Katholieke
Universiteit
Leuven

**Department of
Computer Science**

BIG DATA ANALYTICS PROGRAMMING

Product Quantization and Index Structures

Cas Coopman (r0996992)

Academic year 2023–2024

1 Product Quantization

To ensure a fair comparison, multithreading and multi-core parallelism are disabled on the device for the following experiments. For both implementations, the **CPU time, accuracy, and memory usage** are tracked.

First, PQ 1-NN is tested with varying parameters. Figures 1 and 2 illustrate which metrics are impact by the centroid and subvector parameters respectively.

Figure 1 shows that more centroids increases compression time because it slows down the kMeans fitting process. However, this longer compression time is correlated with higher accuracy. The lookup table size doubles when increasing the centroid count proportionally, but remains negligible. Figure 2 indicates that fewer subvectors significantly compresses the dataset. Compression reduces the prediction time at a cost of lower accuracy. On average, increasing the amount of subvectors from 4 to 98 halves the compression time (not plotted).

Secondly, also the classic scikit-learn 1-NN method is ran. It achieves an accuracy of 97% and took 11 seconds to predict, but used 359 MB of memory. To compare, PQ 1-NN can achieve 94% in 12 seconds with only 6MB, after compressing for 34 seconds.

The choice of preference boils down to the trade-off between high accuracy without compression time, and low prediction time with low memory usage. Approximated kNN is especially advantageous when the computed centroids can be reused often or in environments with large datasets and relatively low memory. The lower accuracy and exponential increasing compression time are the main drawbacks compared to standard kNN. When using PQ, it is paramount that the cost of compression and loss of accuracy are offset by sufficiently leveraging the reduced prediction time and decrease in memory requirements.

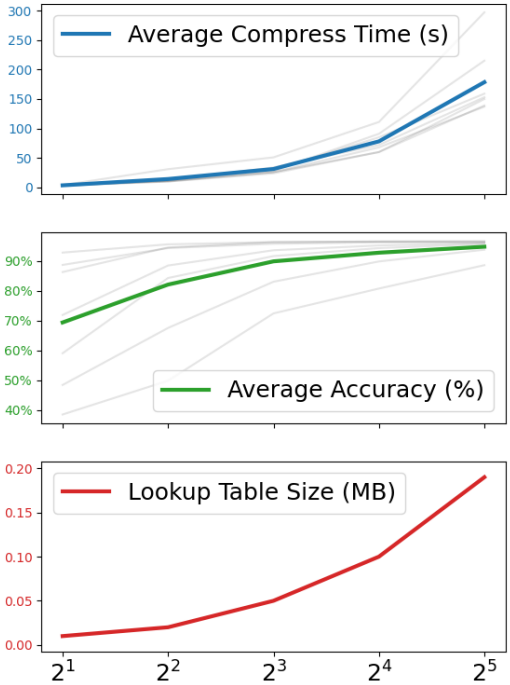

Figure 1: Impact of **centroid count**

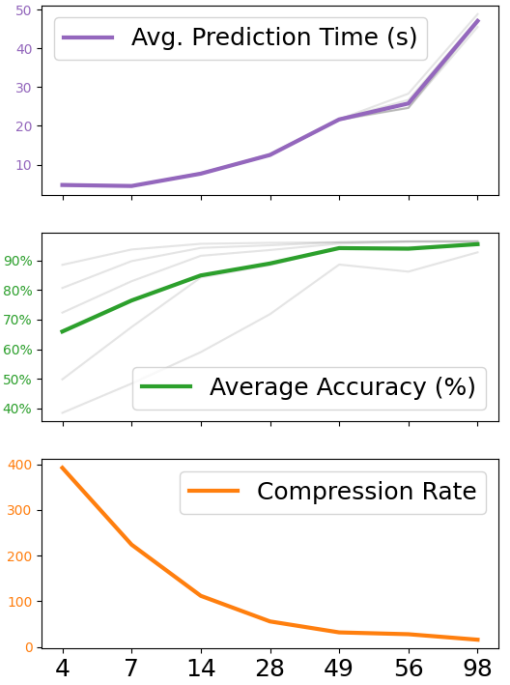

Figure 2: Impact of **subvectors count**

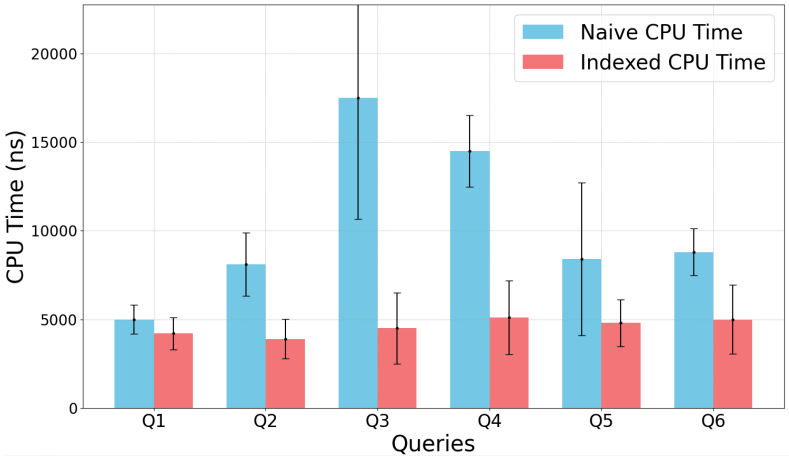

Figure 3: Comparison of CPU time with deviation between Naive and Indexed versions per Query

2 Index Structures

Before tracking the metrics, an initial 1000 pre-runs were performed. This step ensured that the bitmaps and bitslices were already created, effectively isolating the computational gains of this implementation from external factors. It should be noted, however, that in real-world scenarios, the creation of these indexes also requires setup time. Figure 3 presents the average CPU times over 10 runs, comparing the naive and indexed methods for each query. Standard deviations are also plotted on the bar chart. As illustrated, the **index-based approach results in significant time savings** for each query.

The first two queries exhibit the lowest CPU times. These fast results can be attributed to the pre-sorting effect achieved by building the bitmaps prior to the query. Query 2, which requires a count operation, demonstrates that counting with bits is computationally efficient. Query 3 serves as a prime example of the advantages of summing over bitslices. For each slice, the set bits are simply counted and multiplied once by a factor. In contrast, the naive implementation necessitates a summation calculation for every row in the dataset. However, it is important to note that bitslices convert numbers into integers, which can lead to rounding errors if the amplification and encoding lengths are not sufficiently large. To achieve the same precision as the naive method in Query 3, a bitslice with an encoding length of 16 and an amplification factor of 1000 is required. Queries 4 and 6 demonstrate that the CPU times for the indexed methods scale more favorably with increasing query complexity.

A similar trend is observed when measuring Wall time. The largest speedup occurs in Query 3, which sees a **4.55 times** reduction in average Wall time. On average, across all queries, the indexed methods achieve a **2.25-fold** speedup in CPU time. Another noteworthy observation is that the deviation in times is much lower, indicating more consistent operations. To fully understand the benefits and limitations of these techniques, additional experiments should be conducted on larger datasets and more complex queries. Moreover, the time required to construct the indexes should be tracked separately.