Downsampling and geometric feature methods for EEG classification tasks with CNNs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We experimentally investigate a collection of feature engineering pipelines for use with a CNN for classifying electroencephalogram (EEG) time series from the Bonn dataset Andrzejak et al. [2001]. We compare ϵ-series of Betti-numbers and ϵ-series of graph spectra (a novel construction)—two topological invariants of a latent geometry of the timeseries—to raw time series of the EEG to fill in a gap in the literature for benchmarking. Additionally, we test these feature pipelines' robustness to downsampling and data reduction. This paper seeks to establish clearer expectations for both time-series classification via geometric features, and how CNNs for time-series respond to data of degraded resolution.

Topological Data Analysis (TDA) (Zomorodian and Carlsson [2004],Edelsbrunner et al. [2000]) has gained much attention due to applications for data analysis and machine learning. In particular, persistent homology (Scopigno et al. [2004],Edelsbrunner et al. [2002]) has been leveraged for machine learning purposes in numerous tasks. The methods attempt to describe the shape of the data in a latent space particularly amenable to feature engineering. The efficacy of topological features has been demonstrated in various tasks (Chazal and Michel [2017]).

In this paper, we investigate the performance of various topological feature engineering approaches for EEG time-series classification using one dimensional CNN as the classifiers. While CNN architectures are heavily experimented on, less research has explored models for feature engineering using modern geometric techniques (Seo et al. [2016], Bronstein et al. [2017]). Usually, CNNs are trained on the raw time-series data where a convolutional kernels of a fixed sizes and strides are applied to the series with moving windows to compute higher-order features. Persistent homology of the *Takens' embedding* provides one geometric procedure to engineer features for a time-series (Umeda [2017]).

For a time-series, the k-dimensional *Takens' embedding* of the time-series is the Euclidean embedding of points defined by a sliding window of size k—this provides a point-cloud representation of the time-series. From this point cloud the usual TDA approach provides persistent features. Both the raw series and the persistent features can be exploited for machine learning tasks. As a first step towards better understanding feature engineering on time-series, we compare the performance of these two approaches for the classification task.

Furthermore, we propose a novel geometric method beyond homology theories utilizing eigenvalues of sequences of graph Laplacians. Again, we utilize the point-cloud representation's ϵ-neighbor graph, and compute the normalized graph Laplacians' eigenvalues. Density counts of these eigenvalues are encoded as m discrete ϵ-series. We demonstrate the superiority of our approach over the homological features and compare to the raw time-series via classification experiments while keeping the classifier architecture fixed.

Generally, CNN architectures are optimized via network features like batch size, learning rate, kernels, pooling layers, and the like. Some papers have experimented with resizing to improve training time —

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

in (Howard [2018]) they introduced dynamic resizing with progressive resolution; (Roy et al. [2018]) have investigated the CNN robustness under noise. We have not found in the literature examinations of explicit downsampling algorithms' effects. To this effect, we study the performance of the feature engineering methods across multiple regimes of time-series resolution, and effective resolutions, comparing degradation across the feature types and downsampling methods—–a practice common in the signal processing literature but less so in classification of time-series. Note that the encoding methods in this paper need not only apply to sequential collections.

The principle contributions of this work include:

- Introduce a new feature engineering technique utilizing latent geometric properties of the time series.
- Apply the theory and methods of downsampling to time-series classification problem.
- Propose and demonstrate a comparison framework and baseline results for time series clustering via varying features and CNN architectures.

Overview of model pipelines

Raw Time Series Features: We feed the sequential series values into two kernel layers of one-dimensional convolution and max pool layers followed by a fully connected layer.

Persistent Betti Numbers Features: We encode each time series with the 'k-step' Takens' embedding into $\mathbb{R}^k$. This point cloud's ϵ-neighbor graph generates the Vietoris-Rips filtration up to dimension 3. The order of the degree n simplicial homology (or n'th Betti number) is computed for each ϵ neighbor complex, and encoded as n discrete ϵ-series. We feed the sequential ϵ-series values — each on their own channel — into two kernel layers of one-dimensional convolution and max pool layers followed by a fully connected layer.

Persistent Laplacian Eigenvalue Features: We encode each time series with the 'k-step' Takens' embedding into $\mathbb{R}^k$. This point cloud's ϵ-neighbor graph is collected and it's normalized graph Laplacians are computed. The eigenvalues of these Laplacians are computed and bucketed into a partition of m buckets. The counts of eigenvalues in each bucket are encoded as m discrete ϵ-series. We feed the sequential ϵ-series values — each on their own channel — into two kernel layers of one-dimensional convolution and max pool layers followed by a fully connected layer.

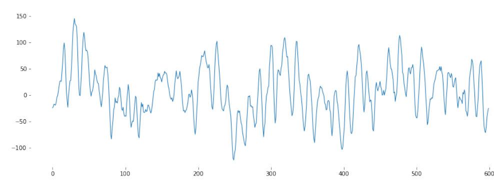

Figure 1: Raw time series from a segment of time where the patient's eyes were closed, segmented to 600 time steps, not downsampled.

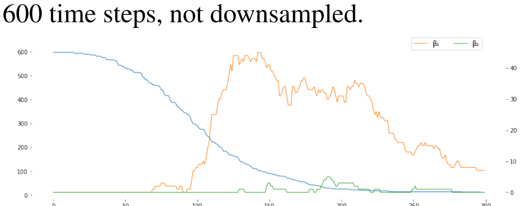

Figure 2: $\beta_j(\epsilon)$ computed for the time series shown in Fig. 1.

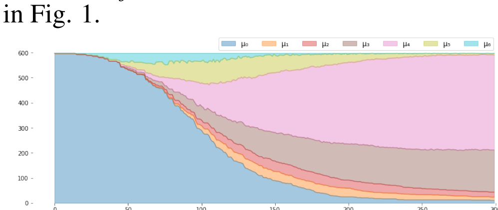

Figure 3: Area plot of $\mu_j(\epsilon)$ for the time series in Fig. 1 as described in Section 3.1.

Downsampled input data: In each of the pipelines, we prepend the model pipeline with a downsampling step using one of three downsampling algorithms and several downsampling resolutions.

Experimental design: Our design matrix consists of the three downsampling methods applied to $\{200, 300, 400, 500, 600\}$ initial data resolutions downsampled by steps of 50. Each of these initial data sets are fed through each of the model pipelines and subsequent CNNs. We use cross-validation and accuracy to evaluate the performance.

1 Experiments

Data set and classification task The data used in this work are time series EEG signals, and are provided by the University of Bonn, explored in Andrzejak et al. [2001]. This data set is comprised of five sets (labeled A-E), each containing 100 single-channel EEG segments 23.6 seconds in duration, with 4097 observations. The segments were hand selected from within a continuous multi-channel EEG recording, chosen for absence of artifacts as well as fulfilling a stationarity condition. Set E contains segments of seizure activity, sets C and D are taken from epilepsy patients during an interval where no seizure activity was occurring, and sets A and B are observations from non epilepsy diagnosed patients. The observations in set A occur during times when the patient's eyes were open, while those in set B occur during times when the patient's eyes were closed. We study the classification task of A vs. B.

CNN architectures All of the prediction algorithms used in this paper are CNNs, each with two sets of one dimensional convolution and max pool layers, followed by a fully connected layer to predict the class label. Architecture parameters are vectors representing $\langle \text{input}, \text{channels}, \text{factor}, \text{kernel1 size}, \text{kernel2 size} \rangle$; $\langle \text{res}, 1, 5, (\text{res}/600) * 18, 2 \rangle$, $\langle 300, 3, 7, 6, 2 \rangle$, $\langle 300, 7, 3, 6, 2 \rangle$ for raw time-series, Bettis, and eigenvalues respectively. `factor` refers to the multiplicative factor of input channels to output channels in each convolution layer, and `res` refers to the resolution to which the time series was downsampled to. Stride and dilation in both conv layers are 1 for each model; first pooling layer has size 7, second has size 3; each model is trained for 10 epochs.

Experiments and Results For a given chunk size, downsampling method, and downsampling rate, we run 10-fold cross validation for each of the three prediction methods described. The average and standard deviation of accuracy is recorded and displayed in Figure 4 for non-dynamic bucketing downsampling methods, and Figure 5 for dynamic bucketing downsampling methods.

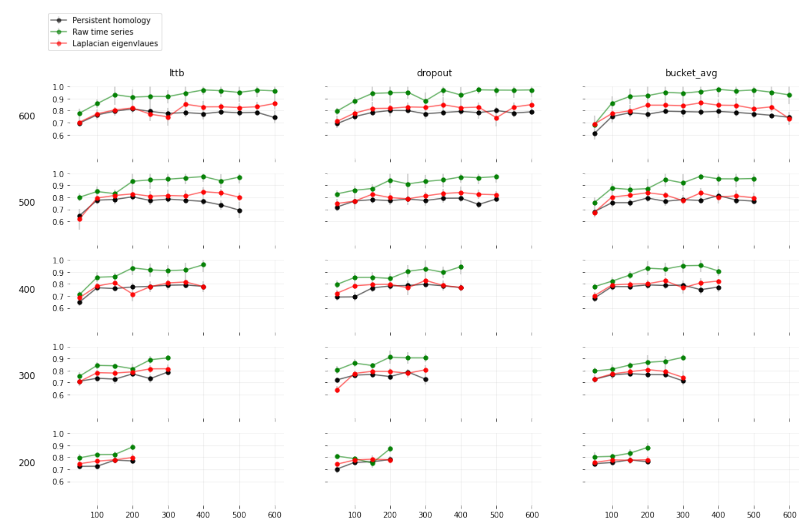

Figure 4: Experiment results for non dynamic bucketing downsampling methods.

2 Outcomes

We've explored a collection of 'experiments' in training and testing CNNs built on EEG data to predict if a patient's eyes are open or closed. The aforementioned experiments primarily sought to establish performance comparisons while varying the feature engineering choice, the chunk size, and the downsampling resolution used.

We established a baseline performance using a raw time series feature set and reproduced performance in (Umeda [2017]) to compare to this baseline. We saw that the performance of this baseline actually outperforms the TDA feature engineered experiment as reported. This suggests that for a task of this type the TDA approach is not SOTA, but may hold value in other regimes, or under more specific hyperparameter tuning. Building on these results, we explored the novel geometric feature engineering method of persistent eigenvalues of the Laplacian. This method also outperforms TDA, but does not significantly outperform the raw time series experiment.

We showed the performance of these networks under the strain of reduced data samples, and in resolution reduction. The impact on performance as we iterate through the parameter space is relatively smaller for the eigenvalue features, but perform worse than the raw time-series when a significant difference is detectable.

Finally, we provided a testbed for further iteration on these sorts of prediction tasks, and opened up a discussion around sensor resolution, sample data size, downsampling, feature engineering, and CNNs. The comparison pipelines are easily extensible for further experimentation with this dataset or others. All of the code for feature engineering and testing is available on GitHub.

Variable resolution training has been employed on ImageNet (Howard [2018]) to dramatically reduce training time, it's interesting to consider the implications of explicitly controlling downsampling schemes for this ansatz. Larger scope, we have left open the question of "multi-resolution" sensor networks and the impact on geometric feature engineering and downsampling.

3 Methods

3.1 Our Approach

Spectral graph theory is an integral facet of graph theory (Chung and Graham [1997]) and one of the key objects of this theory is the Laplacian matrix of a graph, as well as its eigenvalues. We assume all graphs are undirected and simple. For a graph G, let A and D be the adjacency matrix and the degree matrix of G respectively.

The *Laplacian* of G is defined to be $L = D - A$. The *normalized Laplacian* of G is then defined to be $\tilde{L} = D^{-1/2}AD^{-1/2}$.

Denote the eigenvalues(or *spectrum*) of $\tilde{L}$ by $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$. Recall:

Theorem (Lemma 1.7, Chung and Graham [1997])**.** *For a graph G with n vertices, we have that*

1. $0 \leq \lambda_i \leq 2$*, with* $\lambda_0 = 0$*. Further,* $\lambda_{n-1} = 2$ *if and only if a connected component of* G *is bipartite and nontrivial.*
2. *If* G *is connected, then* $\lambda_1 > 0$*. If* $\lambda_i = 0$ *and* $\lambda_{i+1} \neq 0$ *then* G *has exactly* $i+1$ *connected components.*

Persistent Laplacian Eigenvalues for Time Series Analysis Denote by $\tilde{L}_\epsilon(X)$ the normalized Laplacian of $G_\epsilon(X)$. Define $\hat{\lambda}_\epsilon(X) = [\lambda_\epsilon(X)_0, \lambda_\epsilon(X)_1, \ldots \lambda_\epsilon(X)_{n-1}]$ to be the vector of eigenvalues of $\tilde{L}_\epsilon(X)$, in ascending order: $0 = \lambda_\epsilon(X)_0 \leq \lambda_\epsilon(X)_1 \leq \cdots \leq \lambda_\epsilon(X)_{n-1} \leq 2$. When the context is understood, we will drop the designation (X) in the above notations; e.g. G_ϵ or $\lambda_{\epsilon 0}$.

Let I be an interval, $\hat{v} = [v_0, v_1, \ldots, v_{n-1}]$ be a vector, and define

$$\textbf{count}_I(v) := \#\{v_i \mid v_i \in I\}. \tag{1}$$

For a given interval $[0, r]$ (this will be our range of resolutions), and a finite collection of real numbers $0 = \tau_0 < \tau_1 < \cdots < \tau_k = 2$, define for $\epsilon \in [0, r]$:

$$\mu_j(\epsilon) := \begin{cases} \textbf{count}_{[\tau_j, \tau_{j+1})}(\hat{\lambda_\epsilon}) \text{ for } 0 \leq j < k-1 \\ \textbf{count}_{[\tau_j, \tau_{j+1}]}(\hat{\lambda_\epsilon}) \text{ for } j = k-1. \end{cases} \tag{2}$$

That is, $\mu_j(\epsilon)$ counts the number of eigenvalues of $\tilde{L}_\epsilon$ that lie between τ_j and τ_{j+1}. Observe that $\textbf{count}_{[0,0]}(\hat{\lambda_\epsilon})$ is equal to the number of connected components of G_ϵ. We will view the collection $\{\mu_j\}$ as a collection of j real-valued functions with domain $[0, r]$. We refer to the collection of μ_j's as *persistent Laplacian eigenvalues*. Given a time series $\{f(t_i)\}_{i=0}^n$ we form T^m, and compute $\{\mu_j\}_{j=0}^l$ for some choice of $\tau_0, \ldots \tau_l$.

References

Ralph G. Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E. Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Phys. Rev. E*, 64:061907, Nov 2001. doi: 10.1103/PhysRevE.64.061907. https://link.aps.org/doi/10.1103/PhysRevE.64.061907.

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: fundamental and practical aspects for data scientists. *ArXiv*, abs/1710.04019, 2017.

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. American Mathematical Soc., 1997.

H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, page 454. IEEE Computer Society, 2000. ISBN 0769508502.

Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, Nov 2002. ISSN 1432–0444. doi: 10.1007/s00454--002--2885--2.

W. D. Hall. Representation of blacks, women, and the very elderly (aged> or= 80) in 28 major randomized clinical trials. *Ethnicity and disease*, 9(3):333–340, 1999.

Jeremey Howard. Now anyone can train imagenet in 18 minutes. https://www.fast.ai/2018/08/10/fastai-diu-imagenet/, 2018.

Prasun Roy, Subhankar Ghosh, Saumik Bhattacharya, and Umapada Pal. Effects of degradations on deep neural network architectures. *CoRR*, abs/1807.10108, 2018. http://arxiv.org/abs/1807.10108.

R. Scopigno, D. Zorin, Gunnar Carlsson, Afra Zomorodian, Anne Collins, and Leonidas Guibas. Persistence barcodes for shapes, 2004.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. *arXiv*, 2016. URL https://arxiv.org/abs/1612.07659.

Sveinn Steinarsson. Downsampling time series for visual representation. Master's thesis, University of Iceland, 2013.

Yuhei Umeda. Time series classification via topological data analysis. *Transactions of the Japanese Society for Artificial Intelligence*, 32(3):D–G72_–12, 2017. doi: 10.1527/tjsai.D--G72.

Ancker JS. Veinot TC, Mitchell H. Good intentions are not enough: how informatics interventions can worsen inequality. *J Am Med Inform Assoc.*, 25(8):1080–1088, 2018.

Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 347–356, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138857. doi: 10.1145/997817.997870.

Broader Impacts

As the primary application of study for these experiments lies within the medical space, both positive and negative applications jump to mind. Large, high-resolution datasets both for training and evaluation come at the benefit of those in developed and wealthy communities. Our research – through its focus on developing methods robust to degradation – provides an opportunity for an improvement in prediction methods in lower fidelity data regimes; i.e. methods designed with downsampling and data reduction in mind alleviate needs for larger and more complete datasets.

The present study examines EEGs and eye states, which could easily extend to other ailments. More accurate models for predicting seizure, for example, could greatly benefit those privileged enough to share characteristics with those used to train the model in the first place. But what of those not sampled?

As (Hall [1999]) has observed: Blacks, women, and the elderly have historically been excluded from clinical trial research. Such arrangements can lead to what (Veinot TC [2018]) has referred to as intervention-generated inequalities (IGI), a social arrangement where one group gets better, while others don't. On top of their original ailments, groups left out are burdened with continued medical involvement and the associated costs (e.g. additional tests, transportation, childcare, and missed opportunities).

We offer the following suggestions for those in the medical industry hoping to combat some of this inequity: 1. Insist on multiple representative datasets including those from underrepresented groups – incentivization where appropriate. 2. Identify and assist in eliminating barriers to involvement in data collection or diagnostics.

A Downsampling

A.1 Time series downsampling

We consider a downsampling a selection of a subsequence of points, or a smaller set of points that summarize the timeseries. We assume the timeseries has $n+2$ points, and construct a downsample of $m+2$ points.

Naive Bucketing: Select the first and last points of the timeseries; cover the the rest of the points with m even-width intervals(up to integer rounding). We call this a *bucketing*.

Consider a sequence of sequences:

$$\left\{\{x_0\}, \{x_1, \dots, x_k\}, \{x_{k+1}, \dots, x_{2k}\}, \dots, \left\{x_{(m-1)k+1}, \dots, x_{mk}\right\}, \{x_{n+1}\}\right\}$$

and for simplicity, call the sub-sequences $\{b_i\}_{0\leq i\leq n+1}$ such that $b_0 = \{x_0\}$, $b_{n+1} = \{x_{n+1}\}$, and $b_j = \left\{x_{(j-1)k+1} \dots x_{(j-1)k}\right\}$, we refer to these as *buckets*.

Dropout: For each bucket, select the first point in the subsequence.

Bucket Averaging: For each bucket in a naive bucketing, average the x and y coordinates and take this as the reprsentative point; take also the first and last points.

For a bucket, we compute the 2-dimensional average of the points contained within: μ_j. For convenience of notation, we write elements of b_i, as $\left\{x_i^j\right\}$.

Largest-Triangle Three-Bucket Downsample (LTTB) We compute the subsequence via the optimization problem:

$$\textbf{compute } l_i = \arg\max_{x_i^j} \triangle\left(l_{i-1}, x_i^j, \mu_{i+1}\right) \text{ such that } l_0 = b_0 \text{ and } \mu_{n+1} = b_{n+1}.$$

The sequence $\{l_0, \dots, l_{n+1}\}$ is the *largest triangle three bucket downsample.* For more details and intuition around this construction we recommend the original paper.

Remark 1. *This is computed via a recursive optimization process iterating through the buckets; a non-recursive formulation to find the global optima is also possible. The distinction between these two solutions is that in the recursive solution each optimization is conditioned on the previous bucket, where-as the global solution conditions on all buckets simultaneously.*

A.2 Dynamic downsampling

In the above bucketing strategies, points in all regions of the time series are given equal weight in the downsample. Often times, the lagging-variance of a time series is not uniform across the time-domain. One might expect that regions of higher variance might warrant higher resolutions in the downsample, while low variance might require lower resolutions. A simple implementation of this idea, *(inspired by Steinarsson [2013])* is demonstrated in Algorithm 1(implementation included in Appendix). Downsampling methods are then applied to this bucketing of the timeseries.

Algorithm 1 Variance weighted dynamic bucketing

Precondition: $\mathcal{B}$ a naive bucketing, and P an iteration count

function DYNAMICBUCKETS($B : List[List[Float]]$)
  $[b_j] \leftarrow \mathcal{B}$
  for $i \leftarrow 1$ to P **do**
    for $j \leftarrow 1$ to m **do**
      $S(b_j) \leftarrow SSE(OLS(b_j))$
    $z \leftarrow \arg\max_j(S(b_j))$
    $b_z^l \leftarrow \left\{x_1^z, \ldots, x_{\lfloor k/2 \rfloor}^z\right\}$
    $b_z^r \leftarrow \left\{x_{\lfloor k/2 \rfloor+1}^z, \ldots, x_k^z\right\}$
    $\mathcal{B} \leftarrow \mathcal{B} \setminus \{b_z\} \cup \left\{b_z^l, b_z^r\right\}$
  for $i \leftarrow 1$ to P **do**
    for $j \leftarrow 1$ to $m + P$ **do**
      $S(b_j) \leftarrow SSE(OLS(b_j))$
    $a \leftarrow \arg\min_a(S(b_a) + S(b_{a+1}))$
    $b_a^* \leftarrow \{x_1^a, \ldots, x_k^a\} \cup \left\{x_1^{a+1}, \ldots, x_k^{a+1}\right\}$
    $\mathcal{B} \leftarrow \mathcal{B} \setminus \{b_a, b_{a+1}\} \cup \{b_a^*\}$
  return $\mathcal{B}$

Remark 2. *Rather than serially splitting, and then combining buckets to arrive at the rebucketing, it's natural to ask how alternating these operations effects the result. The authors carried out several simulations of this technique and found that convergence to 'stable' bucketing took place much more quickly, but produced far worse results with respect to total SSE.*

A.3 Dynamic downsampling results

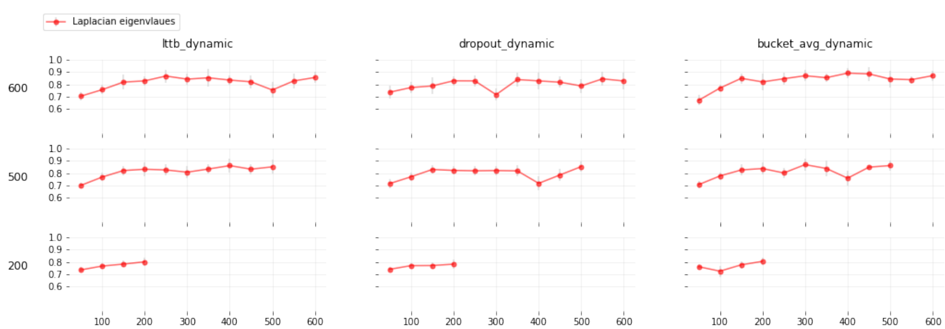

Figure 5: Experiment results for dynamic bucketing downsampling methods.