Master 1
IAD

Data science project report

SUMMARIZATION PROJECT

Tomi COTTRELLE - Wacim BELAHCEL - Assia BOURAÏ

Année universitaire : 2019 – 2020

Encadrant : Themis PALPANAS

Table of contents

1. Abstract

Many solutions have been proposed on the subject of data reduction and similarity search, which often works in pair, including Principal Component Analysis (PCA), Discrete Wavelet Transform (DWT), Piecewise Aggregate Approximation (PAA).

In this project report we introduce two well-known methods efficient on resolving the problems of data summarization and similarity search, namely, Product Quantization for data reduction, and Symbolic Aggregate approximation Index (iSAX) for exact similarity search.

To compare the efficiency of our summarization methodology, we also implemented three other summarization methods that will be briefly presented later.

2. Introduction

Recently a lot of interest has been given to the problematic of similarity search in time series data, principally due to its use in many big data application

When we talk about similarity search, we principally talk about 2 main classes, namely, K nearest neighbor search and range queries.

Given a time series X as a sequence of N values, this time series can be represented as a point in N dimensions, nearest neighbor search is the optimization problem of finding the point in a given set that is closest to a given point or in our case a given time series, The distance between the two time series is calculated using the Euclidean distance.

In our case we principally focused on the problem of K nearest neighbors where K equal to 1.

The problem with time series database is that they are often extremely large, and we can't afford brute force search (sequential scanning), to avoid this issues many solution have been proposed using different techniques that use indexes to speed up this process, the problem with that method is that most used indexes don't work well when the dimension of the data is higher than a certain thresh hold (12 – 16), so we need to reduce the data using a data reduction technique, which also reduce the final space needed to store our data.

Most recent works on similarity search using dimensionality reduction and special access methods (SAMs) follow the Generic Multimedia INdexIng method (GEMINI), which works by first reducing the time series dimension (with algorithm of choice), then inserting the reduced time series into a SAM.

Depending on the problem we are working on (data size reduction, fast approximate search, exact similarity search), different algorithm can be proposed.

For the case of exact similarity search; the data reduction algorithm has to follow special rules that will prevent false dismissals we will go more in detail on that subject on the similarity search section.

The rest of this report is presented as follow, first we describe the datasets used, the rest is divided in two parts : dimensionality reduction (for data size reduction / data visualization) and exact similarity search.

Each part is sub-divided as follow: we first present the problematic in detail; we then present the proposed solution and the different experiments setup.

We will finally conclude this report and provide our remarks and possible optimization.

3. Datasets

Two datasets were provided to us to work on it. Both of them contain 50,000 univariate time series. And each time series consists of 256 (IEEE 754) float numbers.

The first dataset is a synthetic one and the second is a seismic dataset.

The synthetic time series are generated from random walks.

The seismic dataset is sampled from real world. The provided seismic dataset has also been z-normalized as stated in the above procedure.

4. Dimensionality reduction

4.1. Problem definition

In this project, the first task is to summarize each time series of a dataset of 50,000 time series of 256 (float32) values into 32, 64, 128bytes, using a data reduction technique.

Dimensionality reduction is a way to efficiently extract patterns in data, a feature extraction method is applied to a time series data to decrease its dimension from a d-dimensional space to a k-dimensional subspace.

A reverse process can be done by using the extracted features to reconstruct the original time series, a good data reduction technique should minimize the reconstruction error between the original time series and the reconstructed time series, the reconstruction error is calculated using the Euclidean distance between the two.

$$E = \sqrt{\sum_{i=1}^{n} (x(i) - recx(i))^2}$$

The final error will be an average over all the 50k time series.

Moreover, in practice, a method which can encode the extracted features values of the time series using a lower encoding space (ex: from float32 to integer 8) will reduce the total size more efficiently.

4.2. Solution: Product Quantization

The proposed methodology is based on the encoder of the High-Dimensional Indexing for Efficient Approximate Nearest Neighbor Search (HDIdx) [3] paper, which is based on the work done on Product Quantization for Nearest Neighbor Search [5].

The basic idea behind the encoder is to compresses the input vectors into compact hash codes which requires little storage resource. Using a pre-trained encoder.

First, we define how the encoder encode / decode the data, we will then present how to train such an encoder.

3.2.1. Defining the encoder

The encoder that we use (Product quantizer) is composed of multiple Vector quantizers. Formally, a vector quantizer *Q* of dimension *n* and size *s* is a mapping Q: $R^n \rightarrow C$, from a point in n dimension (like a time series) to a set C of codewords c_i.

As stated in [6], in vector quantization (VQ) a codeword c_i is a vector which is used to represent points that are close to him (much like a word in SAX) forming a cluster of points, the number of possible codewords is predefined (and they will be trained using Kmean), the set of all possible codewords $C = \{c_1, c_2..., c_s\}$ is called a codebook, a codebook of size *s* can output s different representation, the final encoding of a point will be the index *i* of the codeword c_i in the codebook C.

for a given codebook, the optimal (closest) codeword index from a sequence is the index *i* of the codeword that satisfies (from [6]):

$$d(x, c_i) \leq d(x, c_j) ; \forall j$$

To encode (reduce the dimension of) a vector / time-series $X_n=x_1,x_2,\ldots,x_n$ of size *n* to m values, (from n dimensions to m), we sub-divide our sequence into *m* vectors, we then use *m* different codebooks of size *s* (m sets of *s* codewords) to encode each subdivided vector, the encoded sequence will be a sequence of indexes representing the closest codeword from each sub-sequence.

In the figure below, an example of reducing a vector of 8 dimensions into 4-dimensional vector using 4 codebooks:

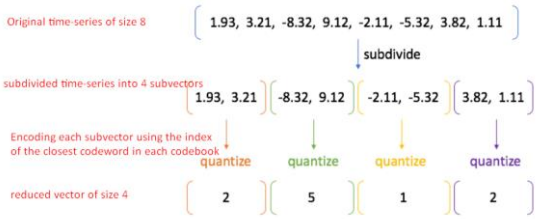


Figure 1 product quantization encoding example

The pseudo-code for the encoding:

Algorithm encoding(ts, codebooks) ;
Input:
ts: time-series of size n to reduce to size m.
codebooks: list of m codebooks composed of s codewords (vectors of centroids) used to encode each sub-vector of original time-series.
Output: reduced-ts: reduced-time-series of dimension m.

```
m = length(codebooks);
reduced-ts = zeros(m);    #initialize vector with m zeros
Sub-vectors = sub_divide(ts, m); # divide ts in a list of
                                 # m subvectors
For  index in range(sub-vectors):
        close_index =
        closest_codeword_index(sub-vectors[index],
         codebooks[index]);
        reduced-ts[index] = close_index;
end;
return reduced_ts;
```

To reconstruct our time-series from the reduced ts we simply append the codeword of each codebooks at the index specified by the reduced-ts :

Algorithm decoding(reduced-ts, codebooks) ;
Input:
Reduced-ts: time-series of size m to reconstruct to n.
codebooks: list of m codebooks composed of s codewords (vectors of centroids) used to decode each index.

```
Output: rec-ts: reconstructed- ts of size n.


m = length(codebooks);
rec-ts = [];
For  index in range(m):
        Cw_index = reduced-ts[index]
        rec-ts.append(codebooks|index][cw_index])
end;
return rec_ts;
```

Moreover, the fact that the encoded sequence is in encoded using indexes basically means that the values are encoded using integer (and not float numbers), which make even smaller in size compared to other methods for the same reduced dimension (for example pca).

3.2.2. Training Encoder:

Training our encoder basically mean generating *m* codebooks that will be used for encoding and decoding.

Each codebook is composed of a set of *s* codewords that we can view as centroids, the number of centroids is pre-defined.

to train the codebooks we first generate all their centroids with random values, we then train the centroids using an unsupervised clustering algorithm (like k mean clustering) on the dataset (or a subset of the dataset) that it will then encode.

Each codebook at partition *m* is trained on the sub-vectors of the dataset at position *m*.

The pseudo code of the training:

```
Algorithm codebooks_generation(dataset, m, s) ;
 Input:
 dataset: dataset matrix dimension w*n used for training, same dataset that will be reduced.
 m: the reduction dimension
 s: the number of centroids for each codebook.
 Output: codebooks: the codebooks that will be used for encoding.

codebooks = list of size m
 Sub-vectors = sub-divide(dataset,m); #subdivide dataset into m*w*l
                                      #matrice (m*w subvectors)
 For  index in range(m):
        codebook = kmean(sub_vectors[m],
                         num_centroids=s);
        codebooks[m] = codebook;
 end;
 return codebooks;
```

4.3. Experiments

To compare the efficiency of our methodology we compare our results to three other techniques, namely PCA, PAA and DWT, the full code of each implemented methodology will also be shared.

3.2.3. Parameters

As mentioned before, when defining a PQ encoder, we have to predefine the number *m* and size *s* of its codebooks, the total number of centroids to train will be *m*s*, we train a different encoder with different settings for each reduced size (32 bytes, 64 bytes, 128 bytes), below a table that summarize the initial parameters:

	codebooks	Centroids per codebook	Total number of centroids
Encoder_32B	32	128	4096
Encoder_64B	64	64	4096
Encoder_128B	128	32	4096

Figure 2 PQ encoders parameters

Two things influenced the choice of parameters:

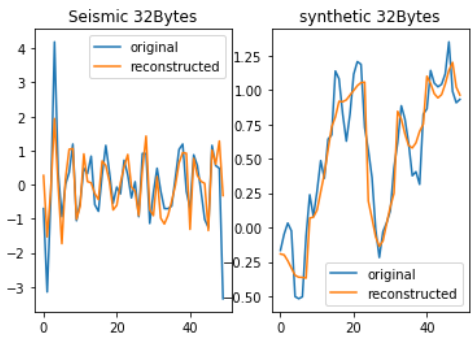
- The number of codebooks depend on the reduced time series size, since each reduced time-series of size *m* (same as the number of codebooks) is composed of a sequence of *m* integer (codeword index, each integer encoded on 1 byte), this basically mean that :

$$m \leq time\_series\_size$$

- The number of centroids per codebook only depend on the hardware used, since more centroids will take longer to train with kmeans, we limited the total number of centroids to 4096, theoretically we could add more centroids, but this would make training much slower.

3.2.4. Results

In the graphs below, we can see the comparison between the original and reconstructed time series for the two datasets, we can clearly see that it is much harder for the reconstructed time series to remains close to the original in the seismic dataset case.



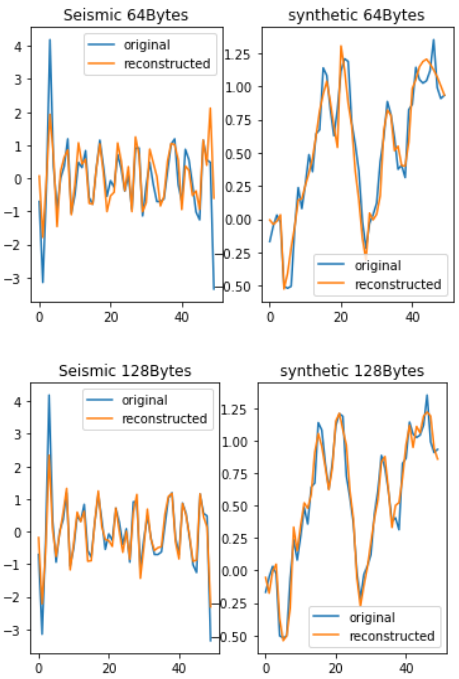


Figure 3 PQ Encoders recustruction results

This is due to the fact that the PQ methodology only replace each sub-vector by the closest centroid of each codebook when reconstructing, with a dataset with higher variance, it become harder for a centroid (which basically is the mean of its cluster) to generalize the sub-vector around it, which makes the average distance between them (the variance) higher.

Nonetheless, as we can see it in the tables below, this methodology gives better results than others reduction techniques on all the experiments setups:

	Synthetic			Seismic		
Reduced size	32	64	128	32	64	128
PAA	6.79	4.88	3.47	15.97	15.94	15.80
PCA	5.46	3.95	2.80	15.46	14.92	13.83
DWT	6.79	4.88	3.47	15.97	15.94	15.80
PQ encoder	**2.35**	**1.91**	**1.56**	**7.9**	**5.99**	**3.79**

Figure 4 comparison of different reduction methods (metric : MSE)

This is due to the many benefits that this methodology has:

- The Fact that the reduced data use 1-byte integer values make it possible to use higher reduced dimension than other methods, which means that more features can be kept.
- Compared to non-data adaptive methods (Wavelet, PAA), PQ encoders adapts to the data and can keep relevant information on the dataset to "generalize" the information.
- Compared to PCA, PQ encoder can keep more information on the time domain.

5. Exact Similarity Search

5.1. Problem formulation

For the second part of the project, we were asked to build a similarity search algorithm that uses dimensionality reduction to speed up the exact similarity search process, the efficiency of the final methodology will be calculated using the pruning ratio.

The pruning ratio is the fraction of skipped time series (time series we did not calculate the Euclidian distance with the query) in the whole dataset, a larger pruning ratio implies that the designed summarization is more efficient.

A crucial criterion for a reduction algorithm used in exact similarity search is to guarantee no false dismissals.

False dismissals, also called false negative, means that true values (in our case the most similar time series) won't be detected by our algorithm, to ensure that we avoid false dismissal the distance measure in the reduced space must satisfy the following condition:

$$D_{reduced}(A,B) \leq D_{original}(A,B)$$

We call this theorem the lower bounding lemma, given the lower bounding lemma, a GEMINI indexing method requires:

- Establishing a distance metric for the original time series (in our case Euclidian distance).
- Producing a dimensionality reduction technique (in our case PAA then ISAX).
- Producing a distance measure on the reduced space that respect the bounding lemma.
- Producing an index that can handle our reduced space efficiently.

The algorithm we decided to implement for this part is called the Symbolic Aggregate approximation Index (iSAX) [7], it is one of the most popular algorithms for time series fast exact similarity search, it also follows the GEMINI methodology requirements.

5.2. Solution ISAX

4.2.1. PAA and SAX detailed explanation

The *Piecewise Aggregate Approximation* (PAA) [4] is a method which allows to represent a time series with *n* values by a step function with *w* segments. Basically, the PAA allows to get a representation with reduced dimensionality using segments averages. The length of segment *l* (with *l* = *n*/*w*).

Each segment is represented by its mean *m*. More precisely, with the original time series $T = [t_1, t_2, \ldots, t_n]$ and the reduce time series $\bar{T} = [\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_w]$ each i^{th} element of T is computed by using the following formula:

$$\bar{t}_i = \frac{1}{m} \sum_{j}^{m*i} t_j \; with \; i = 1, \ldots, w \; and \; j = m * (i-1) + 1$$

This method has O(n) of complexity. In the following figure we see the reduced dimensionality (from 16 to 4) with PAA representation.

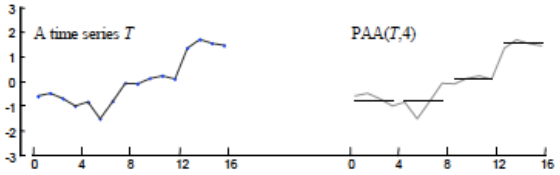


Figure 5 Paa reduction example

The *Symbolic Aggregate approXimation* (SAX) [1] is a method based on PAA representation. SAX split PAA representation into a small alphabet of symbols (the size of alphabet named cardinality). Breakpoints are placed on a virtual x-axis in parallel of PAA representation x-axis. Each zone between breakpoints are labelled in function of the location of PAA value inside the labelled region.

The next figure allows to visualize the idea.

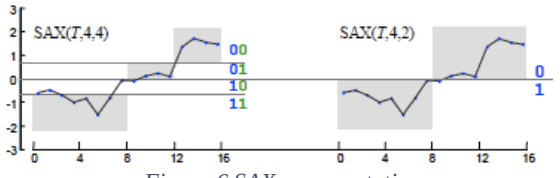


Figure 6 SAX representation

A SAX word is a vector of symbols. We will use this notation:

$$SAX(T, w, a) = T^a = \{t_1, t_2, \ldots, t_w\}$$

where *T* is original time series, *w* is number of symbol and *a* is the cardinality.

The binary numbers used in SAX representation allows to do several things. The first property allows to reduce cardinality easily. For example to go from SAX(T,4,4)= T4 = {11,11,01,00} to SAX(T,4,2) = T2 = {1,1,0,0} we just ignore the last bits of each symbols.

When we have two SAX representation, we can compute the lower bounding (an approximation of the classical Euclidean distance) with the following formula:

$$MINDIST(T^2, S^2) = \sqrt{\frac{n}{w}} * \sqrt{\sum_{i=1}^{w} (dist(t_i, s_i))^2}$$

The SAX method is very competitive with other methods like PAA or DWT, when we compare the three methods with the same coefficients, SAX representation is faster.

4.2.2. ISAX representation

The *indexable Symbolic Aggregate approXimation* (iSAX) [7] is based on SAX representation and allows to build an index on a dataset. The main advantage of iSAX is that we can compare words with different cardinalities. It also allows different cardinalities in a single word.

We will use the following notation:

$$iSAX(T, 4, 8) = T^8 = \{6^8, 6^8, 3^8, 0^8\}$$

For example, with two time series T and S as well as their iSAX words:

$$iSAX(T, 4, 8) = T^8 = \{110, 110, 011, 000\}$$
$$= \{6^8, 6^8, 3^8, 0^8\}$$
$$iSAX(S, 4, 2) = S^2 = \{0, 0, 1, 1\} = \{0^2, 0^2, 1^2, 1^2\}$$

Before giving the iSAX words to the MINDIST function we need to "promote" the lower cardinality (in this case S2). The next figure is a pseudo-code which allows to do so:

```
IF S^k_i forms a prefix for T^8_i THEN,
        *_i = T^8_i for all unknown bits.
ELSE IF S^k_i is lexicographically smaller than
corresponding bits in T^8_i THEN,
        *_i = 1 for all unknown bits.
ELSE,
        *_i = 0 for all unknown bits.
```

To find the "missing bits" of S8 we examine the known bits of S8 for each "letter" in the word and its corresponding bits in Ti8. With this function we can find an admissible representation of S8:

S^8= {011,011,100,100}

Finally, it's possible to compare iSAX words with different cardinalities such as {111,11,101,0} = {78, 34, 58, 02}.

.

4.2.3. ISAX Index

The use of iSAX representation allows for the creation of index structure.

The index is constructed given

- The base cardinality b,
- The word length w or number of segments
- The threshold th, that represents the maximum number of time series indexed in a node.

The index structure is implemented using a tree, where each node represents a subset of the SAX space. A node's representative SAX space is congruent with an iSAX word and evaluation between nodes or time series is done through comparison of iSAX words.

The three classes of nodes found in a tree and their respective functionality are described below:

Terminal Node: A terminal node is a leaf node which contains a pointer to a list with all time series. Each time series in this list is characterized by the terminal node's representative iSAX word and its cardinality.

Internal Node: An internal node designates a split in SAX space and is created when the number of time series contained by a terminal node exceeds *th*.

It contains the list of all children and their cardinalities, the length of segments and the node's representative iSAX word and its cardinality.

The internal node splits the SAX space by promotion of cardinal values along one or more dimensions as per the iterative doubling policy.

Time series from the terminal node which triggered the split are inserted into the newly created internal node and hashed to their respective locations. If the hash does not contain a matching iSAX entry, a new terminal node is created prior to insertion, and the hash is updated accordingly.

Root Node: The root node is representative of the complete SAX space (it contains the list of all his children) and is similar in functionality to an internal node.

The root node evaluates time series at base cardinality.

Pseudo-code of the insert function used for index construction is shown below. Given a time series to insert, we first obtain the iSAX word representation. If the hash table does not yet contain an entry for the word iSAX, a terminal node is created to represent the SAX space and the time series is inserted into it.

If this node is an internal node, the insert function is called recursively. If the node is a terminal one, we determine whether a division takes place. If this is the case, a new internal node is created and all the entries of the overloaded terminal node are inserted there. Otherwise, if there is enough space, then the entry is simply added to the terminal node.

```
Function Insert (ts)
iSAX_word = iSAX (ts,this.parameters)

if Hash.ContainsKey (iSAX_word)
   node = Hash.ReturnNode (iSAX_word)
   if   node is terminal
        if  SplitNode () == false
            node.Insert (ts)
        else
              newnode = new internal
              newnode.Insert (ts)
              foreach ts in node
                        newnode.Insert (ts)
              end
              Hash.Remove (iSAX_word, node)
              Hash.Add (iSAX_word, newnode)
        endif
   elseif node is internal
          node.Insert (ts)
   endif
 else
         newnode = new terminal
         newnode.Insert(ts)
         Hash.Add (iSAX_word, newnode)
 endif
```

If there is an entry in the hash table then the corresponding node is retrieved.

5.3. Approximate search and exact search using ISAX

The iSAX model supports both approximate and exact similarity search.

4.3.1. Approximative search

The approximation method is derived from the intuition that two similar time series are often represented by the same word iSAX. The approximate result is therefore obtained by trying to find a terminal node in the index with the same iSAX representation as the query.

More precisely, this is done by browsing the index tree from the root to the most promising leaf node and matching the iSAX representations to each internal node. If such a terminal node exists, it is identified and then all the distances between the query and each time series in the leaf are computed, and finally, the time series with the smallest distance, the Best So Far distance (BFS), is returned.

In the case where a corresponding terminal node does not exist, the nodes whose last divided dimension has an iSAX value corresponding with the time series of the request is selected. If no such node exists at a given junction, we simply select the first one and continue the descent in the tree structure.

4.3.2. Exact search

The algorithm begins by obtaining an approximate best-so-far (BSF) distance, using approximate search. Once a BSF is obtained, it is used to prune the rest of the index leaves. The leaves that cannot be pruned are examined and the BSF distance is updated if needed. At the end of this process we get exact answer.

5.4. Experiments

4.4.1. Parameters and queries

Two major parameters can be tuned on isax that will make the tree building phase and / or the research different, namely, the number of segments in PAA and the base cardinality.

To keep it simple we chose an 8 PAA segments (which correspond to a 32bytes reduction) and a base cardinality of 2 values.

We had two different queries dataset to work on, one for the seismic dataset and one for the synthetic, and there is some key difference in how those two queries compares to each of their dataset.

The seismic queries dataset are pretty similar to the original dataset.

The synthetic queries dataset is completely different from the original synthetic dataset (a random walk).

4.4.2. Results

Below a small table comparing the results on the two queries datasets on their datasets:

	Pruning ratio
Seismic queries	0.98
Synthetic queries	0
Both	0.49

Figure 7 similarity search results (metric: pruning ratio)

We can see a big different between the two datasets, on the seismic queries we have a 98% pruning ratio which extremely good, but on the synthetic queries we have 0%, which mean that isax was pretty much useless on this experiment.

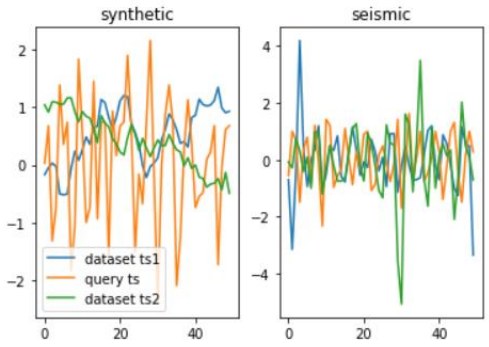


Figure 8 queries data compared to their original dataset

the principal reason for the poor result on the synthetic queries is that the time series used as a query where completely different from the dataset itself (the queries seems to be from real data much like the seismic dataset), as a result, the MINDIST _PAA_ISAX function gave really small distance results when used on an ISAX tree node, but when the exact search algorithm find a BSF time series, the actual distance between the query and this BSF is much further compared to the result of MINDIST_PAA_ISAX computed earlier, as a result, Exact search has to check all the nodes because they are all closer from the PAA segment then the BSF results.

6. Conclusion

On this project, we tried to tackle the problematic of data reduction and similarity search on time series.

Even though the PQ encoder was pretty efficient In our experiment, the fact that it has to compute centroids before reduction makes it impossible to generalize, moreover, computing centroids for larger dataset will become even harder, other methods using deep learning models like auto encoders could be a better alternative in those situations.

Even though ISAX speed up the process of similarity search dramatically using an index, building the index can take a lot of time and is exponentially slower the bigger the dataset, other implementation where proposed to resolve this issue (iSAX 2.0 [8]).

7. References

[1] Jessica Lin , Li Da Wei, Experiencing SAX: A Novel Symbolic Representation of Time Series, Data Mining and Knowledge Discovery [August 2007]

[2] Franky Chan, Ada W. Fu, Haar Wavelets for Efficient Similarity Search of Time-Series: With and 2Without Time Warping, Transactions on Knowledge and Data Engineering [June 2003]

[3] Ji Wan, Sheng Tang, Yongdong Zhang Pengcheng Wu, Steven C.H. Hoi, HDIdx: High-Dimensional Indexing for Efficient Approximate Nearest Neighbor Search

[4] Eamonn Keogh Kaushik Chakrabarti Michael Pazzani Sharad Mehrotra, Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases

[5] Hervé Jégou, Matthijs Douze, Cordelia Schmid, Product Quantization for Nearest Neighbor Search

[6] Qiang Wang and Vasileios Megalooikonomou, A Dimensionality Reduction Technique for Efficient Time Series Similarity Analysis

[7] Jin Shieh Eamonn Keogh, iSAX: Indexing and Mining Terabyte Sized Time Series.

[8] Alessandro Camerra Themis Palpanas Jin Shieh Eamonn Keogh, iSAX 2.0: Indexing and Mining One Billion Time Series.

[9] Bahri, A., Naïja, Y., Jomier, G. and Manouvrier, M., n.d. Recherche Par Similarité De Séquences Temporelles Dans Les Bases De Données : Un État De L'art. [ebook]