Neural activity classification with machine learning models trained on interspike interval series data

Ivan Lazarevich[1,2], Ilya Prokin[3], and Boris Gutkin[1,4]

[1] *École Normale Supérieure, Laboratoire de Neurosciences Cognitives, Group for Neural Theory, Paris, France*

[2] *Lobachevsky State University of Nizhni Novgorod, Nizhny Novgorod, Russia*

[3] *Dataswati, Orsay, France*

[4] *Center for Cognition and Decision Making, National Research University Higher School of Economics, Moscow, Russia*

October 10, 2018

Abstract

The flow of information through the brain is reflected by the activity patterns of neural cells. Indeed, these firing patterns are widely used as input data to predictive models that relate stimuli and animal behavior to the activity of a population of neurons. However, relatively little attention was paid to single neuron spike trains as predictors of cell or network properties in the brain. In this work, we introduce an approach to neuronal spike train data mining which enables effective classification and clustering of neuron types and network activity states based on single-cell spiking patterns. This approach is centered around applying state-of-the-art time series classification/clustering methods to sequences of interspike intervals recorded from single neurons. We demonstrate good performance of these methods in tasks involving classification of neuron type (e.g. excitatory vs. inhibitory cells) and/or neural circuit activity state (e.g. awake vs. REM sleep vs. nonREM sleep states) on an open-access cortical spiking activity dataset.

Introduction

Modern advances in multineuronal recording technologies such as calcium imaging [1] and extracellular recordings with multielectrode arrays [2] allow producing single-neuron resolution brain activity data with remarkable magnitude and precision. In addition to experimental technique development, various data analysis methods were introduced over the years which enable better processing as well as understanding of neural activity data. Recent developments range from accurate inference of spiking events from calcium fluorescence traces based on a variety of machine learning approaches [3] to a myriad of spike sorting techniques for identification of originating neurons in multi-electrode recordings [4, 5]. Modern machine learning techniques were successfully applied both to neural activity decoding (predicting stimulus/action from spiking activity) [6] as well as neural encoding (predicting neural activity from stimuli) [7]. These neural decoding approaches typically focus on firing rate signals of multiple neurons from a population upon stimulus presentation/action execution. In this context, the fine temporal structure of neuronal firing patterns is not considered as a predictor of cell or network properties in question. However, it is known that the temporal structure of neuronal spike trains may significantly vary across cell types and also across particular activity states within a single neuron type. Thus, it may be hypothesized that certain features of neuronal spike trains carry information about cell type or network activity state that can be, in principle, decoded from these activity patterns. In this study, we demonstrate that effective feature representation of neuronal spike trains enables good performance in supervised classification tasks which involve identifying a particular neuron type or activity state of a neural circuit (for instance, pyramidal cell vs. interneuron classification, REM vs. nonREM phase sleep circuit state classification, etc.).

A number of previous studies on feature vector representation of spike trains usually focused on defining a spike train distance metric [8] for identification of synchronized neuronal assemblies [9]. Several different definitions of spike train distance exist such as van Rossum distance [10], Victor-Purpura distance [11], SPIKE- and ISI- synchronization distances [12] (for a thorough list of existing spike train distance metrics see [8]). These distance metrics were used to perform spike train clustering and classification based on the k-Nearest-Neighbors approach [13]. In a recent study, Jouty et al. [14] employed ISI and SPIKE distance measures to perform clustering of retinal ganglion cells based on their firing responses.

In addition to characterization with spike train distance metrics, some previous works relied on certain statistics of spike trains to differentiate between cell types. Charlesworth et al. [15] calculated basic statistics of multi-neuron activity from cortical and hippocampal cultures and were able to perform clustering and classification of activity between these culture types. Li et al. [16] used two general features of the interspike interval (ISI) distribution to perform clustering analysis to identify neuron subtypes. Finally, not only spike timing information was used to characterize neurons in a supervised classification task. Jia et al. [17] used waveform features of extracellularly recorded action potentials to classify them by brain region of origin.

In this work, we propose usage of general time series classification/clustering methods to perform feature vector representation of neuronal spike trains. We demonstrate that effective representations of spike trains obtained with these methods enable good classification results in both cell type classification and network activity state classification tasks. We provide baseline performance estimates for a range of machine learning algorithms trained on feature vector representations of spike trains. These estimates are obtained using a cortical spiking activity dataset for which we consider tasks of excitatory vs. inhibitory cell classification and awake/sleep-phase state classification.

Methods

Overview of time-series classification methods

Within the scope of our approach, we apply general time-series feature representation methods for classification/clustering [18] of neuronal spike train data. Most approaches in time series classification are focused on transforming the raw time series data before training and applying a machine learning classification model. Here we give a brief overview of state-of-the-art approaches one could utilize in order to transform time series data into feature vector representation for efficient neural activity classification.

Neighbor-based models with time-series distance measures

The go-to algorithm to obtain a good baseline for time-series classification is considered to be the k-nearest-neighbors (kNN) model. The kNN algorithm compares a new time-series to all the series in the database of series with known class labels based on some time-series distance metric. For the new series that needs to be classified, k closest series are selected from the database and most frequent class label among them is taken as a label of the new series (so-called voting algorithm). To obtain good results with kNN, it is important to choose the right distance metric. The most common metrics used in kNN for time series classification are the Euclidean distance (or, more generally, the Minkowski L_p distance) [18] and the Dynamic Time Warping (DTW) distance [19]. Conversion to interspike-interval (ISI) series representation of the spike train can be done prior to calculating the L_p inter-train distance. Moreover, the inter-train distance can be defined based on differences of ISI distributions within the trains. For this one can utilize distribution similarity measures (e.g. Kolmogorov-Smirnov distance, Kullback–Leibler divergence, Wasserstein distance) and compute its values between ISI distributions of given spike trains. Such a spike train distance definition would only use the information about the ISI distribution in the spike train, but not about its temporal structure. Alternatively, one can keep the original event-based representation of the spike train and compute the spike train similarity metrics such as van Rossum or Victor-Purpura distances or ISI/SPIKE distances [8]. Finally, any number of distance measures can be combined (e.g. linearly) to give a weighted estimate of the spike-train-distance, and weighting coefficients can be learned to produce optimal classification quality on a given task.

The choice of the distance metric determines which features of time-series are considered as important. Instead of defining a sophisticated distance metric, one can explicitly transform time-series into some feature space by calculating various features of the series that are deemed important (e.g. mean, variance). After assigning appropriate weights to each feature one can use kNN with any standard distance metric. Moreover, such a representation allows the application of any state-of-the-art machine learning classification algorithm. In the following, we discuss various feature space representations available for time series data.

Manual time-series feature engineering

One of the useful and intuitive approaches in time series classification is focused on manually calculating a set of descriptive features for each time series (e.g. their basic statistics, spectral properties, other measures used in signal processing etc.) and using these feature sets as vectors describing each sample series. There exist approaches which enable automated calculation of a significant number of time series features which may be typically considered in different application domains. Such approaches include automated time series phenotyping implemented in the *hctsa* MATLAB package [20] and automated feature extraction in the *tsfresh* Python package [21]. Here we utilize the *tsfresh* package which enables calculation of 794 descriptive time series

features for each spike train, ranging from Fourier and wavelet expansion coefficients to coefficients of a fitted autoregressive process.

Once each time series (spike train) is represented as a feature vector, the spiking activity dataset has the standard form of a matrix with size $[n_{\text{samples}}, n_{\text{features}}]$ rather than the raw dataset with shape $[n_{\text{samples}}, n_{\text{timestamps}}]$. This standartized dataset can be then used as an input to any machine learning algorithm such as kNN, random forests, gradient boosting machines [22] and artificial feedforward neural nets. We found this approach to yield good benchmark classification results in both cell type identification and neural activity state classification tasks.

One advantage of this approach over other time series classification methods is its potential tractability: i) engineered features are interpretable, and ii) if an interpretable model such as a linear model or a decision tree-based model is used for classification (e.g. random forest or gradient boosted decision trees), feature importance ranks can be estimated, and, furthermore, more advanced techniques (e.g. SHAP [23] or LIME [24]) can be used to infer contributions of each manually engineered feature to the resulting classification output.

Quantization / bag-of-patterns transforms

Some state-of-the-art algorithms in general time-series classification use text mining techniques and thus transform time series into bag(s) of words (patterns). This is typically done the following way. First, a time series of real numbers is transformed into a sequence of letters. One of the methods to perform this transform is Symbolic Aggregate approXimation (SAX) [25]. In SAX, bins are computed for each time series using gaussian or empirical quantiles. After that, each datapoint in the series is replaced by the bin it is in (a letter). Another algorithm commonly used for this task is Multiple Coefficient Binning (MCB). The idea is very similar to SAX and the difference is that the quantization is applied at each timestamp. The third algorithm for the series-letter transform is Symbolic Fourier Approximation (SFA) [26]. It performs a discrete Fourier transform (DFT) and then applies MCB, i.e. MCB is applied to the selected Fourier coefficients of each time series. Once the time series is transformed into a sequence of letters, a sliding window of fixed size can be applied to define and detect words (letter patterns) in the sequence. After that, the bag-of-words (BOW) representation can be constructed whereby each "sentence" (time series) turns into a vector of word occurrence frequencies.

Several feature generation approaches were developed utilizing the BOW representation of time series data. One such method is Bag-of-SFA Symbols (BOSS) [27]. According to the BOSS algorithm, each time series is first transformed into a bag of words using SFA and BOW. Features that are created after this transformation are determined by word occurrence frequencies. Another similar approach is Word ExtrAction for time SEries cLassification (WEASEL) [28]. In WEASEL, one similarly has to first transform each time series into a bag of words. WEASEL is more sophisticated in the sense that the selected Fourier coefficients are the most discriminative ones (based on the one-way ANOVA test), several lengths for the sliding window are used and the most discriminative features (i.e. words) are kept (based on the chi-2 test).

Some classification algorithms which use this bag-of-patterns approach represent whole classes of samples with a set of features. One example of such a method is an algorithm called SAX-VSM [29]. The outline of this algorithm is to first transform raw time series into bags of words using SAX and BOW, then merge, for each class label, all bags of words for this class label into a single class-wise bag of words, and finally compute term-frequency-inverse-document-frequency statistic (tf-idf) [30] for each bag of words. This leads to a tf-idf vector for each class label. To predict an unlabeled time series, this time series is first transformed into a term frequency vector, then the predicted label is the one giving the highest cosine similarity among the tf-idf vectors learned in the training phase (Nearest Neighbor classification with tf-idf features). A very similar approach is Bag-of-SFA Symbols in Vector Space (BOSSVS) [31] which is equivalent to SAX-VSM, but words are created using SFA rather than SAX. A more computationally expensive but potentially more accurate approach is to train a standard classifier model on full extracted bag-of-patterns feature vectors. This is for example done with the WEASEL transform [28] where logistic regression is applied to extracted features. Any other classification algorithm can in principle be applied once time series are transformed to bag-of-patterns feature vectors.

All aforementioned time series representation methods are implemented in the *pyts* Python package [32], which was also used in the present work.

Image representation of time series

Several methods to represent time series as images (matrices with spatial structure) were developed and utilized for classification as well. One such image representation method is called the recurrence plot [33]. It transforms a time series into a matrix where each value corresponds to the distance between two trajectories (a trajectory is a sub time series, i.e. a subsequence of back-to-back values of a time series). The obtained matrix can then be binarized using some threshold value. Another method of time-series image representation is called Gramian Angular Field (GAF) [34]. According to GAF, a time series is first represented as polar coordinates. Then the time series can be transformed into a Gramian Angular Summation Field (GASF) when the cosine of the

sum of the angular coordinates is computed or a Gramian Angular Difference Field (GADF) when the sine of the difference of the angular coordinates is computed. Yet another image representation method is the Markov Transition Field (MTF). The outline of the algorithm is to first quantize a time series using SAX, then to compute the Markov transition matrix (the quantized time series is treated as a Markov chain) and finally to compute the Markov transition field from the transition matrix. These image representations can be effectively used in junction with effective deep learning models for image classification (e.g. various available architectures of convolutional neural nets [34, 35]).

Deep learning approaches

Lastly, one can make use of modern deep learning approaches [35] to perform algorithm training on raw time series data. Recurrent neural networks such as Long Short-Term Memory (LSTM) nets [36] and their variations based on one-dimensional convolutional neural networks (CNNs) [37] were shown to enable good classification quality on general time series datasets [38]. Some of the frequently used neural network architectures are plain LSTMs [39], Convolutional Neural Network LSTMs (CNN-LSTMs) [40] and Convolutional LSTMs (ConvLSTMs) [41]. These approaches (together with image-based representations of spike trains as described above) are to be explored in further work.

Finally, all method classes listed in the above subsections (e.g. neighbor-based models, models based on engineered features, bag-of-patterns classifiers, computer vision models on image representations of time series and deep learning models on raw time series data) are fundamentally different in their underlying feature respresentation of time series, such that predictions of these models may be effectively combined in an ensemble of models using model stacking/blending [42] to improve classification results.

Data

For the tasks of spike train analysis and classification, high-quality datasets with recordings of neural firing activity are of foremost importance. Here we used an open-access neural activity dataset from the Collaborative Research in Computational Neuroscience (CRCNS) repository (`http://crcns.org/`) [43]. Specifically, the following dataset was used to define classification benchmarks:

- **fcx-1** dataset [44, 45]: Spiking activity and Local-Field Potential (LFP) signals recorded extracellularly from frontal cortices of male Long Evans rats during wake and sleep states without any particular behavior, task or stimulus. Around 1100 units (neurons) were recorded, 120 of which are putative inhibitory cells and the rest is putative excitatory cells. Figure 1 shows several examples of spiking activity recordings that can be extracted from the fcx-1 dataset. The authors classified cells into an inhibitory or excitatory class based on the action potential waveform (action potential width and peak time). Some of the cells which were classified as excitatory/inhibitory were not found to have an excitatory/inhibitory effect on cross-correlograms with other cells (CCGs), these cells are labelled as "excitatory-like" and "inhibitory-like". Cells with a clear effect on CCGs were labelled as "excitatory-definite" or "inhibitory-definite". Sleep states were labelled semi-automatically based on extracted LFP and electromyogram features, in particular, rapid eye movement (REM) sleep state was characterized by a pronounced theta-band LFP component, while slow-wave sleep (SWS or nonREM) state was characterized by a delta-band LFP component. Non-sleep state was labelled as a WAKE activity class. Thus, original wake/sleep state labelling is based on population-level integral signals. Several classification problems might be addressed with this dataset: (a) excitatory vs. inhibitory cell classification from spike train data with similar mean firing rate, (b) WAKE vs. SLEEP state classification and, finally, (c) REM vs. nonREM/SWS sleep state classification.

Cross-validation scheme and data preprocessing

Suppose we are given a dataset containing data from several mice each recorded multiple times with a large number of neurons captured in each recording. For each recorded neuron we have a corresponding spike train captured over a certain period of time. The number of spikes within each train is going to be variable from train to train, so vectors of spike times for each neuron would have different length. A natural way to standardize the length of spike-train-vectors would be dividing the full spike train into chunks of N_{size} spike times, where N_{size} is fixed for each chunk. The chunks can be taken sequentially from the full spike train or, as a method of data augmentation, can be produced by moving a sliding window across the spike-timing-vector. Thus, each neuron would contribute a different number of spike-timing-chunks depending on its average firing rate. To remove the trend component from each chunk, differencing can be applied to get the ISI-series representation of the spike train chunk. The size of each spike train chunk N_{size} is a free data preprocessing parameter which is a subject of a trade-off: large chunk size would provide good spike train statistics/feature quality for each sample and

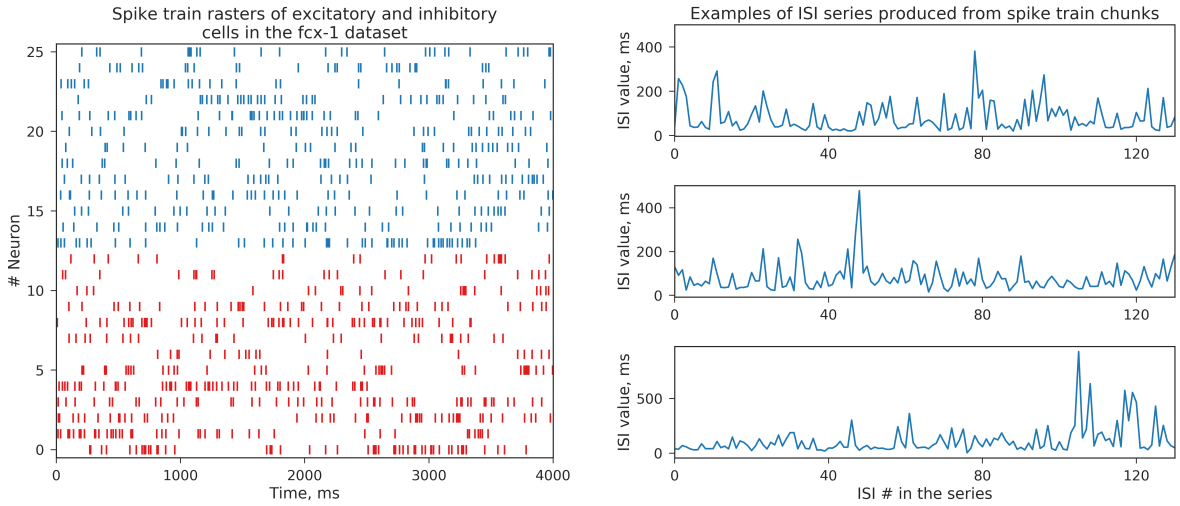

Figure 1: Examples of spiking activity recordings in the CRCNS fcx-1 dataset. Left: spike train raster of a random subset of excitatory cells (red) and inhibitory cells (blue). Right: three examples of ISI series produced from spike train chunks of inhibitory cells in the fcx-1 dataset.

smaller chunk size would allow for more data samples and/or lower computational complexity of the problem. Here we first choose an empirical value of N_{size} for each dataset small enough to reach sufficient dataset size (e.g. $> 10^4$ samples). Further, the optimal value for N_{size} can be inferred from cross-validation (the evaluation of the algorithm on the data that was not used during fitting) of the classification pipeline.

Here we used a classical cross-validation strategy [46] whereby the data is split to two parts: i) training data that is used to fit the parameters of a machine learning model and ii) testing data used to evaluate the model fitted on training data. A simple approach is to take the whole dataset of shape $[N_{\text{chunks}}, N_{\text{size}}]$ with corresponding class labels and perform the train-test subset split for classification quality assessment. However, it would lead to an overly-optimistic estimate of algorithm's performance, because similar ISI-series chunks coming from the same neuron/recording session/mouse can become a part of both train and test datasets. A more correct validation scheme would be to first split animal IDs/recording sessions/neurons into train and test subsets, so that test prediction is performed on a neuron/session/mouse not present at all in the train set. After this splitting by spike-train ID is done, one could generate fixed length chunks of spiking activity within each subset. Here, if not stated otherwise, we divide the dataset into a holdout validation set (40-45% of the data, keeping the class proportions as in the full dataset) and and the rest is used for stratified k-fold cross-validation (with fixed $k = 5$). Neuron IDs/recording sessions/animal IDs do not overlap between folds. The choice between splitting by individual neurons or sessions/recorded animals depend on initial problem statement. One can use spike train recordings from all recorded animals simultaneously for both train and test, or train on one subset of animals and test on a different subset; the latter could be a demonstration that a model trained on certain animals is actually transferable onto new animals which would suggest the generality of the machine learning model.

Algorithm quality assessment

Metrics we used to assess classification performance are accuracy (when class distribution balancing was performed in the dataset beforehand) and AUC-ROC [46] (when probabilistic estimates for a sample to belong to a certain class are available). In general, choice of a particular metric of interest should be determined according to the underlying classification task. For instance, in the classification of diseased states of a neural circuit, more emphasis can be drawn to recall values – due to foremost importance of diseased state recognition and not vice versa.

If the classification task is to determine certain activity states of the neural circuit, one could collect activity of several neurons at a time (e.g. by sorting spikes from MEA recordings or from calcium imaging) and correspondingly perform classification for each measured neuron independently. If the final classification is done by majority voting from all single-neuron predictions and we assume that recorded neurons are randomly sampled from the whole ensemble, the optimistic estimate for accuracy increase with the number of neurons N_{cells} would be

$$\mu = \sum_{i=m}^{N_{\text{cells}}} C^i_{N_{\text{cells}}} p^i (1-p)^{N_{\text{cells}}-i}$$

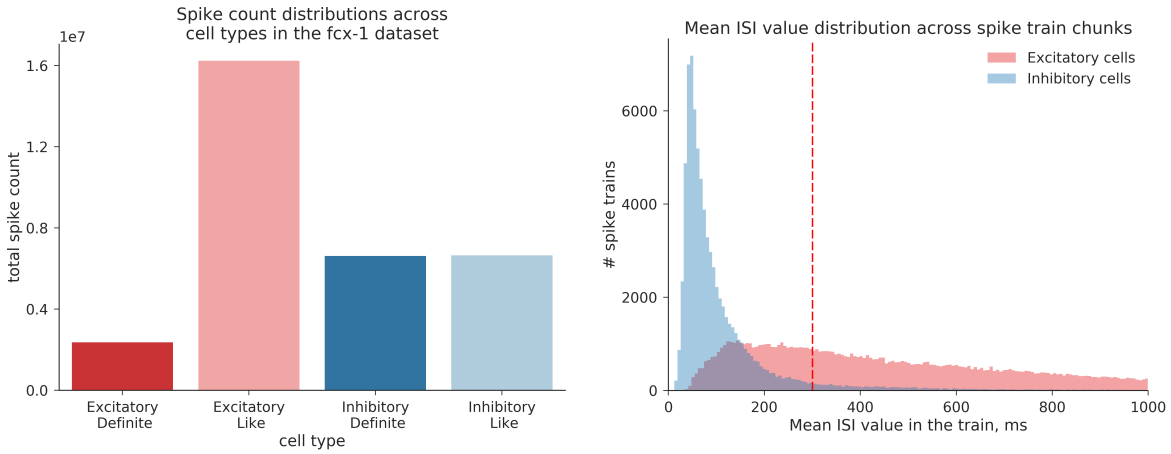

Figure 2: Left: total spike count distribution across cell types in the fcx-1 dataset. Right: mean ISI value distributions for excitatory and inhibitory spike train chunks in the training set. Red dashed line denotes the cutoff value (300 ms) for mean ISI of a spike train to be included in the dataset.

where μ is the probability that the majority vote prediction is correct, p is the probability of a single classifier prediction being correct, N_{cells} is the number of predictions made, $m = \lfloor N/2 \rfloor + 1$ is the minimal majority of votes. If single neurons are to be classified (e.g. into subtypes), one can split a large spike train into several chunks with N_{size} spikes and perform predictions on each of those independently (with final output being the major vote from every chunk). In this case the above estimate is likely overly optimistic, since ISI sequences within a single spike train are probably distributed differently than ISI sequences across different neurons.

Results

As we have discussed above, one could consider posing several types of classification tasks utilizing available spiking data – for instance, cell type classification or network activity state classification. Here, while considering the fcx-1 dataset, we test if it is possible to infer both cell type (e.g. excitatory or inhibitory) and network activity state (e.g. WAKE or REM sleep or nonREM sleep) from spiking activity of individual neurons.

Excitatory vs. inhibitory cell spike train classification

We start with a basic excitatory vs. inhibitory cell classification task for the fcx-1 dataset. If we merge the "definite" and "like" cell subtypes into single classes (inhibitory/excitatory), we end up with a fairly balanced class distribution in the dataset (Fig. 2, left). Moreover, we add spike train samples to the dataset regardless of the underlying network state (sleep/wake). After merging the cell subtypes, we are left with 995 excitatory cells and 126 inhibitory cells recorded in total. We leave cells corresponding to ∼40% of the total spike count from both classes for the validation subset and the rest for the training subset. Every spike train in the data is then represented in an ISI-series form. For both train and validation subsets we extract ISI series chunks of size $N_{\text{size}} = 200$ ISIs by applying a rolling window with a step = 100 ISIs. Mean ISI value across spike train chunks follows a heavy-tail distribution which is different depending on the cell type (Fig. 2, right). It makes sense to consider spike trains in a common mean-ISI interval for both cell types for classification, as there are almost no inhibitory-cell spike trains with mean ISI > 400 ms present in the dataset. We therefore choose a cutoff value for the mean ISI of the train (here it is taken to be 300 ms, Fig. 2) and keep only the trains with mean ISI below this value, so that mean ISI distributions are largely overlapping for both cell types. This means we only keep excitatory-cell spike trains with high firing rates (comparable to inhibitory cells) in the dataset. In this setting the classification problem has a slight class imbalance, since we obtain $\sim$ 26000 excitatory spike train chunks and $\sim$ 75000 inhibitory spike train chunks in the train set after preprocessing. The validation (holdout) set consists of $\sim$ 22000 and excitatory and $\sim$ 48000 inhibitory spike train chunks. Spike train chunks are extracted regardless of neural activity state (e.g. sleep or awake) during particular time intervals in the recording. In other words, we assume that cell type classification can be achieved with training and validation on data extracted regardless of circuit activity state.

The presence of class imbalance in the dataset means that either undersampling or oversampling techniques shall be utilized further in this task to justify usage of accuracy as a metric for classification quality. In this study, we perform undersampling prior to training the classifiers and evaluating accuracy on the validation set.

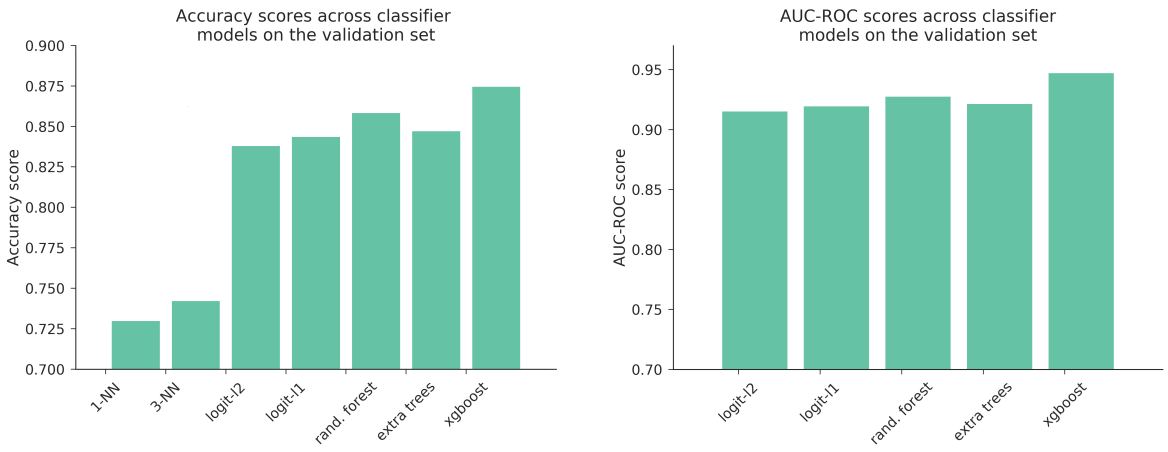

Figure 3: Classification accuracy values (left) and AUC-ROC scores (right) of the classifiers trained on manually engineered (*tsfresh*) features of spike trains (full set of 794 features) achieved on the validation set in an excitatory vs. inhibitory spike train classification task.

1. Manual feature extraction + standard classifier models baseline

To obtain the first classification quality metric baseline we use the manual feature extraction approach without feeding the raw temporal ISI sequence data directly to the classification model. We use the *tsfresh* Python package to calculate a representative set of 794 features for each spike train chunk. This can be done independently for the training and validation subsets as only individual-sample characteristics are being calculated. Mean and variance are calculated for each feature from the training set and then standard scaling with these values is applied to both training and testing sets to avoid overfitting. Low-variance features are removed using the condition $std/(mean + \varepsilon) < thr$, where thr is set to 0.2 and ε to 1e-9. Some of the features extracted from ISI series data by *tsfresh* are highly correlated with each other, as measured by the Pearson correlation coefficient between feature values. Removal of correlated features can be implemented by leaving a single feature out of each pair of features which have Pearson correlation value $> corr_{\text{thr}}$. Exact value of $corr_{\text{thr}}$ is generally a free parameter, however we found that even slight removal of correlated features ($corr_{\text{thr}} < 1$, but close to unity) had a negative effect on classification performance, and a negative general trend for classification accuracy was observed as $corr_{\text{thr}}$ was decreased. Hence, in most of the tests presented we did not perform preliminary removal of correlated features for classification. It was, however, performed prior to applying a dimensionality reduction algorithm (e.g. Principal Component Analysis), as these methods are known to be sensitive to feature correlations. Once ISI series samples are converted to feature-based vector representation, standard classifier models such as random forests and gradient boosting machines can be used to evaluate baseline accuracy scores for the particular task. To compare accuracy scores across different classification model classes, we use (a) a kNN model with Minkowki L_p distance metric, (b) a linear logistic regression model (with either l_1 or l_2 regularization terms) and (c) several nonlinear tree-based models: random forest classifier, randomized decision trees (extra trees) classifier and an implementation of decision tree gradient boosting from the *xgboost* library.

Fig. 3 shows accuracy and AUC-ROC classification scores achieved with the models trained and tested on a balanced dataset obtained by undersampling the inhibitory ISI series class (∼53000 samples in the training set total and ∼44000 samples in the validation set). We were generally able to achieve accuracy higher than 80% even for a linear model (logistic regression) trained on samples from the full 794-dimensional feature space.

To further perform selection of *tsfresh*-extracted features, we first trained a random forest classifier on the full dataset and looked at the resulting feature importance values in the trained model. We selected the top 15 features according to the feature importance ranks and used dimensionality reduction techniques on this reduced 15-dimensional dataset to visualize the data structure with respect to excitatory/inhibitory labels in two dimensions. Results are shown in Fig. 4 for both a linear (Principal Component Analysis, PCA) and two nonlinear (Uniform Manifold Approximation and Projection, UMAP [47] and t-distribution Stochastic Neighbor Embedding, t-SNE [48]) low-dimension embedding algorithms. In both cases there is a clear spatial separation of the excitatory-cell and inhibitory-cell classes of spike trains; it can be seen that classes can be linearly separated with good precision even in these two-dimensional embedding spaces. Furthermore, we trained a supervised version of UMAP on a certain subset of data samples (by providing class label information to the algorithm along with feature vectors) and applied the trained UMAP transform to another test subset of data samples. Results of this transformation are shown in Fig. 4 as well, one can see that supervised UMAP learned a quite effective two-dimensional embedding of the data for excitatory vs. inhibitory class separation. It is to

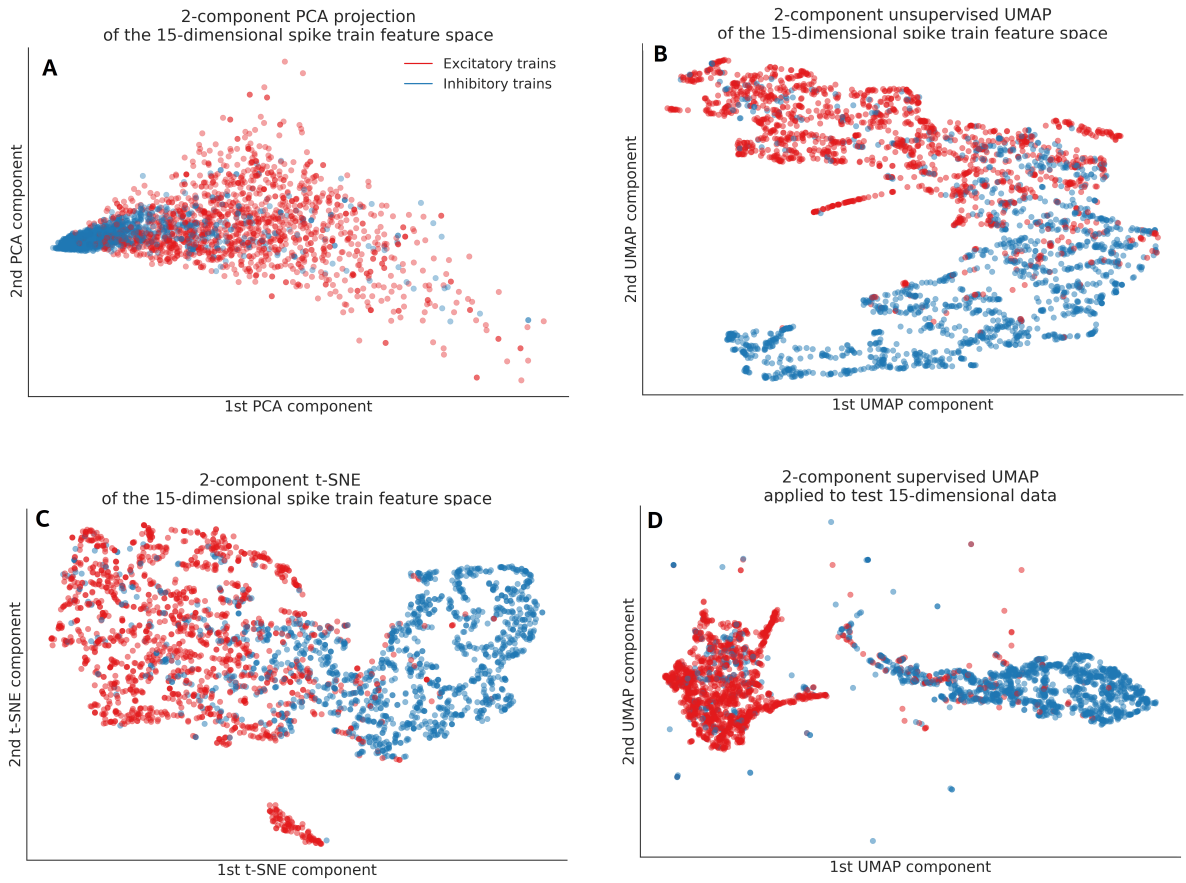

Figure 4: Two-dimensional embeddings of the (15-dimensional) selected-*tsfresh*-feature space using (A) PCA, (B) unsupervised UMAP (C) t-SNE and (D) supervised UMAP embedding algorithms for inhibitory (points marked blue) vs. excitatory (points marked red) spike trains. For supervised UMAP, the transform was fitted on a train subset of the data (7000 samples) and applied a validation subset (3000 samples, shown on the figure). For all methods significant separation of excitatory vs. inhibitory spike train classes can be observed even in the two-dimensional embedding subspace.

be expected that adding these nonlinear UMAP features to the dataset can result in improved performance of classification algorithms. Results presented in Fig. 4 show transformations done by dimensionality reduction algorithms trained and applied to a subset of the full training dataset (consisting of 3000 samples total; except for supervised UMAP, for which 7000 train samples were used for fitting and 3000 test samples for transformation).

2. Neighbor-based models on raw ISI series

Next, we evaluated the performance of nearest-neighbor models with several different distance metrics trained on raw time series and how it compares to classification on engineered features. We trained the kNN model with the Minkowski metric ($p = 1$ and $p = 2$), DTW distance, Kolmogorov-Smirnov distance for ISI value distributions within the train and, finally, the SPIKE synchronization measure (for the latter we used implementation available in the *pyspike* package [12]). Fig. 5 shows achieved accuracy values on the validation dataset for each distance metric trained and validated on several random subsamples of the full dataset (3000 samples of both classes in both train and validation sets). Mean accuracy scores were generally found to exceed 70% for each distance metric used, and the Kolmogorov-Smirnov distance metric was found to perform well compared to the rest of the metrics even for the single nearest-neighbor-based predictions in this task. This means that effective classification of excitatory vs. inhibitory cell spike trains can be accomplished by considering the properties of the ISI value distribution in the spike train, not the exact ordering of ISI values in the series. In other words, random shuffling applied to the order of ISIs in the spike train series would not affect the results obtained with the KS distance metric.

3. BOW representation time series classification algorithms

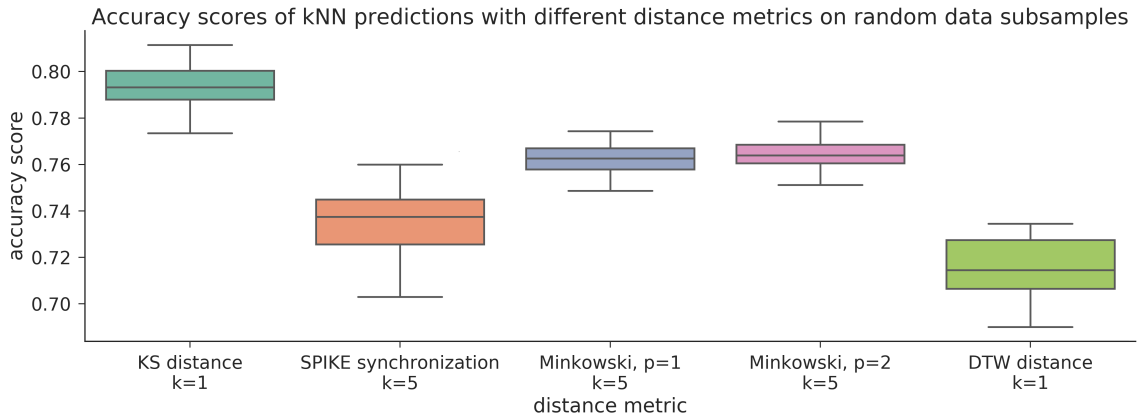

Figure 5: Accuracy score distributions of kNN classifiers trained and validated on random subsamples of the full excitatory vs. inhibitory dataset (6000 samples in the training set, 6000 samples in the validation set, balanced) depending on the distance metric used.

Finally, we examined performance of two bag-of-patterns time series classification methods: BOSSVS and SAX-VSM. We found that performance quality of both these algorithms strongly depends on the choice of their hyperparameter values, with accuracy scores ranging from the chance level to values comparable with other classification methods, depending on the exact choice of hyperparameters. We thus performed global hyperparameter optimization using the Parzen Estimator Tree method available in the *hyperopt* Python package. We collected hyperparameter values and corresponding accuracy scores on each iteration of the global search in order to evaluate how validation accuracy is distributed across the hyperparameter space. All possible hyperparameter values were considered during the initial global search, however results of the initial search revealed that, for some hyperparameters, particular value choices delivered consistently better accuracy scores. Therefore, we fixed some hyperparameter values for both algorithms: for SAX-VSM, we set `quantiles = 'empirical'`, `numerosity_reduction = False`, `use_idf = True`; for BOSSVS we set `quantiles = 'empirical'`, `norm_mean = False`, `norm_std = False`, `smooth_idf = True`, `sublinear_tf= False`. For the detailed description of what these parameters correspond to, please refer to the API documentation of the *pyts* Python package [32]. The remaining free hyperparameters which we searched over are: for SAX-VSM, `n_bins, window_size, smooth_idf, sublinear_tf`; for BOSSVS, `window_size, n_bins, variance_selection, variance_threshold, numerosity_reduction`. Fig. 6 shows distributions of validation accuracy scores for different hyperparameter values of BOSSVS and SAX-VSM algorithms evaluated on a balanced dataset (a random subsample of the full inhibitory vs. excitatory dataset was used, consisting of 3000 samples of both classes in both train and validation sets). BOSSVS algorithm was generally found to outperform SAX-VSM on this task, with BOSSVS validation accuracy scores being in the range of 75-80% (comparable to other classification methods), while most of obtained SAX-VSM accuracy scores were in the 50-65% interval. Overall, the BOSSVS classification method was found to give significantly better performance scores than SAX-VSM on this task.

WAKE/SLEEP network state classification from single neuron spiking patterns

In the previous section, we evaluated how time series classification methods perform on a cell type classification task (in particular, classification of principal cells vs. interneurons). However, our approach is not limited to classification of cell types. In order to demonstrate this, we evaluate classification performance between two distinct activity states of the neural circuit. We make use of the CRCNS fcx-1 dataset, which contains information about the time periods when mice were in an awake or asleep states. Moreover, particular sleep phase intervals (e.g. REM sleep vs. nonREM/SWS sleep) are also labelled. First, we approach the problem of WAKE vs. SLEEP (REM+nonREM) state classification. We extract spike train data from interneurons (both "inhibitory-definite" and "inhibitory-like" cells) at time intervals corresponding to WAKE and SLEEP phases of the recording. The resultant dataset contains spiking patterns of 118 cells, from which we take 70 cells for the training subset and the rest for the validation subset, so that 60% of the total ISI count is used for training. Figure 7 (left) shows mean ISI value distributions for inhibitory spike trains extracted during the episodes of WAKE and SLEEP activity states. One can see that these mean ISI distributions are quite similar, so that limiting a specific mean ISI interval is not necessary. We only limit the tail of the distribution by not considering the trains with mean ISI values > 400 ms. We repeat the same procedure for spike train extraction, with fixed number of ISIs $N_{\text{size}} = 200$ in each chunk generated by a rolling window of size 100 ISIs. We end up with

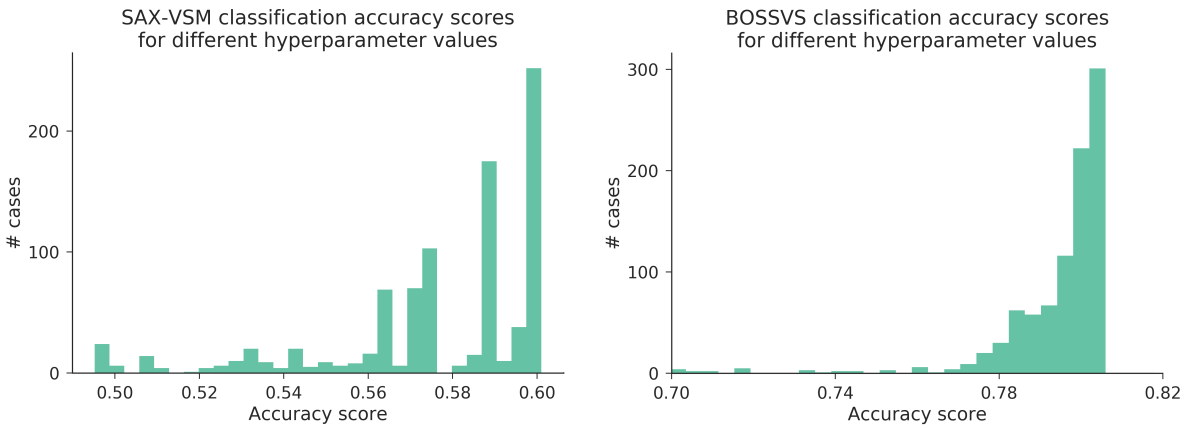

Figure 6: Distribution of accuracy scores achieved by SAX-VSM (left) and BOSSVS (right) classifiers on the excitatory vs. inhibitory trains dataset during iterations of the global hyperparameter search. Classifiers were trained and accuracy was calculated on a subsample of the full dataset consisting of 3000 samples from both classes in both training and validation subsets.

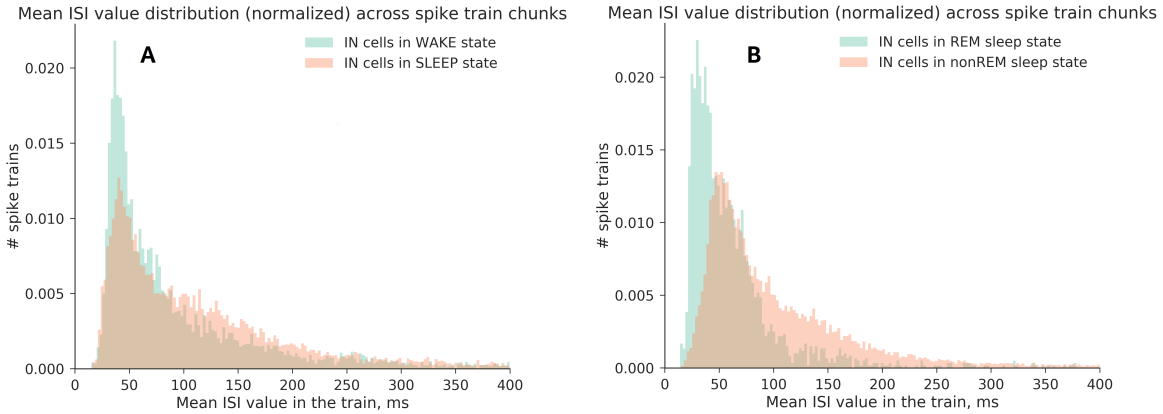

Figure 7: Mean ISI value distributions (normalized) extracted from inhibitory neuron spike train chunks in WAKE vs. SLEEP activity states (left) and REM sleep vs. nonREM sleep activity states (right)

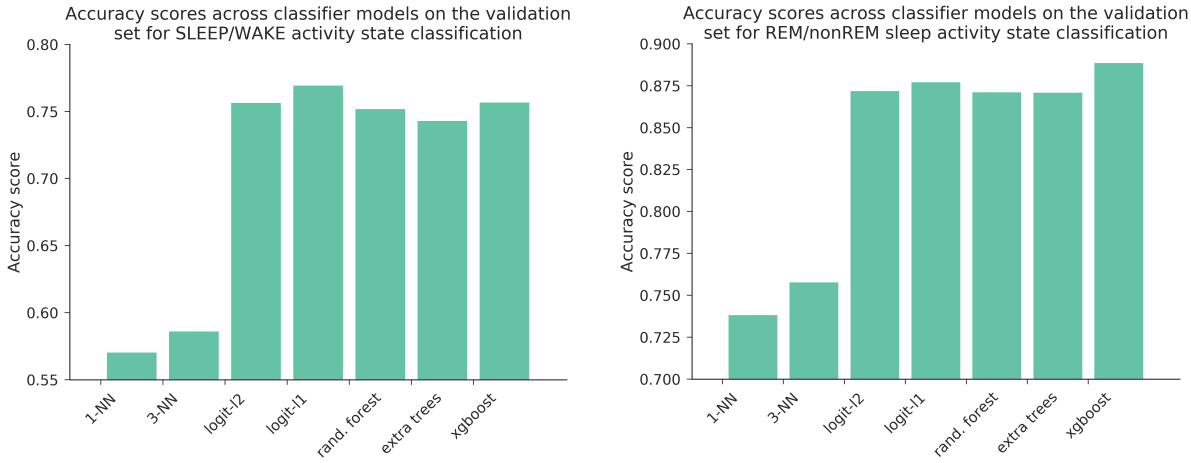

Figure 8: Validation accuracy scores achieved on neural circuit activity classification tasks (left: WAKE vs. SLEEP state classification, right: REM sleep vs. SWS sleep state classification) with different classifier models.

Figure 9: Spike train feature embeddings for WAKE (points marked red) vs. SLEEP (points marked blue) activity states of the neural circuit. Two-dimensional embeddings of the (15-dimensional) selected-*tsfresh*-feature space using (A) PCA, (B) unsupervised UMAP (C) t-SNE and (D) supervised UMAP embedding algorithms for spike trains corresponding to WAKE vs. SLEEP activity states.

$\sim$ 7900 spike train chunks in the WAKE state and $\sim$ 12200 chunks in the SLEEP state for the training dataset. Validation subset contains $\sim$ 4300 spike train chunks in the WAKE state and $\sim$ 9100 chunks in the SLEEP state. We then applied the same classification pipeline as was done for cell type classification: we calculated the full set of *tsfresh* features for each spike train chunk and performed standard scaling and removal of low-variance features. Dataset undersampling was performed to further use accuracy scores as performance measures for classifier algorithms, which resulted in the training set consisting of $\sim$ 7000 samples from both WAKE and SLEEP classes. We then trained several different classification models on these feature vectors to estimate the baseline accuracy scores that can be obtained with this approach. Achieved validation accuracy scores are shown in Fig. 8. We found that performance of kNN models on *tsfresh* feature vectors is significantly worse than for linear and decision-tree-based models. Furthermore, performance of linear models (logistic regression with l_1 and l_2 regularizations) was found to be comparable and slightly better compared to tree-based models (e.g. random forests and gradient boosting machines). In order to perform dimensionality reduction of the spike train data during WAKE/SLEEP states, we looked at the feature importance values for the trained random forest classifier and left the 15 features with largest importance ranks. As it was done for the cell type dataset, we applied several embedding algorithms (PCA, t-SNE, unsupervised and supervised UMAP) to visualize class separation in two dimensions. Indeed, good class separation can be observed for the WAKE vs. SLEEP activity state classes, as it can be seen in Fig. 9.

REM/nonREM sleep network state classification from single neuron spiking patterns

Finally, we looked at whether activity states corresponding to different sleep phases (e.g. REM and non-REM/SWS) could be predicted from properties of individual spike trains. The procedure for data extraction was the same as for the WAKE/SLEEP data, but spike trains were taken from specific time intervals which were detected as REM/nonREM sleep phases. After undersampling was performed (SWS sleep state was initially a dominant class), we ended up with $\sim$ 5000 spike train chunks of size 200 ISIs in the train set, and $\sim$ 3800 spike train chunks in the validation set. Validation accuracy scores achieved on *tsfresh* extracted features for the REM vs. SWS sleep classification task are shown in Fig. 8. Overall, good classification accuracy up to $\sim$ 90% was

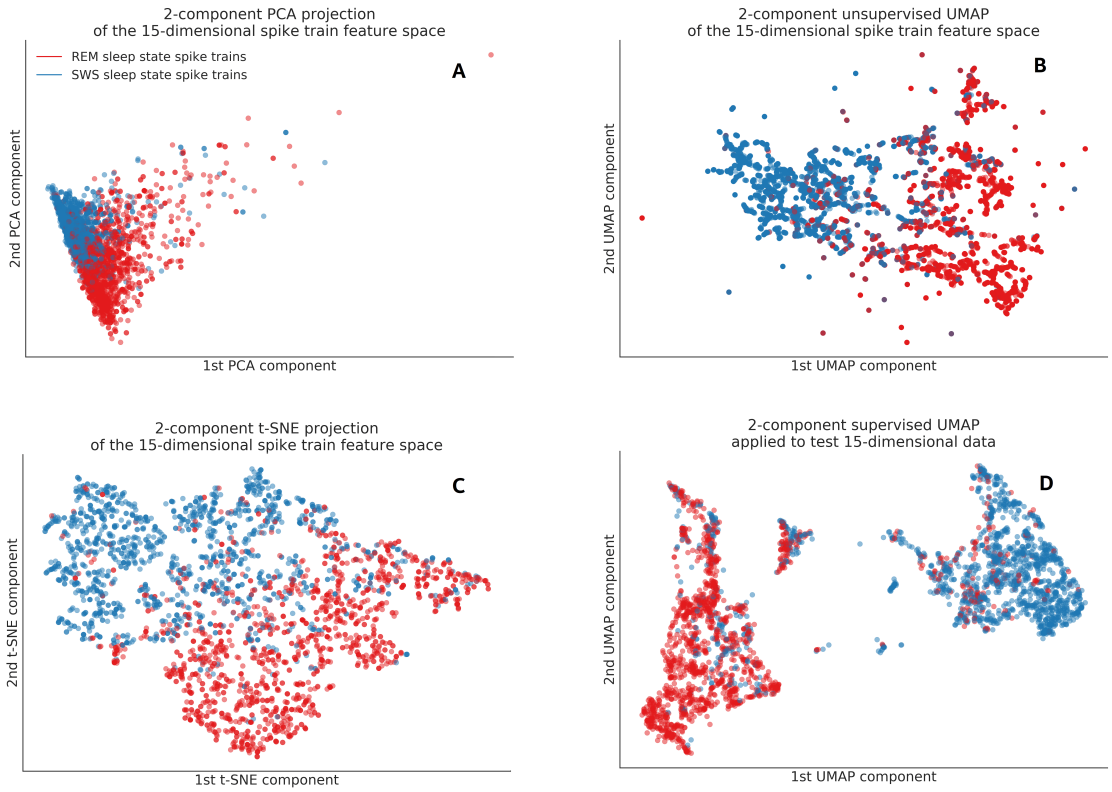

Figure 10: Spike train feature embeddings for REM sleep (points marked red) vs. nonREM/SWS sleep (points marked blue) activity states of the neural circuit. Two-dimensional embeddings of the (15-dimensional) selected-*tsfresh*-feature space using (A) PCA, (B) unsupervised UMAP (C) t-SNE and (D) supervised UMAP embedding algorithms for spike trains corresponding to REM sleep vs. SWS sleep activity states.

achieved, and linear models (logistic regression) were found to be comparable in accuracy to tree-based models such as random forest and gradient boosting. We also applied two-dimensional embedding methods similarly to what was done for WAKE/SLEEP data, good class separation in the two-dimensional embedding subspace can be seen in Fig. 10.

Conclusions

In summary, we have demonstrated good performance of a range of time series analysis models applied to spiking pattern activity classification. The methods described here are very general and can be applied to various tasks from cell type classification to neural circuit's functional state classification. The latter can cover both different functional states of the same circuit (e.g. WAKE/SLEEP state of the cortex) or disease-induced activity states vs. healthy controls. Detection of disease-driven neural activity might be of high importance in this context, and usage of an ensemble of various predictive models might enable precise detection of such activity patterns. In general, we expect that the approaches discussed here could be applied to a range of classification/clustering tasks involving spiking activity in a straightforward way. In this study, we were able to achieve good classification results in both cell-type (excitatory vs. inhibitoty cells) and circuit-state classification (awake activity vs. activity in different sleep phases) tasks using open-access spiking activity data obtained from frontal cortices of rats. Further work will be focused on incorporating more data from different neuron types and different brain areas. A particularly interesting problem in this context is investigating how the structure of the spike train data in the feature vector space depends on the brain area/cell type, and which spike train representation (embedding) is best at encoding the crucial features which differentiate spiking activity recorded across the brain. These are the topics which are going to be tackled in future work.

References

[1] M. Pachitariu, C. Stringer, S. Schröder, M. Dipoppa, L. F. Rossi, M. Carandini, and K. D. Harris, "Suite2p: beyond 10,000 neurons with standard two-photon microscopy," *Biorxiv*, p. 061507, 2016.

[2] D. Tsai, E. John, T. Chari, R. Yuste, and K. Shepard, "High-channel-count, high-density microelectrode array for closed-loop investigation of neuronal networks," in *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pp. 7510–7513, IEEE, 2015.

[3] P. Berens, J. Freeman, T. Deneux, N. Chenkov, T. McColgan, A. Speiser, J. H. Macke, S. C. Turaga, P. Mineault, P. Rupprecht, *et al.*, "Community-based benchmarking improves spike rate inference from two-photon calcium imaging data," *PLoS computational biology*, vol. 14, no. 5, p. e1006157, 2018.

[4] J. J. Jun, C. Mitelut, C. Lai, S. Gratiy, C. Anastassiou, and T. D. Harris, "Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction," *bioRxiv*, p. 101030, 2017.

[5] P. Yger, G. L. Spampinato, E. Esposito, B. Lefebvre, S. Deny, C. Gardella, M. Stimberg, F. Jetter, G. Zeck, S. Picaud, *et al.*, "Fast and accurate spike sorting in vitro and in vivo for up to thousands of electrodes," *BioRxiv*, p. 067843, 2016.

[6] J. I. Glaser, R. H. Chowdhury, M. G. Perich, L. E. Miller, and K. P. Kording, "Machine learning for neural decoding," *arXiv preprint arXiv:1708.00909*, 2017.

[7] A. S. Benjamin, H. L. Fernandes, T. Tomlinson, P. Ramkumar, C. VerSteeg, R. H. Chowdhury, L. E. Miller, and K. P. Kording, "Modern machine learning as a benchmark for fitting neural responses," *Frontiers in computational neuroscience*, vol. 12, 2018.

[8] T. Tezuka, "Multineuron spike train analysis with r-convolution linear combination kernel," *Neural Networks*, vol. 102, pp. 67–77, 2018.

[9] M. D. Humphries, "Spike-train communities: finding groups of similar spike trains," *Journal of Neuroscience*, vol. 31, no. 6, pp. 2321–2336, 2011.

[10] M. v. Rossum, "A novel spike distance," *Neural computation*, vol. 13, no. 4, pp. 751–763, 2001.

[11] J. D. Victor and K. P. Purpura, "Metric-space analysis of spike trains: theory, algorithms and application," *Network: computation in neural systems*, vol. 8, no. 2, pp. 127–164, 1997.

[12] M. Mulansky and T. Kreuz, "Pyspike—a python library for analyzing spike train synchrony," *SoftwareX*, vol. 5, pp. 183–189, 2016.

[13] T. Tezuka, "Spike train pattern discovery using interval structure alignment," in *International Conference on Neural Information Processing*, pp. 241–249, Springer, 2015.

[14] J. Jouty, G. Hilgen, E. Sernagor, and M. Hennig, "Non-parametric physiological classification of retinal ganglion cells," *bioRxiv*, p. 407635, 2018.

[15] P. Charlesworth, E. Cotterill, A. Morton, S. G. Grant, and S. J. Eglen, "Quantitative differences in developmental profiles of spontaneous activity in cortical and hippocampal cultures," *Neural development*, vol. 10, no. 1, p. 1, 2015.

[16] M. Li, F. Zhao, J. Lee, D. Wang, H. Kuang, and J. Z. Tsien, "Computational classification approach to profile neuron subtypes from brain activity mapping data," *Scientific reports*, vol. 5, p. 12474, 2015.

[17] X. Jia, J. Siegle, C. Bennett, S. Gale, D. Denman, C. Koch, and S. Olsen, "High-density extracellular probes reveal dendritic backpropagation and facilitate neuron classification," *bioRxiv*, p. 376863, 2018.

[18] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[19] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," *Pattern Recognition*, vol. 44, no. 9, pp. 2231–2240, 2011.

[20] B. D. Fulcher and N. S. Jones, "hctsa: A computational framework for automated time-series phenotyping using massive feature extraction," *Cell systems*, vol. 5, no. 5, pp. 527–531, 2017.

[21] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)," *Neurocomputing*, 2018.

[22] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[23] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint arXiv:1802.03888*, 2018.

[24] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.

[25] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[26] P. Schäfer and M. Högqvist, "Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets," in *Proceedings of the 15th International Conference on Extending Database Technology*, pp. 516–527, ACM, 2012.

[27] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[28] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 637–646, ACM, 2017.

[29] P. Senin and S. Malinchik, "Sax-vsm: Interpretable time series classification using sax and vector space model," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pp. 1175–1180, IEEE, 2013.

[30] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[31] P. Schäfer, "Scalable time series classification," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, pp. 1273–1298, 2016.

[32] J. Faouzi, "pyts: a Python package for time series transformation and classification," May 2018.

[33] J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle, "Recurrence plots of dynamical systems," *EPL (Europhysics Letters)*, vol. 4, no. 9, p. 973, 1987.

[34] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," *arXiv preprint arXiv:1506.00327*, 2015.

[35] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[37] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[38] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "Lstm fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[39] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel, "Learning to diagnose with lstm recurrent neural networks," *arXiv preprint arXiv:1511.03677*, 2015.

[40] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 4580–4584, IEEE, 2015.

[41] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810, 2015.

[42] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?," *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[43] J. L. Teeters and F. T. Sommer, "Crcns. org: a repository of high-quality data sets and tools for computational neuroscience," *BMC Neuroscience*, vol. 10, no. S1, p. S6, 2009.

[44] B. Watson, D. Levenstein, J. Greene, J. Gelinas, and G. Buzsaki, "Multi-unit spiking activity recorded from rat frontal cortex (brain regions mpfc, ofc, acc, and m2) during wake-sleep episode wherein at least 7 minutes of wake are followed by 20 minutes of sleep. crcns.org," 2016.

[45] B. O. Watson, D. Levenstein, J. P. Greene, J. N. Gelinas, and G. Buzsáki, "Network homeostasis and state dynamics of neocortical sleep," *Neuron*, vol. 90, no. 4, pp. 839–852, 2016.

[46] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, NY, USA:, 2001.

[47] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[48] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.