Product quantization

Quantization-based method has been vastly adopted by vector database solutions. For Milvus, it prefer quantization-based approach to LSH for the concerns of accuracy. In the latest Alibaba AnalyticDB-V, the scholars proposed a method based on product quantization to address the .

In PQ algorithm, the product refers to the cartesian product, and the quantization refers to vector quantization. In PQ, the high-dimensional feature vector is compressed in such a manner that the PQ-encoded code can efficiently approximate the original vector, and the code can be stored in memory, thereby avoiding costly disk accesses.

The following codes demonstrate our proposed PQ-based attributes filtering.

|  |  |
| --- | --- |
| **Algorithm 1**: PQ\_offline\_train  Global constant variable: K = 256 /\*The number of codebook per cluster\*/  Input: vec /\*the list vectors used for training, assuming all vectors in the lists are of the same dimension \*/, M /\*The original vector will be divided into M subspace (sub-vector) \*/  Output: codeword | |
| 1. | Ds = vec.shape[1] / M /\*get the dimension of the vector\*/ |
| 2. | codeword = empty array of shape (M, K, Ds) |
| 3. | for each m in range(M) |
| 4. | vec\_sub = extract the m-th sub-vectors from vec |
| 5. | codeword[m] = kmeans\_clustering(vec\_sub, K) |

|  |  |
| --- | --- |
| **Algorithm 2**: PQ\_encode  Input: codeword /\*This is obtained from PQ\_offline\_train method\*/, vec  Output: pqcode | |
| 1. | M, K, Ds = the shape of the codeword |
| 2. | pqcode = empty array of shape (M, K) |
| 3. | for each m in range(M) |
| 4. | vec\_sub = extract the m-th sub-vectors from vec |
| 5. | pqcode[:, m] = the closest codeword index to each subvector in vec\_sub |

|  |  |
| --- | --- |
| **Algorithm 3**: PQ\_search (Basic)  Input: codeword /\*This is obtained from PQ\_offline\_train method\*/,  pqcode /\*This is obtained from PQ\_encode method \*/,  query /\*This is the query vector\*/  Output: dist | |
| 1. | M, K, Ds = the shape of the codeword |
| 2. | dist\_table = empty array of shape (M, 256) |
| 3. | for each m in range(M): |
| 4. | query\_sub = extract the m-th subvectors from the query |
| 5. | dist\_table[m, :] = compute\_distances(query\_sub, codeword[m]) |
| 6. | dist = aggregate the distances according to pqcode |

Here we demonstrate how to implement pre-query, post-query and concurrent query as Zhilin proposed. First, we have to train with a large amount of image datasets to obtain a fine codeword. For the pre-query, it will do attributes filtering first before encoding the features that meet the requirements, and finally, do basic search. For the post-query, however, it will do encoding and basic search, and then filter the search result. Based on the idea of concurrent filtering Zhilin eminently proposed, we therefore modify the basic search method as shown below.

|  |  |
| --- | --- |
| **Algorithm 4**: PQ\_concurrent\_search  Input: codeword /\*This is obtained from PQ\_offline\_train method\*/,  pqcode /\*This is obtained from PQ\_encode method \*/,  query /\*This is the query vector\*/  Output: dist | |
| 1. | M, K, Ds = the shape of the codeword |
| 2. | dist\_table = empty array of shape (M, 256) |
| 3. | for each m in range(M): |
| 4. | query\_sub = extract the m-th subvectors from the query |
| 5. | dist\_table[m, :] = compute\_distances(query\_sub, codeword[m]) |
| 6. | dist = aggregate the distances according to pqcode |
| 7. | if it fails to meet attribute filtering: |
| 8. | dist = |