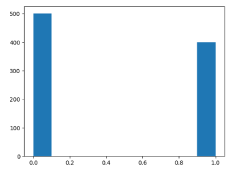
FDA ASSIGNMENT (COURSE PART 1)

PART 1 – NO AI

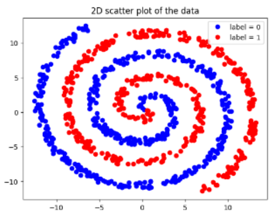
1.

a) 2 features, 900 samples b) We have binary classification since we only have 2 label values, 0 and 1

c) We can infer that we have 500 0 samples and 400 1 samples, not too unbalanced.



2.



a) ...

b) The data does not look linearly separable at first glance, since it has a spiral-like shape.

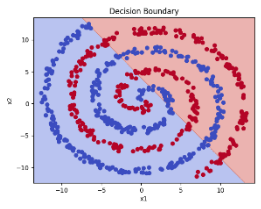
c) No since the data is not linearly separable.

d) The bayes optimal classifier would follow the spiral pattern that we can observe in the image. It would be able to achieve an empirical risk of 0, since the data can be perfectly separated due to the lack of noise (the only somewhat problematic part might be at the very middle but since I am not completely sure I am making a bit of an assumption that we have 0 noise overall)

3.

a) The split is necessary, because you want to be able to train on a part of the data and then when you think you have a good model, you can verify it by using your test data, since if the model is robust it will be able to work on them just fine, otherwise it might have overfitted. b) Well, LDA is out of the question since it makes the assumption that we have Gaussian distributions (which in this case we don't) as for the other 2, I would choose Logistic Regression, mainly because it is simpler and because it works with probabilities, which might be useful in this case since a poor result is a given, considering all 3 classifiers are linear classifiers. But I can't find any other reason to not use SVM, so that could also be a "solid" choice ("solid" as in maybe better but still very bad due to being a linear classifier)

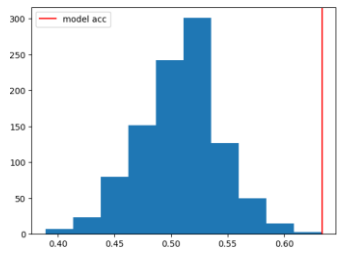
c) 0.6333333333333333, well the classifier is hardly learning since it still isn't performing well (to be expected) since an accuracy of 0.5 would be purely guessing.



d)

4.

a) 0.6, so it's almost as good as the classifier

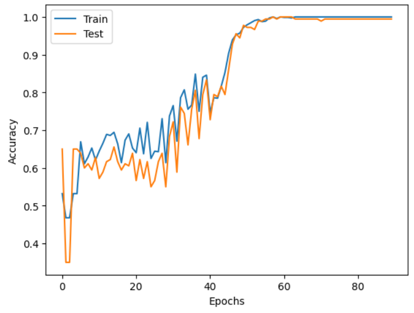


b)

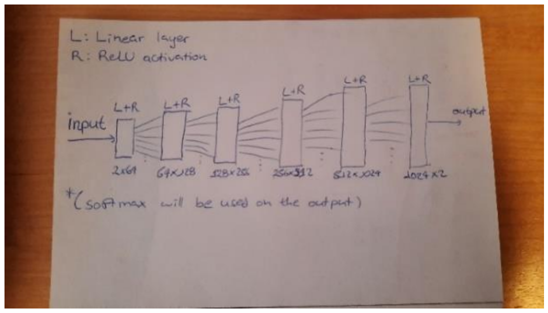
c) The plot suggests that we are doing a little bit better than guessing. We are doing slightly better, however there is a small chance that even by just guessing, this accuracy could be achieved. d) Well, I repeated it 1000 times (as suggested in the pdf). I believe 1000 is enough so we can get a good idea of how the accuracies are truly distributed. If we used less then there exists the possibility that outlier values (such as >63%) could appear a few times and skew the distribution. (when I say less, I mean considerably less, like 100 or 50. Obviously 900 would be just fine) e) 0.65, so they are not the same which is to be expected. Afterall the chance accuracy will almost certainly be different every time I run the code of that cell, whereas this accuracy will always stay the same, since the number of 0's and 1's in the test set doesn't ever change. Still they are close! f) From what we can observe guessing, picking the label that belongs to most samples and training a linear classifier give almost the same results. The reason for this is because the classifier simply can't do that good of a job, because the data is not linearly separable. Looking at the end result, I don't think it would be worth it to train a classifier in this case, since the result is slightly better than guessing.

5. a) … (code)

b) The learning curves suggest robustness and good generalization, due to both curves being close to one

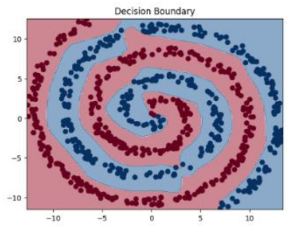


another.



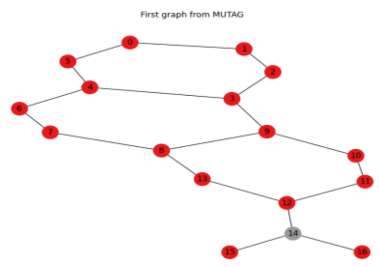
c) Well, my idea was to first find a good architecture, where with enough epochs we can achieve almost perfect learning on the training data. I did this by adding one layer at the time, while also increasing the number of neurons (after a point, no matter how many neurons you have, you won't be able to get a better accuracy without adding another layer). After doing this manually and finding a good architecture, I used 10Fold cross validation to fine tune the learning rate (There are countless of research papers and articles like this: https://mohitmishra786687.medium.com/the-learning-rate-a-hyperparameter-that-matters-b2f3b68324ab which show that just by tuning the learning rate, you should achieve the best result). Finaly, I found that a learning rate of 0.01 gave me the best results.

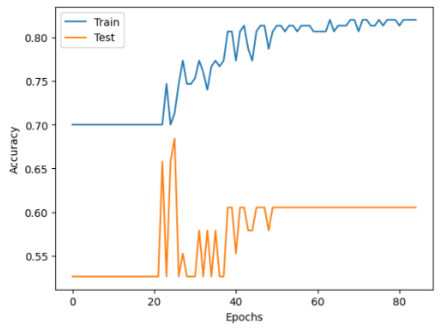
d)



PART 2 – AI ASSISTANCE

1. a) 188, 7, 2   b) Per the ai model (prompt: Can you explain what the MUTAG dataset represents? Specifically tell me what real-world object does a sample represent? What do the nodes, edges, features, and labels correspond to.) and google (https://paperswithcode.com/dataset/mutag) Sample: Chemical compound modeled as a graph Nodes: Atoms, Edges: Bonds between the atoms, Features: Features are one-hot encodings, so in essence the feature vector tells you what type of atom you have (Oxygen, Hydrogen, etc.) and you have 7 of those possible types, Labels: 1 if chemical compound is mutagenic, 0 if not. c) The first sample has 17 nodes and 38 edges (prompt: how to get the nodes and edges of the first sample in the MUTAG dataset?) d) (prompt: plot the first sample of the mutag dataset with nodes colored by second feature)

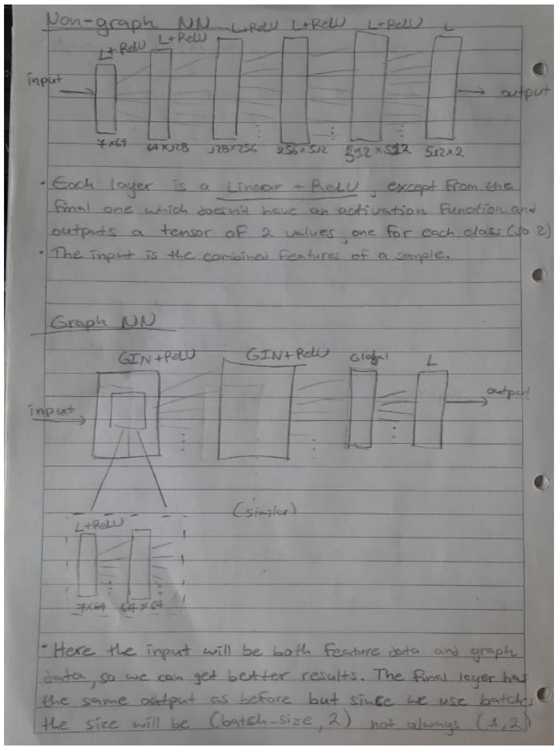




2.

We shall use the same linear model as before, so Linear Regression. a After asking the AI "How can I prep the data so that I can train without using graph connections? I am using the mutag dataset. Give me suggestions" one of the many things it suggested was that I should get the feature vectors of all the nodes in each graph add them together and then divide the whole vector by the number of nodes in that set. It seemed sound to me, so I decided to follow this approach. I tried a few architectures and hyperparameters on the train data, before settling for the one in the notebook. b) , fairly poor results, we aren't really learning much, peaking at around 68% accuracy and then consistently plateauing at 60% c) Choosing most frequent label = 0.6648936170212766, Guessing = 0.42105263157894735, Linear = 0.5263157894736842, NN = 0.6053, overall we are doing barely better than just guessing, considering that even with the NN the accuracy remains low (plateauing at around 60%)

3.

a) 

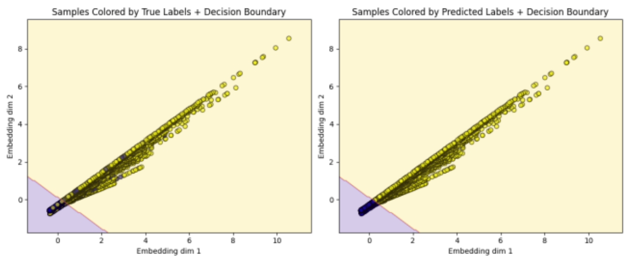
- Each layer is a Linear + ReLU, except from the final one which doesn't have an activation function and outputs a tensor of 2 values, one for each class (so 2)
- The input is the combined features of a sample.

- Here the input will be both feature data and graph data, so we can get better results. The final layer has the same output as before but since we use batches the size will be (batch-size, 2) not always (1,2)

Different inputs, layers and overall architecture (+ graph nn uses pooling)

b) AI prompt : "How does a GNN take into account graph info and what does the layer number mean?" Per the ai and a bit of cross referencing/ searching on google I understood that the high level idea is that each node doesn't just use it's own information but through message passing it tries to take into account information of its surrounding/ neighboring nodes before updating it's weights. As for the layers, they tell us how far the neighbor you receive info from can be. Layer 1 -> immediate neighbor , Layer 2 -> neighbor of neighbor, etc. c) In essence I just created a validation dataset from the training dataset and I manually tried out learning rates and architectures. I also looked a bit on batch sizes but didn't get much of a change (even though I believe this may vary a lot if you use a GPU) d) Accuracy = 0.79, P-value = 4.051328899727381e-06 (not exactly 0 probably but pretty much 0). Looking at the accuracy, it's indeed better than before and by quite a bit. Overall on google I saw that models such as mine should reach around 85% accuracy and considering the fact that that's the range of accuracies I was able to get on previous runs, I would say the model is doing a solid job.

4.



a)

For this question I felt it was appropriate for the prompt to be the question itself and some extra info about the model architecture. b) After some tests I found that using 2 + 16*2 (16 GIN layers with 2 linear layers each) with almost all having 6 input and 6 output layers (the input was 7in 6out and the final 2 layers where 6in 2out and 2in 2out respectively) gave me really good results (around 85% accuracy) on the test data. c) Surprisingly, the result was really good! To me it would seem that even though we don't have enough neurons, the transformations due to the layers + the ReLU activation function make up for it (granted I need to use a lot of layers). HOWEVER, it's important to know that through different runs I got vastly different results, which may also suggest that learning in this case is very unstable and perhaps not that great!

5.a) I used the AI wherever I believed I truly needed assistance. I always tried to google first to hopefully find something and I also cross referenced the AI results with either stuff I later found on google or just my own logic. b) Cross referencing on google + my own logic. c) Well the AI was helpful in the last graph part as well as in the beginning of part 2. I didn't really encountered any problems with the ai, worst case if the first response seemed weird I just restarted, ran the same prompt and got a slightly different result that hopefully seemed correct.

References: https://www.edureka.co/community/72971/how-to-load-numpy-array-in-jupyter-notebook, https://scikit-learn.org/stable/modules/generated/sklearn.inspection.DecisionBoundaryDisplay.html, https://www.w3schools.com/python/numpy/numpy_random_permutation.asp, https://www.geeksforgeeks.org/plot-a-vertical-line-in-matplotlib/, https://psrivasin.medium.com/plotting-decision-boundaries-using-numpy-and-matplotlib-f5613d8acd19, https://paperswithcode.com/dataset/mutag, https://medium.com/@james-coffey/gcn-with-neo4j-and-pytorch-using-mutag-dataset-in-ten-steps-ea3169c9c023, https://milvus.io/ai-quick-reference/what-is-graph-neural-network-gnn-in-deep-learning, https://matplotlib.org/stable/users/explain/colors/colormaps.html

Instructions: unzip the data folder and place it in the same directory as the code, run everything cell by cell, use python version 3.10.8