Data Mining Assignment IV

Fang Zhou

1. Introduction

In data mining area, the scale of feature space is very crucial to the efficiency and efficacy of data mining algorithms. The goal in this assignment is to get deep into k-minhash algorithm. The main idea of k-minhash is to find a smaller feature space instead of the original large feature space. In this assignment, I first extract the feature vector for each document with different n-grams and dimensions. Then I measure the similarity of each pair of documents with Jaccard similarity as true similarity. I also run k-minhash approach with different sketches working on different datasets generated above. I evaluate the efficiency and quality of k-minhash approach and conclude some interesting findings in the end.

2. Methods

2.1 Article Feature

In this assignment, I extract the feature from documents based on the n-shingling frequency. A shingling is a contiguous sequence of n items from a given sequence of text or speech. In common case, a group of words can represent more meaningful information than single word. So it is a very important feature for an article. I choose m most frequent n-shinglings in the whole corpus, where m is equal to 1000, 2000, 3000, 4000, or 5000 and n is equal to 1, 3, 5, 7, or 10. If one n-shingling from the most frequent ones appears in an article, then I set this feature value to 1 in its feature vector. Otherwise, this feature's value is set to 0. More details can be seen in the code and files submitted.

2.2 Data Pre-processing

However, the feature vectors collected in the first step don't fit well for this assignment. For example, some articles may have an empty body part so this article is meaningless for the assignment and algorithm. I pre-process the data following the rules shown below:

- If an article does not have the body part or its body part is empty, then skip this article.
- If an article does not include any shingles in the most frequent n-shingling, then skip this article.

The first rule is very easy to understand. The second rule is used to exclude the article with a feature vector including all zeroes. After this processing, the number of articles with different n-shingling decreases and the data set become more reasonable.

Table I: The number of articles

Shingling	1-shingling	3-shingling	5-shingling	7-shingling	10-shingling
Number of Articles	19042	17314	8799	4822	2585

As we can see, the numbers of articles decrease with the increasing shingling numbers. This result follows my mind. The common knowledge for n-shingling is that when n is equal to 5, it works well for short documents; when n is equal to 10, it works well for long documents. In our dataset, most articles are short documents. So we can see that the number of 5-shingling articles has an appropriate value. However, the number of meaningful 10-shingling articles is only 10% of the articles in the original dataset. In this case, 10-shingling is not a good feature to evaluate the whole corpus because it loses the other 90% articles. It can be concluded that if a document is not long document, the n-shingling should use little n, although larger n can provide more meaningful semantic meaning. In this assignment, I want to evaluate the k-minhash algorithm with different n-shinglings, so I still test the performance of dataset with large n-shingling.

2.3 Similarity Algorithms

The baseline approach is Jaccard similarity. Jaccard similarity leverages the feature sets of two different documents and compute the result of the number of intersection result between two feature sets divided by the number union result between two feature sets. Jaccard similarity can be defined:

$$JS(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

K-minhash algorithm is a technique for quickly estimating how similar two sets are. The scheme was invented by Andrei Broder and initially used in several search engines to detect duplicate web pages and eliminate them from search results. In this assignment, I choose fast k-minhash algorithm to speed up the efficiency of k-minhash algorithm.

Fast k-minhash algorithm need prepare k hash function in advance. The hash function can be defined:

$$h_{a,b}(x) = \big((a \times x + b) \ mod\ p\big) \ mod\ N$$

In this function, a and b are random integers; p is a prime number larger than N; and N is the number of shingles. For each article, k-minhash records the minimal hash value for a specific hash function. The similarity of two articles is measured by comparing the minimal hash values for these two articles, as we discussed in the class.

3. Experiments

3.1 Testbed

I test the program on a single machine. The machine has a 2.4 GHz 6-core Intel Xeon CPU E5-2630, 24 GB memory and one 500 GB Western Digital SATA hard drives.

3.2 Experiments Details

In this assignment, I first extract the feature vector for each document with m (m = 1, 3, 5, 7, and 10) shingles and n (n = 1000, 2000, 3000, 4000, and 5000) dimensions. Then I measure the similarity of each pair of documents with Jaccard similarity as the baseline. I run k-minhash approach with different k (k = 16, 32, 64, 128, 256) working on different datasets generated above.

I run each test for 10 times and calculate the average value of them. All the results are the average value without special notifications.

3.3 Implementation

I implement all the code by myself. The language I choose is Python. I do not optimize the feature vector, which means I use the traditional matrix to store the feature vector. Although I can optimize the feature vector with compressed sparse row, the feature value in the matrix is only 0 or 1 and the value can be processed by bitwise operation quickly. So I do not take this optimization and the performance is still good enough.

4. Results

In this section, I will show the scalability and accurate results of Jaccard similarity algorithm and k-minhash algorithm.

4.1 Efficiency

First, I will show you the scalability of Jaccard similarity algorithm and k-minhash algorithm. For all the tests, I generate datasets with different k-shingles and numbers of features. For Jaccard similarity, I just compare each pair of articles in the corresponding datasets. In the following each table, the first row shows the number of features in the datasets with a specific shingle. The first column shows the different minhash modes, like 16-minhash, 32-minhash, and so on. The unit of result is second. Datasets with different shingles have different numbers of articles. This is the reason I conclude and show the results in the unit of number of shingles. The results can be seen in Table II, III, IV, V, VI, for 1-shingle, 3-shingle, 5-shingle, 7-shingle, and 10-shingle, respectively.

Table II: Running time (second) for similarity algorithms with 1-shingle

# Features	1000	2000	3000	4000	5000
Jaccard	735.92	1347.22	1956.17	2582.98	3192.26
16-minhash	79.50171515	77.67891802	81.24747999	76.81007065	79.11219986
32-minhash	161.1368681	154.1860751	158.0935907	151.0199812	161.1006316
64-minhash	314.2583008	308.5132411	312.8658453	303.8209845	317.6546248
128-minhash	628.2197919	610.1735452	619.144368	603.5932376	630.279262
256-minhash	1253.61	1217.2	1235.63	1206.8	1254.66

Table III: Running time (second) for similarity algorithms with 3-shingle

# Features	1000	2000	3000	4000	5000
Jaccard	593.901	1104.91	1612.06	2127.11	2635.16
16-minhash	80.2210146	78.49789775	77.25693693	75.83301224	80.86017949
32-minhash	160.7821124	155.3422722	158.1413313	151.9826627	159.0868164
64-minhash	313.5677406	307.481286	311.5625805	305.2677485	315.6152046
128-minhash	627.0490745	611.1984087	618.7173623	605.2154543	631.5725011
256-minhash	868.839	855.799	886.421	864.177	856.083

Table IV: Running time (second) for similarity algorithms with 5-shingle

# Features	1000	2000	3000	4000	5000
Jaccard	146.721	282.02	413.357	546.844	677.214
16-minhash	15.58346588	14.70733537	15.01461591	17.5884262	17.29646199
32-minhash	31.3275359	29.95613977	30.02128969	29.75655242	33.09317835
64-minhash	60.84957493	60.38615368	60.50151179	58.82347238	60.32134413
128-minhash	120.5310832	117.4929879	117.3108203	115.8465289	116.029977
256-minhash	239.464	228.092	224.671	225.314	227.206

Table V: Running time (second) for similarity algorithms with 7-shingle

# Features	1000	2000	3000	4000	5000
Jaccard	43.686	83.695	124.317	163.653	204.538
16-minhash	16.07756713	19.15556658	17.3281672	16.77297219	16.45781999
32-minhash	33.46585236	29.24076993	30.04581519	29.65554431	28.72090263
64-minhash	62.79530652	61.95648869	60.19705622	61.01834571	60.74717363
128-minhash	122.9344476	118.8196034	113.8320431	114.5785082	114.8256817
256-minhash	73.706	71.962	70.241	71.398	71.848

Table VI: Running time (second) for similarity algorithms with 10-shingle

# Features	1000	2000	3000	4000	5000
Jaccard	11.674	22.107	36.129	47.924	59.603
16-minhash	2.975814155	5.978517877	3.511131189	4.588992749	3.916688161
32-minhash	3.746207948	3.291374687	5.469562329	4.559542409	5.989893416
64-minhash	7.384870219	8.971361531	8.532865109	7.487865644	6.056257036
128-minhash	13.61498104	14.01797902	12.73686322	11.2661884	13.85613625
256-minhash	23.754	23.662	21.324	21.788	22.643

As we can see, the scalability of Jaccard similarity algorithm increases linearly with the increasing number of features. Jaccard similarity algorithm needs to compare according feature for each pair. The time complexity of Jaccard is O(N^2*m), where N is number of articles in the dataset and m is the number of features. When the number of feature increase, the cost increases. So the trend of Jaccard similarity algorithm is linear. For k-minhash algorithm, the results with different k value change very little. In k-minhash algorithm, before comparing the similarity, algorithm first generate the signatures of all articles, whose time complexity is O(N*m*k), where N is the number of articles in the dataset, m is the feature number, and k is the number of hash functions. When comparing the similarity of pairs, k-minhash algorithm needs O(N^2*k). It is very obvious that O(N^2*k) is changed with different k values and O(N^2*k) is larger than O(N*m*k). In each dataset, the N is constant and k is changed for different k-minhash algorithms. This can explain k-minhash has very similar performance with a fixed k value and increases with increasing value of k.

In most cases, k-minhash algorithm outperforms Jaccard similarity algorithm. What's more, k-minhash use fewer features instead of a large number of original features.

4.2 Quality

In this section, I will use several tables to show the accuracy rate of each algorithm. To measure the quality, I choose mean-squared error (MSE) between true similarity (baseline) and estimated similarity (k-minhash). The results shown in table VII, VIII, IX, X, and XI.

Table VII: MSE for k-minhash algorithms with 1-shingle

# Features	1000	2000	3000	4000	5000
16-minhash	0.005542324	0.007068952	0.007124374	0.006782387	0.006788737
32-minhash	0.008264552	0.007870953	0.004705244	0.006348293	0.003580238
64-minhash	0.012159921	0.004352744	0.008328064	0.006876199	0.005922045
128-minhash	0.00311083	0.001889014	0.001643899	0.003119485	0.002572331
256-minhash	0.030431053	0.009574542	0.015358266	0.008535415	0.006981575

Table VIII: MSE for k-minhash algorithms with 3-shingle

# Features	1000	2000	3000	4000	5000
16-minhash	0.040125058	0.039987997	0.040005829	0.040118968	0.040006225
32-minhash	0.011393989	0.010563725	0.010292144	0.010343664	0.01016334
64-minhash	0.003078779	0.002879606	0.002743484	0.002770117	0.002844451
128-minhash	0.001594358	0.001201107	0.001074627	0.000843582	0.000825036
256-minhash	0.000715924	0.000498174	0.000300123	0.000260537	0.000276773

Table IX: MSE for k-minhash algorithms with 5-shingle

# Features	1000	2000	3000	4000	5000
16-minhash	0.139816673	0.13975265	0.139646101	0.139651615	0.139658062
32-minhash	0.055708178	0.055583153	0.055560542	0.055506276	0.055475696
64-minhash	0.020638396	0.020421458	0.020463882	0.020424742	0.020383027
128-minhash	0.004770839	0.004809571	0.004762155	0.004732566	0.004734532
256-minhash	0.000218619	0.000132411	0.000126505	0.000131308	0.00010397

Table X: MSE for k-minhash algorithms with 7-shingle

# Features	1000	2000	3000	4000	5000
16-minhash	0.16586767	0.165830537	0.165820967	0.165778881	0.165827161
32-minhash	0.075843556	0.075825767	0.075705533	0.075714916	0.075695215
64-minhash	0.030758394	0.030862551	0.03058651	0.030514606	0.030502535
128-minhash	0.014221438	0.014202534	0.014167555	0.014163544	0.014158496
256-minhash	0.000110906	0.000535821	0.00044673	0.00014951	9.7553E-05

Table XI: MSE for k-minhash algorithms with 10-shingle

# Features	1000	2000	3000	4000	5000
16-minhash	0.352620784	0.352587607	0.352571979	0.352644207	0.352599474
32-minhash	0.272441106	0.27234372	0.272367109	0.272344605	0.272335927
64-minhash	0.037130648	0.037217408	0.037098326	0.037090056	0.037044472
128-minhash	0.004730127	0.004536654	0.004556654	0.004532252	0.004521389
256-minhash	0.000606954	0.000463084	0.000202486	0.000127534	9.33E-05

As we can see, with fixed number of features, the quality becomes better with increasing k value. Meanwhile, with fixed value of k, the quality becomes better with increasing number of features. Another observation is in most cases, with fixed feature number and k-minhash algorithm, quality with a large shingle is better than the one with a little shingle. For instance, when we use 256-minhash algorithm and feature vector including 3000 features, MSE value is 0.0003 and 0.0002 for 7-shingle and 10-shingle, respectively. However, the size of 10-shingle dataset is much smaller than the one of 7 shingle dataset, as we mentioned in the section 2.

5. Conclusion

It is very complicated and difficult to explain the phenomenon we have tested and observed. However, we can conclude some results we have observed:

- K-minhash algorithm works well to decrease the dimensions in the feature vector.
- Although the efficiency is not the main reason we bring k-minhash algorithm, in our detailed dataset, K-minhash algorithm is more efficiency in most cases, compared to Jaccard similarity algorithm.
- The efficiency of K-minhash algorithm decreases when the k value is larger.
- In the assignment, we have three parameters: the number of sketch in k-minhash (k), the number of features (m), and the shingle value used to create datasets (n). Each conclusion about quality below has one changeable parameter and the other two parameters have fixed values.
  - K-minhash algorithm performs better when the k value is larger.
  - K-minhash algorithm performs better when the m value is larger.
  - K-minhash algorithm performs better when the n value is larger.
- It is still a problem about how to decide the value of n in n-shingle processing. Larger n can bring better quality. However, larger n also reduces the scale of dataset. Once we prepare a dataset, we need to make a balance between accuracy and efficiency.