CS 580: Artificial Intelligence (P4)

Md. Ridwan Hossain Talukder

I. Letter Classification with Neural Networks

In this project our goal is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

There are 20000 instances in the dataset (16000 training data and 4000 testing data). Each instances have 16 numeric features. There are no missing attribute values and and as this is a multi class classification problem there is a distribution of target class which is not imbalanced.

A. Data Preprocessing

As we can see in Figure 1, the heatmap shows the correlation between the 16 features and we can see that feature 1 to 5 is positively correlated with each other while the others have a week correlation (positive or negative) or no relation. But we did not remove the columns with high correlation as this is already a preprocessed data and with a very low dimension (16), so removing more dimension may result in too few features in the dataset. Which is supported by an experiment done using PCA with n_components=2, which resulted in a very low accuracy of 13% using RandomForestClassifier, so we opted out for using any feature reduction technique in this classification problem. But as the class labels are some categorical values, we used a LabelEncoder to convert the class labels into integer.

B. Multi-layer Perceptron (MLP) Classifier

The Multi-layer Perceptron (MLP) is a feedforward artificial neural network model that maps the input datasets to a set of appropriate outputs. It has multiple layers, but at least three layers of nodes, an input layer, a hidden layer and an output layer. Each layer is fully connected to the following one. The nodes of the layers are neurons with non-linear activation functions, except for the nodes of the input layer. Between the input and output layer there can be number of hidden layers. So this is the basic architecture of the MLP. I am using the MLPClassifier from Sci-Kit Learn toolkit for this multi label classification problem. To design my neural network I used 1 Input layer with a dimension of 16 node and 2 Hidden Layers with a dimension of 128 and 256 node respectively, and for the output layer there is 26 node. I Used the 'tanh', the hyperbolic tan function which returns $f(x) = tanh(x)$, as the activation function for my neural network. The structure is shown in Figure 2 with a node like structure where the first layer is the input layer, and the last layer is the output layer and the middle two layers are the hidden layers.

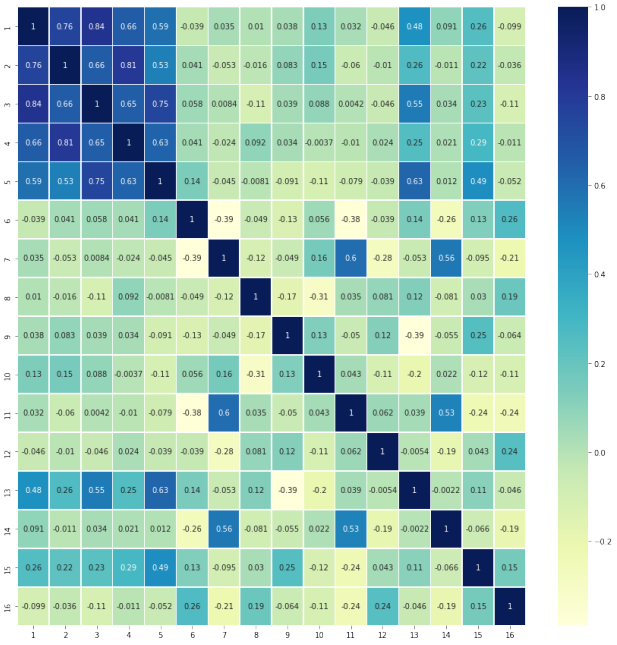

Fig. 1. Correlation between 16 attributes

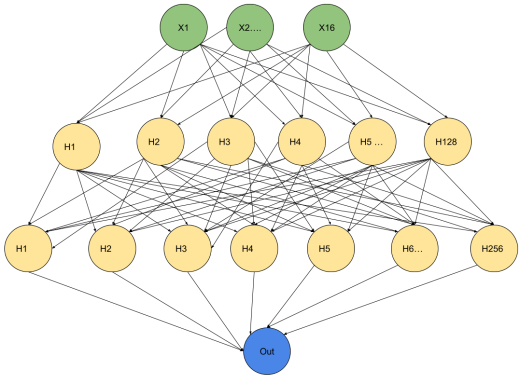

Fig. 2. Multi Layer Perceptron with 2 Hidden Layers

1) Parameter Tuning for MLPClassifier: There are number of parameters in MLPClassifier that can be tuned to get a higher accuracy for this datasets, so I have done some experiments to tune some of the parameters eg: solver, learning_rate, learning_rate_init and activation. To find out the optimized model with this parameters tuned I randomly created a com-

bination of these parameters and using those combinations created a bunch of mlp models and plotted the loss curve for each model (Figure 3) and from Figure 3 we can see that the loss rate is high initially and gradually decreases while converging but if we use sgd solver with momentum or inv-scaling learning-rate the loss rate remains high at the time of convergence while sgd solver and adam solver with constant learning rate reduces greatly and with adam solver it reaches to close to 0 loss rate. So we decided to use the adam solver with constant learning rate and as for the activation 'tanh' performs slightly better in terms of accuracy (0.9668) in test data than using relu (0.9625) activation function so we used relu as our activation function.

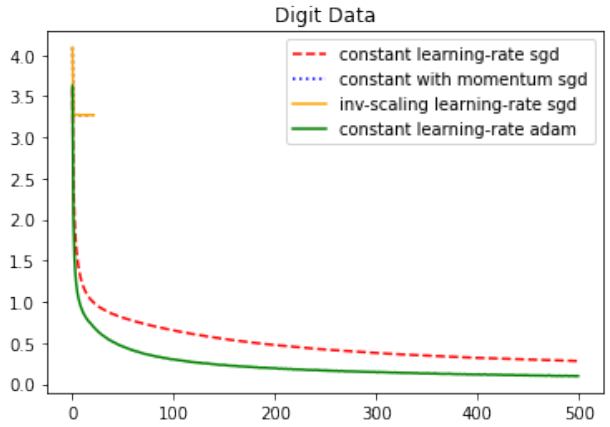

Fig. 3. Experiment on parameter tuning for the MLPClassifier model

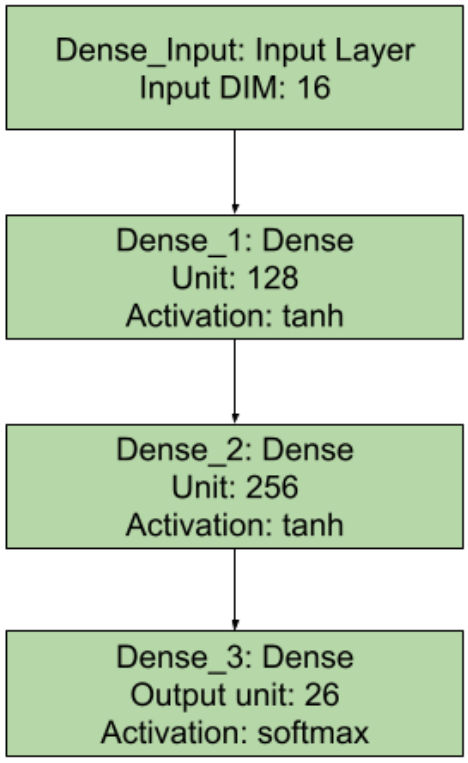

Fig. 4. Keras Sequential Model

C. Neural Network using Keras Sequential Model

I have used the Sequential Model class from Keras to build the Neural Network model for the classifier. The Sequential class is used to define a linear stack of network layers which then collectively builds the model. In our model Figure 4 we added two Dense Layers (which is regular densly-connected NN layer). The first Dense Layer has an output size of 128 and an input size of 16 (equal to the number of features) and the second dense layer has an output size of 256 with the same input size. As I got a better result using tanh activation function in previous experiments I used tanh as the activation function for these dense layers as well. And the final layer which is the output layer has an output size of 26, which is the number of target classes in our dataset and the activation function is softmax for the layer since it's a multi class classification problem. To configure our learning process I used the adam optimizer and the loss function is "categorical_crossentropy" as the target is a categorical attribute and the metrics to evaluate the training is set to accuracy.

1) Parameter Tuning for Sequential Model: As I used the same values for the number of units in layers and activation function that I used for MLPClassifier, I first train my sequential model with those parameters as I already had a better result in MLPClassifier for those values. And for the batch size 32 and using 100 epocs the model had an accuracy of 99.99% on training dataset. To check if there is an overfiting occurring in the training dataset I ran an experiment for number of epocs and found out that the model overfits after it crosses the 100 number of epocs where the accuracy decreases (Figure 5). So for our model we fixed the number of epocs to 100 and it gave us on an average a 97% accuracy on test data which is better than MLPClassifier as well. So did not do anymore parameter tuning for this sequential model as the accuracy was already higher.

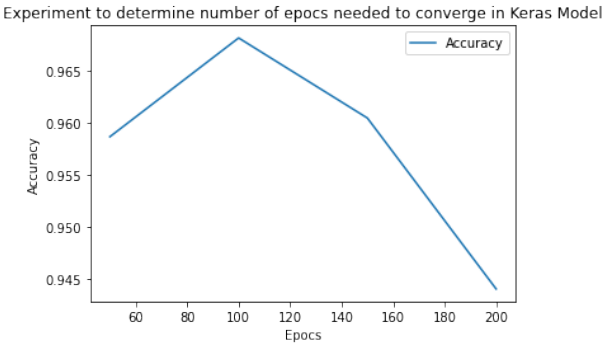

Fig. 5. Experiment on number of epocs to check for overfitting

D. Comparison with RandomForestClassifier

To compare the performance of MLPClassifier and Keras Sequential Model I used the accuracy metric from the SK-Learn toolkit, at the same time I also captured the time it takes to train the model and finally predicted the labels to see how well they perform in terms of time. As we can see that the baseline which is RandomForestClassifier does a really good job in terms of both accuracy and time, it has a accuracy of 96.298% within 3 seconds of time. Where MLPClassifier has a slightly higher accuracy 96.68% than base line but it takes a huge amount of time (100 seconds) compared to the baseline model. The scenario is same for the Keras Sequential Model Classifier too. It has a better accuracy 97.03% but it also takes a good amount (80 seconds) of time in comparison with the baseline model. So the performance is slightly better than the baseline in terms of accuracy but both the neural networks takes more time than baseline.

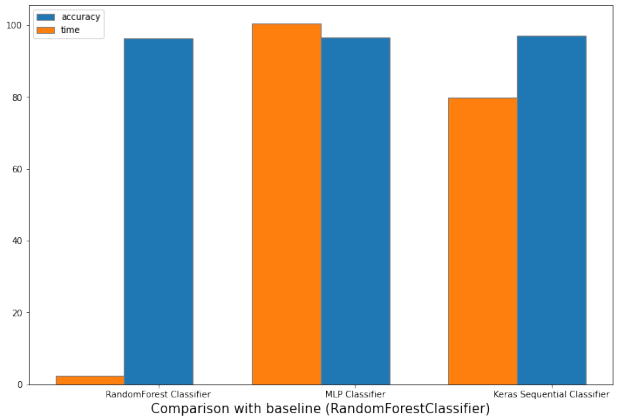

Fig. 6. Comparison with the Baseline Model

E. Results

TABLE I
ACCURACY AND TIME FOR ALL THE 3 MODELS

Metric	Base	MLP	KSM
accuracy (%)	96.298	96.681	97.03
time (seconds)	2.42	100.57	79.84