Classification of 2D and 5D data - Final Project for ECE 8527

David Ryskalczyk
Department of Electrical and Computer Engineering, Temple University
david.rysk@temple.edu

Introduction: Machine learning can be a powerful tool for classification of arbitrary data. The goals of this project are to train a classifier to classify the evaluation data as accurately as possible, using both a neural network and a non-neural network approach, and to evaluate the performance of the classification.

Datasets of two-dimensional and five-dimensional data were provided. These datasets consist of training data, development data, and evaluation data. The 5D evaluation data was provided without labels — even so, the labels on the development and evaluation data must not be used to train the classifier.

Algorithm No. 1 Description: K-Nearest-Neighbors: This is a supervised machine learning algorithm that finds a set number of training samples (K) that are nearest to the point being classified and uses them to predict the label of this point. This is a member of the family of nearest-neighbor algorithms, which includes both supervised and unsupervised algorithms. There are several parameters used by the algorithm, which are important for classification. **`n_neighbors`** determines how many neighbors are considered for each point; **`weights`** determines the weight function used for prediction: whether all points are to be weighted equally, or if the weight should be based on distance; **`algorithm`** determines the algorithm used to compute the nearest neighbors; **`leaf_size`** determines at what point the chosen algorithm switches to the brute-force method; **`metric`** determines what distance metric is used for the tree, and **`p`** determines the power parameter in case the Minkowski metric is used. The available algorithms are **`BallTree`** and **`KDTree`**, which refer to two different data structures for storing a sorted hierarchical structure, as well as a brute-force approach. The brute-force approach has acceptable performance for small training sets with few dimensions, but its complexity rapidly increases as the training set size and number of dimensions increase.

The optimal parameters are best determined via a search process. In order to make this easier, Scikit-Learn provides several meta-classifiers. We are using **`sklearn.model_selection.GridSearchCV`**, which does a brute-force search of all the parameters we would like to try. By default, it performs cross-validation by 5-fold segmentation of the training data, but the training data and the dev data can also be provided separately. **`GridSearchCV`** is able to parallelize the searching across multiple CPU cores, which helps when there are many parameters being tested.

When the training data is applied on a K-nearest-neighbors model trained with said data with distance-based weighting, we appear to obtain an error rate of 0%. This is because the algorithm is comparing the training data points to themselves. We cannot exactly call this overfitting, but the error rate produced this way is not useful regardless. Therefore, we must either split the training dataset in order to measure the error rate or stick to uniform weighting. As the error rate difference between uniform weighting with a Euclidean distance metric and distance-based weighting with a Manhattan distance metric was minimal for the 5D dataset, we used uniform weighting. (I believe the two parameters compensate for each other in that case — I found this phenomenon to be common when using **`GridSearchCV`**: it does not provide much information on whether combinations of parameters compensate for each other and therefore produce nearly the same performance. I believe it offers a way to retrieve scores for each cross-validated slice, but I ran out of time to do this.)

The parameters used for the K-nearest-neighbors models are listed in Table 1, and the final error rates are in Table 3. The leaf size was fixed to 30 for the 5D dataset, as using a smaller leaf size increases computation time with little benefit. Plots of the algorithm being used on the train and dev datasets, with the decision surfaces highlighted, can be seen in Figures 1 and 2, respectively, for the 2D dataset. (Plots for the 5D dataset are left out, as displaying all five dimensions would be difficult.)

Data Set	n_neighbors	algorithm	metric	weights	Leaf_size
2D	18	ball_tree	manhattan	uniform	30
5D	600	ball_tree	euclidean	uniform	30

Table 1. K-nearest-neighbors parameters

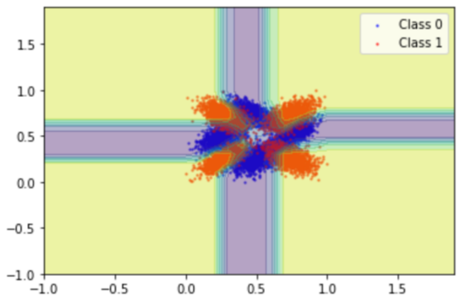

Figure 1: 2D training data with decision surface

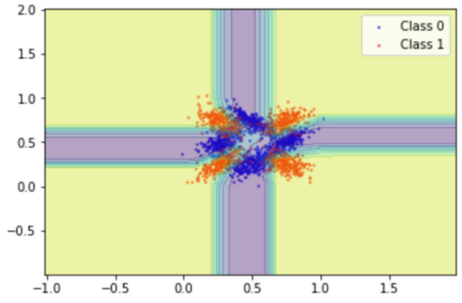

Figure 2: 2D dev data with decision surface

Algorithm No. 2 Description: Multilayer Perceptron: This is a class of feedforward neural network. We were provided with a baseline Pytorch model, which we could adapt for our purposes. The baseline Pytorch model was written for a 26-dimension dataset and needed significant modification to be useful.

Neural network training typically uses an adaptive optimization algorithm. One of the currently popular training algorithms is the Adam optimizer. Adam stands for adaptive moment estimation; this optimizer combines a stochastic gradient descent (SGD) approach with use of the first and second moments of the learning rate to adapt the learning rate.

The learning rate is a hyperparameter that sets the amount the weights of the model are updated during each training run. If it is too large, the error rate may increase during training. If it is too small, the model will take much longer to train, and may provide a non-optimal solution. Tuning the learning rate is critical to obtain useful output from a neural network model.

As training progresses, learning decreases. If the learning rate is too large, the model will stop learning before a good solution is found. The **`ReduceLROnPlateau`** scheduler was used to further dynamically adjust the learning rate between epoch runs. As the training progressed, the learning rate needed to be decreased for training to continue. Otherwise, the loss would increase rather than decrease. This scheduler monitors the learning rate and decreases the learning rate as needed. Termination of training was implemented where if the decrease in loss was small enough (1e-5) for enough epochs (25), training was terminated. Additionally, the model was saved after each epoch, to allow for manual termination of training with Ctrl-C.

The model used for the 2d dataset has three hidden layers, with the two outer hidden layers having 20 neurons each and the inner hidden layer having 50 neurons. The model used for the 5d dataset also has three hidden layers, with the hidden layers having 50 neurons each. The rectified linear (ReLU) activation function is used on each layer. Even though the models were chosen via a fair amount of testing, adjusting the layers as well as number of neurons did not necessarily provide significant improvement.

Tuning the hyperparameters were critical for obtaining reasonable performance from the neural network — perhaps more so than the model design. The final hyperparameters for the models are in Table 2, and the final error rates are in the combined Table 3.

Data Set	Learning rate	Batch size	Number of epochs
2D	1e-5	500	309
5D	5e-6	500	787

Table 2. MLP hyperparameters

Results: The combined classification results can be seen in Table 3. The confidence levels for the difference in performance between the K-nearest-neighbors algorithm and the MLP algorithm can be seen in Table 4. Both classification methods provided comparable performance, with the K-nearest-neighbors algorithm providing slightly better results. With further tuning, it should be possible for both algorithms to perform about the same; however, these results are very much data dependent.

	2D Data			**5D Data**		
Algorithm	**Train**	**Dev Test**	**Eval**	**Train**	**Dev Test**	**Eval**
K-nearest-neighbors	7.69%	7.80%	8.15%	36.84%	37.33%	36.87%
Multilayer Perceptron	8.30%	8.50%	8.35%	38.24%	37.78%	38.38%

Table 3: Final error rates

Data Set	T rain	Dev Test	Eval
2D	94.41%	79.08%	59.09%
5D	100%	74.44%	98.62%

Table 4. Confidence levels of Knn scores vs. MLP scores

Conclusions: For the purposes of this project, it is unclear which algorithm is superior. While both can be tuned to provide somewhat better results, the maximum classification rate seems to be limited, as the output is highly dependent on the quality of the data, and the improvement is minimal.

That said, for the purposes of detecting seizure or no seizure, even the error rates that we are getting for the 2D data set are too high. The provided data is a single EEG vector from the middle of a seizure event. This does not provide sufficient information to classify seizure or no seizure with high accuracy. Use of temporal data, together with a neural network well suited for this data, such as an RNN, would likely produce better results, as would a spectral estimation process that extracts frequency domain data over time intervals. Several papers discuss the use of these techniques[1,2,3].

---

[1] V. Srinivasan, C. Eswaran, a.N. Sriraam. "Artificial Neural Network Based Epileptic Detection Using Time-Domain and Frequency-Domain Features", J Med Syst 29, 647–660 (2005). https://doi.org/10.1007/s10916-005-6133-1

[2] S. Pravin Kumar, N. Sriraam, P.G. Benakop, B.C. Jinaga. "Entropies based detection of epileptic seizures with artificial neural network classifiers", Expert Systems with Applications, Volume 37 Issue 4, 3284-3291 (2010). https://doi.org/10.1016/j.eswa.2009.09.051

[3] Ahmad R. Naghsh-Nilchi, Mostafa Aghashahi, "Epilepsy seizure detection using eigen-system spectral estimation and Multiple Layer Perceptron neural network", Biomedical Signal Processing and Control Volume 5 Issue 2 , 147-157 (2010), https://doi.org/10.1016/j.bspc.2010.01.004