Traditional Machine Learning Classification Vs. Deep Learning Methods

Abdullah Aljebreen
Department of Computer & Information Sciences, Temple University
aaljebreen@temple.edu

Introduction: The provided dataset contains 21,438 data points divided into training and development test sets (90-10%). An extra set of 1174 data points (anonymized) is available for evaluation purpose only. Each data point consists of 26 features, all of them are real numbers. With this number of dimensions, picking suitable classification algorithms was not a trivial task. In this report, we compare two ways of classification, a traditional machine learning method and a deep learning method. We had to perform multiple experiments to help us pick an appropriate method for our dataset in each of these two categories. For traditional methods, K-Nearest Neighbors seems to be an appropriate choice, as suggested to us. In addition to K-NN, we tried some other traditional methods. The only method that was close to the K-NN results was Random Forest which gave us a slightly lower error rate than K-NN. However, we decided to stick with K-NN because Random Forest took much longer time to train and classify (about 8 times longer). Also, in Random Forest we achieved a very low training set error rate which can make it overtrained model and therefore implies that the development results is less credible. In the second task, we experimented with CNN, RNN, and LSTM models, but a simple MLP Neural Network with four layers showed to be the most effective approach to this dataset.

We used Matlab and Python for the following experiments. In the K-NN method, we used the Statistics and Machine Learning Toolbox in Matlab which provides us with multiple functions to train and evaluate K-NN models and perform many other operations as well. For the Neural Networks algorithm, we used Keras, a Python high-level neural networks API, with TensorFlow, an open-source software library for Machine Learning. Keras allowed us to define and train neural network models in a few short lines of code and carry out the process with an impressive computational performance.

Algorithm No. 1 Description: The first algorithm is K-Nearest Neighbor (KNN) which is a traditional ML classification technique. In K-NN, the nearest-neighbor rule for classifying any test point is to assign it the label associated with the majority of the closest K data points from the training set. K-NN is known for its excellent performance in term of error rate, especially with large datasets. Surpassingly, in our case, the computational efficiency of both training and classifying was also very good and surpassed other traditional methods. To improve the error rate and the computation time as well, we tried multiple dimensionality reduction and feature selection methods with KNN, but none of them gave us a better accuracy than the original dataset. This might be a result of the nature of this dataset. When we tried to reduce the number of features to get to a lower dimension space, the two classes in the remaining features usually overlapped which made the reduced features less informative unless we combined them with all other features. Also, we tried multiple distance metrics and the standard Euclidean distance proves to be the right choice, especially since the numeric representation of the features are similar in nature. Before choosing our K, we varied the number of neighbors in our experiments from 1 to 200 to make sure that we picked the right K. Furthermore we used an option that standardizes the data by centering and scaling each column of the training data by the column mean and standard deviation, respectively. Using this option gave us a significant improvement in the performance of 8% error rate reduction.

Algorithm No. 2 Description: Our deep learning algorithm is a multilayer perceptron (MLP) which is a feedforward neural network that consists of four layers of neurons. The first layer is the input layer which includes 26 neurons, one for each feature. We used 128 neurons in each of the second and the third layers (the hidden layers). Finally, the fourth layer has two nodes that produce the predicted class using a binary matrix representation. In the two hidden layers, we used Sigmoid activation function. Sigmoid function helps us introduces non-linearity into our neural network model. The output layer uses Softmax activation to make sure that the outputs of the two unit to be between 0 and 1 and that tells us the probability that any of the classes are true. Using Sigmoid and Sigmoid provided us with the most accurate classification results comparing with other activation functions. We used a learning rate of 0.4 in the first hidden layer

and 0.8 in the second layer. We reached these learning rates by trial and error knowing that these parameters aim to prevent the model from overfitting. In our model, Keras sets the initial random weights for each layer. Also, we didn't use a bias parameters in our model. Another decision to be made was choosing an optimization algorithm. Optimization algorithms help us to minimize the error rate in the training phase by using a mathematical function that's dependent on the learnable parameters. One of the available algorithms is called Adam, which is an algorithm for first-order gradient-based optimization of stochastic objective functions. It computes adaptive learning rates for each parameter and keeps an exponentially decaying average of past gradients. In our model, we chose Adamax, which is variant of Adam based on the infinity norm. Again, these decisions were based on series of experiments and trials.

Results: Our experiments were performed on a personal laptop with these specifications: (CPU: 2.3 GHz Intel Core i7, Memory: 16 GB 1600 MHz DDR3, GPU: Intel Iris Pro 1536 MB). In all of our experiments, we tried to commit to the approach of closed-loop in the training process and open-loop for the testing. By that, we maintained the credibility of our development test set results which should assist us achieving a decent performance in the evaluation test set. Both of our methods produce similar training set (closed-loop) and development set (open-loop) results.

For our first method (K-NN) and with this number of dimension in our dataset, it made sense to choose a small number of neighbors K. However, the results show that the error rate on the development test goes down as we increase K and reaches a convergence point after using 170 neighbors giving us 37.77% error rate using 172 neighbors. The error rate on the training data goes up as we increase K and converges in a similar neighborhood with an error rate of 39.72%. The time of training and classification also increases as we increase K from about 2.5 seconds to 3.9 seconds which is not a serious rise in our case. When using K=172, the total execution time for training and classifying was 3.76 seconds. Looking at these results, K-NN with its simple structure produced a reasonable error rate and at the same time, it was extremely computationally efficient. In our Neural Network model, the results were slightly better in the development set. Our 4-layer network reaches 35.51% error rate after 50 epochs. For training data, the model yields a 40.96% error rate (closed-loop). Although 50 epochs may be considered a high number, our results showed that the error rate starts to converge after 48 epochs and we got our best error rate at 50 epochs. A large number of training steps didn't affect the overall efficiency since the training time for each step is minimal. The total training and classification time is 92.88 seconds which is faster than any other neural network model we tried.

Conclusions: At first, there was a pressure to make sure that we avoid overtraining. We had to balance between the training set performance and the development test set results. For example, when experimenting with K-NN in Matlab, there was an option that performs an automatic hyperparameter optimization using 5-fold on the training data. Unfortunately, this didn't turn out well since the optimized model gives a low error rate only on training data, but not on the development test set. As a result, we didn't pay much attention to the error rate of closed-loop experiments. The nature of the dataset taught us that a good classifier should give closed-loop results around 40% error rate. Any model with much less error rate is somehow suspicious and fails to accomplish good results in the development test.

When comparing our results of both models, the Neural Network achieved a better error rate (one point less than K-NN). Nevertheless, these results might fall short of outperforming K-NN simplicity and fast performance. In addition to the impressive execution time of K-NN, it is a much simpler model with fewer parameters. Occam's razor tells us when there are two ways of solving a problem that yields the same results, the simpler way is usually better. In our case, the two results are close enough and therefore our choice is in favor of the K-NN model and its prediction.

Algorithm	Dataset		
	Train	Dev Test	Eval
172-Nearest Neighbor	39.72%	37.77%	46.17%
4-Layer Neural Network	40.96%	35.51%	45.66%

Table 1. Comparison between the open-loop snd closed-loop error rates of our K-NN and Neural Network models.