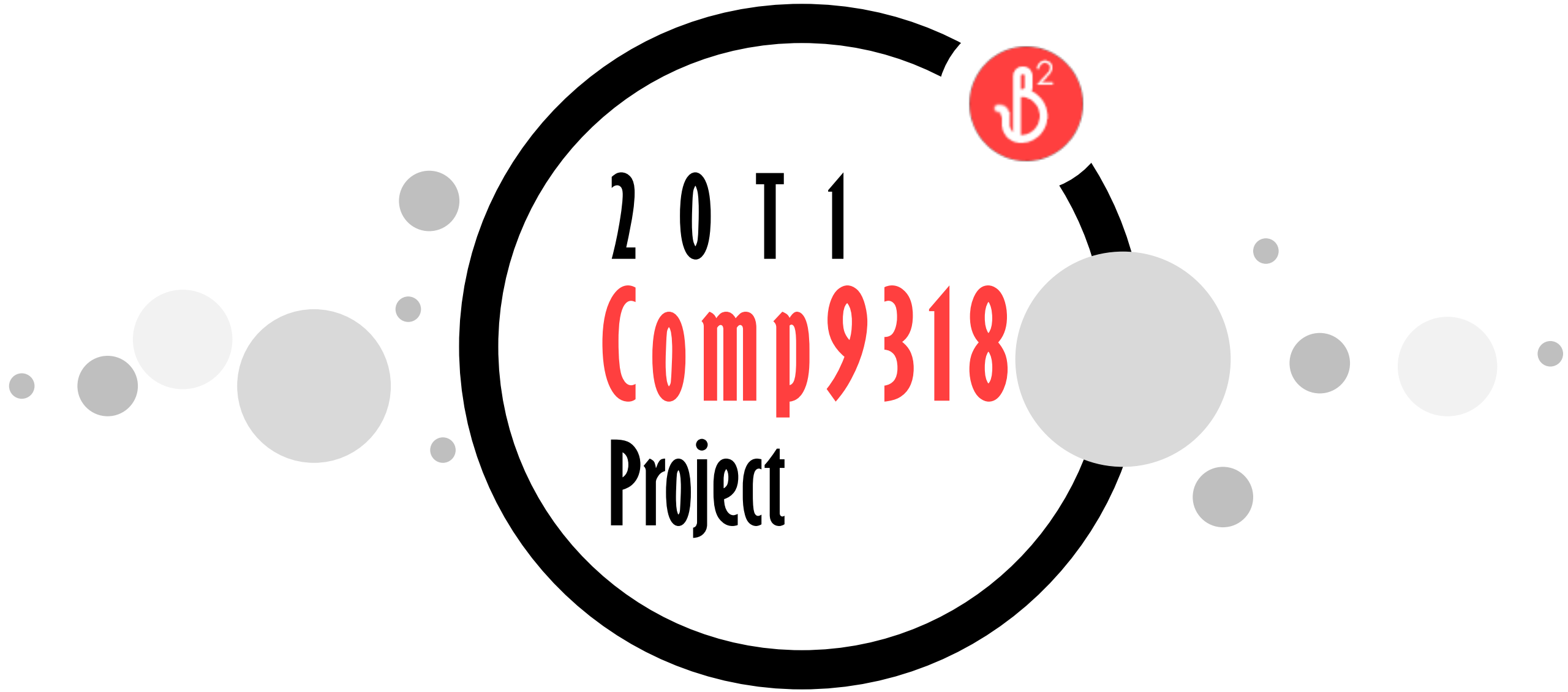

2 0 T 1
Comp9318
Project

Comp9318
Libraries

- Scipy 1.4.1
- Numpy 1.81.6
- Python 3.6

Comp9318
Part1

H. Jégou, M. Douze and C. Schmid, "Product Quantization for Nearest Neighbor Search," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 1, pp. 117-128, Jan. 2011.

- Idea:
  - Partition the dimension into m partitions
    - Accordingly a vector ➔ m subvectors
  - Use separate VQ with k codewords for each chunk
- Example:
  - 10-dim vector decomposed in m = 2 subvectors
    - $o^T = [o_1 : o_2]^T$
  - Each codebook has 4 codewords, denoted as $c_{i,j}$
  - Total space in bits:
    - data: m log(K)
    - codebook: m * ((D/m) * K)

7	2	-1	5	6	4	3	-5	-1	-7
-2	3	0	6	4	2	2	4	5	-8
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...

0	1	0	0
1	0	1	1
...	...	...	...
...	...	...	...
...	...	...	...

$C_{1,1}$	5.9	2.3	-2.7	3.9	6.1
$C_{1,2}$	-1.3	1.8	7.4	5.5	0.9
$C_{1,3}$	...	...	...	...	...
$C_{1,4}$	...	...	...	...	...

$C_{2,1}$	-0.1	3.5	1.4	9.6	5.5
$C_{2,2}$	...	...	...	...	...
$C_{2,3}$	...	...	...	...	...
$C_{2,4}$	...	...	...	...	...

Comp9318
Part1

In this question, you will implement the product quantization method with L_1 distance as the distance function. **Note** that due to the change of distance function, the PQ method introduced in the class no longer works. You need to work out how to adjust the method and make it work for L_1 distance. For example, the K-means clustering algorithm works for L_2 distance, you need to implement its L_1 variants (we denote it as K-means* in this project). You will also need to explain your adjustments in the report later.

Specifically, you are required to write a method `pq()` in the file `submission.py` that takes FOUR arguments as input:

1. **data** is an array with shape (N,M) and dtype='float32', where N is the number of vectors and M is the dimensionality.
2. **P** is the number of partitions/blocks the vector will be split into. Note that in the examples from the inverted multi index paper, P is set to 2. But in this project, you are required to implement a more general case where P can be any integer >= 2. You can assume that P is always divides M in this project.
3. **init_centroids** is an array with shape (P,K,M/P) and dtype='float32', which corresponds to the initial centroids for P blocks. For each block, K M/P-dim vectors are used as the initial centroids. **Note** that in this project, K is fixed to be 256.
4. **max_iter** is the maximum number of iterations of the K-means* clustering algorithm. **Note** that in this project, the stopping condition of K-means* clustering is that the algorithm has run for `max_iter` iterations.

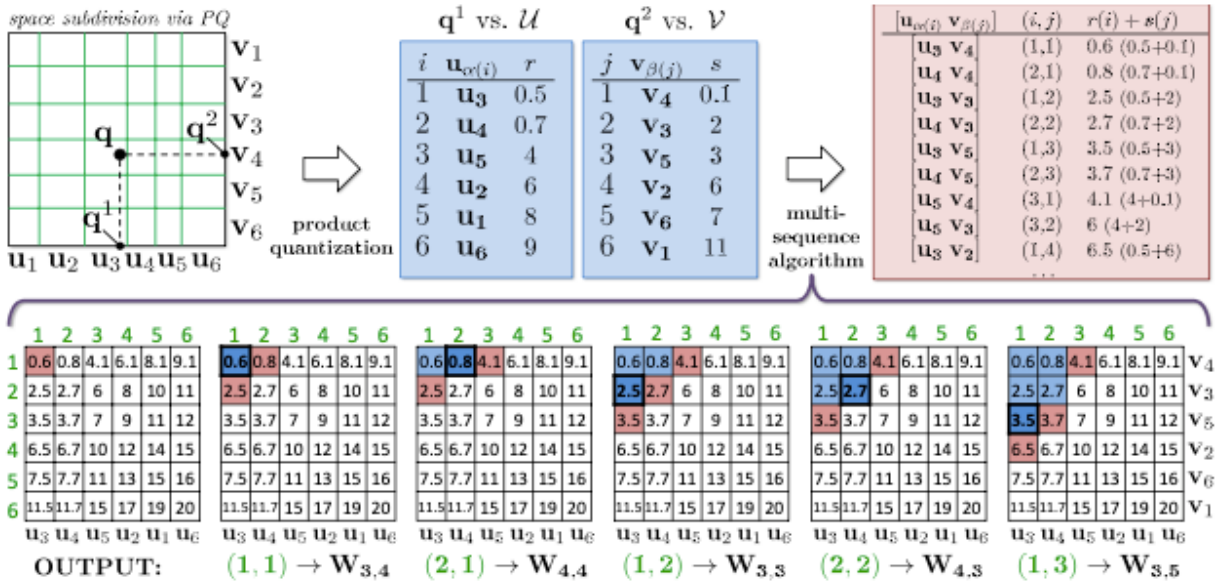
Comp9318
Part1

The `pq()` method returns a codebook and codes for the data vectors, where

- **codebooks** is an array with shape (P, K, M/P) and dtype='float32', which corresponds to the PQ codebooks for the inverted multi-index. E.g., there are P codebooks and each one has K M/P-dimensional codewords.
- **codes** is an array with shape (N, P) and dtype=='uint8', which corresponds to the codes for the data vectors. The dtype='uint8' is because K is fixed to be 256 thus the codes should integers between 0 and 255.

Comp9318
Part2

A. Babenko and V. Lempitsky, "The Inverted Multi-Index," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 6, pp. 1247-1260, 1 June 2015.



Algorithm 3.1: MULTI-SEQUENCE ALGORITHM()

INPUT : $r(:)$, $s(:)$ % *The two input sequences*
OUTPUT : out(:) % *The sequence of index pairs*
% *Initialization:*
out $\leftarrow \emptyset$
traversed (1:length(r), 1:length(s)) $\leftarrow$ **false**
pqueue $\leftarrow$ new PriorityQueue
pqueue.push ($(1,1)$, $r(1)+s(1)$)
% *Traversal:*
repeat
 $((i,j),d) \leftarrow$ pqueue.pop()
 traversed$(i,j) \leftarrow$ **true**
 out $\leftarrow$ out $\cup \{(i,j)\}$
 if $i <$ length(r) **and** (j=1 **or** traversed$(i+1,j-1)$)
  then pqueue.push ($(i+1,j)$, $r(i+1)+s(j)$)
 if $j <$ length(s) **and** (i=1 **or** traversed$(i-1,j+1)$)
  then pqueue.push ($(i,j+1)$, $r(i)+s(j+1)$)
until (enough traversed)

Comp9318
Part2

In this question, you will implement the query method using the idea of inverted multi-index with L_1 distance. Specifically, you are required to write a method `query()` in the file `submission.py` that takes arguments as input:

1. **queries** is an array with shape (Q, M) and dtype='float32', where Q is the number of query vectors and M is the dimensionality.
2. **codebooks** is an array with shape (P, K, M/P) and dtype='float32', which corresponds to the `codebooks` returned by `pq()` in part 1.
3. **codes** is an array with shape (N, P) and dtype=='uint8', which corresponds to the `codes` returned by `pq()` in part 1.
4. **T** is an integer which indicates the minimum number of returned candidates for each query.

The `query()` method returns an array contains the candidates for each query. Specifically, it returns

- **candidates** is a list with Q elements, where the i-th element is a **set** that contains at least T integers, corresponds to the id of the candidates of the i-th query. For example, assume $T = 10$, for some query we have already obtained 9 candidate points. Since $9 < T$, the algorithm continues. Assume the next retrieved cell contains 3 points, then the returned set will contain 12 points in total.

Comp9318
Hints

The implementation of `query()` should be efficiency. You should work out at least

1. How to efficiently extend Algorithm 3.1 in the paper to a general case with P > 2.
2. How to efficiently make use of `codes` returned by Part 1. For example, it may not be wise to enumerate all the possible combinations of codewords to build the inverted index.

We will test the efficiency by setting a running time limit (more details later).

Comp9318
Evaluation

Your implementation will be tested using 3 testcases (**30 points each, and another 10 points from the report**), your result will be compared with the result of the correct implementation. Part 1 and part 2 will be test **seperately** (e.g., you may still get 45 points from part 2 even if you do not attempt part 1), and you will get full mark for each part if the output of your implementation matches the expected output and 0 mark otherwise.

Note: One of these 3 testcases is the same as the one used in the **submssion system**.

We will provide immediate feedback on your submission **based on small sample testcases**. You can view the feedback using the online submission portal on the same day.
For **Final Evaluation** we will be using more different testcases, so your final scores **may vary** even you have passed the testcase.
You are allowed to have a limited number of Feedback Attempts **(15 Attempts for each Team)**, we will use your **LAST** submission for Final Evaluation.