CSOC-IG(Assignment-2)

immediate

Abstract—Deep Learning is a powerful subset of Machine Learning that uses multilayered neural networks to learn from data. The primary difference between traditional machine learning algorithms—such as regression or basic classification—and deep learning lies in their complexity and implementation. Dense neural networks in deep learning are significantly more complex and large-scale, often making the entire system appear like a "black box": we provide input data, but the internal workings are not easily interpretable, even though we obtain the desired output.
Despite this opacity, deep learning excels at capturing intricate patterns in data, handling class imbalances, and solving problems involving massive datasets. In such cases, the emphasis shifts from understanding every mathematical detail to achieving accurate and reliable results, which is where deep learning truly shines.

1. Understanding the provided Dataset

The dataset that has been provided for this week's assignment is the **'No-show' dataset**, which basically captures the information of multiple patients and the fact whether they show up for a medical appointment or no. The features as provided in the dataset to predict this fact are PatientID, AppointmentID, gender of the patient, Neighbourhood of the patient, their time of scheduling the appointment v/s the time of the actual appointment, and a few aspects of medical history like diabetes, hipertension, alcoholism etc. Thus the dataset is a comprehensive collection of features and figures which may answer some interesting questions like *people falling under which category of age are likely to be the most regular in attending medical appointments?*-or *who among men and women are more concerned about their health?*

2. Data pre-processing techniques

2.1. Dimensionality reduction

It is very important in machine learning to reduce the dimensionailty of the dataset so as to contain the most relevant and independent features. Besides, highly correlated features are often problematic in training the model since in determining the model fucntion, often the weights of these features are unnessarily computed and are anyways numerically related so as to exhibit the relationship between these correlated features. Apart from highly correlated features, there may exist some features which are simply irrelevant and have values much greater or lesser in magnitude as compared to other important feature values because of which weights corresponding to these features are indescriminately updated unless compensated by regularization. They may be imbalanced values too. Thus removing them is the best choice.

In this dataset there are two quantities with unusually high numbers because they correspond to a form of identification- that is the PatientID and the AppointmentID. ***Now the approach behind dropping AppointmentID and retaining PatientID*** is that PatientID corresponds to a specific patient, irrespective of the number of appointments taken or the number of visits. It basically captures the ***behavioural pattern*** of specific people towards their awareness about their own health, or simply how they think they benefit from the appointmnets and who are regular. Thus this feature is retained.

However the AppointmentID feature *corresponds to the ID alloted corresponding to every appointment made*, this may not at all be signficant in predicting the choice of the patient to show up at the appointment, thus it has been dropped.

2.2. Target Encoding for Patient IDS

Since the patient ID basically represents every individual patient, thus it is a feature constituting multiple categories-each category an individual and new patient. One more observation is the imbalance in the target prediction, No-show indicates their abstinence from attending the appointments and the maximum category is 'No'- most patients tend to redeem their fee for making the appointment ;) Thus the minority class is 'YES'-they dont show up. Thus the target encoding of the patient ID has been done taking into account this class imbalance. Thus the target encoding has been done based on the mean of the minority class-tendency of the patient to miss the appointment with respect to all the occurrences of a given patient in the dataset.

2.3. Feature Engineering

Feature engineering refers to the assembly of new, relevant features pertaining to our given dataset either by combining two or more existing features in a manner that combines both their essential contribution to the overall dataset, or altogether creating a new relevant feature with new data values. In our case we have two such features which can actually be augmented to give a more relevant feature and these two thereafter can be dropped individually from the dataset. ***These two features are-Scheduling Time and Appointment Time.*** *Firstly, we can observe on analyzing the dataset using pandas that the appointment time is actually standard 00:00:00 for all data samples thus clearly signifying that either the time has not really been recorded or its not so essential.* Then coming to the Scheduling Time, which includes both the date along with a unique time of scheduling. *Now we can do two constructive things out of this- extract the timing hours of scheduling the appointment, since time of scheduling does act as a wholesome indication of the urgency of the requirement of medical aid and the person will surely not miss such an appointment, or it may just simply be a follow up which the patient with regret misses out on.* Thus t**he hour of the scheduling time has been extracted and encoded using hour sine and cosine cycles to capture the cyclic nature well-This was a better choice in terms just simply assigning buckets/ categories like morning, afternoon and evening to different time intervals.** ***The second thing is to calculate the delay in the actual organization of the appointment after scheduling in terms of the number of days(this has been done since anyways the time of the appointment is not clear)***. The delay in terms of the number of days again is a good measure of significance and urgency....follow ups may be scheduled every 3 months, a medical emergency can be organizing the appointment on the very same day or even if its a first time visit it most probably will be organized on the same day. Thus dropping the existing two columns and including this feature instead makes more sense.

2.4. One-Hot Encoding

Neural networks work on mathematical data, thus we require numbers only-words dont make any sense to the model. Thus for features like Gender, Neighbourhood-we construct a one-hot encoding, for every specific neighbourhood(this feature is important in throwing light upon the effect of proximity of hospitals and infirmaries for attending the appointment for different age groups of people or different people in general). One hot encoding for one gender is sufficient, as in simply a collection of 0s and 1s for supposingly the female category-if not a female then a male.

2.5. Min-Max Scaling

Neural networks work the best with normalized values. Thus values like age have been brought to a range of 0 to 1 using Min-max scaling for best results, since all other values are primarily categorical.

All these data pre-processing techniques lead to a final number of 93 feature columns in the engineering dataset.

3. Approach 1(Pure python implementation)

The first approach is the pure python implementation of a neural network. The neural network implemented consists of 4 hidden

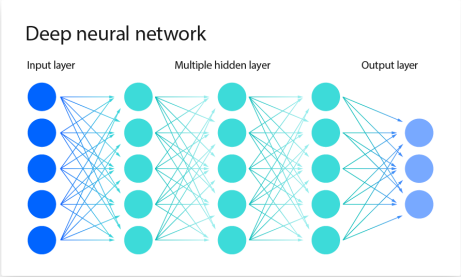

Figure 1. Deep neural network

layers along with one input and one output layer. The 4 hidden layers each consist of 256,128,64 and 32 neurons each respectively and the activation for each is ReLu. ReLu is consisdered to be one of the best choices for introducing non-linearity and to ensure a smooth training. Besides, since the output predicts two labels-Yes and No, the output activation is the sigmoid function. The algorithm is consistent with the foundational skeleton of a neural network. Since our activation function is ReLu , He initialization technique or Kaiming Initialization technique has been utilized, the formula for which is as follows-

$$w = random * \sqrt{\frac{2}{inputsize}}$$

Using OOPS concepts, a class for each hidden layer has been utilized to initialize the weight matrices and bias vectors via a parameterized constructor and another specific class for the output layer has been utilized in order to take into account the difference in the final activation along with the computation.

Hyperparameter tuning includes number of epochs-finally set to 686, learning rate finally set to 0.03.

The algorithm followed first involves the entire process of forward propagation, that is computing the output of each layer and feeding it to the next layer as input recursively. For the hidden layers-

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \tag{1}$$

$$a^{[l]} = ReLU(z^{[l]}) \tag{2}$$

For the output layer-

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \tag{3}$$

$$a^{[l]} = \sigma(z^{[l]}) \tag{4}$$

After undergoing forward propagation and calculating all the predicted values, we move on to calculate the objective/loss function.Since this is a classification task, we have utilized the Cross Entropy Loss, however pertaining to the imbalance of classes in the target variable-that is huge number of Nos as compared to the Yeses-we use a weighted loss function. The weighted cross entropy loss heavily prioritizes the minority class as compared to the majority class using two weights-w1 and w0. w1 for the positive class has been tuned to a value of 0.75 whereas w0 has been tuned to a value of 0.25.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[w_1 \cdot y^{(i)} \cdot \log_2(\hat{y}^{(i)}) + w_0 \cdot (1-y^{(i)}) \cdot \log_2(1-\hat{y}^{(i)})\right] \tag{5}$$

The values of predicted labels are clipped first in order to prevent errors arising due to 0 values inside log in the cross entropy loss formula and that has been sure using a parameter ϵ , which has been set to 10^{-15}. Besides the trend of other accuracy metrics including the F1 score, recall and precision is kept track of. In imbalanced classification problems, where one class significantly outnumbers the other, accuracy becomes a misleading metric because a model can achieve high accuracy by simply predicting the majority class. Metrics like precision, recall, and F1 score provide a clearer picture of model performance on the minority class. Precision measures how many predicted positives are actually correct, while recall captures how many actual positives were correctly identified. The F1 score combines both, offering a balance between precision and recall. These metrics focus on the performance of the model where it matters most—in detecting the minority class. This makes them more reliable and informative for evaluating models in imbalanced datasets.

After all these measurements, finally backpropagation is performed to update the weights across all layers-

Error at output layer-

$$\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial a^{[L]}} \circ \sigma'\left(z^{[L]}\right) \tag{6}$$

Error at the hidden layer-

$$\delta^{[l]} = \left(W^{[l+1]^\top} \delta^{[l+1]}\right) \circ \sigma'\left(z^{[l]}\right) \tag{7}$$

Final update of the parameters via the gradients-

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \delta^{[l]} a^{[l-1]^\top} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[l]}} = \delta^{[l]} \tag{9}$$

Thus this constitutes the complete structure of an ANN from scratch.

Figure 2. Code snippet

4. Approach 2-Using Pytorch

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It provides a flexible and intuitive way to build, train, and deploy neural networks. Unlike traditional static computational graph frameworks, PyTorch uses dynamic computation graphs (also called "define-by-run"), which makes it easier to debug and experiment with models. PyTorch integrates well with Python and popular libraries like NumPy, and it includes tools for automatic differentiation (autograd), GPU acceleration, and model serialization.

To build a neural network in PyTorch, we start by defining a class that inherits from torch.nn.Module. Within this class, we define the layers as fully connected layers 1,2,3,4(fc1,fc2,fc3etc) along with the output layer in the _init_() method and the forward pass in the forward() method. After creating the model, we define a loss function (same cost sensitive cross entropy loss as used in approach 1) and an optimizer (Adam in our case). Training involves looping over the dataset, computing predictions, calculating the loss, backpropagating gradients with loss.backward(), and updating weights using optimizer.step().

The hyperparameter tuning involves setting the weights for the positive and negative class as 0.75 and 0.25 respectively , just that number of epochs have been increased from the first approach to 1000.

5. Approach 1 v/s Approach 2

In both the cases the entire dataset has been divided to give 80% training data and 20% test data.

For approach 1-

1.Final F1 score on the training data-0.75
2.PR AUC- 0.317
3.Final F1 score on the test data-0.7538
4.Final Recall on the test data-0.8601
5.Final precision on the test data-0.6709
6.Test loss-0.13087190988381497

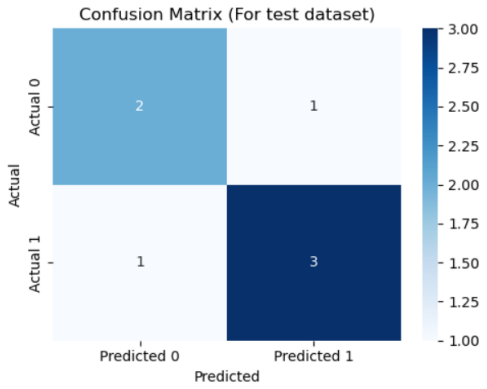

Figure 3. Confusion matrix-approach 1

For approach 2-

1.Final F1 score on the training data- 0.7985
2.PR AUC-0.8498
3.Final F1 score on the test data-0.7523
4.Final Recall on the test data- 0.8795
5.Final precision on the test data-0.6573
6.Test loss-0.1309

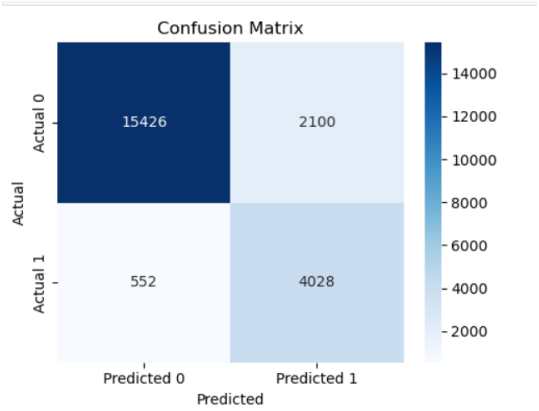

Figure 4. Confusion matrix-approach 2

A pure Python implementation of a neural network, while educational and transparent, tends to be inefficient in terms of memory and speed. It relies heavily on nested loops and basic data structures like lists or NumPy arrays, which are not optimized for large-scale linear algebra operations. These operations are computed sequentially on the CPU, and memory is managed at the Python interpreter level, which adds significant overhead. As a result, training even small neural networks can be slow and memory-intensive, making pure Python impractical for real-world deep learning tasks.

In contrast, PyTorch is a highly optimized deep learning framework that uses torch.Tensor objects, which are more memory-efficient and capable of running on both CPU and GPU. It leverages parallel computation and low-level libraries like cuDNN and MKL to accelerate operations dramatically. PyTorch also supports dynamic computation graphs, automatic differentiation, and in-place operations, which further reduce memory usage and speed up training. This makes PyTorch ideal for building, training, and deploying deep neural networks efficiently, especially when working with large datasets and complex models.

6. Graphs

For approach 1-

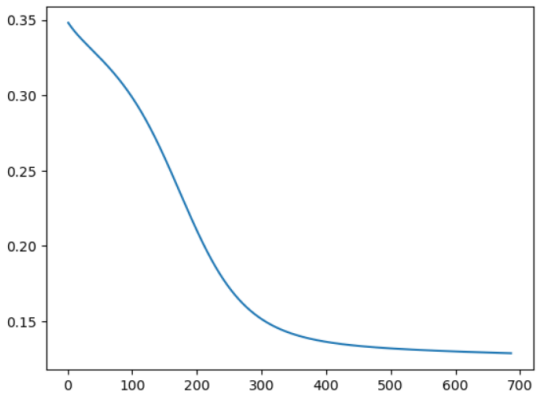

Figure 5. Loss v/s Epochs- Approach 1

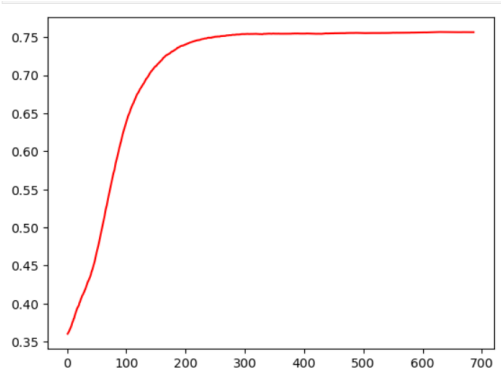

Figure 6. F1 score v/s Epochs- Approach 1

For approach 2-

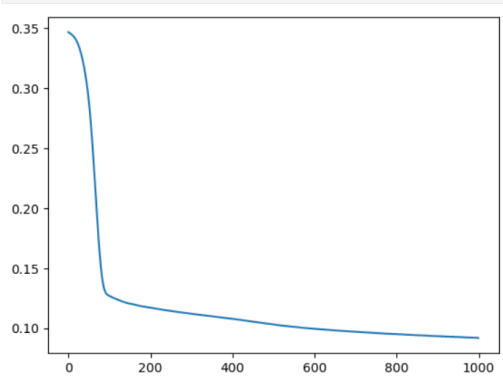

Figure 7. Loss v/s Epochs- Approach 2

The above graphs show the variation of losses and f1-score across the epochs for both the approaches.The values are close, but space and time complexity create a difference.

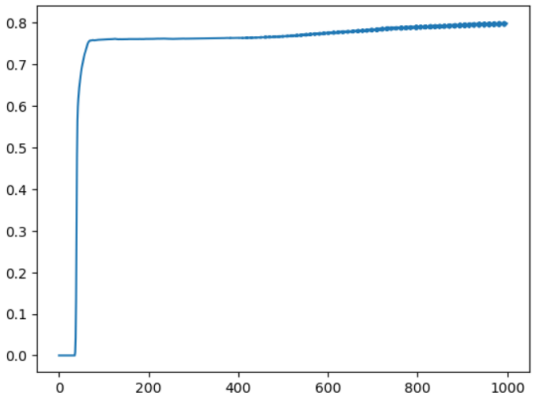

Figure 8. F1 score v/s Epochs- Approach 2

7. References used

1. Kaggle-for learning about the dataset
2. COPS IG resources for learning about ANNs
3. ChatGPT for grammatical errors, factual questions, debugging