Seizure Detection Using a Random Forest and Multi –Layer Perceptron

Victor Espinoza
Department of Electrical and Computer Engineering, Temple University
Tug86727@temple.edu

Introduction: The primary goal of this work is to design two different binary classifiers based on 25 sample sequence or features of data. The training set of data consisted of 19320 instances of these sequences or feature sets, while the class was indicated by a zero or a one. The development set consisted of 2118 sequences and the evaluation set consisted of 1174 sequences. The data is suspected to have been obtained from electroencephalogram (EEG) sessions, a multi-channel signal which describes the electrical activity in the brain via voltages. This work required the implementation of a traditional machine learning algorithm as well as a deep learning method. The mentioned algorithms were designed using Scikit and Keras with a TensorFlow backend, both in python 3.6.

Algorithm No. 1 Description: As for the conventional classifying method for this work we have chosen the random forest algorithm. In order to comprehend the random forest approach we will first explain the decision tree, which is the building block of the random forest algorithm. The decision tree is a classification technique that consists of a set of splitting nodes based on entropy. This finite number of splitting, or branching, will decide a path for the data sample that will ultimately lead to the classification of the data. The questions can be thought of as nodes or visualized as a point where branches of the tree split. As data travels up a finite number of nodes it will ultimately end up in a leaf, where the data is classified. Although this is a practical and computationally inexpensive approach it is easy to overfit the data, meaning that complex trees will not classify non-training data very well. In order to quantify for the efficacy of the trees we will implement ways to measure the impurity in the classifications. Two common methods to quantify this impurity in classification regions are by measuring entropy and the Gini index.

This classifying method was implement using the Scikit data mining toolkit. This toolkit does not specify how it quantifies impurity so it was not controllable by the user. In practice, random forests are used as an ensemble machine learning method, in which we deploy a large quantity of weak learners, decision trees. As numbers of trees increases in our forest, the more computationally expensive out algorithm becomes. The random forest ultimately classifies data based on the majority vote, or classification amongst the decision trees. Pruning, is the practice enforcing a maxim depth of splitting nodes across all tress as well as enforcing a minimum population per class or leaf. Setting the number of tress to 90 and the maximum tree depth to 9, I was able to achieve my best results in the train and dev test.

Algorithm No. 2 Description: For our non-conventional technique we introduce a common deep learning approach, the multilayer perceptron, which can take all 25 points of data as a weighted input. The building block of the multilayer perceptron, commonly referred to as MLP or feed-forward neural network, is the neuron. The input layer consists of all the 25 nodes, the hidden layer is made up of the computational unit, and output layer is comprised of $\hat{y}$ where each hidden layer is applied a bias an activation function. Output $\hat{y}$ is also computed with the use of an activation function. The activation function is typically designed to return a binary value based on a threshold, or a probability value ranging from 0 and 1or from -1 to 1 like hyperbolic tangent function. The ultimate goal of the training is to go through each training set, and solve for the optimal values of each weight to minimize the error. This introduces a series of algorithms that are used to optimize the adjustment of weights; SGD, stochastic gradient descent, Adam, and L-BFGS, Limited Memory Broyden-Flectcher-Goldfarb-Shannon. The one that seems to be the most common in practice is stochastic gradient descent, so it was the chosen algorithm for our feedforward network. SGD updates the weight based on a learning rate η and the gradient of the loss function. Learning rates can also be an adaptive variable.

The rate of convergence using the gradient descent approach can be significantly affected by initial weights configuration of the MLP. Randomizing the weights every time makes our network prone to getting stuck in a local minima or start over from positions that are more distant from our target output. The learning process is therefore required to repeat from several different starting positions, which can misguide us or lead us to make uneducated design decisions. It is believed that a set of optimal initial conditions will result in more efficient epochs. Previous works show that if you consider your neural network as single layer perceptron, you can solve for optimal initial conditions using singular value decomposition or QR decomposition.

Results: Our designed multilayer perceptron performed the worst out amongst the two designed algorithms in both train and dev set with an error rate of 40.19% . The random forest performed much better in the training set above with an error rate of 25.321%. As for the dev test, both our designed algorithm were relatively close to each other in performance with error rates of 35.2% and 36 for the Random Forest and MLP respectively. The designed random forest performed poorly in the Eval seta and displayed 46.85% error, about a 20% increase in error from its own performance in the training set. The feedforward network displayed 45.32% within reasonable range from its own performance in the training set error 40.19%. In addition we also attempted to implement singular value decomposition to try an estimate better initial conditions but we were not able to achieve significant results initial weights but I was able to complete it. The intention behind this was to visualize orthogonal components within the data that would reflect variance amongst the features given.

Conclusions: .The MLP is a standard neural network and arguably of the lowest in complexity. This work shows us that there various parameters to take into consideration in the design process, therefore one should have some sort of a test plan beforehand since there are various moving parts, Additionally, it would have been helpful to graph how error changes as a function of another variable, much like previous assignments in this course in order to visualize how variables affect our outcomes. It is also important to note that the multilayer perceptron is a deep learning approach that requires significantly bigger amounts of training data to work more efficiently. With the sample mentioned sample size, it would also be worth further investigating the learning rate for this size data, or if an adaptive learning rate would decrease the error amongst all sets

I would have liked to have been able to experiment with other parameter updating algorithms within Keras and external to it like in the Scikit toolkit. In the future, I would like to implement the optimization methods that are previously mentioned in this work amongst many other optimizers. Recent work also shows that it is more common in practice to perform batch normalization in order to adapt to the slowing down of learning rates of consequent layers. Additionally, the importance of initial weights in neural networks is worth investing in like Singular value decomposition or QR decomposition to optimize results. As Random forests, although it is not very computationally expensive and relatively reliable, it is important to refrain from overfitting the data by pruning the forest. If we wanted to decrease maximum tree depth even more we could implement the Adaboost algorithm, another ensemble approach. The Adaboost algorithm proposes even weaker learners in larger abundance that are just slightly better than randomly gusseting. It is also worth mentioning that the maximum number of epochs used for this experiment was 25 and was not investigated into further values since we had already ran the Eval set. In future work it would be wise to see if percent error converges as the number of epochs increases.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
Random Forest	25.321%	35.223%	46.8484%
MLP	40.19%	36.02 %	45.3152%

Table 1. Given 3 data sets, we implemented a conventional classifier, Random Forest, and a deep learning classifier a multilayer perceptron. The table displays the error rate for each approach and set.