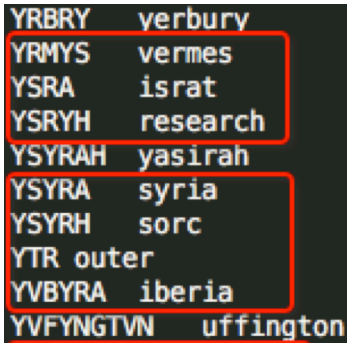


Figure 11 Translating errors

Figure. 12 Example for predicting multiple names

2) Predict multiple names, if they make a tie with the GED. Shown in Figure.12. Obviously, it can improve the Recall but reduce the Precision.

After improving, the performance shown in Table.2 :

Table.2. Tricky for Trained Matrix Method's Performance

Method Name	Correct Pre	Total Pre	Precision	Recall
train1000Lines Matrix	6126	13437	45.59%	45.59%
tran1000Lines Matrix + translating table	6377	13437	47.46%	47.46%
train1000Lines Matrix + multiple predict	6740	17126	39.36%	50.16%
tran1000Lines Matrix + translating table + multiple predict	6989	16895	41.37%	52.01%

5. Conclusions

The main method used in the project is trained matrix with some tricky technique into the GED algorithm. After modifying original approach, Recall is from 30.59% to 52.01%. It has already increased a lot, but still in modest performance. There are many reasons for that, for example:

1) Some of the correct names are not the name with best global edit distance. So need to change another based approximate method.
2) The data set for training matrix isn't big enough. It's not ideal to evaluate on the same collection that is used to build the model.
3) Not find a good strategy for calculating the cost of insertion and deletion and the score of replacement between a letter and a blank space.
4) Easily to find there're some same Persian names in train.txt transfer into different Latin name. It indicates when translate the real Persian name to what it shows in the train.txt, it has already cut down some important information, so it's more difficult to transfer the "incomplete Persian name" to a correct Latin name.

However, These drawbacks can be guaranteed as the direction for future improvement.

References

[1] Navarro, G (2001), 'A Guided Tour to Approximate String Matching.' ACM Computing Surveys, 33, 1, pp. 31–88.

[2] Kondrak, G. (2005), 'N-Gram Similarity and Distance.' String Processing and Information Retrieval (SPIRE 2005). 3772, pp. 115-126.

[3] Odell, Margaret King (1956). 'The Profit in Records Management' Systems Magazine, pp.20-21.

[4] Henikoff, S, Henikoff, J.G. (1992). 'Amino Acid Substitution Matrices from Protein Blocks' National Academy of Sciences, 89, 22, pp.10915–10919.