Deep Learning Model for Predicting Diseases: Team 27

Sabrina Chen, xc234

Demir Degirmenci, dwd20

Lizzie Wang, yw703

Parin Vora, prv6

Motivation and Data Understanding

In Modern Medicine, doctors face a major issue when diagnosing and treating disease. Diagnosis of the wrong disease can lead to patients being given the wrong treatments, which could lead to further issues down the line in the form of lawsuits, or worse death. If doctors could use frequently reported symptoms to more accurately predict the diseases, they could reduce the risks of complications arising in the future. We've found a Kaggle database which tracks symptoms often reported in doctors offices, logged as categorical variables. Doctors could use this model to either cross examine their initial diagnosis, or even to track development of symptoms over time allowing them to make more informed decisions. A model like this will allow for more accurate and efficient online medicine practices. For example our group could sell this off to a hospital system like Duke where patients are invited to take an online assessment before they see the doctor where they're asked if they have each of these symptoms (listed in the model). Depending on their responses then, they can be sent to a wing of the hospital which specializes in cases similar to their symptoms. This will allow for quicker and better results, saving Duke time and reducing the risk that patients get misdiagnosed.

We created two comparative models to study symptoms in relation to their prognosis. A Neural Network with dropout and a Logistic Regression model with Multinomial classification as we’re looking to correctly identify the Prognosis in a set of many possible outcomes. We’ll seek to maximize our accuracy in predicting outcomes, if not make it 100% accurate, as misdiagnosis can be fatal for patients and it would be a struggle to get doctors to work a faulty model.

Data Preparation

Since the dataset we downloaded has already been divided into training and testing data, there is no need for us to perform any further splitting. After removing rows with missing values, both datasets now have the same number of columns (133), with the training dataset containing 4,920 rows and the testing dataset having 42 rows. The concern is that the testing dataset is relatively small, which may not be ideal for effectively evaluating the model. In that case, we put 1500 rows from the training dataset to the testing dataset so it can be more accurate. The columns represent various symptoms, such as itching and skin rash, while the rows correspond to individuals and whether they exhibit these symptoms. To ensure that the training dataset is balanced—meaning that each prognosis has a roughly equal number of cases, which prevents the model from struggling to detect less common diseases—we visualized the frequency of each prognosis. The results showed uniformity across the diagnoses, confirming that the dataset is well-balanced and requires no further adjustments.

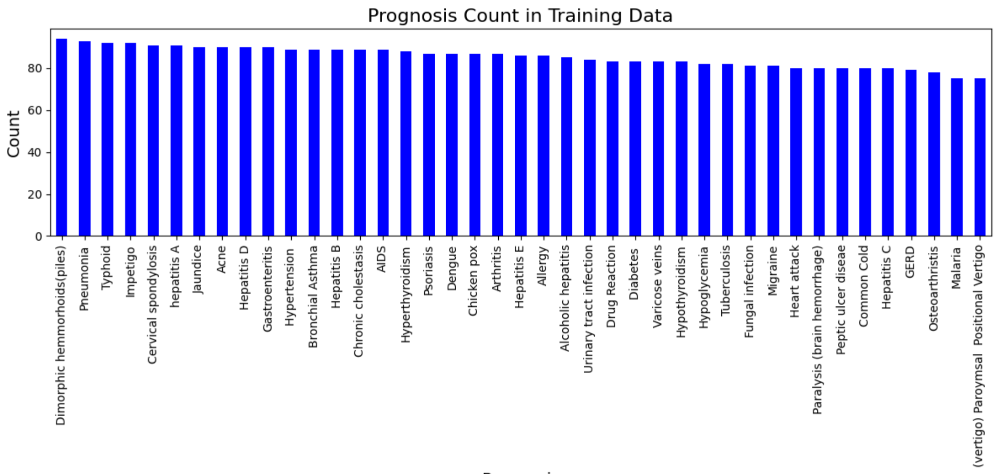


Figure 1: Prognosis Count in Training Data

We also created a correlation matrix for various symptoms, where each cell in the heatmap represents the correlation between two symptoms. The analysis reveals a strong relationship between throat irritation and a runny nose, while most other symptoms exhibit only moderate correlations.

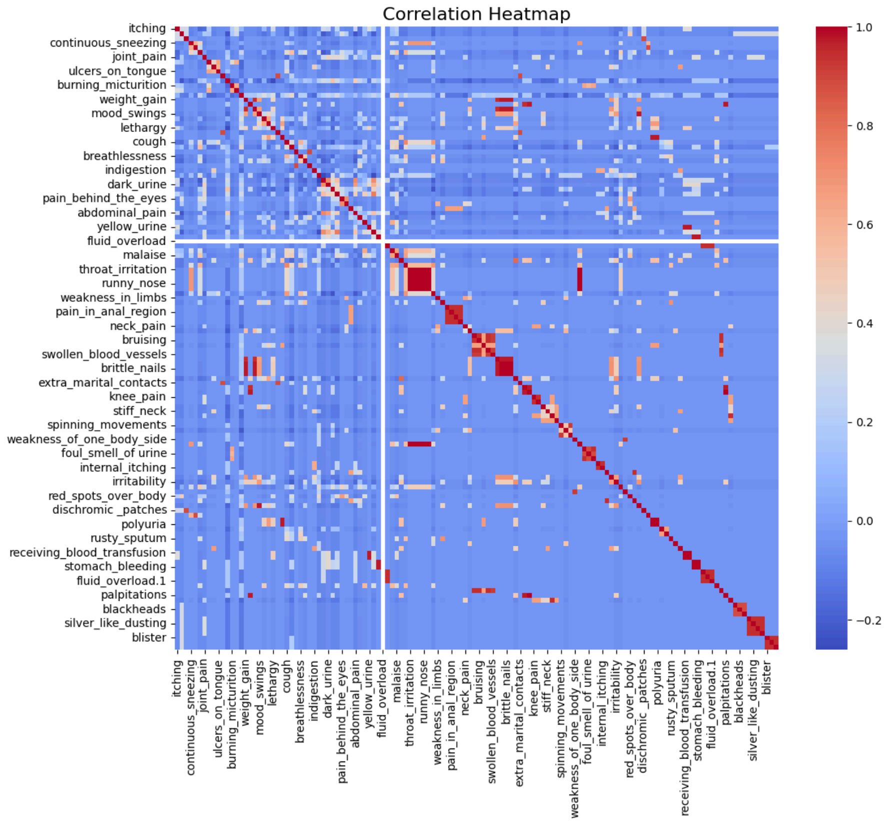
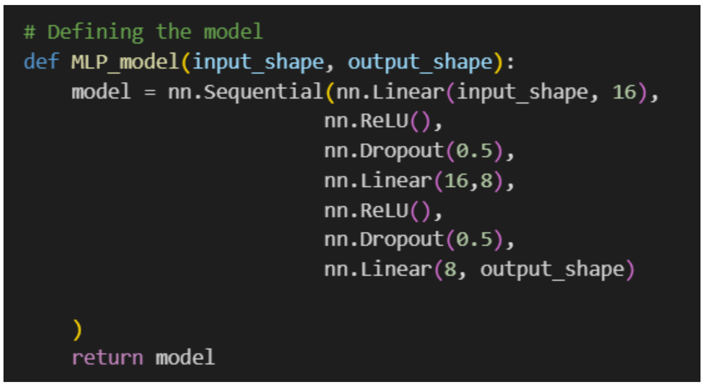


Figure 2: Correlation Heatmap

After removing the target column—prognosis—we standardized both the training and testing datasets to have a mean of 0 and a standard deviation of 1. This ensures that differences in scale do not influence the results. Additionally, we shuffled the training data to ensure that the order of the data does not affect the training outcomes.

Modeling

We started with a very simple multi-layer perceptron model having the following architecture: Input → Hidden Layer 1 (16 neurons) → Dropout (0.5) → Hidden Layer 2 (8 neurons) → Dropout (0.5) → Output. This is the image for the code:



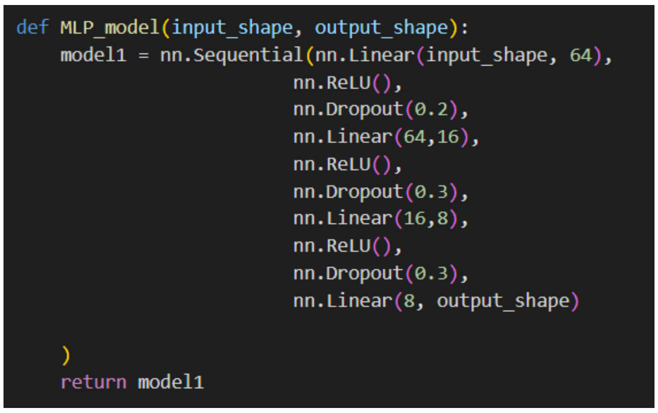
```
# Defining the model
def MLP_model(input_shape, output_shape):
    model = nn.Sequential(nn.Linear(input_shape, 16),
                          nn.ReLU(),
                          nn.Dropout(0.5),
                          nn.Linear(16,8),
                          nn.ReLU(),
                          nn.Dropout(0.5),
                          nn.Linear(8, output_shape)

    )
    return model
```

Figure 3: MLP Model 1

Since we are working on a multi classification problem, we used Cross Entropy Loss as our loss function. We used Adam as the optimizer, and set a relatively higher dropout rate of 0.5 to start with, to prevent overfitting. We set a learning rate of 0.0015, a weight decay of 0.001 and trained the model on 300 epochs. We used higher learning rates and weight decay rates earlier, but the model was overfitting by quite a margin. If we use lower learning rates, we would have to train the model for a larger number of epochs, which would make it computationally expensive. We achieved a test accuracy of 100% and MSE loss of 0.5372 after running this model.

We built another multi-layer perceptron model: Model-2, having the following architecture: Input → Hidden Layer 1 (64 neurons) → Dropout (0.2) → Hidden Layer 2 (16 neurons) → Dropout (0.3) → Hidden Layer 3 (8 neurons) → Dropout (0.3) → Output



```
def MLP_model(input_shape, output_shape):
    model1 = nn.Sequential(nn.Linear(input_shape, 64),
                           nn.ReLU(),
                           nn.Dropout(0.2),
                           nn.Linear(64,16),
                           nn.ReLU(),
                           nn.Dropout(0.3),
                           nn.Linear(16,8),
                           nn.ReLU(),
                           nn.Dropout(0.3),
                           nn.Linear(8, output_shape)

    )
    return model1
```

Figure 4: MLP Model 2

We used the same learning rate (0.0015) and weight decay rate (0.001) as the previous model.

Implementation

We started by training the model on the original data from Kaggle, that is, training data being 4920 rows and testing data being 42 rows. The performance that we achieved on this data using the 1st model, was extremely good, with a 100% test accuracy and a MSE loss of 0.0492. However, we felt that the performance was too good, because of the size of the test dataset. There were a total of 41 diseases in the original training dataset, and 1 row for each disease in the testing dataset. This makes the model evaluation very misleading. To tackle that, as mentioned in the data preparation, we split the data such that the training data had 3500 rows and the testing data had 1500 rows. We used this same model on the new splitted data, and achieved a test accuracy of 100% and MSE loss of 0.5372. However, we felt that we could improve on this model even further. Hence, we made another model: Model-2, which had more hidden layers than the previous one. We set the same hyperparameters as the previous model with the learning rate being 0.0015 and weight decay being 0.001. We had initially tested with SGD as the optimizer function, but the model accuracy was not increasing at all after a certain number of iterations. Hence, we went forward with Adam.

Results and Evaluation

After executing the model, we plotted the loss for both the training and testing datasets over 200 epochs. As observed, the loss significantly decreased from the 1st to the 25th epoch. The final test result, showing a loss of 0.2746, indicates that our model is highly accurate. Additionally, even though it is a bit overfitting, the accuracy rate is pretty good.

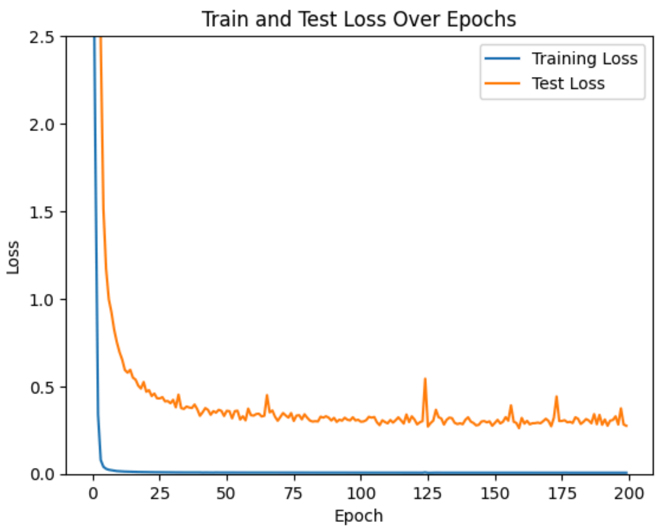


Figure 5: Train and test losses for MLP Model

However, 'loss' may not be the most informative metric for interpretation. Therefore, we introduced a confusion matrix, with the x-axis representing predicted labels and the y-axis representing true labels. Examining this confusion matrix, we can clearly see that all predicted labels match the true labels, with all other entries being 0. This indicates that our model has achieved perfect prediction accuracy.

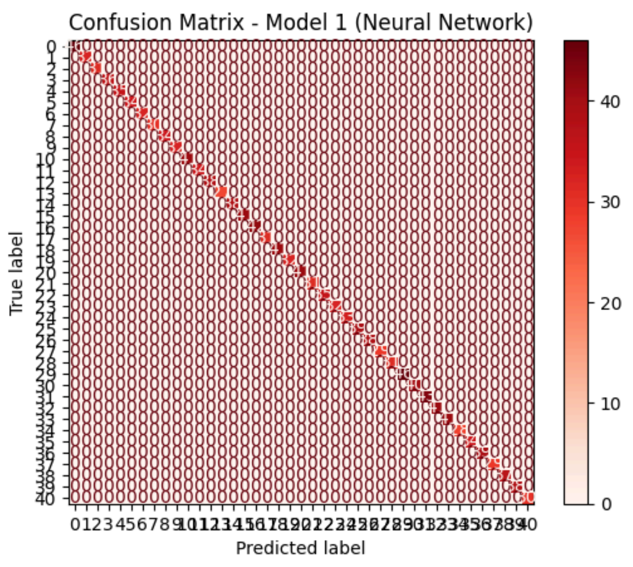


Figure 6: Confusion Matrix for MLP Model

We further calculated the accuracy directly as the ratio of correct predictions to total predictions, and it reached 100% for both the training and testing phases.

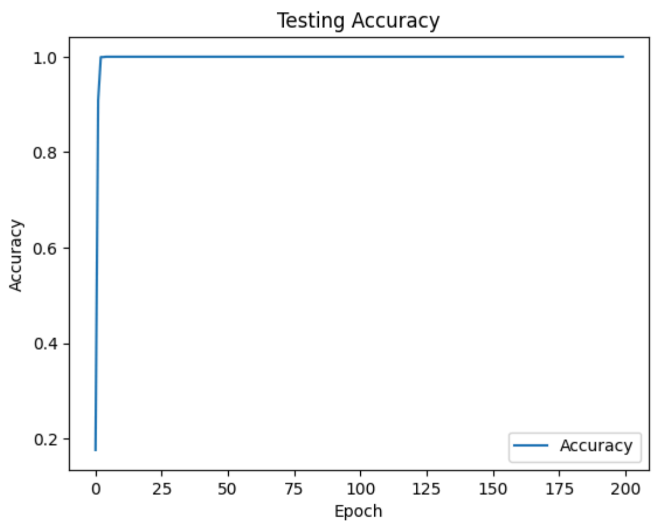


Figure 7: Test Accuracy for the MLP Model

Based on the confusion matrix and accuracy metrics, we can easily visualize and assess the performance of our model.

We used a model from a Kaggle competition as our benchmark, which employs logistic regression with a multinomial multi_class setting. This approach is straightforward for predicting categorical data. The results are as follows: the accuracy reached 100%, and the confusion matrix was identical, demonstrating that our model matches the performance of the benchmark. Although logistic regression may be simpler and easier to use, our multilayer model is suitable for more complex datasets, making it more applicable and flexible despite its complexity in construction. We can utilize this type of model for initial screening in clinical settings to classify diseases and provide preliminary insights for doctors' reference.

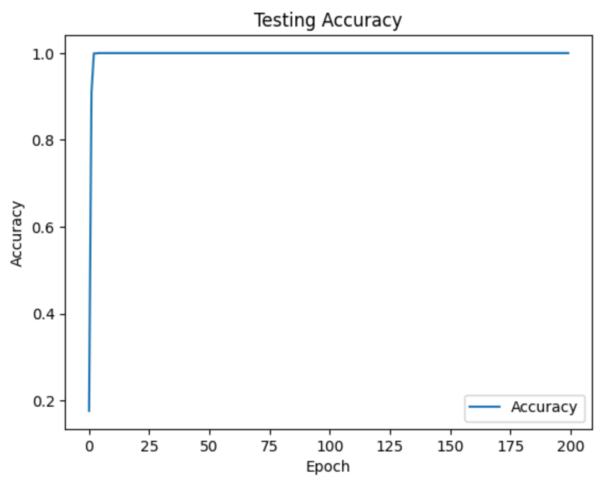


Figure 8: Test Accuracy for Logistic Regression Model

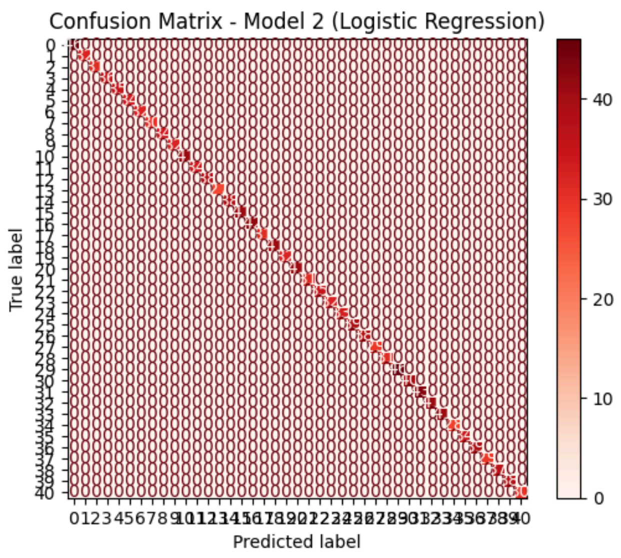


Figure 9: Confusion Matrix for Logistic Regression Model

<u>Deployment</u>

Developing our deep learning model, we've been able to create very accurate predictions for the prognosis of our patients. In real world deployment we can use this as a preliminary screening for patients. Patients will take an examination online asking about their symptoms, and from there our model will run providing our doctors with ideas of what the patients might be facing. As people filling out our examination are in to see a doctor, we shouldn't face too many ethical concerns as their answers will be recorded for medical reasons only.

Whilst this model will be really good at creating a meaningful prognosis for diseases which have already been seen, it will struggle in detecting new and more niche diseases. A way to work around this is to continually train our model which will be strenuous, but it will allow our model to learn new diseases. In order to mitigate this stress, we could bring on a couple dedicated data entry specialists to manage this task.

<u>Team Contribution</u>

Parin Vora: Data Preparation, Data Modeling, Model Review

Sabrina Chen: Data Cleaning, Data Modeling, Hyperparameter Tuning

Demir Degirmenci: Business Insights, Data Analysis, Slide preparation

Lizzie Wang: Benchmark Comparison, Documentation, Slide preparation