KNN And MLP Approach for multiclass Sequence to Sequence Decoder

Nazia Rahman
Department of Electrical and Computer Engineering, Temple University
Rahman.nazia@temple.edu

Introduction: The dataset of this experiment represents 1D signals (amplitude vs time). There is total 10,000 training files, 2,000 dev files and 2,000 blind eval files. Each file contains a combination of 5 types of events- class 0 (background) and class 1,2,3,4 (non-background) event. To evaluate the performance on eval files data, one neural network and one non-neural network approach is used. Multilayer perceptron with three hidden layers is used as a neural network which gives error rate of 5.60% for training data, 5.65% for dev data and x% for eval data. As non-neural network, K-Nearest-Neighbors algorithm is used which provides error rate of 0.68%, 0% and y% for training, dev and eval data respectively.

Data Preprocessing: In this project, there are 10,000 training data files. Each of the training data files are sequential in nature having five types of events: bckg (class 0), and four non-bckg events (classes 1 - 4). In this data preprocessing part, each of these sequential files are first segmented using a sliding window of length 10. Within a particular window, only one label is assigned based on the majority vote. For a particular window, the training data is treated as an observation, and there are multiple observations based on the length of sequence of that specific data sequence. These 10 features (because of the window length) observations are then fed to machine learning algorithm.

Multilayer Perceptron Approach (MLP): In our dataset, there is no linear relationship between the events of the data. MLP is used as neural network, as it can learn the non-linear relation between events and its features. MLP consists of three types of layers—the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as prediction and classification are performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Like a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. In each layer there may have multiple nodes. Each node performs some mathematical operation, i.e., multiply the wights associated with the node with the input and add a bias term. After that, some nonlinear activation function is used so that it can learn nonlinear relationships. After computing the output, the loss between predicted output and actual label is backpropagated to the network and optimize the weights of the network by minimizing the overall cost function. After the training is completed, the network is expected to predict unknown data with satisfactory accuracy.

In this project, pytorch is used to build the MLP model. Each preprocessed data sequence is considered as a batch size for the MLP network. Since there are, 10000 training data files, so there will be 10000 batches. And, since each of the preprocessed data sequence has 10 features, 10 input node is used for this model. Number of hidden layers (3 layers) and nodes in each hidden layer (64) are chosen experimentally. As nonlinear activation function in the hidden layers, the ReLU activation function is used. ReLU activation provides a comparatively faster response. For multiclass classification, softmax activation is used in the output layer. Dropout regularizer is used in the hidden layers for preventing overfitting. Since this is a multiclass classification problem, I use categorical cross entropy loss function. Adam optimizer is used to minimize the overall cost function.

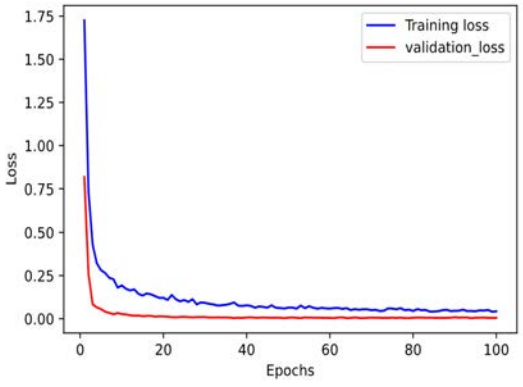

Figure 1: Loss vs epochs performance

From figure 1, it can be noticed that initially there was high loss. But with the increase of epochs, loss is minimized.

K-Nearest-Neighbors (KNN) Approach: KNN is one of the simple and easy to implement algorithm compared to other supervised learning algorithm. Its performance is dependent mainly on the number of neighbors, K. In this way, it can be considered as a single tuning parametric method.

In a neighbor of K points, the test data will consider K nearest points as its neighbors and among them dominant samples' class will be assigned as the estimated class for the test data. The number of K is selected based on the satisfactory performances of the dev and train data. In this experiment K=2 is selected experimentally.
From figure 2, it can be observed that for lower number of K error rate is low. And after certain number of neighbor value, the performance does not change.

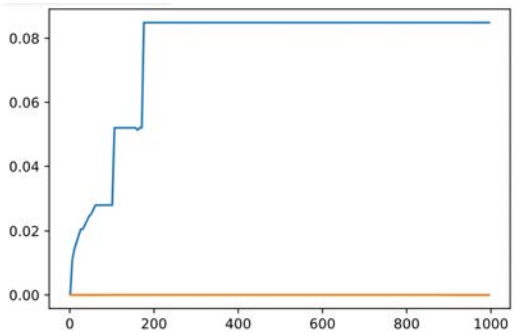

Figure 2: Change of error rate with K

Results: The result of the above discussed algorithm are summarized in Table 1.

Algorithm	Dataset error (%)		
	Train	Dev	Eval
MLP	5.60	5.65	
KNN	0.68	0	

Conclusions: In this project, one neural and one non-neural network have been designed to estimate the performance of 1D signal. I could not optimize the training model properly for the KNN approach. So, comparison between the approaches could not be compared properly.