Multilabel Classification with Hamming Loss and Exact Match Ratio Evaluation

Ahmed Anwer 110045403

IEEE Joint Chapter of Computational Intelligence Society (CIS) and Systems, Man, and Cybernetics (SMC), Windsor Section

Design

Motivation

The algorithm and approach chosen for this multilabel classification was a deep neural network (DNN) or multilayer perceptron. The entire system was written in python. The main libraries used were tensorflow's keras library for model development and scikit learn, numpy and pandas primarily for data processing and model evaluation. One of the most common and robust approaches for multilabel classification is deep learning due to its rapid development capabilities, iterative feature extraction and community support.

Algorithm

A deep neural network (DNN) is a biological brain-influenced artificial neural network that essentially exhibits some decision making capability once it is effectively designed and trained with a sufficient dataset. The employed DNN takes in the 175341 by 42 dataset and outputs two vectors representing the two labels in the given dataset. The output labels are processed and converted to integer values as represented in the dataset; the first label describing 10 mutually exclusive classes and the second label is a binary output describing 1 class. The model leverages optimization, activation and loss functions as well as specific parameters to train effectively against the data. The DNN is essentially an extremely complex function f(x), which takes in data sample x and outputs a result y by processing it through units known as neurons. Each layer of the DNN is a bundle of neurons that can be thought of as mini-functions f(x) by themselves. The function's weight indicates the neuron's output's influence on the next DNN layer after it. When the DNN's prediction y is not equal to the true value of y, it readjusts the weights of its neurons in an effort to better reach the true value of y. Through many iterations, the DNN can be trained to make predictions that are sufficiently or exactly equal to the expected output y.

Architecture

The deep neural network (DNN) consists of six layers of which four are hidden layers. They consist of 1024, 512, 256, 128, 64 and 11 layers from the input to the output, respectively. It must be mentioned that the fifth and sixth layer constituted the overall output layer instead of the output layer being a single-output layer. This was because the first label is a mutually-exclusive multiclass output and the second is a binary output. Normally, multilabel outputs are grouped as one output layer with the sigmoid activation function where the number of neurons represents each binary class. However in this example, label one was not binary whereas label two was. Using the conventional approach where the output is a unified vector, the model wrongly believes that a zero vector is the most correct vector since the true y values also have many zeros. This would be correct for the binary label, but incorrect for the ten class label since it must have exactly one 1 and nine 0s. The multilabel problem was essentially split into a smaller multiclass problem.

As such, keras' functional API allowed for a dual-output layer, one for the ten class label, the other for the binary class label. Each layer is a regular dense layer. Dropout layers were initially used but later omitted since they did not provide a significant impact. The number of neurons were also chosen by trial and error and with computing powering in mind. Additionally, adding more neurons did not provide a significant advantage within what was tested. The first four layers use the Rectified Linear Unit (ReLU) function, the fifth layer uses softmax and the sixth uses sigmoid. ReLU is a piecewise function that is commonly used with classification and regression DNNs. The softmax function outputs a probability distribution for the ten class label, where the class with the highest probability is chosen as this label's prediction. On the other hand, sigmoid is used for the binary class label as it outputs a value from 0 to 1 that is rounded up or down.

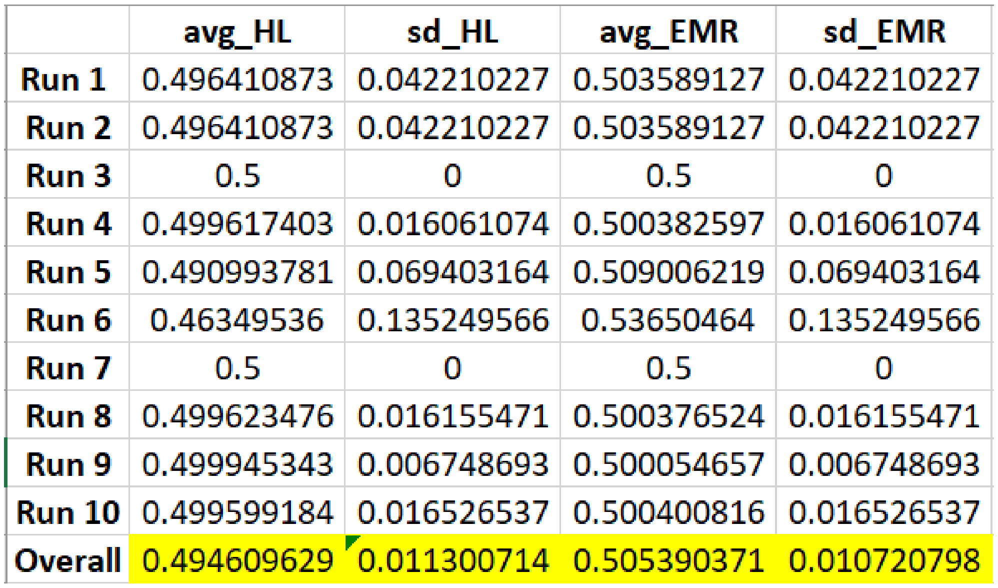
Experimental Results

Experimental Setting

The Adam optimizer function was used with the learning rate of 0.000001, batch sizes were 64 samples, and 20 epochs were chosen. 30 percent of the training data was allocated for validation. The loss functions chosen were binary cross entropy for the binary class label and categorical cross entropy for the ten class label. Furthermore, the data was slightly processed. One-hot encoding was performed on the ten class label, thereby producing a one-hot vector representing the ten numerically labeled classes. This was fed into the output layer using the softmax function. The binary class label was kept as is and fed into its output layer. For evaluation, the output vector was decoded back to the two integers representing the labels

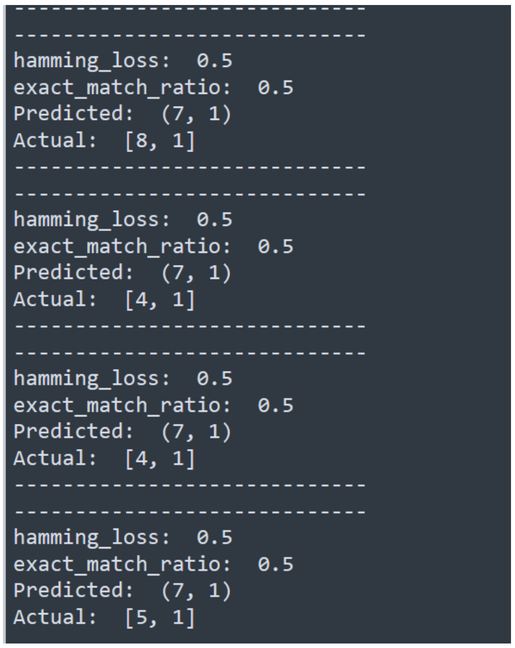
Analysis

The results of ten runs are shown in the following table:



	avg_HL	sd_HL	avg_EMR	sd_EMR
Run 1	0.496410873	0.042210227	0.503589127	0.042210227
Run 2	0.496410873	0.042210227	0.503589127	0.042210227
Run 3	0.5	0	0.5	0
Run 4	0.499617403	0.016061074	0.500382597	0.016061074
Run 5	0.490993781	0.069403164	0.509006219	0.069403164
Run 6	0.46349536	0.135249566	0.53650464	0.135249566
Run 7	0.5	0	0.5	0
Run 8	0.499623476	0.016155471	0.500376524	0.016155471
Run 9	0.499945343	0.006748693	0.500054657	0.006748693
Run 10	0.499599184	0.016526537	0.500400816	0.016526537
Overall	0.494609629	0.011300714	0.505390371	0.010720798

The overall hamming score is about 0.4946 +/- 0.0113 and exact match ratio 0.5053 +/- 0.0107. Hamming score essentially reports the incorrect labels divided by the correct labels and exact match ratio reports the degree to which the predicted label exactly matches the actual label. From the results, it is apparent that there is roughly a 50-50 split between the two metrics, and in runs three and seven, this was exactly the case. Ideally, the hamming score should be 0 and exact match ratio 1.0. Additionally, since there were only two labels of which one was binary, the model had a 50 percent chance to score 0.5 in the hamming loss if it were to ineffectively predict the same binary label value each time. On the other hand, the ten class label is more difficult to predict. The following are a sample of predictions from the test data and the actual labels:



```
----------------------------
----------------------------
hamming_loss:  0.5
exact_match_ratio:  0.5
Predicted:  (7, 1)
Actual:  [8, 1]
----------------------------
----------------------------
hamming_loss:  0.5
exact_match_ratio:  0.5
Predicted:  (7, 1)
Actual:  [4, 1]
----------------------------
----------------------------
hamming_loss:  0.5
exact_match_ratio:  0.5
Predicted:  (7, 1)
Actual:  [4, 1]
----------------------------
----------------------------
hamming_loss:  0.5
exact_match_ratio:  0.5
Predicted:  (7, 1)
Actual:  [5, 1]
```

These were taken from rows 400-450. It is apparent that the ten class label incorrectly predicts many times with seven, whereas the two class labels seem reasonable as it is correct in this given range.

Therefore, this deep neural network is roughly 50% accurate. It is comfortable with predicting the binary label but inaccurate with the multiclass label. This model can be improved by editing the model structure but may also benefit from more data processing. The data provided was anonymous and so it was difficult to process it further without knowing its real life implications. Nevertheless, this algorithm is a strong first-step solution in tackling this multilabel classification problem.