Report

1. The approach / procedure I take

Step1: Separating lines and words, using unified lowercase, doing Lemmatization and deleting words whose length is smaller than 2. (Similar to the procedure in the first programming homework)

Step2: ① Use each word's existence as a part of feature, the length of each feature is 4379.

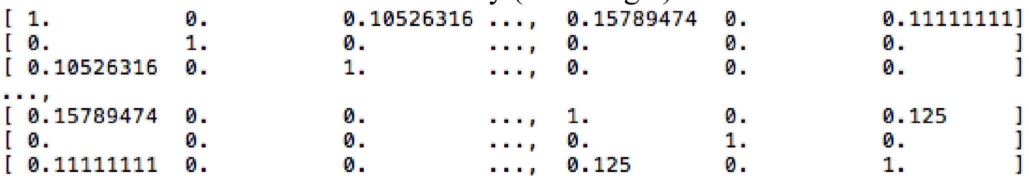
② Use three consecutive words' existence as a part of feature, the length of each feature is 24918 (3-word shingle)

Step3: Computing the exact Jaccard Similarity as the true similarity baseline.

Step4: Creating k-minhash signatures, computing the estimated Jaccard similarity and calculating the mean-squared errors. (k = 16, 32, 64, 128, 256) Here, $h_{a,b}(x) = (a * x + b\ )\ mod\ p$ (a, b are random integers, N = 3000, p is a prime number larger than N and I choose p = 3001)

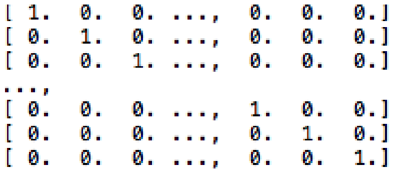
2. Creating a baseline (The exact Jaccard similarity)

Jaccard Similarity (No shingle)



```
[ 1.          0.          0.10526316 ...,  0.15789474  0.          0.11111111]
[ 0.          1.          0.         ...,  0.          0.          0.        ]
[ 0.10526316  0.          1.         ...,  0.          0.          0.        ]
...,
[ 0.15789474  0.          0.         ...,  1.          0.          0.125     ]
[ 0.          0.          0.         ...,  0.          1.          0.        ]
[ 0.11111111  0.          0.         ...,  0.125       0.          1.        ]
```

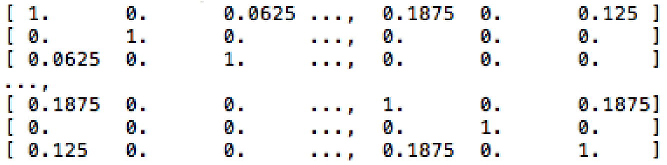
Jaccard Sim (3-word shingles)



```
[ 1.  0.  0. ...,  0.  0.  0.]
[ 0.  1.  0. ...,  0.  0.  0.]
[ 0.  0.  1. ...,  0.  0.  0.]
...,
[ 0.  0.  0. ...,  1.  0.  0.]
[ 0.  0.  0. ...,  0.  1.  0.]
[ 0.  0.  0. ...,  0.  0.  1.]
```

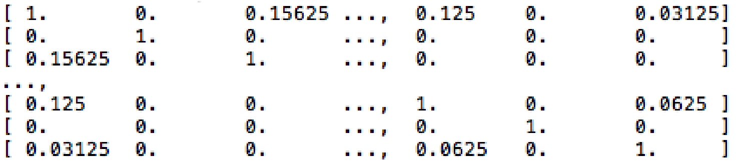
3. Creating a k-minhash sketch (Estimated Jaccard similarity)

16-MinHash Similarity (No shingle)



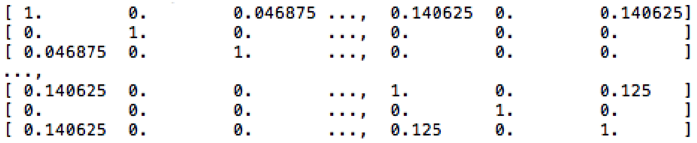
```
[ 1.      0.      0.0625 ...,  0.1875  0.      0.125 ]
[ 0.      1.      0.     ...,  0.      0.      0.    ]
[ 0.0625  0.      1.     ...,  0.      0.      0.    ]
...,
[ 0.1875  0.      0.     ...,  1.      0.      0.1875]
[ 0.      0.      0.     ...,  0.      1.      0.    ]
[ 0.125   0.      0.     ...,  0.1875  0.      1.    ]
```

32-MinHash Similarity (No shingle)



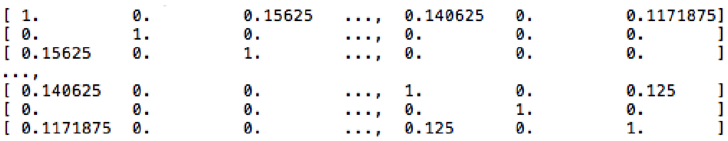
```
[ 1.       0.       0.15625 ...,  0.125    0.       0.03125]
[ 0.       1.       0.      ...,  0.       0.       0.     ]
[ 0.15625  0.       1.      ...,  0.       0.       0.     ]
...,
[ 0.125    0.       0.      ...,  1.       0.       0.0625 ]
[ 0.       0.       0.      ...,  0.       1.       0.     ]
[ 0.03125  0.       0.      ...,  0.0625   0.       1.     ]
```

64-MinHash Similarity (No shingle)



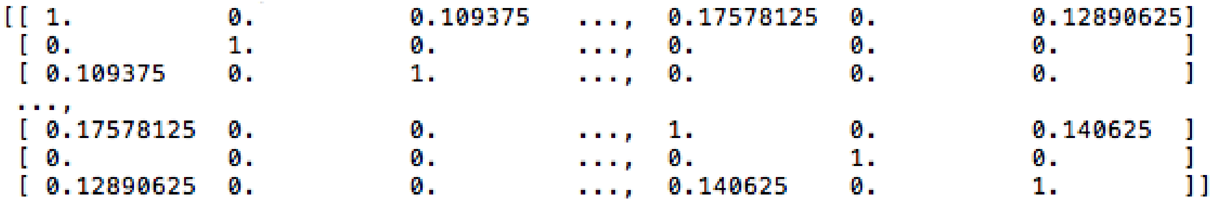
```
[ 1.        0.        0.046875 ...,  0.140625  0.        0.140625]
[ 0.        1.        0.       ...,  0.        0.        0.      ]
[ 0.046875  0.        1.       ...,  0.        0.        0.      ]
...,
[ 0.140625  0.        0.       ...,  1.        0.        0.125   ]
[ 0.        0.        0.       ...,  0.        1.        0.      ]
[ 0.140625  0.        0.       ...,  0.125     0.        1.      ]
```

128-MinHash Similarity (No shingle)



```
[ 1.         0.         0.15625   ...,  0.140625   0.         0.1171875]
[ 0.         1.         0.        ...,  0.         0.         0.       ]
[ 0.15625    0.         1.        ...,  0.         0.         0.       ]
...,
[ 0.140625   0.         0.        ...,  1.         0.         0.125    ]
[ 0.         0.         0.        ...,  0.         1.         0.       ]
[ 0.1171875  0.         0.        ...,  0.125      0.         1.       ]
```

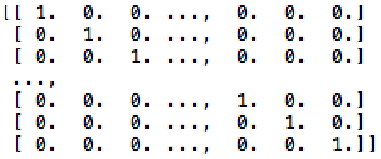
256-MinHash Similarity (No shingle)



```
[[ 1.          0.          0.109375   ...,  0.17578125  0.          0.12890625]
 [ 0.          1.          0.         ...,  0.          0.          0.        ]
 [ 0.109375    0.          1.         ...,  0.          0.          0.        ]
 ...,
 [ 0.17578125  0.          0.         ...,  1.          0.          0.140625  ]
 [ 0.          0.          0.         ...,  0.          1.          0.        ]
 [ 0.12890625  0.          0.         ...,  0.140625    0.          1.        ]]
```

16~256-MinHash Similarity (3-word shingles)

(Values being shown are happened to be the same, but the actual similarity matrix is different)

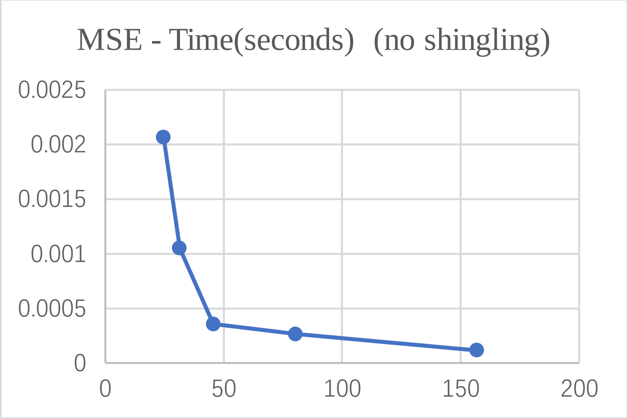


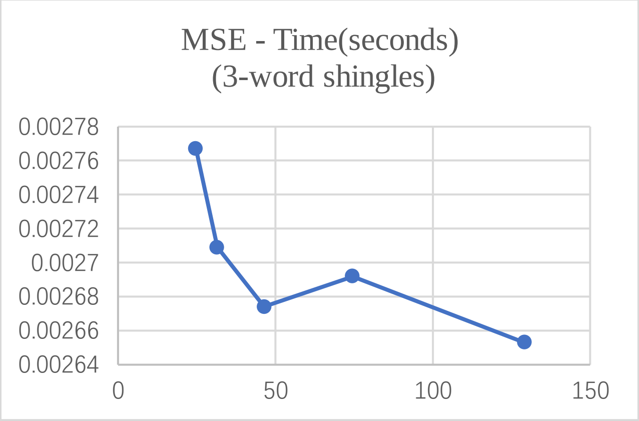
```
[[ 1.  0.  0. ...,  0.  0.  0.]
 [ 0.  1.  0. ...,  0.  0.  0.]
 [ 0.  0.  1. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  1.  0.  0.]
 [ 0.  0.  0. ...,  0.  1.  0.]
 [ 0.  0.  0. ...,  0.  0.  1.]]
```

4. Results

No shingle	Base-line	16-minhash	32-minhash	64-minhash	128-minhash	256-minhash
Efficacy (Mean-squared error)	0	0.002064	0.001050	0.000357	0.000264	0.000115
Generating Signatures' Time (sec)	/	0.22	0.40	0.65	1.12	3.14
Comparing Time (seconds)	17.69	24.40	31.07	44.93	79.24	153.72
Efficiency (Total Time) (sec)	17.69	24.62	31.47	45.58	80.36	156.86

3-word shingles	Base-line	16-minhash	32-minhash	64-minhash	128-minhash	256-minhash
Efficacy (Mean-squared error)	0	0.002767	0.002709	0.002674	0.002692	0.002653
Generating Signatures' Time (sec)	/	0.21	0.34	0.58	0.97	1.69
Comparing Time (seconds)	10.41	24.56	31.18	45.84	73.56	127.45
Efficiency (Total Time) (sec)	10.41	24.77	31.52	46.42	74.53	129.14





(In the plot, the 5 points from left to right are k = 16, 32, 64, 128, 256)

5. Analysis

① Increasing k causes the higher cost time. Meanwhile, the mean-squared errors become smaller. (There is an exception point k = 128 in 3-word shingles. The MSE error between k = 64 and k = 128 is 0.000018, which may be caused by randomness).

② The time used by Minhash should be smaller than baseline's comparing time theoretically. In this homework, it's larger because I use for-loop to compute the Minhash methods' similarity, which is slower. However, when I calculate the baseline's Jaccard similarity matrix, I use existing function intersection and union without for-loop, which will be faster.

③ According to my experiment results, no shingling's mean-squared errors are smaller than 3-word shingles. May be these sentences are short, which can be handled by no shingling quite well. Instead, 3-word shingles decrease the opportunities for two sentences to have the same feature, since it is harder to have the same "3 consecutive words" than just one same word.