CS168# minipro2

16337266 徐原

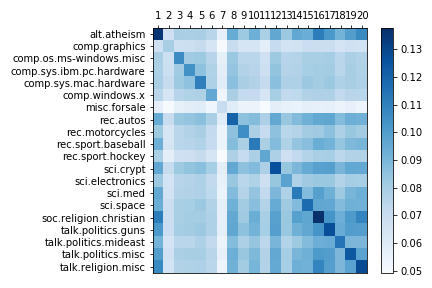
16337270 杨滨好

# Part1

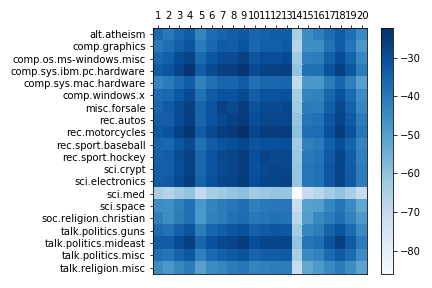
1. (2 points) Make sure you can import the given datasets into whatever language you’re using. For example, if you’re using python, read the data50.csv ﬁle and store the information in an appropriate way. Remember that the total number of words in the corpus is huge, so you probably want to work with a sparse representation of your data (e.g., you don’t want to waste space on words that don’t occur in a document). If you’re using MATLAB, you can simply import the data using the GUI.

**Answer: See code project2.py**

1. (8 points) Implement the three similarity metrics described above. For each metric, prepare the following plot. The plot will look like a 20×20 matrix. Rows and columns are index by newsgroups (in the same order). For each entry (A,B) of the matrix (including the diagonal), compute the average similarity over all ways of pairing up one article from A with one article from B. After you’ve computed these 400 numbers, plot your results in a heatmap. Make sure that you label your axes with the group names and pick an appropriate colormap to represent the data: the rainbow colormap may look fancy, but a simple color map from white to blue may be a lot more insightful. Make sure to include a legend



**Figure-1.1 Jaccard**



**Figure-1.2 l2**

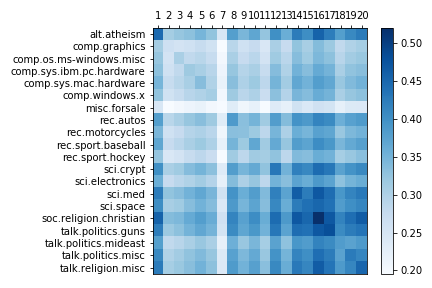


Figure1-3 cosine

1. (4 points) Based on your three heatmaps, which of the similarity metrics seems the most reasonable, and why would you expect that/those metrics to be better suited to this data? Are there any pairs of newsgroups that are very similar? Would you have expected these to be similar?

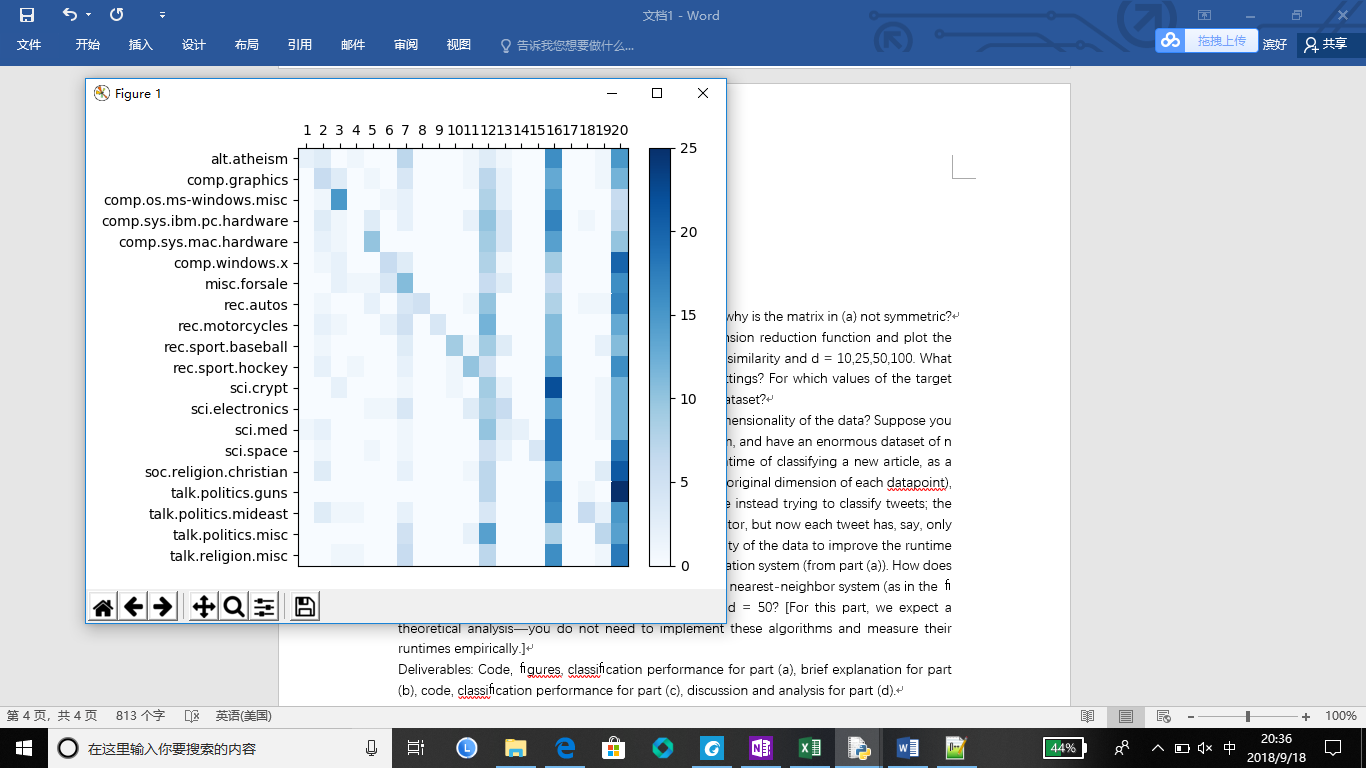
Answer: Jaccard seems the most reasonable. Firstly, the same groups must have the most similarity, so the diagonal of matrix should perform in deep color. Secondly, we can analysis some groups, for example: comp.os.ms-windows.misc , comp.sys.ibm.pc.hardware, and comp.sys.mac.hardware. All of them three may belong to the “same news” . Jaccard shows that both characteristic simultaneously.

# Part2

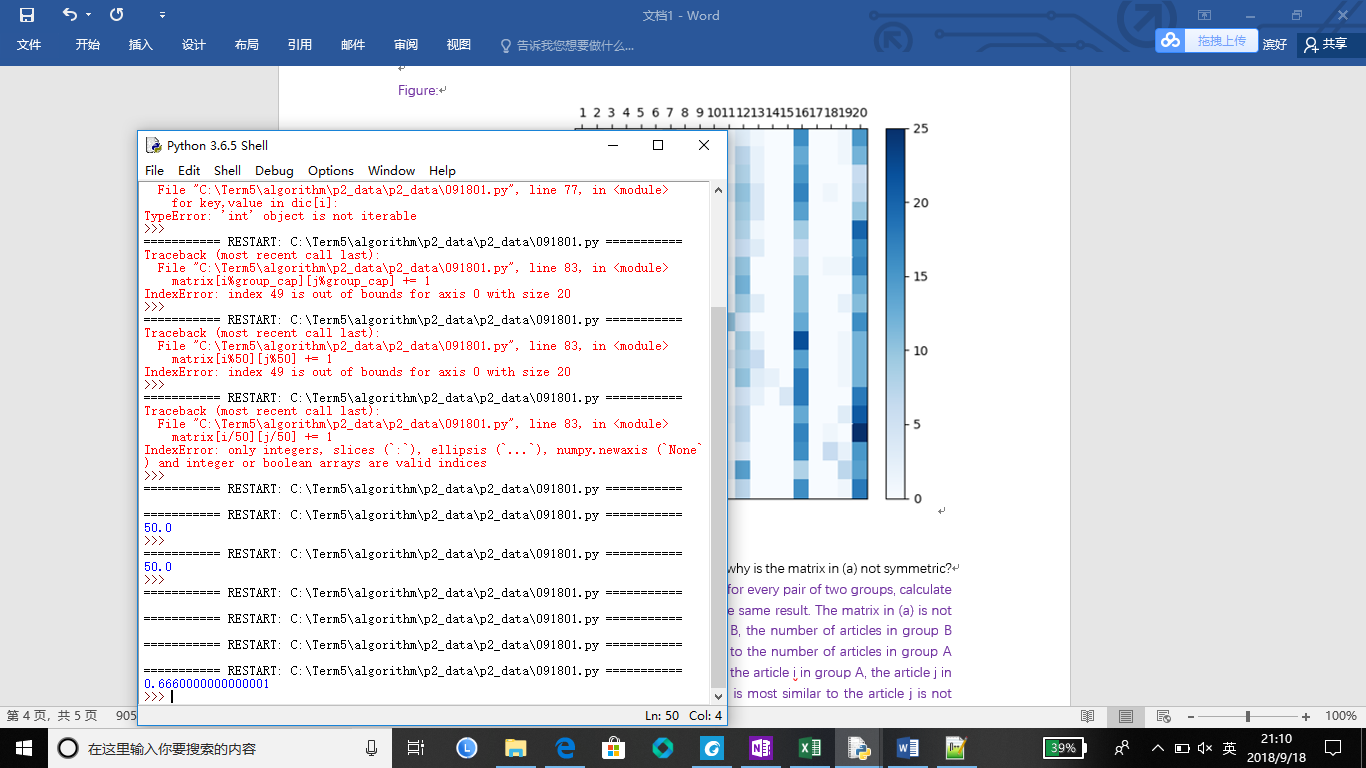
1. (3 points) (Baseline Classification) Implement the baseline cosine-similarity nearest-neighbor classiﬁcation system that, for any given document, ﬁnds the document with largest cosine similarity, and returns that newsgroup/label. (Do each computation using brute-force search.) Compute the 20×20 matrix whose entry (A,B) is deﬁned by the number of articles in group A that have their nearest neighbor in group B. Plot these results in a heatmap.

What is the average classfication error (i.e., what fraction of the 1000 articles have the same newsgroup/label as their closest neighbor)?

**Answer: see code part2**



Classification Error:



1. (2 points) Your plots for Part 1(b) were symmetric—why is the matrix in (a) not symmetric?

Answer: Plots for part1(b) were symmetric because for every pair of two groups, calculate the similarity of them using the average will get the same result. The matrix in (a) is not symmetric because for every pair of group A and B, the number of articles in group B most similar to articles in group A does not equal to the number of articles in group A most similar to articles in group B. For instance, for the article i in group A, the article j in group B is most similar to it, but the article which is most similar to the article j is not necessarily the article i.

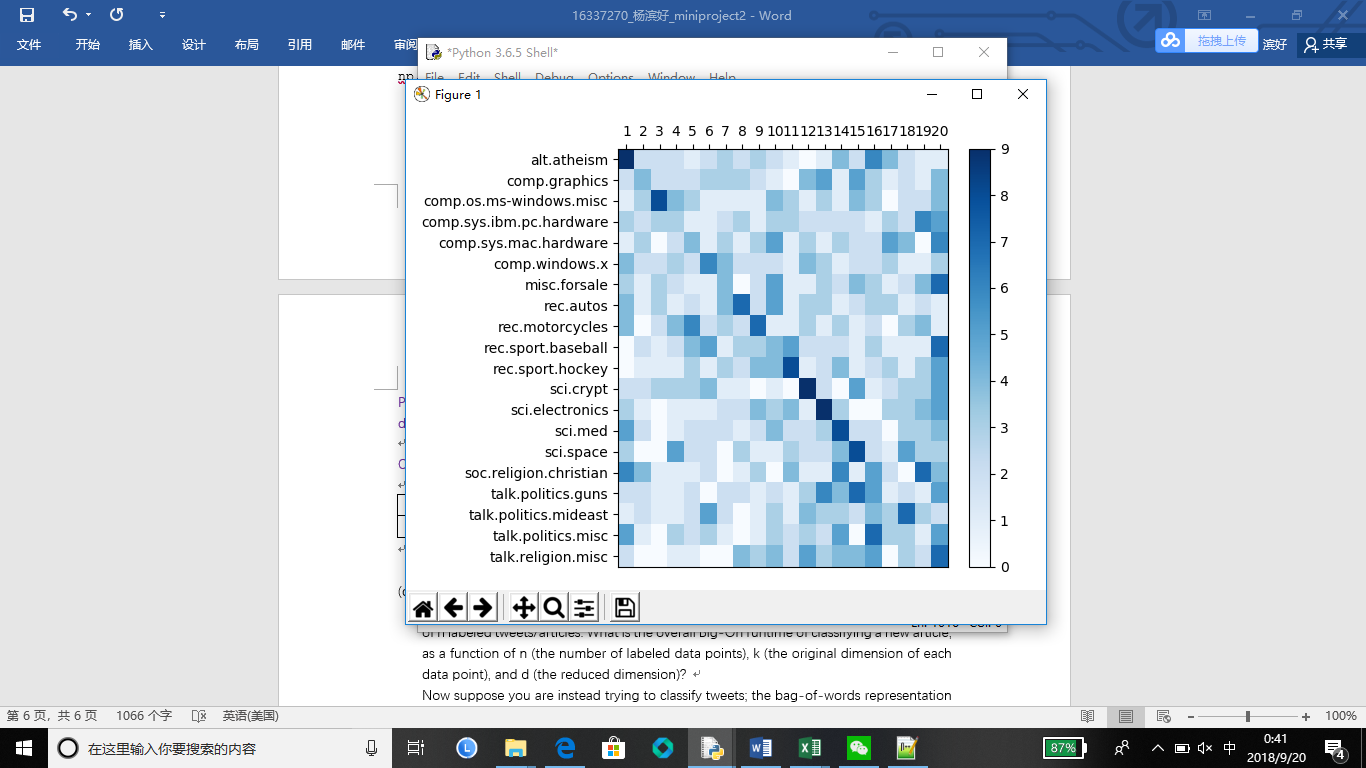
1. (7 points) Implement the random projection dimension reduction function and plot the nearest-neighbor visualization as in part (a) for cosine similarity and d = 10,25,50,100. What is the average classification error for each of these settings? For which values of the target dimension are the results comparable to the original dataset?

Solution:

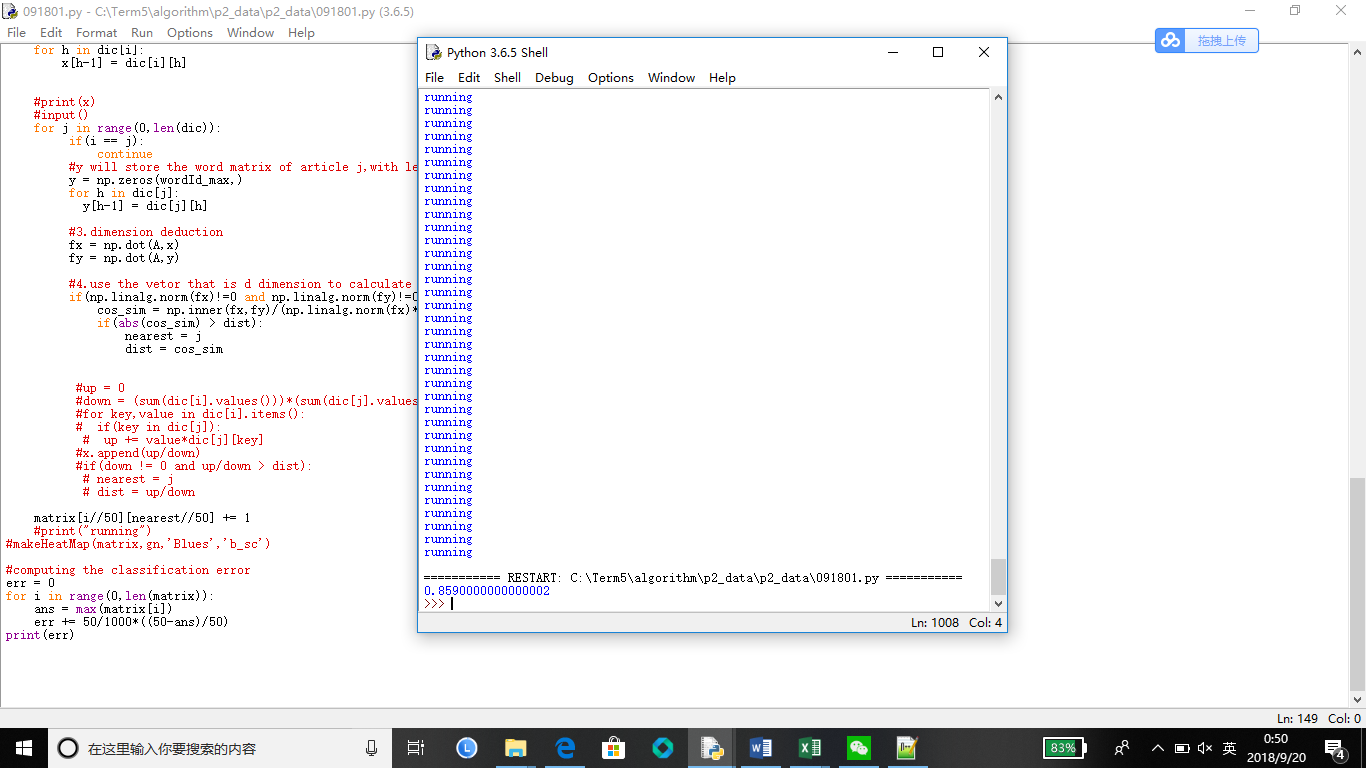
Code is similar to (a), the important part:

Plot the heatmap:

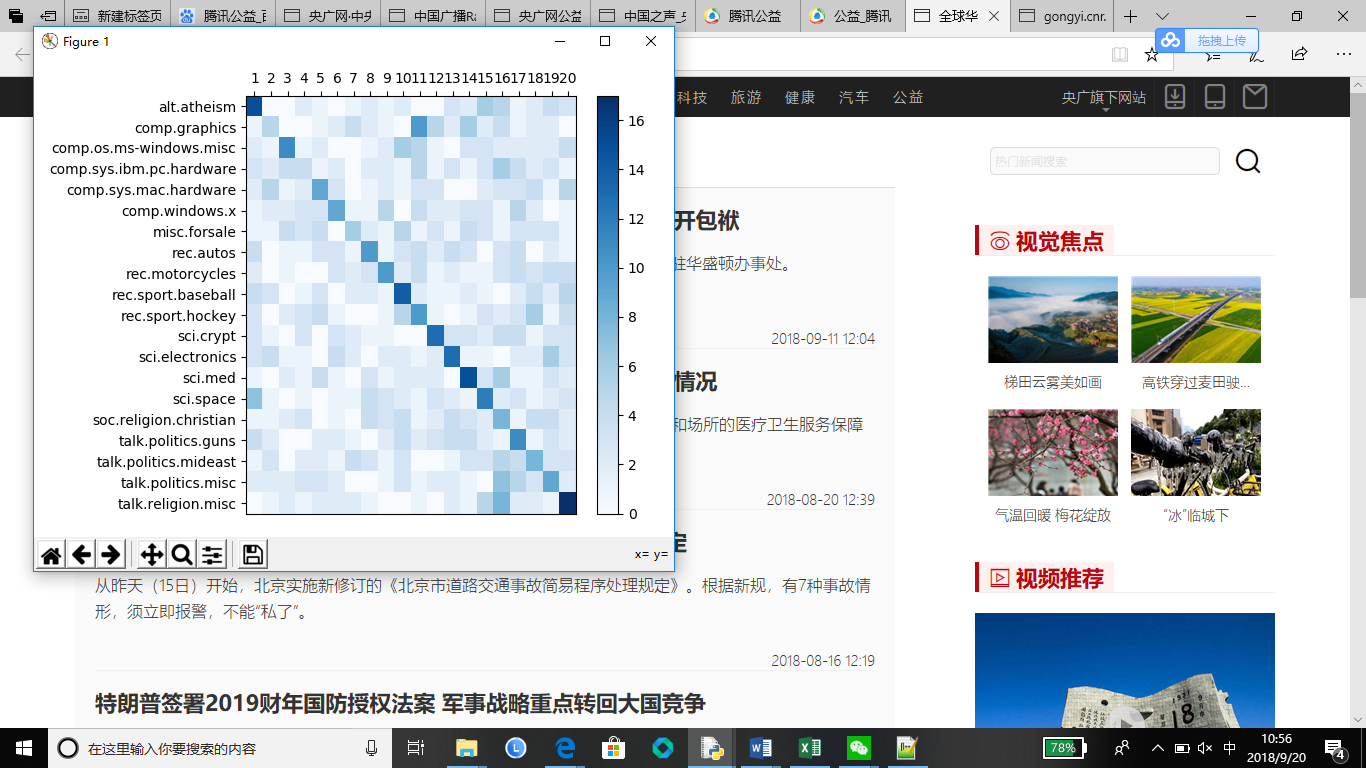
d = 10：



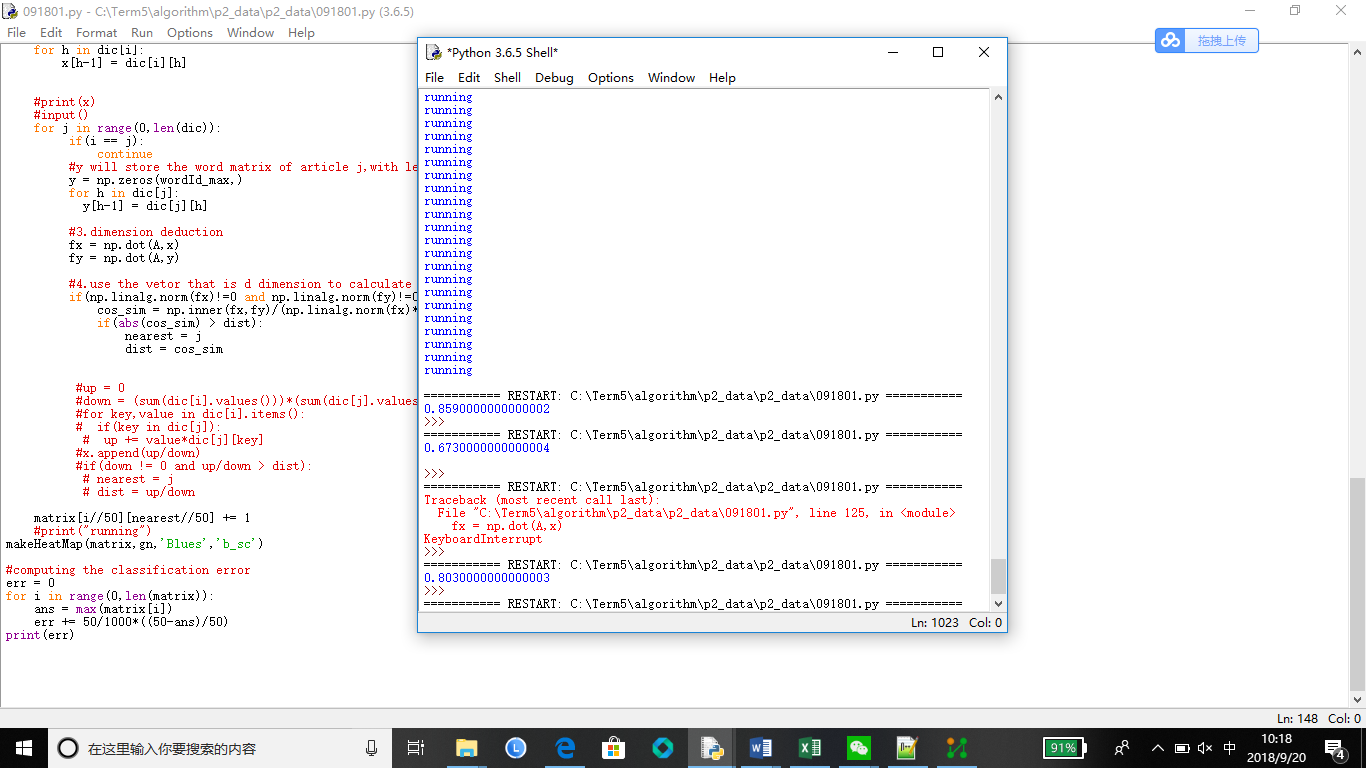
Classification error:



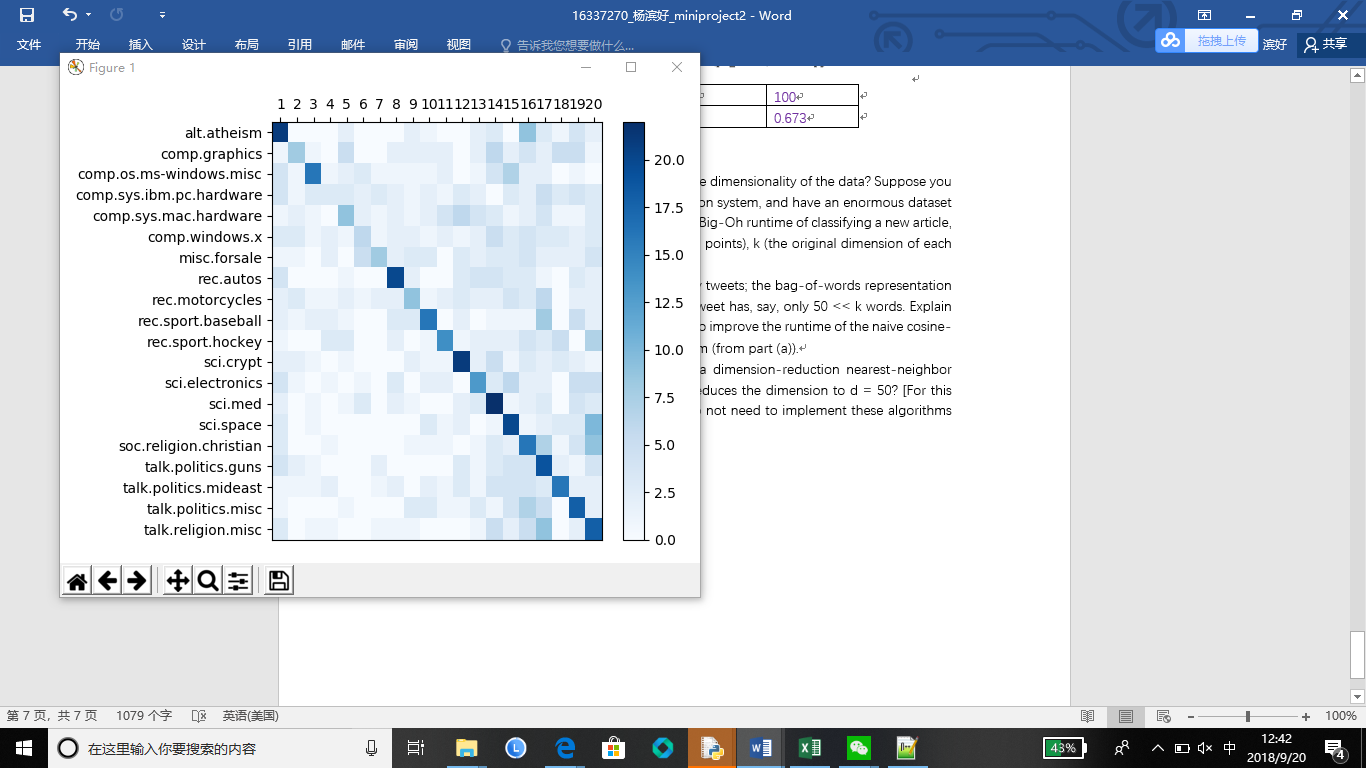
d=20:



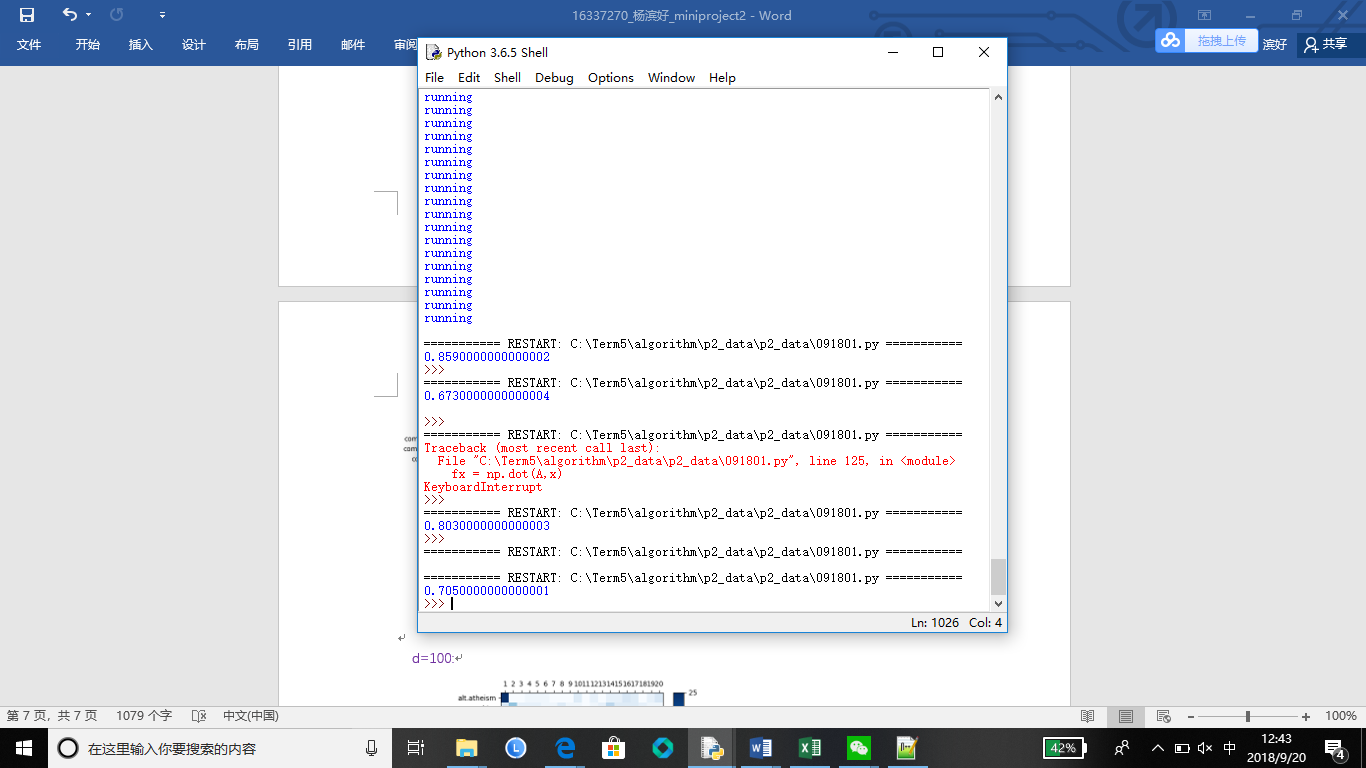
Classification error:



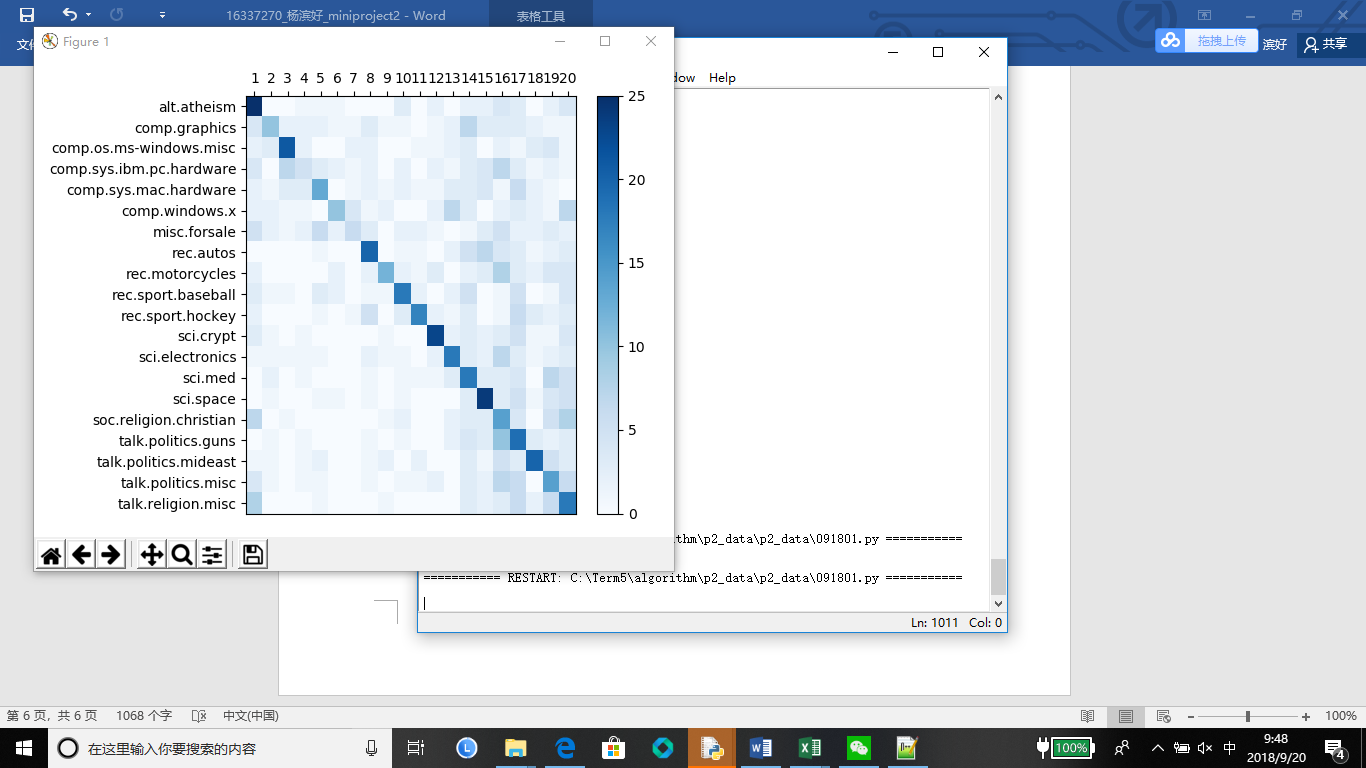
d = 50:



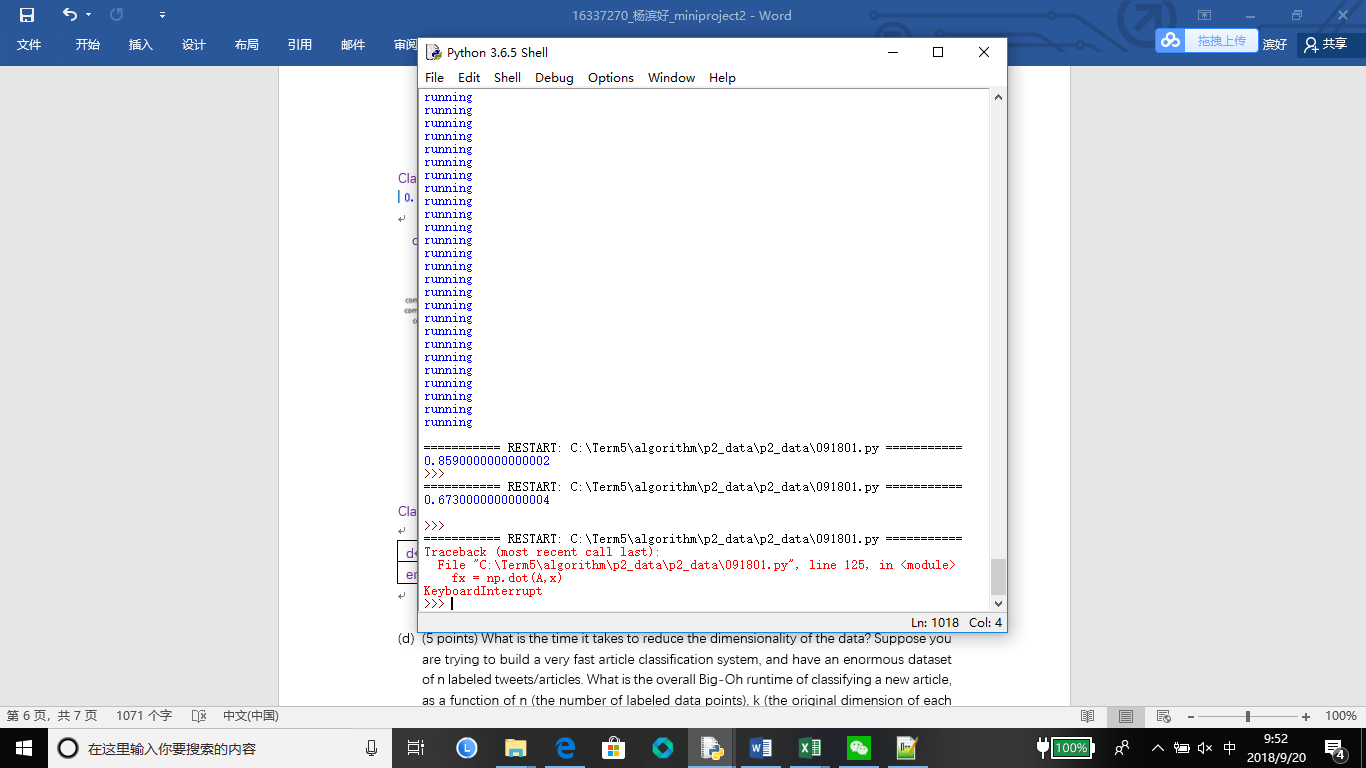
Classification error:



d=100:



Classification error:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| d | 10 | 20 | 50 | 100 |
| error | 0.859 | 0.803 | 0.705 | 0.673 |

When d = 100, the results are comparable to the original dataset.

1. (5 points) What is the time it takes to reduce the dimensionality of the data? Suppose you are trying to build a very fast article classification system, and have an enormous dataset of n labeled tweets/articles. What is the overall Big-Oh runtime of classifying a new article, as a function of n (the number of labeled data points), k (the original dimension of each data point), and d (the reduced dimension)?

Now suppose you are instead trying to classify tweets; the bag-of-words representation is still a k-dimensional vector, but now each tweet has, say, only 50 << k words. Explain how you could exploit the sparsity of the data to improve the runtime of the naive cosine-similarity nearest-neighbor classification system (from part (a)).

How does this runtime compare to that of a dimension-reduction nearest-neighbor system (as in the first step of this part) that reduces the dimension to d = 50? [For this part, we expect a theoretical analysis—you do not need to implement these algorithms and measure their runtimes empirically.]

Solution:

To reduce the dimensionality of the data, it takes time to multiply the Gaussian matrix (d,k) with the original k-dimension vector to get the d-dimension vector, approximately O(d\*k) for an article.

For a k-dimensional vector, as in part(a), I choose to store its non-zero values into a “dictionary” so that it can take less space and easy to calculate.

For data that are sparse like the “data50.csv” provided, using the “dictionary” may be more effective, because compared to the k-dimensional vector, a little “dictionary” is capable for words in an article. For data that are dense and it’s necessary to use the vector, using the dimension deduction can really deduce the runtime.

# Part 3: Locality Sensitive Hashing

Goal: The goal of this part is to implement a basic Locality-Sensitive-Hashing nearest-neighbor classification system, and experiment with the tradeoﬀ between bucket size and number of hash table. This part is largely an illustration that such techniques can be applied for fast classification—a larger dataset would have illustrated this better (though Parts 1 and 2 would have taken much longer :).

Description: You will implement the Random Hyperplane Hashing LSH scheme, which has the property that vectors with larger cosine similarity will have a higher probability of colliding (i.e. hashing to the same value). [You will be able to reuse much of the code from Part 2.] The hashing scheme, and associated nearest-neighbor classification system, is defined as follows: • Hyperplane Hashing: Construct ` hashtables in the following manner: for the i’th hashtable, deﬁne a d×k matrix Mi by drawing each entry randomly (and independently) from a normal distribution of mean 0 and variance 1. The ith hashvalue of the k-dimensional vector v is deﬁned as the binary vector sgn(Miv) ∈{0,1}d, where each positive coordinate of Miv is replaced by a “1” and each nonpositive coordinate by a “0”. Note that each hashtable has 2d buckets, and each data point is placed in exactly one bucket of each of the hashtables.

• Classification: Suppose each original datapoint v has already been hashed (to bucket sgn(Miv) of the ith hashtable, for each Then, to predict the label of a (new) query vector q, do the following:

1. compute its ` hashvalues (bucket sgn(Miu) of the ith hashtable);
2. consider the set Sq of the original datapoints that were placed in at least one of these ` buckets;
3. among all points of Sq, compute the data point x that is most similar to the query q (using brute-force search over Sq); and
4. label q with x’s label.
5. (3 points) Consider the ith hash tables in the above scheme, corresponding to matrix Mi. For two vectors, x,y ∈ Rk that form an angle of angle(x,y) = θ < π/2 radians (i.e. x and y form an acute angle of θ), what is the probability (over the randomness in the construction of the matrix Mi) that they hash to the same bucket in this ith hash function? [Hint: for each of the d coordinates that define the hash of x and y, what is the probability that they are equal, as a function of θ?] Prove your claim in at most two sentences.

Solution: For each of the d coordinates that define the hash of x and y, the possibility that the hash of x and y are equal is (1-θ/180), which dot(v,x) and dot(v,y) has the same sign. Therefore, the possibility that they hash to the same bucket is (1-θ/180)^d.

1. (1 points) If angle(x,y) ≥ 2θ, what can you say about the probability that x and y hash to the same bucket in the ith hash table? Prove your claim in at most one sentence.

Solution: The possibility that x and y hash to the same bucket in the ith hash table is (θ/180)^d if angle(x,y) > π/2 and (1-θ/180)^d if angle(x,y) < π/2, for dot(v,x) and dot(v,y) has the same sign(thus getting the same hash value) when the hyperplane is appropriate.

1. (5 points) Suppose you have a dataset X = {x1,x2,...,xn} with n =1,000,000 points, and consider the scheme described in the “Hyperplane Hashing” and “Classification” protocols, where you have ` diﬀerent hash tables, and, for a query point y ∈ Rk, you check every point in your dataset X = {x1,x2,...,xn} that collides with y in at least one of the ` diﬀerent hash tables. Suppose you know that there is some point xi ∈ X with angle at most 0.1 radians from y, and that there are not too many points xj with an angle angle(xj,y) ∈ (0.1,0.2). How should you pick d and l such that ：

1) With probability at least 0.9, the point xi with angle(xi,y) ≤ 0.1 ends up hashing to same bucket as y in at least one of the ` hash tables, and

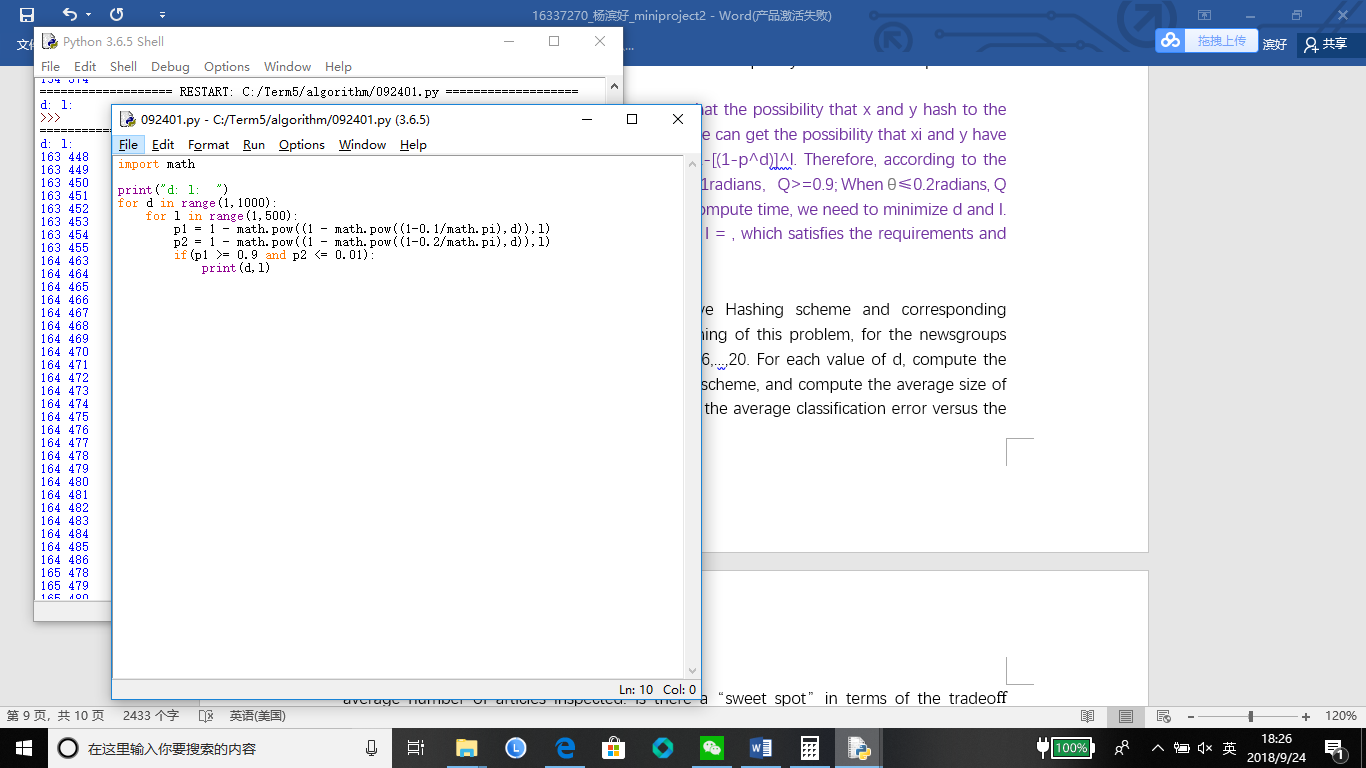
2) The expected number of points xk that have angle greater than 0.2 from y that you end up needing to consider (i.e. that hash into the same bucket in at least one of the ` hash functions) is small, and

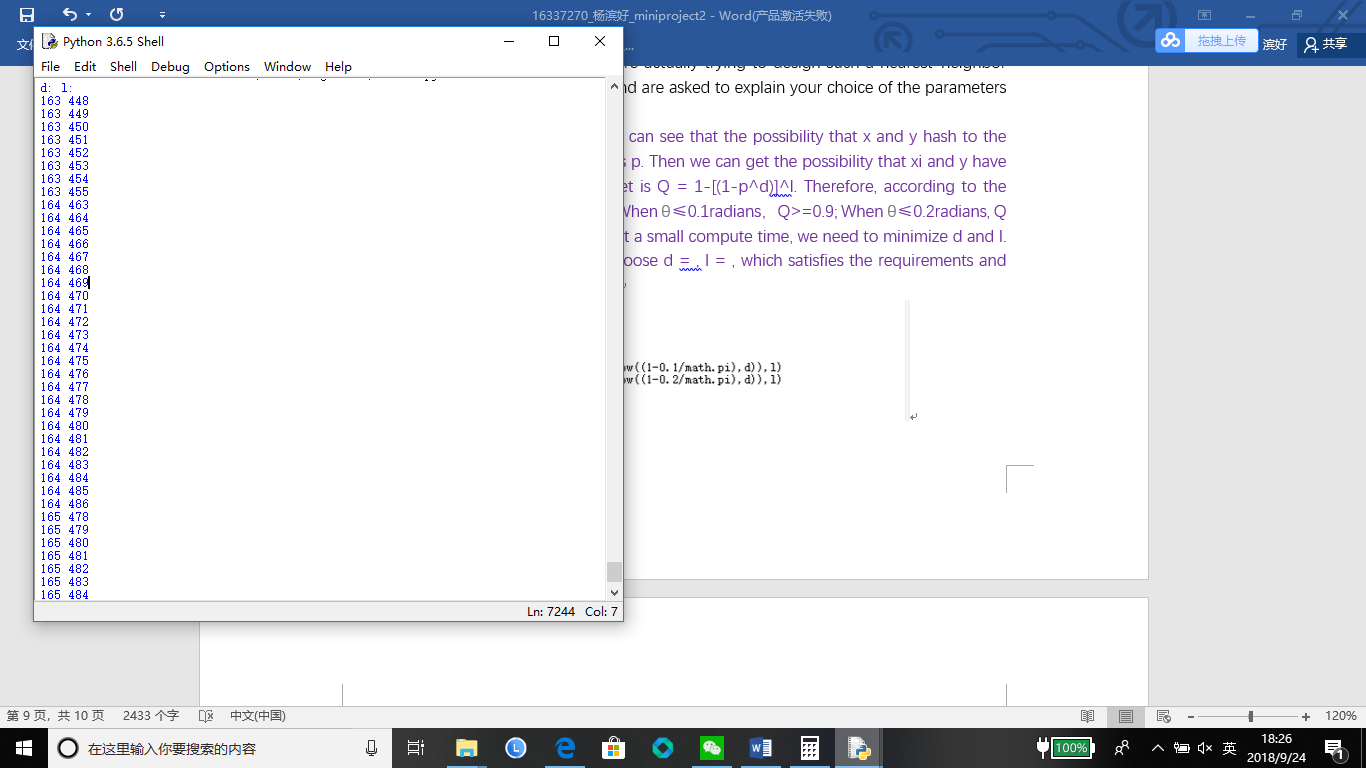
3) for a given datapoint, the total time to compute all ` hashes is relatively small.

Compute actual numeric values for d and l in the case that n = 1,000,000; if you like, in addition you may also give an answer as a function of n. Discuss any natural tradeoﬀs. [Hint: There is no single right answer here, and please reference your answers to the two previous parts. Imagine that you are actually trying to design such a nearest-neighbor search algorithm for a company, and are asked to explain your choice of the parameters to your supervisor.]

Solution: According to part (a), we can see that the possibility that x and y hash to the same value is (1-θ/π)^d, named as p. Then we can get the possibility that xi and y have the same value at least in a bucket is Q = 1-[(1-p^d)]^l. Therefore, according to the problem description, we can get：When θ≤0.1radians，Q>=0.9; When θ≤0.2radians, Q -> 0.For n = 1,000,000, if we expect a small compute time, we need to minimize d and l. There is no unique answer, so I choose d =163 , l =448 , which satisfies the requirements and may have a shorter compute time.

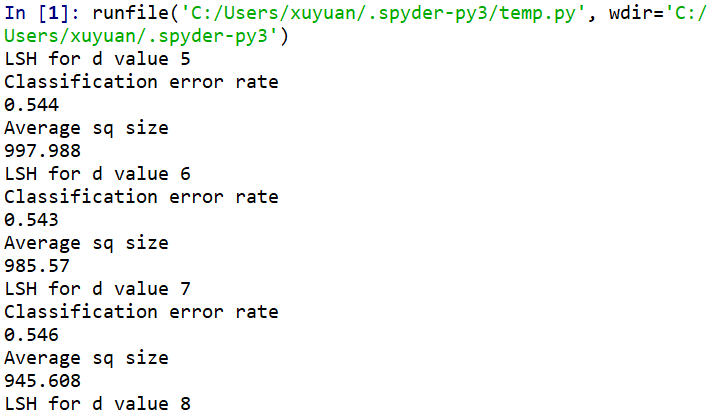
The computation and the result are as follows:

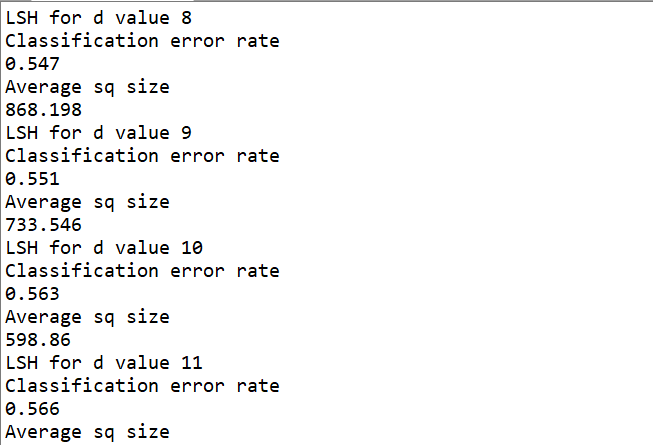


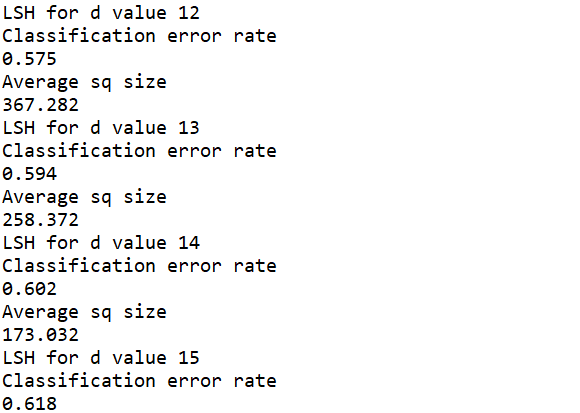


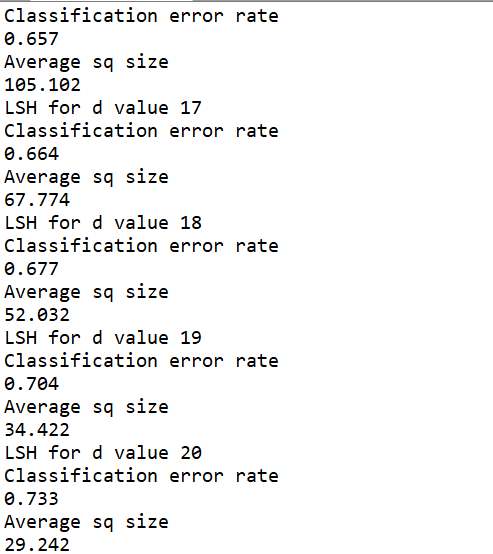
1. (6 points) Implement the Locality Sensitive Hashing scheme and corresponding “Classification” protocol described at the beginning of this problem, for the newsgroups dataset, with ` = 128 hash functions, and d = 5,6,...,20. For each value of d, compute the average classification error of the corresponding scheme, and compute the average size of the set Sq (averaged over the 1000 articles). Plot the average classification error versus the average number of articles inspected. Is there a “sweet spot” in terms of the tradeoﬀ between classification error and the average size of Sq (which in turn governs the running time of classification)?

**Answer: see code project2.py**

****

****

****

****

1. (4 points) Compare and contrast the performance properties the LSH-based nearest-neighbor classification system in part (d) with those of the dimension-reduction-based one in Problem 2. What properties of an application would suggest that the former would be better choice than the latter, or vice versa? Describe how you might combine the two approaches. For example, could dimensionality reduction help speed up the brute-force computation in the LSH classification system? When, if ever, might such a combination outperform both of the single-approach systems? Justify your answer.



Solution:

The LSH-based nearest-neighbor classification may have a lower classification error when the parameters are appropriate, in contrast, the compute time that the dimension-reduction-based method needs is relatively small. The runtime of LSH is so long ,especially when d becomes bigger. LSH is better when where are a lot of data points and dimension reduction is better when the dimensions is big.

If we combine the two approaches, we may consider that we can replace the brute-force computation with using the dimension-deduction method. When we compare the y with a xi, we can use the dimension-deduction first to lower the dimension so that we can reduce the compute time. Theoretically, it may outperform both of the single-approach systems.