ECE-8527 Final Exam

Richard Sand
Department of Electrical and Computer Engineering, Temple University
richard.sand0001@temple.edu
April 28, 2022

Introduction: We were provided with time-sequence 1D data containing 5 event classifiers. 0 indicates background noise, and classes 1-4 are signal sources. Three data sets are provided: training (train), development (dev), and evaluation (eval). Train has 10,000 labeled samples, dev has 2,000 labeled samples, and eval has 2,000 unlabeled samples. Each sample is of arbitrary duration. For each labeled sample, metadata is provided for the events that occur within the sample – the label, start, and stop times. No other information is provided about the nature of the signals. The goal of the project is to classify the eval data. The deliverables are CSV hypothesis files, one for each sample file, containing the hypothesized event labels, boundaries, and confidence levels.

Two machine-learning algorithms are implemented, the first is a non-neural network approach, and the second is a neural network approach.

A visual inspection of the data via *matplotlib* showed series with low levels of background noise combined with generally square steps of various amplitudes and durations. The events did not seem to overlap.

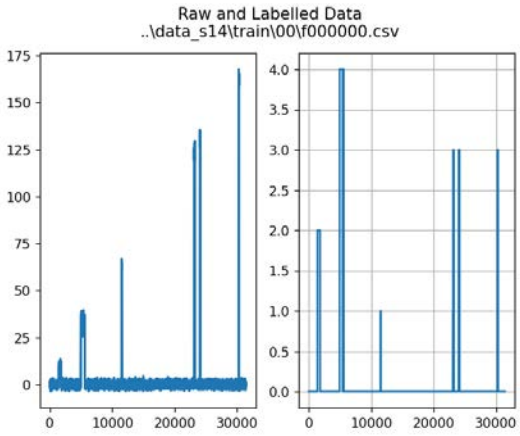

Figure 1: Exemplar Raw and Labeled Data

To preprocess the data, a lowpass butter filter from `scipy.signal` was applied, and the data scaled to [0,1]. Then, to identify the event boundaries, each time sample was compared to its previous sample to calculate the delta. A delta larger than a configurable threshold percentage was considered the start (if positive) or end (if negative) of an event.

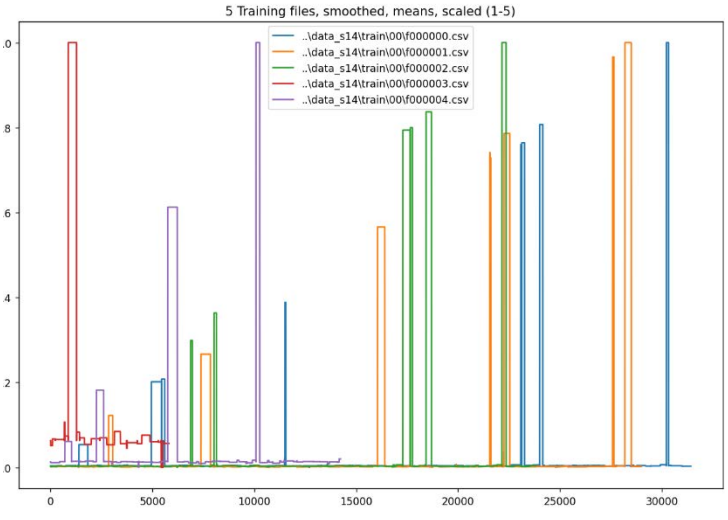

Figure 2: Five Files with Normalized Data

After identifying the events, the average amplitudes of all samples in the event were taken. The durations and amplitudes were combined into a 2D array. For the training and dev data, the labels were attached, and these 2D arrays were then processed with the below algorithms. The models were trained on the training data, and then tuned to provide best results on the dev data. The tuned models were then applied to the eval data and the results recorded as hypothesis files.

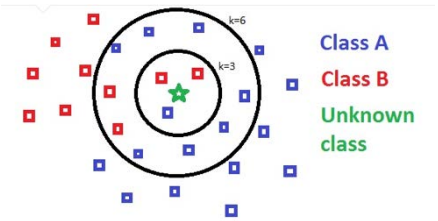

Figure 3: KNN Model

Algorithm No. 1 Description: For algorithm 1, we utilized both K-Nearest Neighbor (KNN) and Random Forests (RNF) algorithms and compared their results. KNN is a supervised classification algorithm where Euclidean (or other) distances are used to find the labelled samples that are closest to the evaluation sample. We used the *sklearn* class `sklearn.neighbors.KNeighborsClassifier` for KNN.

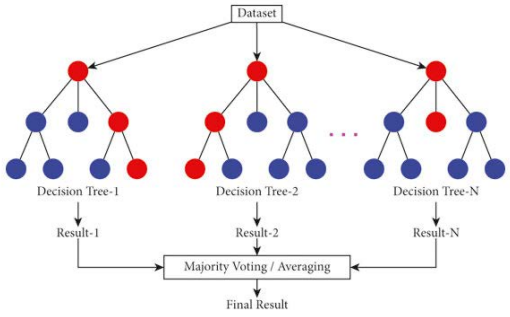

Figure 4: Random Forests Classification

RNF is an "ensemble" training method where multiple decision trees are evaluated and a majority vote taken for final evaluation. For RNF we used `sklearn.ensemble.RandomForestClassifier`. For both algorithms, we varied a parameter *n* which represented the number of neighbors or number of estimators for KNN and RNF, respectively. We evaluated $1 \leq n < 51$ with step 2 for each, measuring the error rates for the train and dev sets.

We selected the RNF model as the better of the two models, and ran the evaluation data through the model with *n* that was optimal for the dev data set.

Algorithm No. 2 Description: Multilayer Perceptron (MLP)

We used the ISIP *PyTorch* based implementation of Multilayer Perceptron for this assignment. An MLP a type of feedforward neural network, where each node is connected to all nodes on the next layer. This implementation uses a network with 3 layers of linear transformation (`torch.nn.Linear`) with a rectified linear unit activation function (`torch.nn.ReLU`, which is $y = x$ for $x > 0$ and $y=0$ for $x \leq 0$) and dropout function between each linear transformation (`torch.nn.Dropout`). It uses a cross entropy loss function (`torch.nn.CrossEntropyLoss`).

Results: First we compared the KNN and RNF algorithms to determine the best number of neighbors or evaluators tuned to the dev data.

	Data Set		*Notes*
Algorithm	**Train**	**Dev**	
KNN	0.2366	0.3557	*Best training result at n= 3* *Best dev result at n= 11*
RNF	0.0546	0.3572	*Best training result at n= 49* *Best dev result at n= 11*

Table 2: KNN vs RNF Training Results

The KNN method was fractionally better than the RNF, a statistically insignificant amount. It should also be noted that RNF has an element of randomness anyway. However, RNF was significantly better than the KNN on the training data. Therefore, we chose to proceed using the RNF algorithm on the evaluation data.

The MLP was able to process the data successfully and come up with predictions. We aggregated all of the training and dev events into a training and dev csv file. Then we executed the MLP to create the model file, and then decoded the training data with the model. Unfortunately, after scoring the training data, the error rate was extremely high, indicating a flaw in the model. The confusion matrix is shown below:

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
RNF	5.46%	35.57%	?
MLP	%	%	?

Conclusions:

```
confusion [[ 0. 93.  0.  0.  0.]
 [ 0.  1.  0.  0. 25.]
 [ 0.  1.  0.  0. 21.]
 [ 0.  0.  0.  0. 23.]
 [ 0.  1.  0.  0. 35.]]
error rate  =    99.5000%
```

Figure 5: Confusion Matrix

At this point we concluded that there was no point in generating the hypothesis for the eval data, since the implementation clearly could not understand the data provided. One observation about the hypothesis data it generated was that none of the predictions were for class 0. A convoluted neural network (CNN) may provide better results.