SAX Acquaintance Report

Denys Sobchyshak

September 17, 2016

1 Introduction

Symbolic Aggregate approXimation (SAX) is a data reduction technique that not only allows for a drastic decrease in memory use of time series storage, but also enables fast approximate and exact searches in huge amounts of data. This report provides a short description of findings and insights obtained during my acquaintance with SAX [10] and its further developments in indexable SAX (iSAX) [12] and iSAX 2.0 [2].

1.1 Implementation details

Due to a proof of concept (POC) nature of this work it was decided to use Python as the implementation language, since it facilitates fast solution delivery and incorporates a vast diversity of scientific libraries. The code is also heavily dominated by functional programming style of implementation, since not only such approach is more convenient for handling mathematical concepts, but it also allows more flexibility in terms of further parallelization of the solution.
It must be noted, that provided code is far from being optimal due to its POC spirit. Thus, occasional bugs and memory use inefficiencies might be discovered, especially in the graphical interface (GUI) part. However, this should not prevent an informed user from utilizing the solution in its fullest capacity on a modern commodity hardware installation.
The following is a list of software installations and libraries, required for the solution to run:

- python **3.4.3**
- matplotlib **1.5.2**
- numpy **1.11.1**
- scipy **0.18.0**

1.2 Definitions

In the further parts of this work we will refer to SAX as a data reduction technique, however, in some works (e.g. [8], [10], [12], etc.) SAX may be referred to as a dimensionality reduction technique. As implied in [13], I'd like to set a clear boundary between the terms and argue that in a general problem setting SAX is not a dimensionality reduction technique.

Specifically, for $n, m, k, l \in \mathbb{N}$ let's define a transformation that receives a $n \times m$ dimensional set with n data points each defined in $\mathbb{R}^m$ and returns a $n \times k$ dataset with $k < m$ and each of n data points defined in $\mathbb{R}^k$ to be a dimensionality reduction technique. Please note, that while according to our definition domains are defined in $\mathbb{R}$, some techniques can use a character space for the domains and we do not wish to exclude those techniques. Given that definition, we will define data or numerosity reduction technique as a transformation that receives $n \times m$ dimensional set with n data points each defined in $\mathbb{R}^m$ and returns a $l \times m$ set, where $l < n$ and the domain of each of m dimensions is arbitrarily defined.

Thus, simply put, a data reduction technique provides a different and compressed representation of the original data, yet retains the number of dimensions of the original feature space, while a dimensionality reduction technique tries to find a linear or non-linear combination in the original feature space and reduces that space to only the most important components. However, it can be argued that for a specific problem setting (e.g. nearest neighbor search) SAX can be considered a dimensionality reduction technique.

2 SAX Representation

To start with, it was needed to provide a SAX representation for a synthetic time series. Thus, it was needed to generate a time series, such that it would not require further preprocessing, or simply put, it would not violate any of SAX assumptions. According to [12] the only assumption SAX makes is that the underlying data has a local Gaussian distribution (i.e. on small intervals), which SAX author claims to be present in over 120 studied data sets from various domains [7]. Moreover, not only the author claims, that the distribution adaptive implementation of SAX which he experimented with would always approximately descend to a Gaussian distribution, but also, even if the underlying data has a different distribution, the SAX approach would still provide valid results only with a degraded performance.

Given the aforementioned, a simple random walk with Gaussian distribution for random number generator would provide the needed results and was implemented as

```
import numpy as np
def random_walk(t=1000):
    if t > 0:
        return np.cumsum(np.random.randn(t))
```

Furthermore, while SAX algorithm itself is pretty straightforward, the PAA [8] calculation for a non-integer size of the window span turned out to be a bit tricky. It was eventually achieved by scaling the entire domain using the PAA's word length as

```python
import numpy as np
def paa(series, w):
    n = len(series)
    series = np.array(series)
    if n == w:
        return series
    if w == 1:
        return [series.mean()]
    if w < 1 or n < w:
        return None
    aggregate = [0]*w
    idx = np.arange(n*w) // w
    for i in range(0, n*w, n):
        aggregate[i//n] = (series[idx[i:i+n]]).sum() / n
    return aggregate
```

To showcase the results we will transform two series

$$x1 = [0.98575863, 0.60495287, 0.83123062, -1.34965375, -1.07228837]$$

$$x2 = [-1.24234567, -0.79303633, -0.06255499, 0.49080739, 1.60712961]$$

into their corresponding SAX representations with PAA word length of 4 and SAX cardinality of 5

$$sax1 = ['100',' 100',' 010',' 001']$$

$$sax2 = ['001',' 010',' 100',' 100']$$

and measure the distance between them while comparing it to the Euclidean between the original series

$$eucl = 4.276009034552973$$

$$mindist = 1.965338463812398$$

which is clearly smaller, than the Euclidean, thus providing us with a lower bounding estimate of the true distance.

3 Alternative Representation

While reading about dimensionality reduction of time series I've stumbled across a novel technique called forecastability component analysis (ForeCA) [5] [6], which seem to heavily rely on Fourier transforms for spectrum analysis of a

series. It's content is rather involving and math intensive, thus was deemed to be out of scope for the purposes of this work. However, as mentioned in [12] and [4], the use of discrete Fourier transforms (DFT) for data reduction and sequence matching have been quite successful and also provides the desirable lower bounding property. Thus it was decided to investigate the method and implement it accounting for considerations noted in [4].
The results were rather interesting and did not follow the recommendations in [4] to use only 2-3 frequency components for the aforementioned synthetic time series, but rather required around 100 components for a data set of 2000 points in order to gain reasonable time series reconstruction. However, given that only the frequency components have to be stored, it is still a great reduction from initial time series size. A showcase of DFT reconstruction as compared to the original data can be seen on Figure 1.

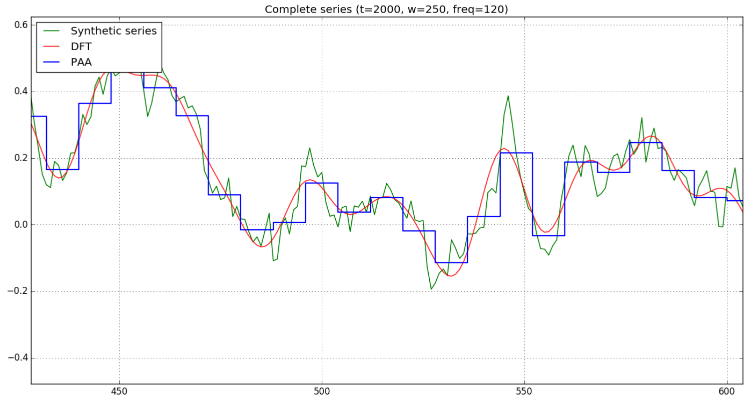

Figure 1: Comparison of DFT reconstructed and original synthetic time series, where t is the series length, w is the PAA word length and $freq$ is the number of DFT components used

4 (Optional) Parallelization

As parallelization of SAX within a computer cluster can not be considered a trivial task, I did not tangle with it in great depth. However, in general such task would not only require a proper infrastructure set up, but also is highly dependent on the underlying hardware specifications, cluster interconnects, etc. For our problem setting of not having enough memory (i.e. RAM) to process the entire data set at once and given that time series is stored on a network attached storage (NAS) or alike external location and each node within a cluster has access to it, one can provide a general description of how this issue can be tackled. Specifically, for a POC style solution I would use an MPI implementation, such as MPICH [1], paired with it's python binding (e.g. [3]) to read different parts of time series and process it in parallel. The processing itself would require nodes to share following data: series length, PAA word length, SAX cardinality, normalization data such as slice mean and variance, potentially boundary values of the series for proper PAA calculation and the resulting SAX part for final output (or the reduction in MapReduce context).

5 Interactive visualization

For interactive exploration of influence of various parameters on the resulting output a matplotlib based GUI was implemented. It comprises four visualizations with several parameter control sliders as explained in Figure 2. In order to interact with the visualization one will need to click on different parts of sliders. Please note, that changing the time series length will result in generation of a new random walk, while change of other parameters only adapts the corresponding transformations.

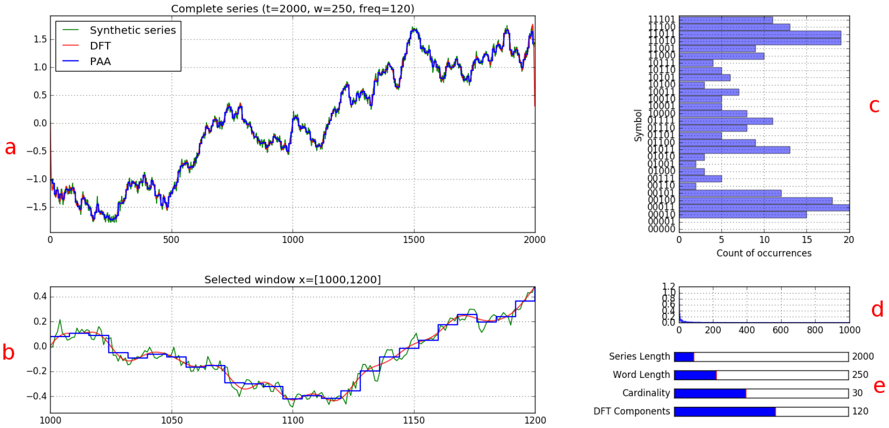

Figure 2: Interactive user interface showcase with next components: a) provides a complete view on the synthetic time series and its transformations and allows selection of a smaller window to be displayed in b b) zoomed in view of the selected window in a c) distribution of SAX symbols in the transformed representation d) values of DFT frequency coefficients e) sliders that control series length, word length of a PAA transform, SAX cardinality and DFT frequency components used for reconstruction

6 Conclusions

As I got familiar with SAX and basics of iSAX, the idea behind the technique seems to provide a great deal of performance gain and decrease in memory usage for time series storage. Thus, any task that requires sub-series matching or similarity search should greatly profit form this technique. However, SAX might be less efficient in streaming or online analysis settings, where the entire data set is not available even on separate nodes of the cluster. This comes from a requirement to normalize the data values into a $\mathcal{N}(0, 1)$ distribution, i.e. to scale it to mean 0 and standard deviation of 1. Thus a further investigation of topics such as [9] and adoption of SAX for algorithms, such as stochastic gradient descent, is absolutely required.

Moreover, as my focus is mainly on predictive analytics, I find it necessary to continue investigating case studies of SAX applications as in [14] and [11]. Naturally, earlier mentioned ForeCA technique is also caught up in the crosshairs.

References

[1] Mpich website. https://www.mpich.org/.

[2] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. isax 2.0: Indexing and mining one billion time series. In *2010 IEEE International Conference on Data Mining*, pages 58–67. IEEE, 2010.

[3] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

[4] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, 1994.

[5] Georg M Goerg. Forecastable component analysis. In *ICML (2)*, pages 64–72, 2013.

[6] Georg M Goerg. Forecastable component analysis talk. http://techtalks.tv/talks/forecastable-component-analysis/58229/, June 2013.

[7] Eamonn Keogh. Saxually explicit images: Data mining large shape databases talk. https://www.youtube.com/watch?v=vzPgHF7gcUQ, May 2006.

[8] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

[9] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms.

In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

[10] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[11] Ian Morgan and Honghai Liu. Predicting future states with-dimensional markov chains for fault diagnosis. *IEEE Transactions on Industrial Electronics*, 56(5):1774–1781, 2009.

[12] Jin Shieh and Eamonn Keogh. isax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

[13] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006.

[14] Indrė Žliobaitė, Jorn Bakker, and Mykola Pechenizkiy. Beating the baseline prediction in food sales: How intelligent an intelligent predictor is? *Expert Systems with Applications*, 39(1):806–815, 2012.