A Comparison of Random Forest and Neural Network Classifiers on EEG Data

James McHugh
Department of Electrical and Computer Engineering, Temple University
james.mchugh@temple.edu

Introduction: Throughout the course, many different machine learning classification algorithms were discussed. These algorithms consisted of traditional classifiers, such as support vector machines, k-nearest neighbor, and Bayesian classifiers, and neural network classifiers, such as multilayer perceptrons, convolutional neural networks, and other deep learning algorithms. Each of these algorithms has its own strengths and weaknesses; for example, deep neural networks are good at modeling complex decision surfaces, but are very data-hungry and can often get caught in local minima when minimizing the loss function. In this final project, two binary classification algorithms will be used to perform seizure classification on EEG data: a traditional classifier and a neural network. For the traditional classifier, a random forest classifier was used, and for the neural network classifier, a multilayer perceptron was used. These algorithms will be discussed in more detail in later sections.

Two datasets were provided for this project: a train dataset and a development dataset. A third dataset, the evaluation set, was withheld to be used as a final evaluation of performance. The train dataset was used as its name suggests, to train the classifiers. The development dataset was used to test the trained classifier before submitting it for evaluation. Each dataset is a text file that consists of multiple samples; each sample is made up of a 26-element feature vector and a boolean indicating whether the sample corresponds to a seizure or no-seizure event. After training the classifier to a point that it performs well on the development data, it was submitted for evaluation to measure the final performance of the system.

Algorithm No. 1 Description: Multiple classification algorithms were investigated for the traditional classifier. These algorithms consisted of a support vector machine, k-nearest neighbor, and random forest classifier. Out of the three algorithms, the random forest classifier performed the best on both the train and development datasets for the tested parameter values, so it was chosen for the traditional algorithm. The k-nearest neighbor algorithm was much more susceptible to overfitting than the other two algorithms, which is another reason it was not chosen. Further reasons for selecting random forest are that it has many parameters that can be adjusted, and it can be parallelized easily since it is an ensemble of multiple decision trees. The parameters to be tuned consisted of the number of estimators in the classifier, the minimum number of samples required to split each node in a tree, and minimum number of samples for a node to be a leaf node. To determine these parameters, the classifier was evaluated at various parameter values, and the parameters that gave the best results were chosen to be the parameters used for classification. Table 2 shows the parameters chosen from this analysis.

Algorithm No. 2 Description: For the neural network classifier, a multilayer perceptron was used. Several other deep learning algorithms were considered. One of these was a convolutional neural network, but it did not seem to fit the application since the data we were given does not contain spatial or temporal information. The topology chosen consisted of one input layer of 26 units, three hidden layers, and an output layer of one unit. The three hidden layers consisted of a dense layer with 100 units, a dropout layer for regularization, and a bottleneck layer whose number of units was varied. The final topology used can be seen in Figure 1. Several optimizers were tested with this topology, and it was found that the Adam optimizer performed the best for the given architecture and application.

Before training the system, the training data was augmented by duplicating the data, adding Gaussian white noise to it in the form $N(0,0.01)$, and concatenating it to the original data. This was done to add extra variation to the data to reduce overfitting and increase performance on the development set. To determine the parameters for this network, such as initialization algorithm, regularization algorithm, activation

functions, batch size, and more, two approaches were taken. The first approach was to choose an initialization, activation, and regularization algorithm based on results on similar applications (such as those discussed in Lecture 42), and then vary the batch size, number of epochs, and units in the bottleneck layer to find the best parameters. When training the classifier through Keras, a callback was used to save the weights from the epoch that gave the best performance on the development dataset. The second approach taken was to automate this process by continuously creating classifiers using random parameters in an infinite loop and then saving the best one. This method provided the best results, which can be seen in Table 1.

Results: When compared to the multilayer perceptron, random forest classifier performed the best on the training data with an error rate of 24.37%. Despite slightly overfitting on the training data, the algorithm did not perform too much worse than the neural network on the development set as can be seen in Table 1. Unfortunately, these results did not carry over to the evaluation set where the error rate was over 10% higher than the development set.

The multilayer perceptron performed the best on the development dataset with an error rate of 34.09%. This can most likely be attributed to its innate ability to model complex decision surfaces, but it did not perform as well on the training data. This is because the MLP did not overfit like the random forest classifier did. Surprisingly, the results from the development set did not carry over to the evaluation set for this classifier either. The error rate increased by about 10% from the development set to the evaluation for this classifier as well. The final performance of the MLP can be seen in Table 1.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
Random Forest Classifier	24.37%	36.45%	47.10%
Multilayer Perceptron	41.19%	34.09%	44.12%

Table 1. Error rates of both classifiers on all three datasets

Conclusions: Two classifiers were created to perform binary classification on seizure data extracted from EEGs. A random forest classifier and a multilayer perceptron were the selected classifiers to be construced and evaluated. Despite both classifiers producing error rates around 35% for the development data, the error rates were much higher for the evaluation datasets. This came as quite a shock, but it can most likely be associated with the fact that the classifiers were specifically tuned to produce good results on the development data. It is possible that these parameters did not generalize well for datasets other than the train and development datasets; in effect, producing much worse results on the evaluation set. For future work, it would be best to set aside part of the training data as a second validation set instead of using it for training. Then, when the classifier produces acceptable results on the development dataset and second validation set, it could be submitted for evaluation. This would help assure that each classifier generalizes well.

As an attempt to build a better classifier from the results of the random forest and neural network classifiers, a voter system was created. This system performed principal components analysis on the feature vectors for each sample in the datasets to reduce them to a single unit. The predictions produced by the MLP and random forest classifiers were then appended to the corresponding feature vectors. A neural network was then used to perform classification on these values. This system produced the same results as the MLP for the development data, but it produced worse results for the evaluation dataset. The voting system gave an error rate of 45.23% for the evaluation set. It is possible that a voter system would have produced better results if implemented in a way different than the way it was implemented in this assignment. In complex systems, different types of classifiers are often chained together to produce better results. For future work on this problem, this method should be investigated further.

	Parameters		
Name	**Min Split**	**Min Leaf**	**Number of Estimators**
Value	1.00% of train set	0.10% of train set	70

Table 2. Parameters chosen for the random forest classifier

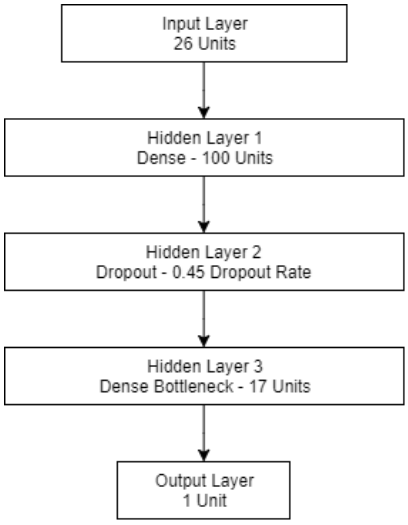

Figure 1: Final neural network topology

	Parameters								
Name	**Out Act.**	**Other Act**	**Reg**	**Bath Size**	**Max Epochs**	**Learn Rate**	**Dropout Rate**	**Bottleneck Units**	**Initializer**
Value	tanh	relu	None	128	1000	10^{-4}	0.45	17	Normal

Table 3. Parameters chosen for the neural network.