USING LOCAL SENSITIVE HASING TO VALIDATE GENERATIVE NEURAL NETWORKS

Adrien Cortes
Mines Saint-Etienne, ISFA Lyon
`adrien.cortes@etu.emse.fr`

September 1, 2023

ABSTRACT

In this article, we present a novel method for validating generative models of temporal series. Our approach involves representing the data using a combination of a CUMSUM filter and a Piecewise Aggregate Approximation (PAA), which allows us to capture the different shapes of the data while retaining its essential features. We then use local sensitive hashing to identify all the recurring patterns in the data, and to describe them visually along with their probability of occurrence. By comparing the patterns found in the generative data with those in the original data, we can propose a metric for evaluating the generative model's performance. We demonstrate the effectiveness of our method using a variationnal autoencoder and a dataset of random walks. Our approach has the potential to be useful in a variety of applications, such as time-series analysis, data compression, and data mining.

1 Introduction

Generative models like GANs [1] and VAEs [2] have a wide range of applications, and evaluating them is challenging due to the difficulty of comparing models. The literature commonly uses three criteria to evaluate generative models: *log-likelihood*, *Parzen window estimates* [3], and *visual fidelity of samples*. [4] showed that these criteria are independent of each other, which motivates a reflection on which criteria to choose to evaluate a model.

Some works, such as [5], use a specific representation of the generative model's output to estimate its quality. Additionally, [6] proposes a new likelihood index to identify overfitting, which classical likelihood cannot do.

We believe that neural networks can achieve good visual outputs, reach a good likelihood, and validate all these criteria, without necessarily replicating specific artifacts or features of the dataset. The features can be dilated and modified in time and space, making it completely impossible to use Euclidean distance to compare some features of our dataset and the samples from the generative neural network. That is why one need to find alternatives.

Our contributions

- Our proposed method, which draws inspiration from SAX, involves the use of a CUMSUM filter in conjunction with Piecewise Aggregate Approximation. As far as we know, this is the first method of its kind.
- By utilizing this representation along with local sensitive hashing, we are able to identify all recurrent items in our database regardless of their dilation in space or time. This provides us with a visual depiction of the various patterns in our database and their probabilities of occurrence.
- We evaluate the efficacy of our approach by applying it to a variational autoencoder.

2 Improving Piecwise Aggregate Approximation (PAA) with CUMSUM Filter

We consider a dataset $\mathcal{D} = \{X_1, \dots, X_N\}$ consisting of N i.i.d. samples from a continuous random vector. In our experiment, we choose X_i to be a random walk of size n with normally-distributed increments. As we will explain later, we center and scale each element of our data.

Our objective is to construct a function $f : \mathcal{D} \longrightarrow \mathcal{U}$ such that $f(X_i)$ is equal to $f(X_j)$ if X_i is very similar to X_j in a precise sense that we will define. Additionally, we expect that the dimension of $\mathcal{U}$ is much smaller than the dimension of $\mathcal{D}$.

Once we have achieved our goal, we propose to create a unique representation of the data $\mathcal{D}$ by concatenating $f(X_1), \ldots, f(X_N)$. We will abuse notation and write $f(\mathcal{D}) = [f(X_1), \ldots, f(X_N)]$. With $f(\mathcal{D})$ in hand, we can compute all the k-mers that can be found in it using local sensitive hashing. This enables us to describe all the patterns present in the data and to precisely characterize $\mathcal{D}$.

Next, we can compare the k-mers[1] of $\mathcal{D}$ to those of the generative data $\mathcal{G}$, much like how one can compare DNA sequences. This will allow us to propose a metric for evaluating our generative model.

To extract meaningful information from a data point X_i, we propose a function $f_1 : \mathcal{D} \longrightarrow \mathbb{R} \times \mathcal{E}$ that identifies the dates (or events) $\mathcal{E}$ and the levels associated with those events. Our goal is to develop a function that captures the different shapes of the data, such that a shape that is dilated in time would give the same code. The temporality of the data is captured in the set of events $\mathcal{E}$, while $\mathbb{R}$ only retains the order and intensity of variations. Figure 1 illustrates two time series, X_1 and X_2, that we would like to capture in this way.

To achieve this, we use a CUMSUM filter to identify the dates at which the events $\mathcal{E}$ occur. When events occur, we use a Piecewise Aggregate Approximation (PAA) method between two consecutive events to compute the corresponding value of $\mathbb{R}$ associated with that event.

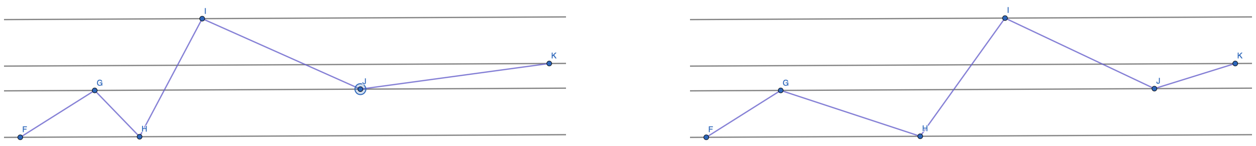


Figure 1: The image shows two time series X_1 and X_2 with the same succession of important levels but different instants. This highlights the need for a method that is not sensitive to time compression or dilatation which can occur with real data. To address this issue, we propose the use of a CUMSUM filter in conjunction with PAA to create a representation that is insensitive to temporal variations. This enables us to capture the different shapes of the data without being overly concerned with the precise timing of each event. In other words, our method allows us to identify similar patterns in the data, even if they occur at different times. This is important for validating generative models, as a good generative model should be able to replicate the different patterns observed in the data regardless of the exact timing of each event.

Reduction of dimension With PAA In order to reduce the dimension of time series, various techniques have been proposed in the literature. One of them is Piecewise Aggregate Approximation (PAA), which was introduced by Yi and Faloutsos [7] and later improved by Keogh et al. [8]. Given a time series X of length n, PAA represents it in a lower dimensional space of dimension w by a vector $\bar{X} = \bar{x}_1, \ldots, \bar{x}_w$, where each element $\bar{x}_i$ is calculated by taking the mean of a fixed number of adjacent values of X. Specifically, $\bar{x}_i$ is computed as follows:

$$\bar{x}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j$$

This technique divides the data into equal frames and calculates the mean value of each frame to create a vector of values representing the data, as shown in Figure 2. Sampling at fixed intervals may not capture important events in the data. To address these issues, we propose using a cumsum filter, which is not sensitive to time compression or dilation, as previously discussed. This allows for a more robust representation of the data and better capture of important events. While there have been some improvements on the PAA technique, such as the extended PAA proposed by Lkhagva et al. [9] or the higher-order PAA proposed by Pham et al. [10], we don't use them and focus on the use cumsum filter to create a more robust representation of the data.

Using CUMSUM filter with PAA We have not come across any existing literature that uses the cumsum filter to determine the intervals for computing PAA. However, we believe this could be a more effective approach. Several studies, including [11] and [12], suggest that the cumulative sum (CUSUM algorithm) is a sequential analysis technique

[1]In the context of bioinformatics, a k-mer refers to a sequence of length k that occurs in a longer DNA or RNA sequence.

commonly used for monitoring change detection. There are various versions of this algorithm, such as Wald's, but we have chosen to use the version proposed by [13] because the author provides a "physical" interpretation of the filter.

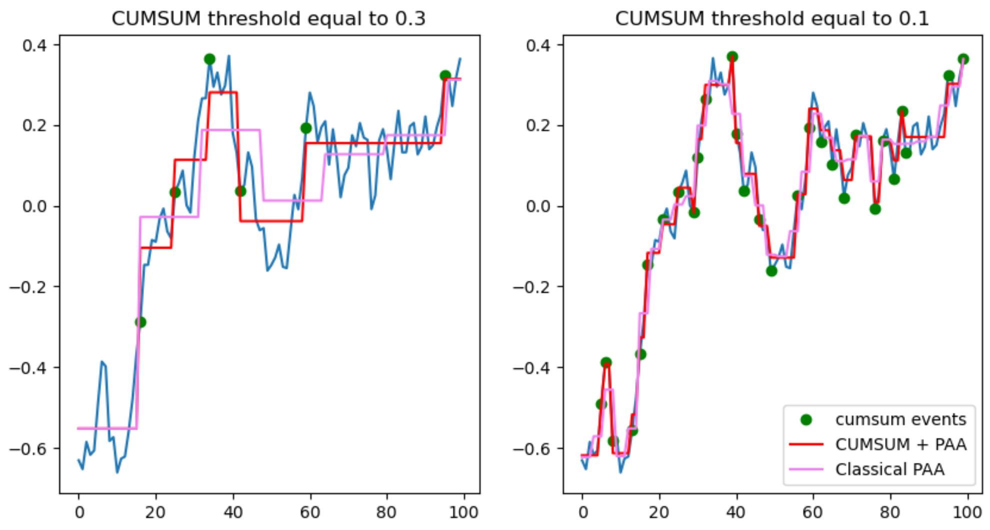


Figure 2: The algorithm is calibrated to use the same number of events in the two situations, with a slight variation for the decoupe but generally with the same number. The important aspect in this context is the shape of the time series, which remains unchanged despite the use of the filter.

Definition 1 (Euclidian Distance) *Given two time series X_i and X_j of length n, the Euclidean distance between them is defined as:*

$$Dist(X_i, X_j) \equiv \sqrt{\sum_{k=1}^{n} \left(x_i^k - x_j^k\right)^2}$$

In order to ensure a rigorous and unbiased comparison between our proposed method and the classical PAA technique, we carefully enforced that both methods utilize an identical number of data points to represent the temporal series. For our method, we additionally kept a record of the dates of events detected by the CUMSUM filter in order to enable more precise reconstruction in time and facilitate the computation of Euclidean distance as a measure of similarity. Alternatively, the use of Dynamic Time Warping [14] could have been considered as an alternative approach to evaluate the distance between the series, but this was beyond the scope of our current study.

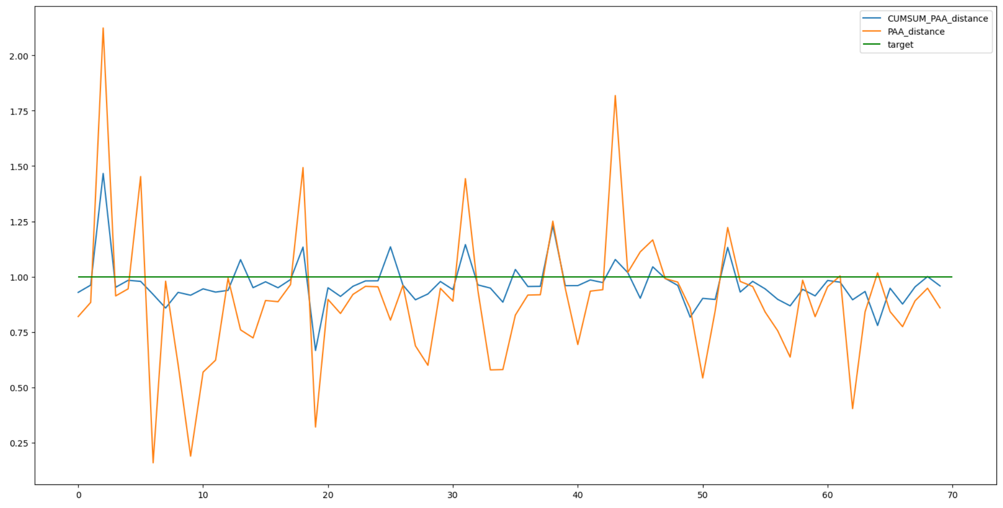


Figure 3: describing the error of reconstruction, giving the value ?

We present the results of our method and the classical PAA technique for different threshold values. For each threshold value, we computed the relative distance error between the time series reconstructed using CUMSUM+PAA and PAA, denoted as distCUM and distPAA respectively. We also computed the variance of this relative error denoted as varCUM and varPAA respectively. The results are shown in the table below:

$threshold$	$distCUM$	$distPAA$	$varCUM$	$varPAA$
0.10	**0.990314**	0.984396	**0.000071**	0.002580
0.15	**0.986079**	0.919452	**0.001596**	0.016332
0.20	**0.971088**	0.866409	**0.008959**	0.085167

Quantatize the rest According to the methodology described in [9], it is recommended to apply a discretization technique that generates equiprobable symbols after the application of PAA and CUMSUM. As suggested in [15], the resulting symbols should be equiprobable. To achieve this, we exploit the Gaussian distribution of normalized time series and introduce breakpoints.

Definition 2 (Breakpoints) *Breakpoints are a sorted list of numbers $B = \beta_1, \dots, \beta_{a-1}$ such that the area under a $N(0,1)$ Gaussian curve from β_i to $\beta_{i+1} = 1/a$ (β_0 and β_a are defined as $-\infty$ and ∞, respectively).*

β_i^a	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

An alphabet Σ is obtained as $\Sigma = \{"-1.22", "-0.84", "-0.52", "-0.25", "0", "0.25", "0.52", "0.84", "1.28"\}$, where the elements of Σ are considered as characters. Let $\mathcal{U}$ denote the set of all words that can be formed with Σ. We use $\mathbf{P}$ to denote a unique code for each series, where $\mathbf{P} \subseteq \mathcal{U}$.

Note that the use of the CUMSUM filter with PAA means that two series of the same length will not have the same code $\mathbf{P}$ and $\mathbf{Q}$ of the same length. However, our objective is not to compare time series, but rather to identify artifacts and patterns that regularly occur in our data $\mathcal{D}$, regardless of whether they are translated in space or differ in time dynamics. Our approach consists of the following process:

- Firstly, we generate the code of our time series in the set $\mathcal{D}$, denoted by $\mathbf{P_1}, \dots, \mathbf{P_n}$.
- Next, we concatenate the codes $\mathbf{P_1}, \dots, \mathbf{P_n}$ to obtain a unique code describing the set $\mathcal{D}$, denoted by $\mathbf{P} = [\mathbf{P_1}, \dots, \mathbf{P_n}] \subseteq \mathcal{U}$.
- Finally, we identify all the k sub-sequences of $\mathbf{P}$ that occur regularly, which represent the important information in the data to capture.

Using this approach, we can compare the features of our original data $\mathcal{D}$ and the generated data $\mathcal{G}$.

3 Local Sensitive Hasing to found similarities between two time series

The original work on local sensitive hashing was introduced by [16] and improved by [17]. These technical proposals provide an unbiased approximation of a similarity measure[2]. There are various families of LSH, depending on the similarity distance considered in our working space. As an example, we will focus on MinHash for Jaccard similarity [18]. We will use the initial version of MinHash, but it should be kept in mind that various improvements of MinHash exist, such as *b-bit MinHash* [19] or *one permutation MinHash* [20].

Definition 3 (Locality Sensitivity Hashing) *A family of function of $\mathcal{H}$ is called (R, cR, p_1, p_2)-sensitive if :*

$$\forall p, q \in M, h \in \mathcal{H} : \begin{cases} dist(p,q) < R \Rightarrow P_r(h(p) = h(q)) > p_1 \\ dijt(p,q) > cR \Rightarrow P_r(h(p) = h(q)) < p_2 \end{cases}$$

[2]The aim is to find nearest neighbors in high dimensions, which can be useful once we have the code for all our data. (This can be relevant to use such metrics in more complex databases)

In our model, because we can assim the code obtain after quantitize the serie as a word composing of letters on an alpahbet, we are going to get inspired of models used in bag-of-words model such as [21].

Definition 4 (Jaccard Similarity) *Given two sets $\mathcal{A}$ and $\mathcal{B}$, the Jaccard similarity is define as :*

$$J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|}$$

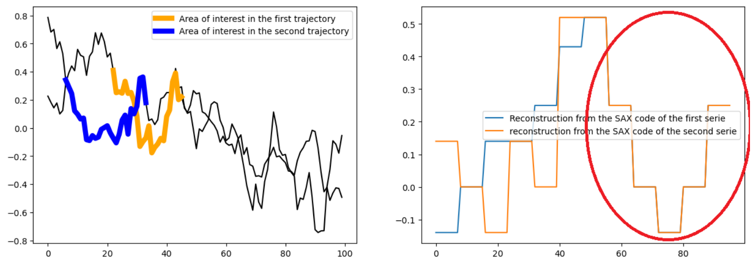
This distance is going to measure code $\mathbf{P}, \mathbf{Q} \subseteq \mathcal{U}$ of temporal series are similar.

MinHash propose an approximation of this measure. We considere a set of D hash function, which are D random permutations here $\{\pi_d\}_{d=1}^{D}$ where $\pi_d : \mathcal{U} \mapsto \mathcal{U}$. It has been proved that for two code $\mathbf{P}, \mathbf{Q} \subseteq \mathcal{U}$,

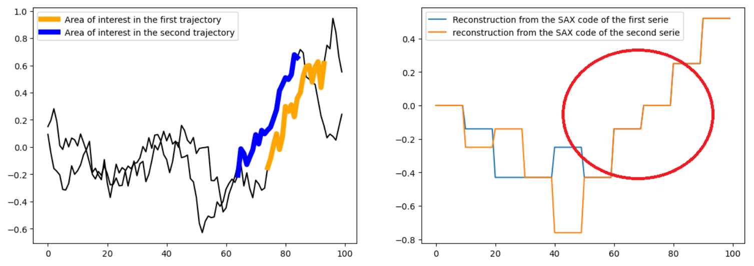
$$Pr\left[\min\left(\pi_d(\mathbf{P})\right) = \min\left(\pi_d(\mathbf{Q})\right)\right] = J(\mathbf{P}, \mathbf{Q})$$

where we say that there is an hash collision if $\min\left(\pi_d(\mathbf{P})\right) = \min\left(\pi_d(\mathbf{Q})\right)$. To estimate the Jaccard distance, we just have to reproduce the experiment and estimate probability of hash collision.

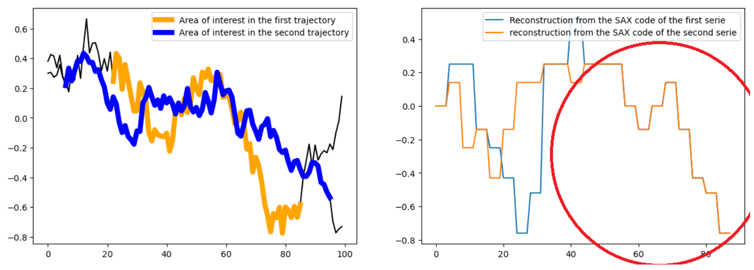
It follows that minwise hasing is (R, cR, R, cR) sensitive. Figure 4 show an application of our method.



(a) An example of similar 6-mers



(b) An example of similar 5-mers



(c) An example of similar 11 mers

Figure 4: We have examined our dataset $\mathcal{D}$ for all possible k-mers, where k takes the values 5, 6, and 11. Several examples of these k-mers are shown in the following plots.

4 Validation of an Variationnal Autoencoder (VAE)

Variationnal autoencodeur Assume that these data are generated by a continuous random process z. The generation of the data follows a well-defined path:

1. A value $z^{(i)}$ is generated from the isotropic multivariate Gaussian prior distribution (an assumption made) $p_\theta(z) = N(z; 0, I)$.
2. A value of $x^{(i)}$ is generated from the unknown multivariate Gaussian conditional distribution $p_\theta(x|z)$.

No assumptions are made on the marginal or posterior probabilities. It is not possible to estimate $p_\theta(x)$ using classical algorithms like EM (intractable integral).

We introduce a recognition model $q_\phi(z \mid x)$ that will approximate the density $p_\theta(z|x)$. We refer to z as the "latent" variables or "codes". Finally, we call $q_\phi(z \mid x)$ an "encoder" and $p_\theta(x|z)$ a "decoder". Note that $p_\theta(z)$ and $p_{\theta^*}(x|z)$ are differentiable almost everywhere with respect to θ and z.

It is essential to note that the recognition model $q_\phi(z \mid x)$ that will approximate the density $p_{\theta^*}(z|x)$ can theoretically belong to a large class of densities. Therefore, we assume that it is approximately Gaussian with a diagonal covariance matrix[3]. Hence, we have:

$$\log q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) = \log \mathcal{N}\left(\mathbf{z}; \mu^{(i)}, \sigma^{2(i)}\mathbf{I}\right)$$

where $\mu^{(i)}$ and $\sigma^{2(i)}$ are practically approximated by neural networks.

And

$$\log p(\mathbf{x} \mid \mathbf{z}) = \log \mathcal{N}\left(\mathbf{x}; \mu, \sigma^2\mathbf{I}\right)$$

We show that

$$\log p_\theta\left(\mathbf{x}^{(i)}\right) = D_{KL}\left(q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p_\theta\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right)\right) + \mathcal{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right)$$

with :

$$\mathcal{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right) = -D_{KL}\left(q_\phi\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\left[\log p_\theta\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]$$

The idea is then to maximise the lower bound, $\mathcal{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right)$ that we can compute[4]. By doing so, you maximise $\log p_\theta\left(\mathbf{x}^{(i)}\right)$. Other details, such as reparametrization trick are not explained here but can be found in the original paper.

Application We considere the function :

$$f : (a, b) \to \left(a + b, 2a, -2b, 6ab, a^2, 6a^2, a, b^3, a^2, 10ab, a, b, 0.76b\right)$$

where $a, b \sim N(0, 1)$. To make the thing more challenging, we replicate each of the value of the function 10 times, in order to obtain a vector in dimension 130. Then we train the VARE to replicate the function, basically, we obtained :

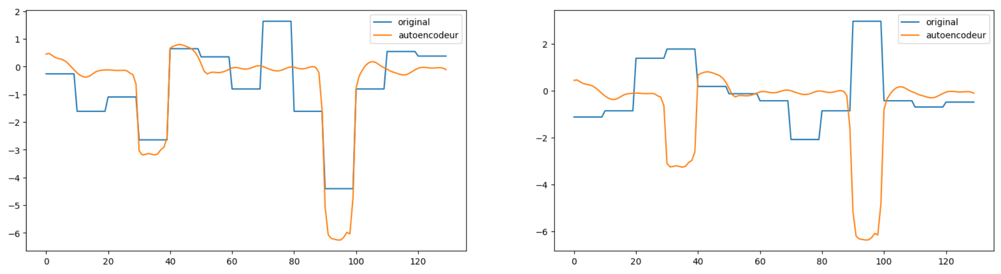


Figure 5: The algorithm is not really got here at reproducing the function. We can easy see it visually. But, we see that the algorithm is able to capture part of the information, such as the shape, how can we describe it ?

In the table below, we compare the length of the k-mers, and the pourcentage of succes in replicating the k-mers.

k	$succes$
2	7%
3	1.26%
4	0.26%
5	0%

[3]This assumption is classical in the literature and has been shown to be relatively robust on empirical tests

[4]Easy because we can explicitely compute Kullback-Leiber divergence for normal distrbutions

We see that is model is unable to correctly model the reality. And is very bad if we considere long sequences. We can go even much more longer, and for the 2 mers, compare the proportions.

$2MersPropGenerate$	$2MesPropData$
67.0	85.0
46.0	42.0
106.0	111.0
5.0	4.0
60.0	64.0
39.0	23.0
3.0	9.0
45.0	55.0
109.0	115.0
63.0	59.0
25.0	29.0
77.0	70.0
10.0	8.0

It should be possible to write statistical tests, such as for Quasi random Generator, and make the models pass by theses tests?

5 Conclusion

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[3] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560. PMLR, 2013.

[4] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[5] Julian Niedermeier, Gonçalo Mordido, and Christoph Meinel. Improving the evaluation of generative models with fuzzy logic. *arXiv preprint arXiv:2002.03772*, 2020.

[6] Marco Jiralerspong, Avishek Joey Bose, and Gauthier Gidel. Feature likelihood score: Evaluating generalization of generative models using samples. *arXiv preprint arXiv:2302.04440*, 2023.

[7] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. 2000.

[8] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 151–162, 2001.

[9] Battuguldur Lkhagva, Yu Suzuki, and Kyoji Kawagoe. Extended sax: Extension of symbolic aggregate approximation for financial time series data representation. *DEWS2006 4A-i8*, 7, 2006.

[10] Ninh D Pham, Quang Loc Le, and Tran Khanh Dang. Hot a sax: a novel adaptive symbolic representation for time series discords discovery. In *Intelligent Information and Database Systems: Second International Conference, ACIIDS, Hue City, Vietnam, March 24-26, 2010. Proceedings, Part I 2*, pages 113–121. Springer, 2010.

[11] Milton Severo and Joao Gama. Change detection with kalman filter and cusum. In *Discovery Science: 9th International Conference, DS 2006, Barcelona, Spain, October 7-10, 2006. Proceedings 9*, pages 243–254. Springer, 2006.

[12] David Tam. A theoretical analysis of cumulative sum slope (cusum-slope) statistic for detecting signal onset (begin) and offset (end) trends from background noise level. *The Open Statistics & Probability Journal*, 1(1), 2009.

[13] Marcos Lopez De Prado. *Advances in financial machine learning*. John Wiley & Sons, 2018.

[14] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[15] Stefano Lonardi. *Global detectors of unusual words: design, implementation, and applications to pattern discovery in biosequences*. PhD thesis, Purdue University, 2001.

[16] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[17] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[18] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, 1998.

[19] Michael Mitzenmacher, Rasmus Pagh, and Ninh Pham. Efficient estimation for high similarities using odd sketches. In *Proceedings of the 23rd international conference on World wide web*, pages 109–118, 2014.

[20] Anshumali Shrivastava and Ping Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *International Conference on Machine Learning*, pages 557–565. PMLR, 2014.

[21] Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *Artificial intelligence and statistics*, pages 886–894. PMLR, 2014.