Action potential model for neurons

Lucas Mariétan

March 7, 2024

1 Introduction

This documents explain my choices and the steps I went through to build a model for determination of species, brain area and cell type based on the characterization of the first action potential of the voltage trace of neurons.

2 Code

My code needs at least Python 3.10. In order to install channelpedia_api package you can follow the tutorial on https://bbpteam.epfl.ch/repository/devpi/bbprelman/release/channelpedia-api/stable. All the other packages can be installed with pip install or the package manager of Pycharm, smote_variants package only works with pip install. My code is available on Github: https://github.com/lmarieta/BlueBrain.git.

3 Data

3.1 Raw data

Each neuron cell was tested over many repetitions, with different current stimuli (sweep) and with different test protocols. The test protocols used for the modelization are APWaveform, IDRest and IV, each with different stimulus profile. Raw data consist of voltage measurement over time. The routine used for vizualisation of raw data is `plot_raw_traces.py`. In order to use it, you need raw data files as .mat files, with `CellXX_Y.mat` in a single folder, with XX and Y the cell numbers. Stimuli data are extracted from the analyzed cell files `aCellXX_Y.json`. Please note that the acell files have been converted to JSON format to be readable by the channelpedia_api library. Cell information is retrieved from a cell list file, for example `CellList30-May-2022.csv`. Some cells are also excluded from the visualization using `outliers.txt` (i.e. Maurizio), with the cells to exclude written as *protocol* $APWaveform$, *cell* 309_1, *repetition* 0, *sweep* 1.

3.2 Feature extraction

Features are extracted from these protocol measurements using Aecode. Features used for prediction are given in Table 2. Other features useful for data pre-processing are *stim* and *spikecount*, where

Feature name	Description
AP_begin_voltage	Voltage before the start of the action potential
AP_amplitude	Difference between the peak of the action potential and AP_begin_voltage
AP_half_wdith	Width at half amplitude
min_AHP_voltage	Minimum after AP (don't know where exactly)
IV_peak_m	Intrinsic resistance of a cell at peak(?)
IV_steady_m	Intrinsic resistance of a cell in steady-state

Table 1: Features used to predict the class of a cell. The resistance of a cell is the same for all traces of a given repetition.

Protocol	Nr. of traces (at least 1 missing value)	N° of traces
IDRest	133	1640
APWaveform	95	716
IV	0	171

Table 2: Missing values in aecode replaced by the median of the group (species, brain area and cell type). 15 cells (Rat Cortex PC-L5) were not tested with IV protocol and therefore the value was replaced by the median of the corresponding group. One cell had no value for the extracted features and was therefore removed from the analysis.

stim refers to current stimulus, constant for the protocol used and *spikecount* counts how many action potentials occur during the test protocol. I use `preprocessing.py` to extract features from acell files `aCellXX_Y.json`. `get_ap_index.py`, `get_features.py`, `get_protocols.py` and `get_trace_indices.py` are used to set the test protocol, the features to be extracted, the action potentials to be extracted and the sweeps to be extracted.

The traces extracted for in my model are the following: first action potential, all sweeps with a spikecount above 0, first repetition and features and protocols as described above. The repetition index can be directly passed as a parameter to the preprocessing routine as it does not depend on the protocol. A cell is included in the analysis if there exists a json acell file and if the cell name is given in the `CellList30-May-2022.csv` file.

3.3 Oversampling

I decided to use an oversampling technique to balance our classes and increase the number of data points for training of the machine learning algorithms (otherwise the model could systematically favour the dominant class to improve the overall performance). I tried a whole range of techniques and the one yielding the best performance was the Synthetic Minority Oversampling TEchnique (SMOTE) [1]. Oversamples are generated by taking the difference between the feature vector under consideration and its nearest neighbor, multiplying this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features.

This technique is only applied for training and not for testing or validating the machine learning algorithm.

4 Models

This section discuss three models to predict the classes based on each first action potential measurement or on cell values: multinomial logistic regression, XGBoost and neural networks. I've implemented other models but they do not perform as well in the case described below. However it is important to notice that other models might perform better with different set of features, protocols, etc... so it might be worth trying at least random forest when changing the inputs of the model. XGBoost is quite fast, but logistic regression and in particular neural network ('custom_nn' in the code) can be much slower, so if possible try to run the neural network with a GPU.

The performance metric used to measure model performance is the F1-score. The F1-score is the harmonic mean of the precision and the recall, where the precision is the ratio of the number of correct prediction over all predictions and the recall is the ratio of correct prediction over all elements of the class we want to predict. A potential downside of this metric is that it gives equal importance to precision and recall, maybe in our application we want something different. This metric is in general well-suited for imbalanced classes as we have here.

All my models are trained with a 80%-20% split between training + validation data and test data. The split is made along the cell names so the trace of a cell cannot be in training + validation and test sets. I also tried to keep the same distribution of each class in each sub-sets. The training + validation set is also splitted 80-20 in the same fashion in a training and validation sets respectively. I create 5 random combinations of training and validation sets within the 80% of the original dataset and find the best performing set of hyperparameter across all these combinations on the validation set(for the neural network I train on less epochs for this step to speed up a bit the process). I then

take these best hyperparameters and train the initial training + validation set and fit to the test set. The performance reported is the performance on the test set.

4.1 Spike analysis

This section presents the models I use to predict the class based on each individual traces first action potential properties. There are therefore many entries per cell. Each feature is either a single feature for each cell, such as the resistance or different for every traces.

4.1.1 Multinomial logistic regression

As in other forms of linear regression, multinomial logistic regression uses a linear predictor function $f(k, i)$ to predict the probability that observation i has outcome k, of the following form:

$$f(k,i) = \sum_j \alpha_{j,k} x_{j,i} \tag{1}$$

Interaction terms can also be present:

$$f(k,i) = \sum_j \alpha_{j,k} x_{j,i} + \sum_j \sum_m \beta_{jm,k} x_{j,i} x_{m \neq j,i} \tag{2}$$

Here we have two-by-two interaction terms but more complex models can be built with higher order interaction terms, for example combine three features in a single term. In my case I took a degree of three, which means three features can be combined in a single term.

Considering K classes and an action potential characterized by the features $\boldsymbol{X}_i$, $i = 1...n$, the probability that the label Y_i corresponds to the class k can be written as:

$$Pr(Y_i = k) = \frac{e^{\sum_{u=1}^{n} \beta_u^j X_u + \sum_{u=1}^{n} \sum_{v \neq u}^{n} \beta_{uv}^j X_u X_v + \sum_{u=1}^{n} \sum_{v \neq u}^{n} \sum_{m \neq u,v}^{n} \beta_{uvm}^j X_u X_v X_m}}{1 + \sum_{j=1}^{K-1} e^{\sum_{u=1}^{n} \beta_u^j X_u + \sum_{u=1}^{n} \sum_{v \neq u}^{n} \beta_{uv}^j X_u X_v + \sum_{u=1}^{n} \sum_{v \neq u}^{n} \sum_{m \neq u,v}^{n} \beta_{uvm}^j X_u X_v X_m}} \tag{3}$$

The parameters for the logistic regression are the following: random search over param_dist = 'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear'], 'max_iter': [100, 200, 300, 400, 500, 600] and interaction terms of degree 3. Best hyperparameters are ('C'=100, 'penalty'='l1', 'max iter'=400). Training data is oversampled with the SMOTE algorithm. Data are not scaled

I obtain a F1-score of 0.85. The confusion matrix is shown in Fig.1.

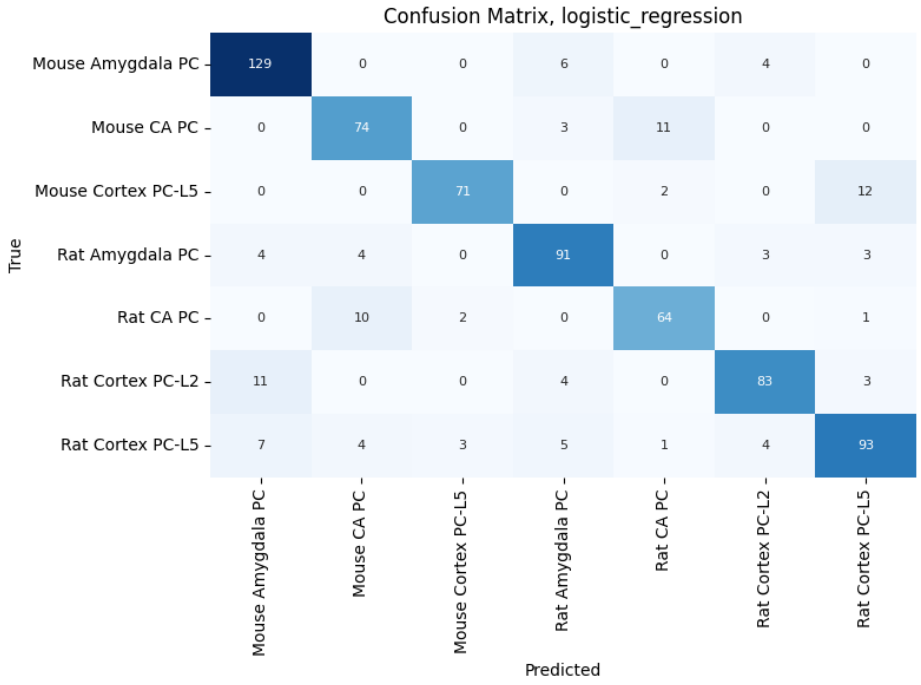

Figure 1: Class prediction confusion matrix. Features used for prediction: IV_steady_m, IV_peak_m, min_AHP_voltage, AP_half_width, AP_amplitude, AP_begin_voltage.

4.1.2 XGBoost

XGBoost is a decision tree ensemble algorithm [2], which means that it combines multiple algorithms to obtain a better model. It performs much better than the logistic regression and is particularly suitable for classification tasks. The downside is that there is no easily understandable reasoning explaining why a data point belongs to a given class.

I performed a random search over the following parameters to find the best model: he model is tested with the following parameters used for search: random search over param_dist = 'objective': ['multi:softmax'], 'num_class': [7], 'max_depth': [3, 5, 7], 'learning_rate': [0.01, 0.1, 0.2], 'n_estimators': [50, 100, 200], 'subsample': [0.8, 0.9, 1.0], 'colsample_bytree': [0.8, 0.9, 1.0], 'gamma': [0, 1, 5], with 10 iterations 5 cross-validations fold and a 'f1-weighted' score to select the best set of hyperparameters ('max_depth'=7, 'learning_rate'=0.2, 'n_estimators'=50, 'subsample'=0.8, 'colsample_bytree'=0.8, 'gamma'=0). A standard scaler is fitted to the training data and then used to transform the test data. A label encoder is used to fit the label data (i.e. classes) and then used to transform the test data.

With this model, I obtain a F1-score of 0.96. The confusion matrix is shown in Fig.2.

An important note is that the F1-score decreases to 0.72 without resistances. However now a trace of a cell cannot be in both training and test set (or validation) so there should not be leakage of information from the training set to the test set which would introduce bias in the results (as we have seen when I introduced the resistance at first).

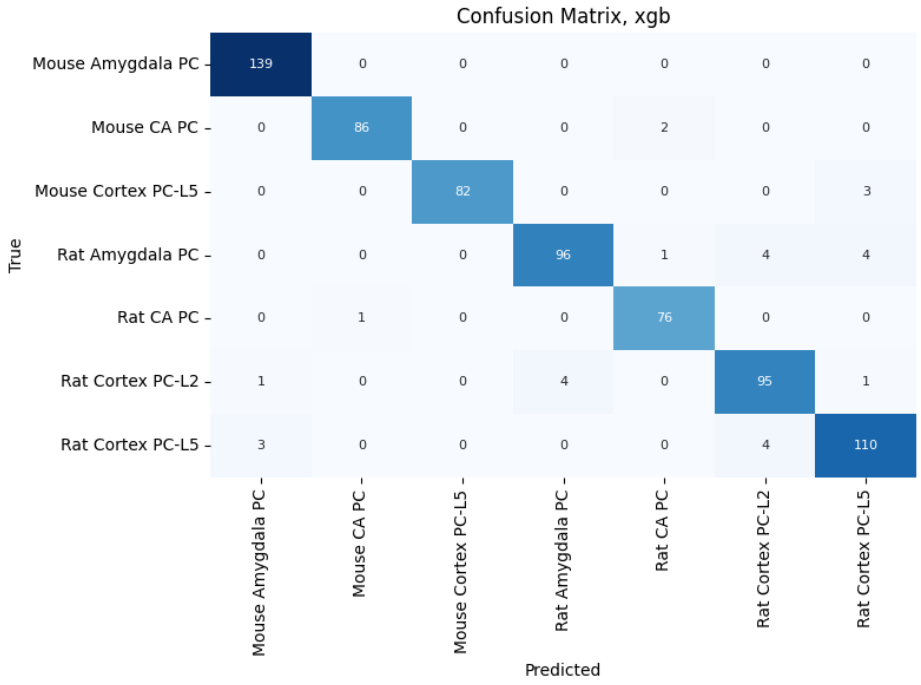

Figure 2: Class prediction confusion matrix. Features used for prediction: IV_steady_m, IV_peak_m, min_AHP_voltage, AP_half_width, AP_amplitude, AP_begin_voltage.

4.2 Cell analysis

This section presents the models I use to predict the class based on each cell properties, as opposed to action potential properties previously. In this case, the dataset is comprised of one entry per cell. Each feature is either a single feature for each cell, such as the resistance or averaged over all traces of each cell.

4.2.1 Neural network

The confusion matrix in Fig. 3 compares the class of each cell to the prediction made with a neural network. The corresponding F1-score is 0.71. The parameters are the following: epochs: 200, early stops when the training loss does not improve after 10 epochs, random search over param_dist = "optimizer__learning_rate": [0.0001, 0.001, 0.1], "dropout_rate": [0, 0.1, 0.5], "num_hidden_layers": [1, 2, 3, 5] with 10 iterations 5 cross-validations fold and a 'f1-weighted' score to select the best set of hyperparameters ("optimizer__learning_rate"=0.0001, "dropout_rate"=0.1, "num_hidden_layers"=5), each neuron layer having 64 units and each hidden layer is followed by a dropout layer, activation function is 'relu' for each layer except for the output layer, which is 'softmax', the optimizer is 'Adam' with a 'categorical_cross_entropy' loss function. Training data is oversampled with the SMOTE algorithm. A standard scaler is fitted to the training data and then used to transform the test data. A label encoder is used to fit the label data (i.e. classes) and then used to transform the test data. The best score achieved with this model is F1 = 0.82 by replacing the mean by the median of all traces to get the intrinsic value of a cell, see Fig.4.

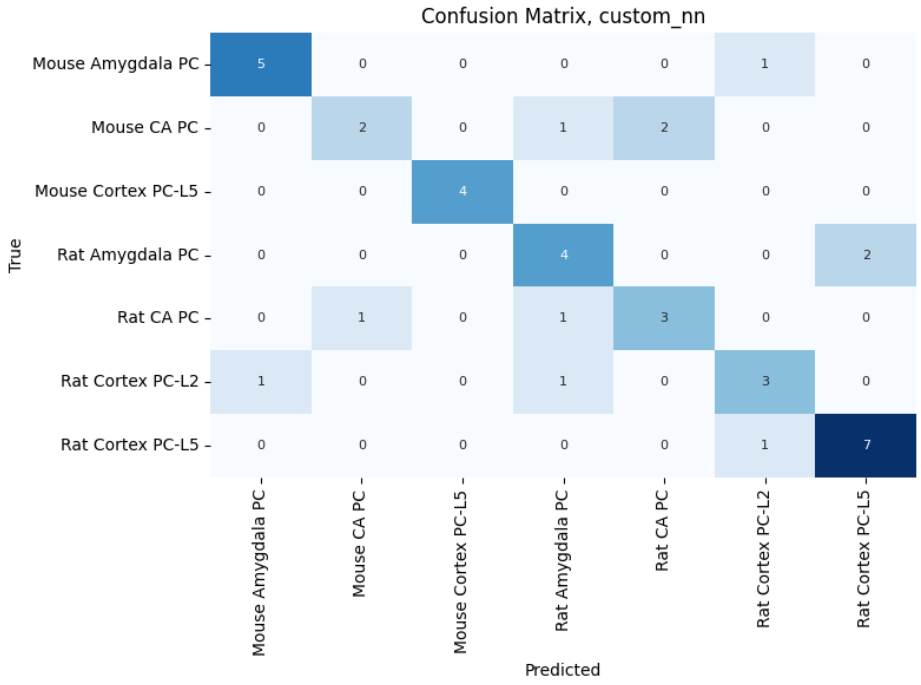

Figure 3: Class prediction confusion matrix. Features used for prediction: IV_steady_m, IV_peak_m, min_AHP_voltage, AP_half_width, AP_amplitude, AP_begin_voltage. Mean used to compute cell value.

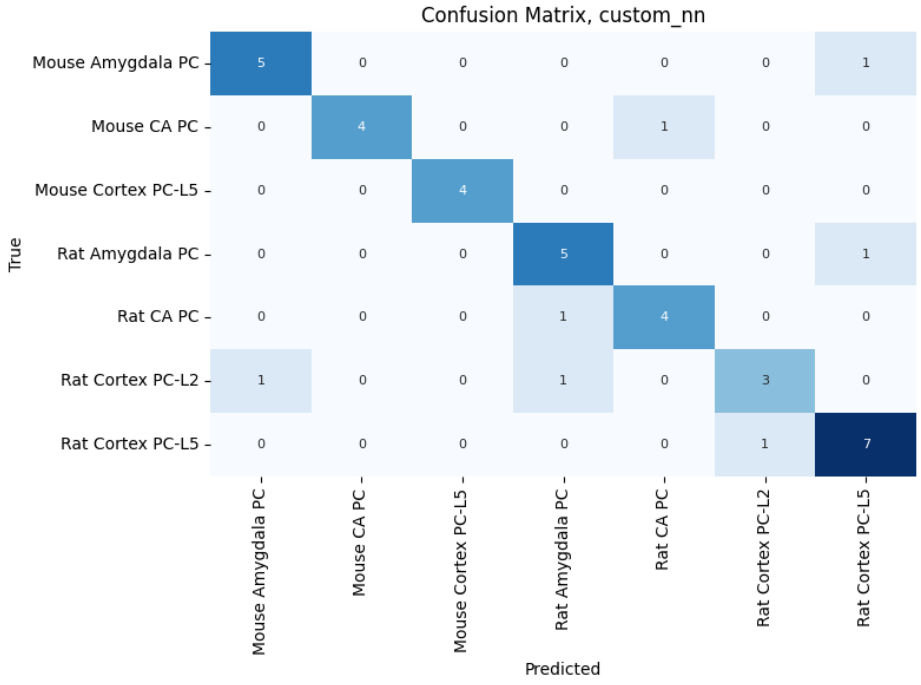

Figure 4: Class prediction confusion matrix. Features used for prediction: IV_steady_m, IV_peak_m, min_AHP_voltage, AP_half_width, AP_amplitude, AP_begin_voltage. Median used to compute cell value.

4.2.2 Logistic regression

This logistic regresion model is only used with the median to compute the intrinsic value of a cell, in agreement with the performance improvement seen with the neural network in 4.2.1.

The parameters for the logistic regression are the following: random search over param_dist = 'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear'], 'max_iter': [100, 200, 300, 400, 500, 600], with 10 iterations 5 cross-validations fold and a 'f1-weighted' score to select the best set of hyperparameters ('C'=100, 'penalty'='l1', 'max_iter'=400) and no interaction terms. Training data is oversampled with the SMOTE algorithm. Data are not scaled. Confusion matrix is shown in Fig.5. The resulting F1-score is 0.77.

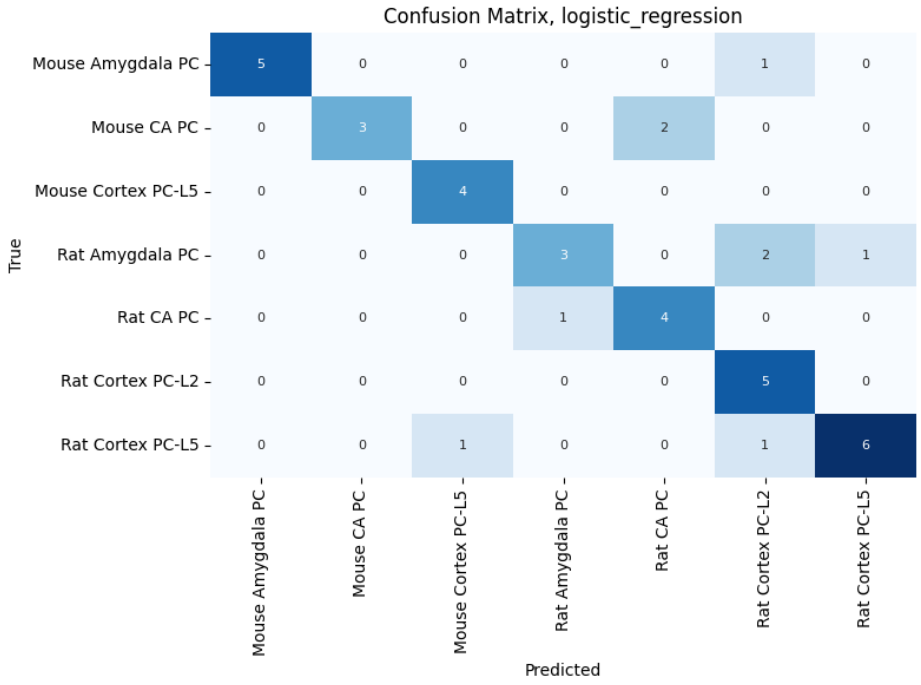

Figure 5: Class prediction confusion matrix. Features used for prediction: IV_steady_m, IV_peak_m, min_AHP_voltage, AP_half_width, AP_amplitude, AP_begin_voltage. Median used to compute cell value.

5 Next steps

What could be done next in my opinion is test the model on never seen cells, just to make sure that it generalizes well. We could also integrate trace resistance measurement. Adding features and protocols can be done easily in `ML_models.py` (check the input definition in main function). If you integrate other AP I would recommend to add 'ap_index' as a feature because the first AP is different from the others. This can be done by modifying the `get_ap_index.py` function. Another interesting thing we could do is monitor the performance as a function of the list of features.

References

[1] N. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *ArXiv* abs/1106.1813 (2002). URL: https://api.semanticscholar.org/CorpusID:1554582.

[2] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016). URL: https://api.semanticscholar.org/CorpusID:4650265.