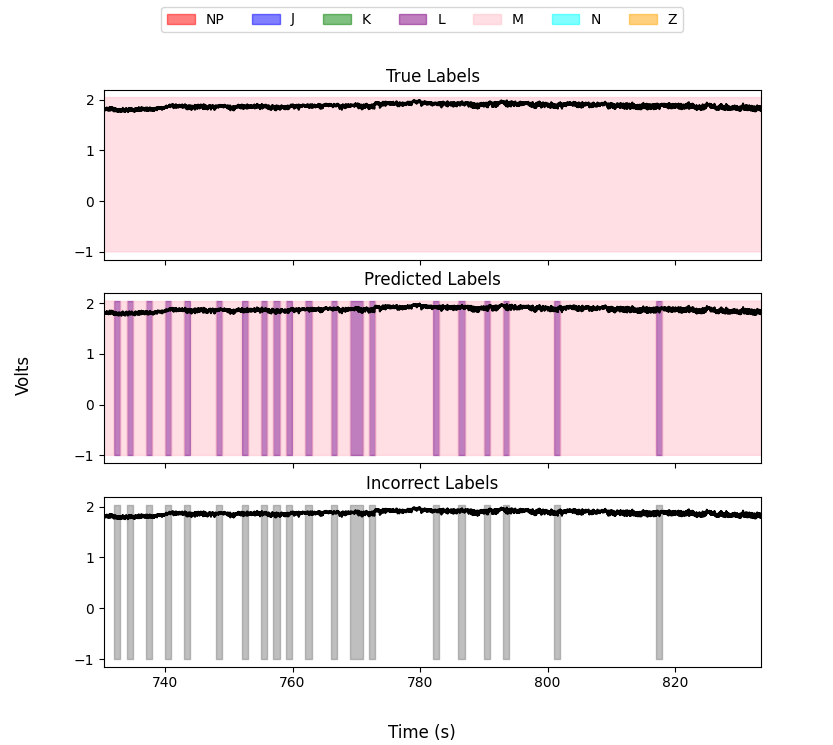
Background and Rationale

The outputs of running our models on EPG data currently exhibit issues like barcoding and missing transition states such as J and Z. While we could attempt to address these issues on the model side, fixing these problems in a post-processing step is comparatively straightforward and will have immediate benefits with regards to useability and hopefully overall performance of our models. Below is a description of problems we have encountered along with proposed post-processing methods to fix them.

Barcoding

Barcoding occurs when there is a rapid alternation between two states model output, so called because this leads to transition plots that look like a barcode (below). It is typically seen between M and L states and is probably a result of uncertainty between the two where the transition occurs. Not only does barcoding hurt performance, but it is also incredibly tedious to manually correct as the user must manually delete dozens of short segments.



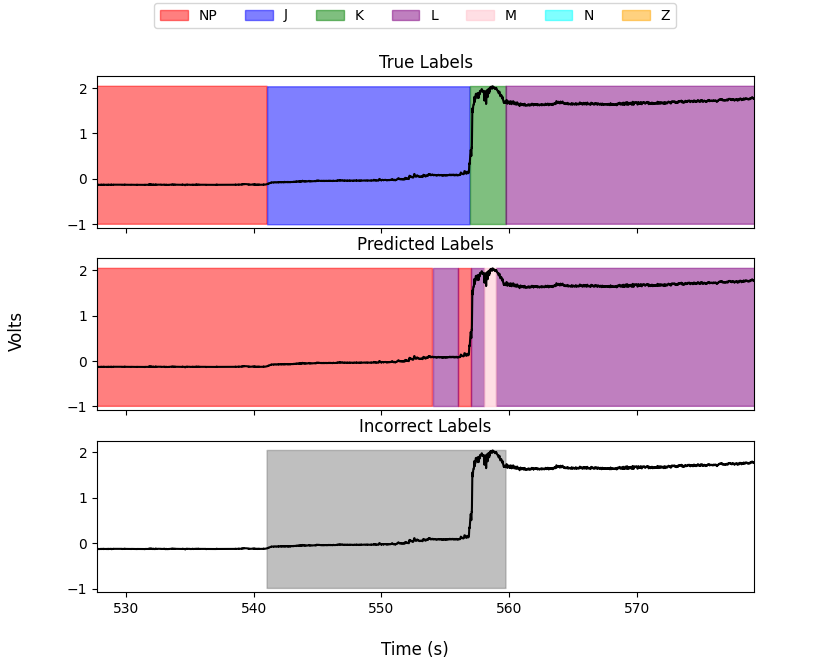
Barcoding Example

Barcoding Correction

Mode Filter: Applying a sliding mode filter with an empirically chosen window size could have the effect of removing barcoding by replacing the label at each time point with the most common label in the window around it. This can be thought of as “smoothing” the model output. This will have the intended effect of removing frequent switching. However, it may delete short waveform types (like those at transition points) that we do not want removed. Therefore we may need to restrict barcoding correction to certain regions or waveform types. This filter is implemented in postprocessing.py in Zach’s folder.

Missed Transition States

The transition states J, Z, and to an extent K are short and characterized largely by their taking place between other states. Therefore it is difficult for our models to identify them because they make up a very small portion of any individual recording and thus the entire dataset. Additionally, as our models label on 1-3 second windows that have the longest waveform type in each window as the target waveform type, they are often “gerrymandered” out of the training data. This leads to our models essentially never labeling a transition as such, hurting performance in these classes and forcing users to manually add them after the fact.



Missed Transition States Example

Missed Transition States Solutions

Transition insertion: While our models may struggle with identifying transitions, if we assume that the other states are well-labeled then it is fairly straightforward for us to know where to add them based on our knowledge of the transition matrix. For example, we know that NP always goes to J, K, and then L. If our model produces labels that go immediately from NP to K, we know J was skipped and can insert a region of J between NP and K. Knowing *exactly* where to put the start and stop of this insertion will likely require a case-specific algorithm.

Impossible State Sequences

Related to the issue of missed transitions is the issue of impossible state sequences, where the output labels go from K to NP for example. While these impossible sequences are easy to identify by making use of our transition matrix, fixing them is nontrivial. In some cases, like if a short segment of NP is present between two long L segments, we know that the NP segment should be deleted, something that should already be handled by the barcoding mode filter. In other instances we may not be able to determine the best course of action. If we were to have two long L segments interleaved with two long NP sequences, it is unclear without further inspection if both, neither, or only one label was actually correct for that region. However, flagging these impossible transitions for the user to address themselves would certainly be useful. Impossible state sequence finding is implemented in postprocessing.py in Zach’s folder.