KNN and Multilayer Perceptron for Data Classification

Md Saidur Rahman Pavel
Department of Electrical and Computer Engineering, Temple University
pavel.saidur@temple.edu

Introduction: The training dataset of this experiment contains 300000 samples, and the development set has 15000 samples. There are a total of 3 classes in the dataset labeled as 0,1, and 2. In order to classify the data as a non-neural network approach, I choose KNN. By meticulously experimenting with multiple machine learning models, I found KNN performs best in this particular case. I got an error rate of **25.38%** for the training set, **25.72%** for the development set, and **64.37%** for the evaluation set. For the Neural network approach, a multilayer perceptron with three hidden layers is used. The error rate is found **27.20%, 27.10%,** and **62.17%** for training, development, and evaluation set, respectively.

K-Nearest-Neighbors: KNN is a supervised learning algorithm where a new data point is classified based on its neighboring data points. In a neighbor of K points, the class that has the dominant number of samples will be assigned to the new data points. Several parameters need to be tuned in order to make the classifier effective. **n_neighbors** is one of the crucial parameters, which decide how many data points should be included in a neighbor. For a minimal value of this parameter, training accuracy will be very high, but this will cause overfitting on the training data. **n_neighbors = 110** is selected experimentally, which provides satisfactory performance on the development data without much compromising training accuracy. Figure 1 demonstrated the change of error rate with respect to the number of neighbors.

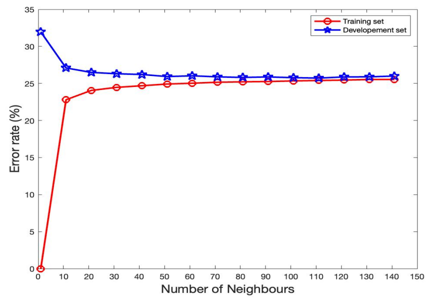

Figure 1: Number of Neighbors Vs Error Rate

For measuring the distances to find the neighbors, several distance matrices are available. They are generally called '**minkowski** distances with an argument '**p**'. for p = 1, it is called Manhattan distance, and p=2 is called Euclidean distance. I got the best result for the **'minkowski** distance with '**p = 6'** in the given data set. The complete parameter set that was used to train the KNN model is demonstrated in Table 1.

n_neighbors	weights	Leaf_size	Metric	p
110	Uniform	30	minkowski	6

Table 1: KNN Parameters

The KNN is trained with 10-fold cross validations. The error rates for each fold are then averaged to yield the final error rate. It is a good evaluation method which ensures a certain result from an algorithm does not come by a random chance.

Multilayer Perceptron: Multilayer Perceptron is a fully connected neural network, which can learn the nonlinear relationship between the input features and labels. It has multiple layers, with each layer having multiple nodes. Each node performs some mathematical operation, i.e., multiply the wights associated with the node with the input and add a bias term. After that, some nonlinear activation function is used so that it can learn nonlinear relationships. After computing the output, the loss between predicted output and actual label is backpropagated to the network and optimize the weights of the network by minimizing the overall cost function. After the training is completed, the network is expected to predict unknown data with satisfactory accuracy.

In this project, Tensorflow is used to build the multilayer perceptron network. Because the problem is a multiclass classification, the labels are encoded to the one-hot encoded form in the beginning. For example, initially, the labels were 0,1 and 2, and after performing one hot encoding, they become 100,010,001, respectively. Therefore, for 300000 training data, the label is in the shape of a 300000×3 vector. The network was created using Keras sequential model. For hidden layers, I tried the values between 2 to 8, but with the increasing number of layers, no noticeable performance improvement was found. As a result, I fixed the number of hidden layers as 3. The number of nodes in each layer is selected experimentally as 50. I also used the ReLU activation function for nonlinear activations. ReLU activation provides a comparatively faster response. For multiclass classification, softmax activation is used in the output layer. In the training process, five fold cross-validation is used. The entire training dataset is divided into five groups, and each time 4 of the groups are used for training, and the remaining one is for validation. The number of epochs is set to 300 because beyond that model's accuracy was not improving much. The complete set of parameters is summarized in Table 2.

hidden layers	**No. of Nodes**	**Batch Size**	**Epochs**	**Learning Rate**	**Optimizer**	**Loss function**
3	50	64	300	0.001	Adam	Categorical Cross Entropy

Table 2: Parameter Set for MLP

For preventing overfitting Dropout and l_2 Regularization was tried, but they provided poorer performance compared to without regularization.

Results: The complete result for the classification is summarized in Table 3. Both KNN and MLP provide a comparable result, although MLP performs slightly better in the blind evaluation set.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
KNN	25.38%	25.72%	64.37%
MLP	27.20%	27.10%	62.17%

Table 3. Summary of the error rate from the two algorithms

From Table 3, the training and development set error is sufficiently close for a given algorithm, but the blind evaluation set is very far apart. This seems unusual, and it's complicated to draw any conclusion without examining the dataset. One possible reason might be the evaluation dataset is not from the same distribution as the Train and Development set.

To further evaluate the performances, some performance measures are tabulated in Table 4,

Algorithm		**Accuracy**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**
KNN	Class 0	0.84	0.81	0.85	0.73	0.77
	Class 1	0.81	0.77	0.83	0.69	0.73
	Class 2	0.85	0.65	0.94	0.85	0.73
MLP	Class 0	0.82	0.87	0.79	0.68	0.76
	Class 1	0.83	0.78	0.81	0.68	0.73
	Class 2	0.84	0.54	0.99	0.95	0.69

Table 4: Performance Measures

Conclusions: In this project, a multiclass classification has been done using two algorithms. From their performance, it is difficult to decide which one is superior. In Training and Development data, the algorithms did a somewhat satisfactory performance. Although theoretically, it is possible for the neural network to provide almost perfect performance, at least for training data using sufficient depth of the layers and adequate numbers of neurons, it could not be possible to build an arbitrarily complex network due to hardware limitations. Proper hyperparameter tuning is another way to get performance-boosting but finding the best set of parameters is challenging. A popular way to find parameters is Grid Search, which is computationally expensive. In this work, I manually tested different parameters to get the possible best performance.