CSE 881 Project Report

Cameron White, Ali Saffary, Brenden Hein

April 2022

1 Introduction

Classification is a major area of interest in both machine learning and data mining. The task of classification involves taking in a multitude of samples, where each sample is represented by a set of features, and the types of these features are the same between samples. For this project, a large scale classification task was assigned. This task, as described previously, involves a set of features for every sample provided. However, the data set used involved a graph structure, as well. Thus, not only does this project involve a data matrix of the samples and their corresponding features, but the samples are also represented by an adjacency matrix representing the structure of the graph and the relationships between nodes.

Throughout this project, multiple methods will be evaluated. Firstly, a couple simple base classifiers will be used. However, they ignore the adjacency matrix, and thus, the results may potentially be sub-optimal. These two techniques are K Nearest Neighbors and Closest Mean. These are simply meant to be used as decent base classifiers and will give a good understanding of where this project sits initially.

After, two more advanced models were tested, this time using the adjacency matrix. These models were a Graph Neural Network and Graph Convolutional Network. With the two models, the hope is to improve the accuracy from the initial to classifiers substantially, through deep learning and using the corresponding adjacency matrix. The overall goal of this project is to accurately classify the testing points with an accuracy of at least 80%.

2 Attempted Methods

2.1 The Data Set

As discussed in 1, the data set being worked with for this project is a data matrix and a corresponding graph to represent the relationship between samples. In terms of what data is initially provided, there was:

- 2048 samples, with 496 samples used for training and 1984 samples used for testing

- 1390 features for every sample, where each feature is binary
- 496 indices corresponding to the 496 training nodes
- 1984 indices corresponding to the 1984 testing nodes
- 496 labels, with each label corresponding to each element in the training nodes

2.2 Data Preprocessing

2.2.1 Cross Validation

The first data preprocessing technique applied was k-fold cross validation. The way cross validation works is by splitting the data into K-folds. For example, if 5-fold cross validation is being performed on 100 data points, there will be 5 folds, each with 20 points.

Once the folds are created, each fold will serve as the testing set and the rest of the folds are combined into a training set. The model will then be ran K times, and for each time, the testing set will be a different one of the folds while the training set is a combination of the rest of the folds. After the error rate for each fold is computed, an average error rate across all the folds is computed as the overall average error rate.

2.2.2 PCA

PCA is a dimensionality reduction technique that reduces the number of features of a data set while simultaneously trying to retain as much information as possible. The way this works is by finding the eigenvectors and eigenvalues of the covariance matrix. Then, these eigenvalues are ranked in descending order, and depending on how many dimensions you want to reduce down to, you select those top eigenvectors. This becomes your feature vector in your new d-dimensional space.

We attempted using PCA in our project to get a feel for how the data was spread out and to see if working in a reduced dimensional space yielded better classification results. However, PCA performed poorly. The data was all over the place. There were many overlapping classes, and some classes, such as class 2, seemed to exist everywhere in our projected feature space. This was tested for 2 and 3 dimensions for visualization purposes. Higher dimensions, such as 50, were tested, as well, with a similar conclusion drawn: PCA is not helpful to our dataset.

This is most likely due to the fact that our features were binary, i,e., the variance of the features are most likely not that significant. To add to this, the overall matrix was very sparse, making it even harder to retain the info needed for PCA to work well.

We also tried MCA, which works well with categorical data, and logistic PCA, which works well with binary data. However, the results were not good

and there were issues running it, so it was scrapped in favor of using the original feature space.

2.3 K-NN

K-NN is a non-parametric density estimation method used for classification in both the binary and multi-class case. The way it works is by going through testing points, and for each test point, it finds all of its neighbors based in a Euclidean space. Then, using the hyperparameter, k, which is how many of its nearest neighbors to look at. Out of those K nearest neighbors, the class that is most prevalent within them will be the class the testing node is assigned to.

2.4 Closest Mean

Closest Means works by first clustering all of the training nodes together by class, so there will be a cluster for class 0, 1, 2, etc. Once the clustering is complete, the centroid is computed for each cluster. Similar to K-NN, Closest Means works by finding the Euclidean distance between a test point and all of the clusters' centroids. Whatever centroid the test point is closest to will be assigned to the class of that centroid.

2.5 GNN

A Graph Neural Network (GNN) is an algorithm that looks back at the neighbors of the current test node by k number of hops. It takes the average of the features of each neighbor node at hop k, produces a learnable weight for the average and adds the average with the k-1 hop's node's features. This algorithm recursively creates neighbor averages and each hop until it reaches the current test node. Finally, the results of the averages of the current node's neighbors will be concatenated to the features of the current node and the concatenated feature set will be passed to linear neural network layers with a final layer of Softmax.

Our implementation of the GNN is a one-hop-limited version of this GNN. That is, we only go back by 1 hop and only take the average over the features of the neighbors of the test node. By doing this we are ensuring that the model does not get too complex and the training and testing times are relatively fast.

We did not use torch geometric or any other pre-built GNN algorithm. We built the model by hand and using basic PyTorch linear neural network layers. The average neighbor feature set was created in the prepossessing stage by looking at the neighbors of each node, taking the average of the features, and concatenating the average of the node itself. This created a 2780x1 feature set instead of a 1390x1 feature set. From here on out it was simply passing each node's new feature set to the neural network.

The best validation accuracy we got was from a GNN with 3 hidden layers with respectively 500, 30, and 7 nodes. Taking 2780 features as input and resulting in a Softmax layer with 7 nodes as the output layer. Each hidden

layer is complemented by a ReLu activation function. The best learning rate found was 0.01 and 30 epochs of training proved to be the appropriate number of epochs to train on.

2.6 GCN

GCN however is a much more complicated graph classification algorithm because it uses convolutional layers instead which are considerably more complex.

Inspired by the code shown in class by Wei, we used the torch geometric library tools as our base of implementation. 1 hidden layer with 36 nodes proved to be the best number of hidden layers. An implementation of 2 hidden layers with respectively 180 and 36 layers was attempted, but was too complex and would not converge as graciously as the model with 1 hidden layer.

Training for 13 epochs was the appropriate number of epochs to train on. Any less and the model would not find the best validation results. On the other hand using more than 13 epochs would start to deteriorate the model's performance with the validation data.

We trained our the model with ReLu, ReLu6, ELU, and Sigmoid activation functions. ELU produced the best validation accuracy by a small margin and Sigmoid produced the worse validation accuracy by a few percents.

3 Performance and Analysis

3.1 K-NN

The results for K-NN were very disappointing. Initially, the model was performing around 18%, which isn't much different from rolling a die. However, after it was discovered that the labels set was incorrect, K-NN was then tested again. And It did perform better, but by no means good; our validation results were around 24%. When we submitted, our results turned out even worse at 23%. So while K-NN was great to test out, the results were very bad, especially considering the adjacency matrix of the graph structure was never used.

3.2 Closest Mean

The results for this model (which, again, does not use the graph's adjacency matrix) were much better. Initially, with testing, it appeared that closest means performed well on the validation set with an accuracy of around 80%. Given how simple of a classification model was, it seemed dubious that the results were that high. However, once the submission results were returned, it appeared that the model did not do as well as hoped, with 66% accuracy. So while better, it was still less than originally planned.

3.3 GNN

As simpler models weren't returning the results desired, more advanced methods had to be used. When using GNN (Graph Neural Networks), the results were better but definitely finicky. On the validation set, the results varied from 40% - 77%. Due to unpredicability of the results of GNNs, it was decided not to waste a submission on that and test of GCNs first.

3.4 GCN

GCN by far outperformed every other model. The submissions were tested twice for this model. The first time, a ReLu activation function was used. After fine-tuning the hyperparemeters, the GCN returned 85% accuracy on the validation set. Once submitted, the GCN still performed about the same on the test results at 84%. These results were what was initially hoped for, and thus, the model was fine-tuned more to see if even higher results could be achieved.

In the second submission, an ELu activation function was used instead, which is similar to ReLu. These results were slighty better by a percent, with the validation results set being at 86% accuracy and the final test results being at 85%. Other activation functions were tested, but they are not mentioned due to the fact that they were yielding worse results. With this, we were happy with the results and stopped finetuning the models.

4 Lessons Learned

During the process of development of this project, there was some very important knowledge that we were able to acquire applying to a multitude of different topics. One of the topics that was very heavily involved in this project was data comprehension. With the particular set of data that was given to us, it was very difficult to understand the format of the data and we had to employ multiple different strategies to compose the data in a way that was useful to us. This will prove very practical in the future when we need to perform data analysis on a different dataset.

While we finished the project with a model that performed well, we certainly did not start with a great one and we created many different models along the way. This helped reinforce the idea that you may not get the best results early on in development, but persevering will almost always be beneficial and you will learn a lot along the way.

In the back half of this course, we learned a lot about using graphs for classification. This topic turned out to be pivotal in our project as our best performing model was a GCN. None of us had much experience in the field before, but this project gave us a great opportunity to learn about the subject and fortify what we learned in class by applying it to a real-life situation.

5 Future Performance Improvements

At the end of developing our final GCN model, we attempted modifying some of the parameters such as the learning rate, number of epochs, and number of layers, but we saw no performance increase. However, one thing that we thought could really make our classifier much better would be the use of Sage software. Sage is a software package that allows for complex mathematical computation and could provide many benefits to our the current methods of our project.

While we saw no improvement of the model after tuning the hyperparameters, there could definitely be further progress made by looking more deeply into methods for optimization such as nested cross-fold validation, grid search, or Bayes search.

As mentioned earlier, the dimensionality reduction techniques that we attempted did not seem to help our model classification accuracy, but working on MCA or logistic PCA more closely could prove very beneficial to help the efficiency and accuracy of our model. Dimensionality reduction is such an important piece of classification and would definitely be worthy of further exploration.