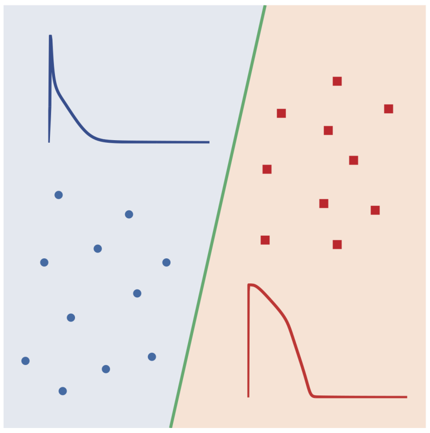
Biomedical Data Science Lab
Fall 2019

Classification of cardiomyocites based on their action potentials



Benjamín Béjar Haro
Assistant Research Professor
Department of Biomedical Engineering
319B Clark Hall, Johns Hopkins University
E-mail: `bbejar@jhu.edu`

1 Lab Description

In this lab experience we will be looking at the problem of classifying cardiac cells by looking at their *Action Potentials* (APs). For the purpose of this task we will be using synthetically generated APs following the models in [1, 2].

1.1 Tasks

Task 1. *[35 points] (Pre-processing and feature extraction)*
In this first part we will be normalizing the data and extracting hand-crafted features that will be later used for the classification task. Load the dataset `Adult_samples.mat` that contains examples of adult atrial and ventricular action potentials generated using the models in [1, 2]. For each of the two classes there are 1000 samples generated with a sampling rate of $f_s = 500$ Hz.

(a) *(15 points) Data preparation and normalization.*
Split the data between test and training sets by randomly selecting 10% of the points as your training set. Make sure the two classes are well represented in the training set (e.g., use the same number for both). Normalize the data so that each AP has zero resting potential and unit maximum amplitude. Create an array of corresponding labels for the data points. For ventricular type use the class label $+1$ and -1 for atrial type. Make two plots displaying your normalized training data for each of the classes. Use time units for the horizontal axis.

(b) *(20 points) Hand-crafted features.*
The *Action Potential Duration* (APD) at $x\%$ is defined as the time it takes to reduce the maximum amplitude of the AP to $x\%$ of its value. Write a function that computes APD at a given percentage $x \in [0, 1]$. Compute also the *Average of the Action Potential* (AAP) and build two-dimensional features by concatenating APD@0.5 and AAP. Make a scatter plot of the training data using these two features. Use different colors and/or markers to represent each class.

- Based on your scatter plot, is the training data using the above features linearly separable? Why?

Task 2. *[55 points] (Classification)*
In this task we will be using hand-crafted features and learned features from the data in order to classify cardiac cells based on their action potentials. You will be asked to implement a 1-Nearest-Neighbor (NN) classifier as well as a simple two-layer neural network.

(a) *(15 points) Nearest-neighbor classifier.*
Implement a 1NN classifier using the Euclidean distance. A 1NN classifier works as follows: Given your training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where N is the number of training samples, $\boldsymbol{x}_i \in \mathbb{R}^D$ is a feature vector and $y_i \in \{-1, 1\}$ its associated label, and a novel sample $\boldsymbol{x}$, the 1NN classifier assigns to $\boldsymbol{x}$ the same label as its closest point in the training set. That is, the estimated label $\hat{y}$ of $\boldsymbol{x}$ is such that:

$$\hat{y}(\boldsymbol{x}) = y_{k^*}, \quad k^* = \underset{i \in \{1,\dots,N\}}{\arg\min} \|\boldsymbol{x}_i - \boldsymbol{x}\|_2. \tag{1}$$

- Compute and display the classification accuracy over the test set using the handcrafted training features of Task 1.

(b) *(40 points) Two-layer Neural Network.*
Implement a two-layer neural network classifier of the form:

$$\widehat{y} = \text{sign}\left(f_\theta(\phi(\boldsymbol{x}))\right), \quad f_\theta(\phi(\boldsymbol{x})) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b, \quad \theta = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix},$$

where $f_\theta(\cdot)$ is a linear prediction function (*i.e.,* classification layer) parametrized by $\theta = [\boldsymbol{w}^T, b]^T$. The feature extraction part of the network consists of a linear layer followed by a ReLu (rectified linear unit) non-linearity:

$$\phi(x) = \text{ReLu}\left(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1\right), \quad \text{ReLu}(x) = \begin{cases} x & x > 0 \\ 0 & \text{else} \end{cases}.$$

In order to find the network's parameters $\Theta = \{\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{w}, b\}$ minimize the following *regularized empirical risk* using PyTorch:

$$\min_{\Theta} \underbrace{\frac{1}{N}\sum_{i=1}^{N} L\big(f_\theta(\phi(\boldsymbol{x}_i)), y_i\big) + \lambda\big(\frac{1}{w}\|\boldsymbol{w}\|^2 + \frac{1}{W}\|\boldsymbol{W}_1\|^2\big)}_{C(\Theta)}$$

where the loss function $L(f, y) = \|y - f\|_2^2)$ is the quadratic (square) loss, and w, W, are the number of elements in $\boldsymbol{w}$ and $\boldsymbol{W}$, respectively.

1. Define a network model in PyTorch according to the definition above. For that purpose you can use `‘torch.nn.Sequential‘`. Follow this example to learn how to use it.
2. Run a gradient descent algorithm to minimize the cost function using $\lambda = 1$. Carefully choose the step-size and number of iterations until you see the method converges (i.e., the cost function gets to a "plateau").
3. Make a scatter plot of the learned features (i.e., prior to classificaiton layer) by your network model. Has your model learned features that are linearly separable? Display in the scatter plot the decision boundary that you have learned. Compute the classification accuracy over the test set.
4. Plot the weights of the learned linear layer. What has your network learned?

1.2 Python cheat sheet

Here goes a list of relevant python functions that you might want to use for this lab experience. This list in only intended to be illustrative. For a detailed description of the different functions, please refer to the particular package documentation.

Command	Description
`import numpy as np`	Imports numpy module with name np.
`import matplotlib.pyplot as plt`	Imports numpy module with name np.
`import random as rnd`	Imports random number generator module with name rnd.
`scipy.io.loadmat('matfile.mat')`	Load Matlab `.mat` file and puts it into a dictionary.
`plt.plot(x,y)`	Plot of y over x.
`plt.scatter(x,y)`	Scatter plot of y over x.
`rnd.sample(population,n)`	Samples n elements at random from population array.
`np.concatenate(x,y,axis)`	Concatenates x and y along specified axis.
`np.argmax(x)`	Returns the index where x is maximum.
`np.sign(x)`	Retruns sign of x (entry-wise).
`np.dot(x,y)`	Computes the inner product of x and y.
`np.outer(x,y)`	Outer (tensor) product of x and y.
`np.matmul(x,y)`	Matrix multiplication of x and y.
`np.linalg.eig(x)`	Returns the eigenvalues/eigenvectors of x.
`np.mean(x), np.std(x)`	Computes mean/std value of x.
`np.where(x)`	Returns non-zero support of x.
`np.arange(N)`	Gives a list of numbers $0, \ldots, N-1$.
`np.abs(x)`	Computes the absolute value of x.

References

[1] A. Nygren, C. Fiset, L. Firek, J. Clark, D. S Lindblad, R. Clark, and W. R Giles. Mathematical model of an adult human atrial cell : The role of k+ currents in repolarization. *Circulation research*, 82:63–81, 01 1998.

[2] T. O'Hara, L. Virág, A. Varró, and Y. Rudy. Simulation of the undiseased human cardiac ventricular action potential: Model formulation and experimental validation. *PLOS Computational Biology*, 7(5):1–29, 05 2011.