SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.039 THEORY AND PRACTICE OF DEEP LEARNING

Predicting Mental Health Treatment Needs

Enhancing Early Detection and Support for Patients

SWASTIK MAJUMDAR 1005802

ADITYA VISHWANATH 1004281

HARIKRISHNAN CHALAPATHY ANIRUDH 1005501

Predicting Mental Health Treatment Needs

Supervisor: Professor Matthieu De Mari

Introduction

Mental health and well-being are significant aspects of healthcare and have a direct correlation to one's quality of life and overall well-being. Mental health is as important, if not more important, than every other facet of one's healthcare in the physical form.

The mental well-being of employees is a critical factor that affects their productivity, engagement, and overall job satisfaction. However, mental health issues are often stigmatized and not addressed openly in the workplace. As a result, many individuals who are struggling with mental health issues do not receive the help they need, and employers do not have a robust way to check in and assess the mental well-being of their employees to provide the necessary support.

To address this issue, this project proposes developing a deep learning model that can predict whether an employee might need mental health support. This model can help companies monitor the mental health levels of their employees and better provide them with the necessary support to improve their well-being, be it counseling, therapy, or more advanced forms of mental health assistance.

It should also be acknowledged that mental health is something we still have much to learn about and something we need to discuss more as a society. This means that it is understood that mental health is a spectrum and not always something one can classify with a fixed range or mode of measure. Hence, it is a delicate issue for this project to tread on while attempting to build a model to classify one's mental health state. However, given this hurdle, this project can be seen as a first step towards making that change and building a more complex model that would allow us to strive for perfection in supporting individuals struggling with mental health. It would also open up the door to start discussions about using technology to assist with mental healthcare, which is what we need as a society to normalize this issue and discussions surrounding it.

Contents

Chapter 1

Dataset

1.1 Mental health survey dataset

The dataset can be downloaded using the following link.

The objective of our project is a supervised task, using a labeled dataset. The dataset comprises of 1259 samples and 27 features. It is a binary classification task to determine if an employee requires mental health support.

The mental health dataset used for this project was collected through a survey conducted in various countries around the world. The dataset contains various features related to mental health, including demographic information, mental health conditions, and various factors that may contribute to mental health issues.

Despite containing a significant amount of data, this dataset required extensive cleaning and preprocessing before it could be used for machine learning tasks. As the data was collected through an online survey, it may have limitations in terms of representativeness and bias. Nevertheless, the dataset's large sample size and variety of features make it a valuable resource for studying mental health.

1.2 Data Preprocessing

Please refer to **Data preprocessing** section of the Jupyter Notebook for the implementation. Below are the steps taken for cleaning and pre-processing the dataset to ensure its quality and reliability for the classification task at hand:

1.2.1 Renaming the columns of the DataFrame

During the data preprocessing stage, it was necessary to rename the columns in the DataFrame to more intuitive column names, as the original column names were survey questions. This was achieved by utilizing the rename() function in Python.

1.2.2 Number of employees

Upon investigation, the `no_of_employees` column in the DataFrame had the following values '1 to 5', '6 to 25', '26-99', '100-500', '26-100', '500-1000', and 'More than 1000'. We used the replace() function to ensure consistency and uniformity in the column and the original values were replaced with '1-5', '6-25', '26-100', '100-500', '500-1000', and '>1000', respectively.

1.2.3 Mental health coverage

Likewise, we also examined the `mental_healthcare_coverage` column and noted that it contained the values 'Yes', 'No', 'I don't know', and 'Not eligible for coverage / N/A'. To maintain consistency, we used the replace() function to replace the value 'Not eligible for coverage / N/A' with 'No'. As a result, the column now exclusively comprises the values 'Yes', 'No', and 'I don't know'.

1.2.4 Openness with family and friends

The unique values in the `openess_of_family_friends` column were: 'Somewhat open', 'Very open', 'Somewhat not open', 'Neutral', 'Not applicable to me (I do not have a mental illness)', and 'Not open at all'. Using the replace() function 'Not applicable to me (I do not have a mental illness)' is mapped with 'I don't know'.

1.2.5 Age

For the age column, we first calculate the median age for rows where the age is greater than or equal to 18 or less than or equal to 75. Rows with ages that are less than 18 or greater than 75 are replaced with the calculated median age.

Additionally, Min-max scaling was applied to scale the age to a specific range of [0,1]. If the `age` column was not scaled it would have a larger range compared to other features in the dataset, introducing numerical instability. Moreover its values may dominate the model's learning process and have a disproportionate influence on the model's predictions. By using Min-max scaling, `age` is brought to a similar scale as other features, allowing the model to give them equal importance during training.

1.2.6 Gender

Upon examination, the `gender` column had various values, including different spellings, capitalization, and variations. The replace() function was used to replace the different variations and standardize gender into to three categories: "male", "female", and "other".

1.2.7 Tech Role

Essentially, the `role_in_company` column is encoded into a binary `tech_role` column that indicates whether an individual's role is related to technology or not.

Four subsets of the `role_in_company` column in the DataFrame, are created based on specific string patterns ("Back-end", "Front-end", "Dev", and "DevOps") using the "str.contains()" method. These subsets are converted to lists and appended to a list called `tech_list`. The nested lists in `tech_list` are flattened into a single list called `flat_list` using a list comprehension. Duplicates are removed from `flat_list` while preserving the order of the elements using the "dict.fromkeys()" method, and the result is assigned back to `flat_list`.

A new column called `tech_role` is added to the DataFrame containing the same values as `role_in_company`. The values in the `tech_role` column that are found in `flat_list` (the tech-related roles) are replaced with the value 1 using the replace() method. A list of remaining unique values in the `tech_role` column, excluding the value 1, is stored in `remain_list`. The values in the `tech_role` column that are found in `remain_list` (the non-tech-related roles) are replaced with the value 0 using the replace() method.

Finally, the `role_in_company` column is dropped from the DataFrame using the "drop()" method to remove it from the DataFrame.

1.2.8 SimpleImputer

The SimpleImputer class from the scikit-learn library was used to fill missing values in the DataFrame with the most frequent value of each column.

1.3 Data Visualization

To recreate these plots, please refer to **Data Visualization** section of the Jupyter Notebook. After performing pre-processing, data visualization was used to explore and analyze the dataset, helping us to identify patterns, trends, and outliers. Moreover, we were able to gain insights from the data, identifying correlations, and discovering hidden relationships that may not be apparent.

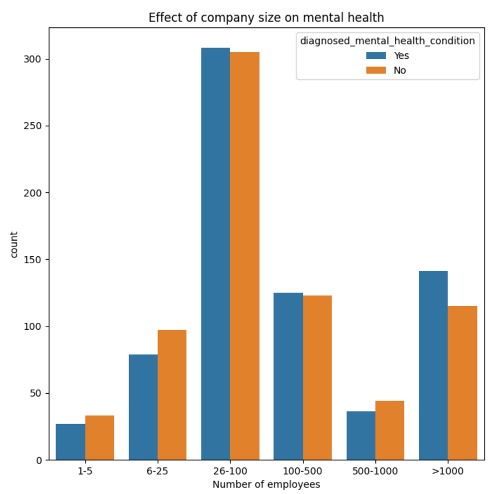

Figure 1.1: Company Size

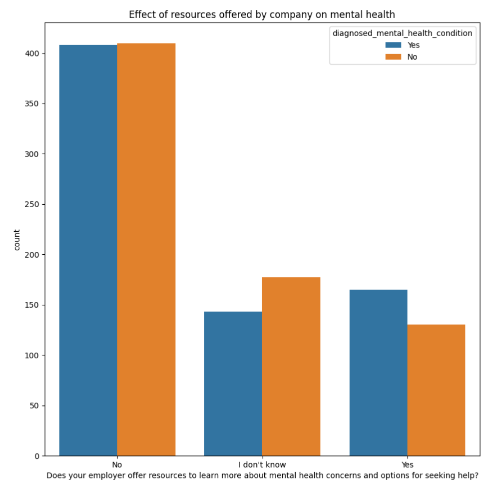

Figure 1.2: Employer Resources

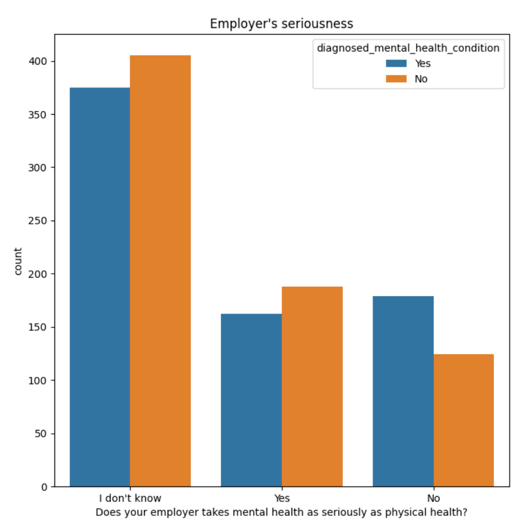

Figure 1.3: Employer Seriousness

The three charts presented above provide insights into the relationship between workplace environment and employees' mental health.

Firstly, Figure 1.1 indicates that mid-sized companies (26-100 employees) have many employees diagnosed with mental health conditions. This could be because mid-sized companies often expand rapidly and do not have the same resources as larger companies. This can increase job demands and stress on employees, leading to potential mental health challenges.

Secondly, Figure 1.2 reveals that employees who report that their employers do not offer resources to support employee mental health are more likely to be diagnosed with mental health conditions.

Lastly, Figure 1.3 shows that employees who believe that their employer does not prioritize mental health as much as physical health have a higher likelihood of being diagnosed with mental health conditions. This result was expected, as neglecting mental health can lead to higher levels of stress and anxiety in employees, ultimately resulting in mental health issues.

Overall, the charts provide valuable insights into the complex relationship between workplace environment and employees' mental health, highlighting the need for employers to prioritize mental health resources and support in the workplace.

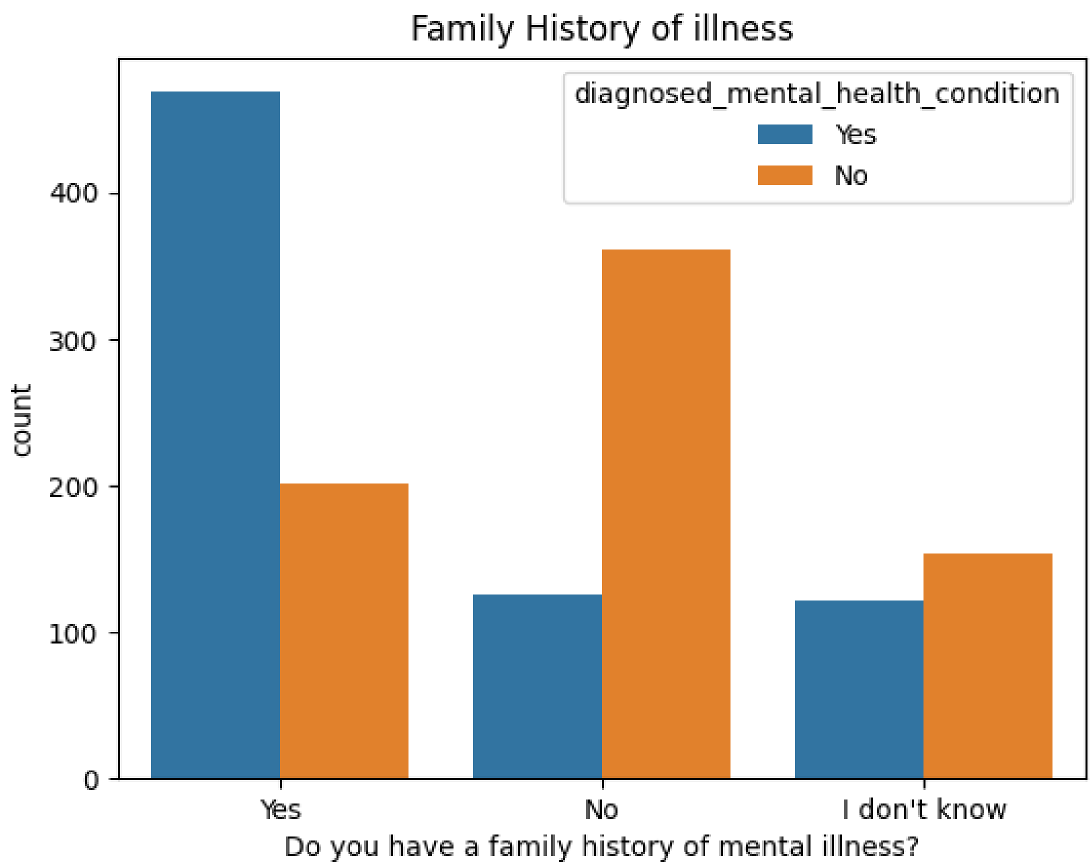

Figure 1.4: Family History

Figure 1.4 provides insights into the impact of an employee's family history of mental health illness on their own mental health. The graph reveals a significant positive correlation between an employee's family history of mental illness and their likelihood of being diagnosed with mental health conditions. Specifically, the data indicates that employees with a family history of mental illness are more likely to be diagnosed with mental health conditions compared to those without such a history.

This correlation is not surprising given that research has established a strong genetic component to mental health conditions, implying that these disorders are hereditary. As a result, individuals with a family history of mental illness may be more genetically predisposed to developing such conditions, thereby increasing their likelihood of being diagnosed with mental health issues.

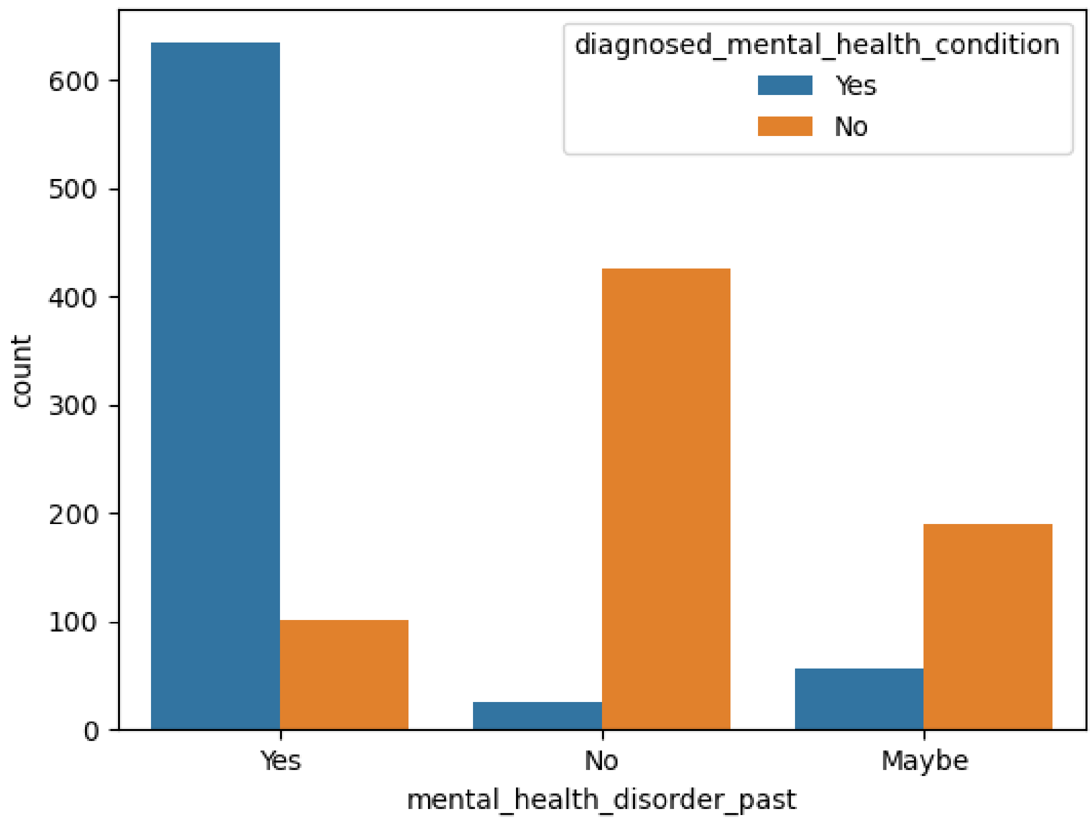

Figure 1.5: Previous Mental Health Illness

Figure 1.5 illustrates the relationship between an employee's history of mental health disorders and their likelihood of being diagnosed with a mental health condition currently. This chart provides a valuable means of understanding the recurrence of mental health conditions amongst employees. The data reveals that employees who have experienced a mental health disorder in the past have a higher probability of being diagnosed with a similar condition again compared to those who have not.

This positive correlation between past mental health disorders and current diagnoses is not surprising, given that research has established the recurrence of mental health conditions due to the tendency of patients to relapse. This highlights the importance of early intervention and preventative measures to address mental health issues before they become more severe and potentially lead to relapse.

Hence, this chart provides important insights into the recurrence of mental health conditions amongst employees, highlighting the need for employers to provide adequate support and resources to employees with a history of mental health disorders to reduce the risk of relapse and improve overall mental health outcomes.

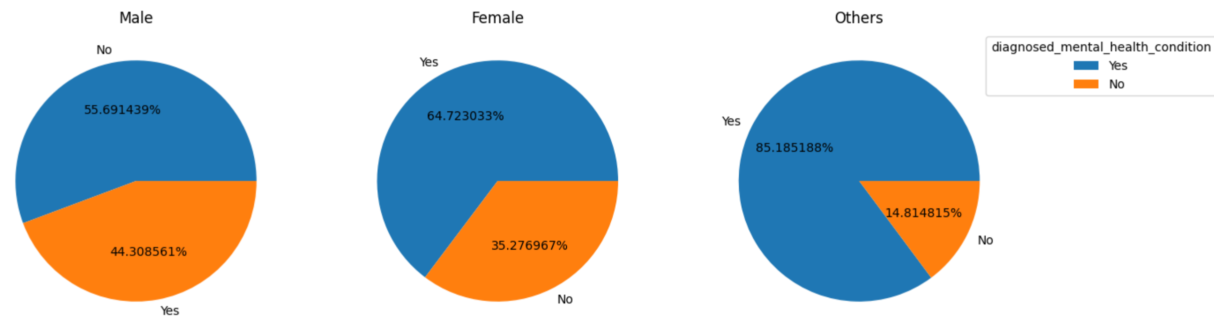

Figure 1.6: Gender

Figure 1.6 provides insights into the relationship between an employee's gender and their likelihood of being diagnosed with a mental health condition. The data reveals a gender disparity in mental health outcomes, with female employees being more likely to be diagnosed with a mental health condition compared to their male counterparts. Interestingly, employees who identify as non-binary or within the gender spectrum have an even higher likelihood of being diagnosed with a mental health condition compared to female employees.

These findings have important implications for workplace mental health support and intervention strategies. The results suggest that certain gender groups may be at a higher risk of developing mental health conditions, and as such, may require more targeted and tailored support. Furthermore, the gender disparities revealed in this study highlight the need for organizations to ensure that mental health support and resources are accessible and inclusive for all employees, regardless of their gender identity.

Overall, this chart provides valuable insights into the relationship between gender and mental health outcomes, emphasizing the need for gender-sensitive mental health policies and interventions in the workplace.

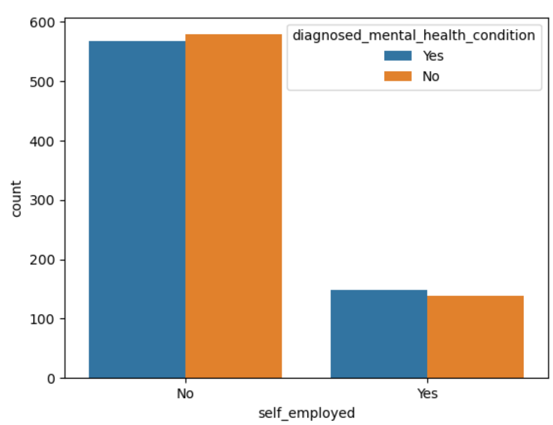

Figure 1.7: Self Employed

Figure 1.7 endeavors to establish a correlation between an individual's self-employment status and the likelihood of being diagnosed with a mental health disorder. This correlation is essential to comprehend, as it may aid in identifying the particular category of employees who require improved mental health support. The graph illustrates a nearly equal distribution among employees who are not self-employed with respect to being diagnosed with a mental health condition. Similarly, employees who are self-employed also exhibit a relatively even distribution between those who have and have not been diagnosed with a mental health disorder. This suggests that a direct correlation cannot be established between self-employment and mental health, and in fact, both groups, i.e., self-employed and non-self-employed individuals, have an equal risk of experiencing mental health disorders.

1.4 One-hot encoding

After performing data pre-processing and visualization, our dataframe consisted of 20 input features and a single output feature representing the class of the employee's diagnosed mental health condition.

Input features:

1. `age`
2. `self_employed`
3. `no_of_employees`
4. `tech_company`
5. `mental_healthcare_coverage`
6. `knowledge_about_mental_healthcare_options_workplace`
7. `employer_discussed_mental_health`
8. `employer_offer_resources_to_learn_about_mental_health`
9. `medical_leave_from_work`
10. `comfortable_discussing_with_coworkers`
11. `employer_take_mental_health_seriously`
12. `openess_of_family_friends`
13. `family_history_mental_illness`
14. `mental_health_disorder_past`
15. `currently_mental_health_disorder`
16. `gender`
17. `country`
18. `country work`
19. `work_remotely`
20. `tech_role`

To convert the categorical data into numerical representations, label encoding was initially applied. However, the model performed poorly. In label encoding, each category is assigned a unique integer label based on alphabetical ordering. For instance, in the case of the "gender" feature with categories "Male", "Female", and "Other", label encoding would assign them the integer labels 0, 1, and 2, respectively. These integer labels are arbitrary and do not possess any inherent ordinal relationship.

When a machine learning model is trained on these integer labels, it may incorrectly interpret them as continuous values with ordinal relationships, leading to inaccurate predictions and diminished performance.

To counter this issue, one-hot encoding was used, which is another popular technique for turning categorical data into numerical representations. Each categorical feature is transformed into a binary variable, where the presence of a category was denoted by 1 and the absence by 0. This approach created a binary feature for each category, enabling the model to independently capture the distinctiveness of each category. As a result, one-hot encoding prevented the model from assuming any ordinal relationship between the categories, thereby enhancing the accuracy of the predictions.

We made a careful decision to not apply one-hot encoding for `age`. This is mainly because:

- **High dimensionality**: One-hot encoding `age` would create a binary variable for each unique age, which could result in a large number of binary features.
- **Loss of Information:**: One-hot encoding of `age` would result in a binary variable for each age value, treating each age as a separate category. This can result in the loss of the inherent ordinal relationship between different age values. For example, the model would not capture the fact that "age 25" is closer to "age 26" than "age 45". This loss of information can impact the model's ability to accurately capture the underlying patterns and relationships in the data.

Instead, we used Min-Max scaling to represent the `age` feature in a more meaningful and interpretable manner in the model. Unsurprisingly, the performance of our model significantly improved with the utilization of one-hot encoding.

Chapter 2

Model Architecture

2.1 Model Description

For this binary classification task, a deep learning model with multiple hidden layers was implemented. Our model has the following components:

- **Input Layer**: Takes in input data with n_x features.
- **Hidden Layers**: The model has n_h hidden layers. To ensure modularity in the neural network, two blocks namely ReLU and Sigmoid were defined.
  - **ReLU**: The ReLU block represents a rectified linear unit activation function applied to the output of a linear layer, with input size n_x and output size n_y.

    $$Relu(z) = max(0, z)$$

  - **Sigmoid**: The Sigmoid block represents a sigmoid activation function applied to the output of a linear layer, with input size n_x and output size n_y.

    $$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  The Hidden Layers are created using the ReLU and Sigmoid blocks.
- **Dropout Regularization**: Dropout is applied to the hidden layers, except for the last hidden layer. This prevents overfitting and improves the model's generalization.
- **Leaky ReLU Activation**: Leaky ReLU activation with a negative slope is applied to the hidden layers, except for the last hidden layer. The negative slope allows the model to learn from both positive and negative gradients, which can help in improving the convergence rate and overall performance of the model.

  $$LeakyReLU(z) = max(0.1z, z)$$

- **Output Layer**: The output layer of size n_y uses a Sigmoid activation function to produce binary classification probabilities.
- **Loss Function**: The model uses Binary Cross Entropy loss as the loss function for training. It is widely used as a loss function for binary classification tasks, as it calculates the difference between the predicted and actual class probabilities.

  $$\text{BCE Loss} = \sum_i [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where

- y_i is the is the true label for the i-th sample. (Either 0 or 1).
- p_i is the predicted probability for the i-th sample being in class 1 (the positive class).

- **Evaluation Metric**: The model uses Binary Accuracy as the evaluation metric for both training and validation.

- **Optimization**: The model uses Stochastic Gradient Descent (SGD) as the optimization algorithm for updating the model parameters during training.

$$\text{SGD Update Rule: } \theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

where

- θ_t represents the model parameters at time step t.
- η denotes the learning rate.
- $\nabla J(\theta_t)$ refers to the gradient of the loss function $J(\theta_t)$ with respect to the model parameters θ_t.

- **Forward method**: The forward method implements the forward pass of the neural network, where input x is passed through the stacked layers in the layers list.

- **Trainer method**: The trainer method is responsible for training the model. It takes input and output data, as well as optional hyper-parameters such as maximum number of iterations learning rate and batch size. It creates a data loader for batch-wise training, defines an optimizer (SGD with specified learning rate), and performs forward and backward passes for each batch in the training loop. It also keeps track of training loss, accuracy, validation loss, and validation accuracy during training.

- **Tester method**: The tester method is responsible for evaluating the trained model on test data. It takes the test input and output data, performs a forward pass to get predictions, calculates loss and accuracy, and prints the results.

Overall, the model architecture consists of stacked layers of ReLU with dropout and leaky ReLU activation functions, followed by a Sigmoid activation function in the last layer for binary classification.

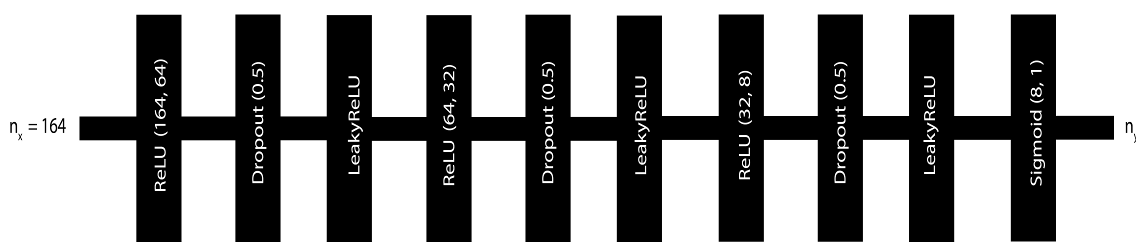

Figure 2.1: Model Architecture

2.2 Model Parameters

Hyper-parameter	Value
Epochs	150
Dropout Probability	0.5
Batch size	32
Loss function	Binary Cross-Entropy Loss
Optimizer	SGD
Learning Rate	0.01
LeakyReLU negative slope	0.1

Table 2.1: Hyper-parameters

- **Epochs**: The number of epochs was determined through experimentation, where the model was trained with different number of epochs and evaluated for performance. 150 epochs was found to give the best performance on the test set.
- **Dropout Probability**: The dropout probability was also determined through experimentation, where the model was trained with different dropout probabilities. A dropout probability of 0.5 was found to be effective in preventing overfitting while still allowing the model to learn from the data. By randomly dropping out some of the neurons in the network during training, the network does not heavily rely on any one input feature.
- **Batch size**: Upon experimenting with batch sizes of 32, 64 and 128, a batch size of 32 was found to be effective in achieving good performance while still allowing the model to converge within a reasonable amount of time.
- **Optimizer**:After experimenting with various optimizers such as Adam and RMSProp, Stochastic Gradient Descent (SGD) was selected as the optimal optimizer.
- **Learning rate**: After testing the model with various learning rates, it was determined that a learning rate of 0.01 was optimal, as it allowed the model to achieve good performance while still converging in a reasonable amount of time.
- **Leaky ReLU negative slope**: Through experimentation with various values, a negative slope of 0.1 was selected as the optimal value for the negative slope of the leaky ReLU activation function. This value was found to be effective in preventing vanishing gradients and achieving good performance.
- **Non-Linear Activation Function**: The Rectified Linear Unit (ReLU) function is widely considered to be the optimal non-linear activation function for Deep Learning models, owing to its superior performance in numerous applications. Consequently, the initial approach in this project was to adopt ReLU as the default activation function in the model's architecture.

  However, the No Free Lunch Theorem posits that there is no universally optimal approach to solve or optimize all tasks, and that a particular approach may work well for one problem but not for another. Therefore, we conducted experiments with various activation functions to identify the best one for the specific task at hand. They were namely:

  - **Gaussian Error Linear Unit (GELU)**: Investigated the effectiveness of the GELU function as a non-linear activation function compared to the widely used ReLU function. GELU function documented superior performance in some previous studies.

    The findings indicate that the GELU function resulted in a significantly lower test accuracy compared to the ReLU function. Specifically, the GELU function produced a test accuracy of approximately 48.3, which is significantly worse than the performance of the ReLU function.

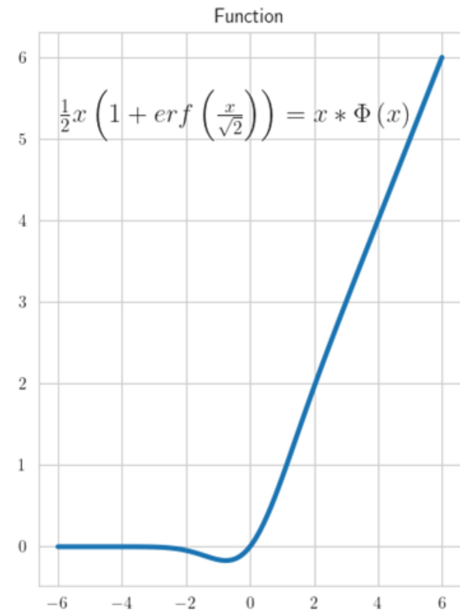

Figure 2.2: GELU Graph and Function.

– **Scaled Exponential Linear Unit (SELU)** : Another Linear Unit Function experimented with was SELU. This was motivated by the curiosity to explore the potential of various LU activation functions in the context of deep learning models. The experimental results revealed that the SELU function performed better than the GELU function, with a test accuracy of approximately 83.2. However, the performance of the SELU function was not as good as that of the ReLU function.

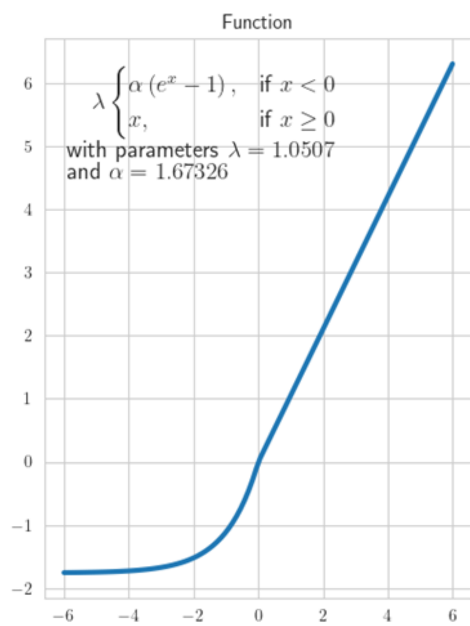

Figure 2.3: SELU Graph and Function.

– **Exponential Linear Unit (ELU)**: ELU was another activation function, like GELU, that was cited to be better than ReLU and it was the closest to ReLU amongst all the activation functions that were tried out.The experimental results indicated that ELU performed similarly to ReLU, with a test accuracy of approximately 92.9. However, the performance of ELU was slightly lower than that of ReLU in this case, albeit to a very small extent.

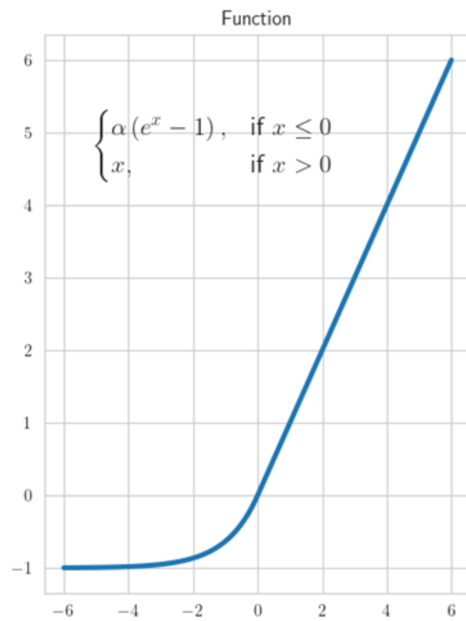

Figure 2.4: ELU Graph and Function.

– **Mish**: Experimenting with the Mish function was motivated by the fact that both functions (Mish and ReLU) operate element wise and the possibility that Mish may perform better than ReLU. However, the experimental results revealed that Mish performed poorly in this model with the test accuracy being considerably lower than desired at 48.5.

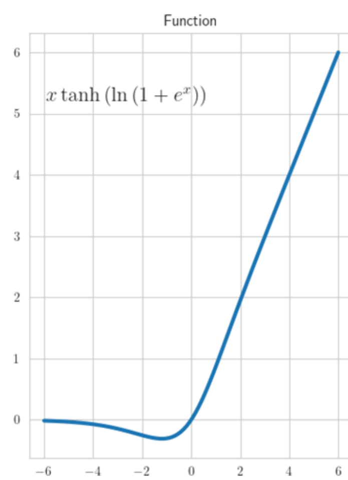

Figure 2.5: Mish Graph and Function.

Chapter 3

Model Performance

3.1 Visualization of model performance

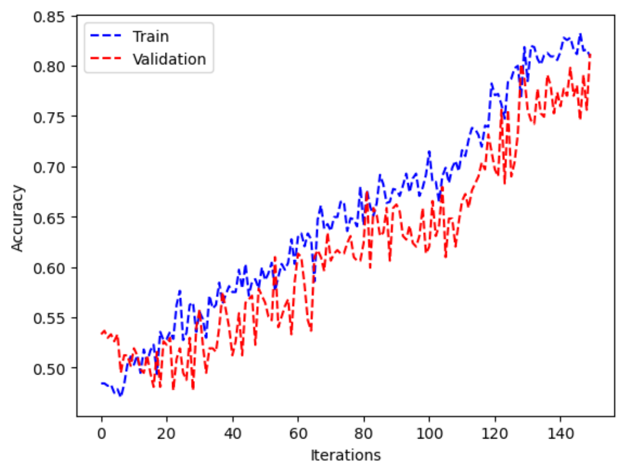

Figure 3.1: Accuracy Curve

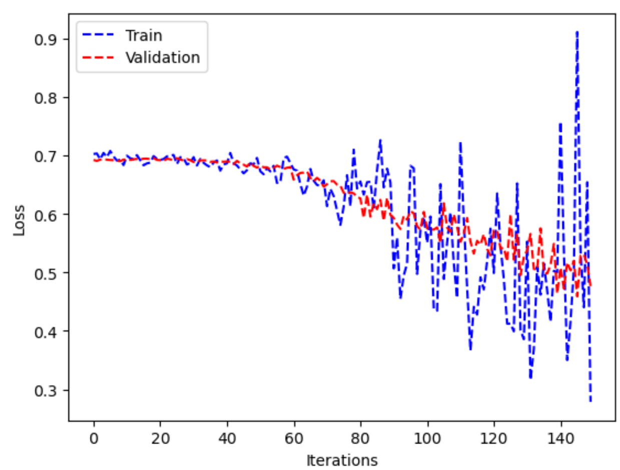

Figure 3.2: Loss Curve

To recreate these images, please refer to **Accuracy and Loss curves** section of the Jupyter Notebook.

The accuracy curve (Figure 3.1) and loss curve (Figure 3.2) presented in this study provide valuable insights into the model's performance. Using these curves, evaluation of the model's learning ability and generalization performance, as well as the detection of overfitting, can be accomplished. By monitoring these curves during the training process, informed decisions were made to optimize the model's performance through adjustments in hyper-parameters or model architecture.

As depicted by the accuracy curve, there is a steady increase in the training accuracy, suggesting that the model is effectively learning from the data. The validation accuracy increases along with the training accuracy, indicating that the model is able to generalize well to new data. The model is not overfitting since the validation accuracy does not decrease while the training accuracy continues to improve.

The training loss, depicted by the loss curve, seems to be decreasing over time, indicating that the model minimizes the error in the training data. However, we observe high oscillations in the training loss from epoch = 70 on wards. Even though our training loss is oscillating, our validation loss is stable, indicating that the model is still generalizing well to unseen data.

We believe that the high oscillations in the training loss were caused by stochasticity in data. The training data might have inherent stochasticity or variability, resulting in high oscillations in the training loss. If the training data contains randomly changing patterns, it can cause the model's loss to fluctuate during training, even if the model is actually learning and improving.

To address this issue regularization techniques such as dropout were implemented. Additionally, tuning of learning rates, constant monitoring of validation loss, and thorough evaluation of model architecture was performed in an attempt to stabilize the training process and enhance the performance of the model.

3.2 Comparison with state-of-the-art models

Model	Accuracy (%)
DeepNeuralNet	93.03
RandomForest	91.64
Gradient Boosting	90.59
Ada Boost	90.94
Decision Tree	86.06

Table 3.1: Comparison with State-of-the-Art Models

Please refer to **Performance against some state-of-the-art models** section of the Jupyter Notebook for the implementation of these models.

The higher accuracy of DeepNeuralNet suggests that it is a promising model for the task at hand.

Chapter 4

Model Recreation Steps (From Scratch)

4.1 Setting up the Python Environment

1. Install Python 3 if you haven't already.
2. Open a terminal or command prompt in the project directory.
3. Create a virtual environment using `venv`:

   ```
   python3 -m venv myenv
   ```

4. Activate the virtual environment:
   - On Windows:

     ```
     myenv\Scripts\activate.bat
     ```

   - On Unix or Linux:

     ```
     source myenv/bin/activate
     ```

5. Install the required packages using `pip`:

   ```
   pip install -r requirements.txt
   ```

   This may take a while.

4.2 Training the Model

1. Make sure the virtual environment is activated.
2. Train the model:

   ```
   python train.py
   ```

   This may take a while.

3. Your model should now be training with the required packages installed.

4.3 Deactivating the Virtual Environment

To deactivate the virtual environment, simply run the following command in the terminal or command prompt:

```
deactivate
```

4.4 Loading Trained Model Weights

Please find saved model parameters in the 'weights' folder of the project directory. To load a pre-trained model, refer to the **Loading the pre-trained model** section of the Jupyter Notebook. Make sure to change the path in the code snippet to load the desired parameter values. The tester() method puts the model on evaluation mode, which reproduces the accuracy we obtained.

Chapter 5

Web Application - WellCheck

5.1 Setup

A video demonstration can be viewed using the following link.

Follow the instructions below to run the app.

Prerequisites

- Node.js and npm (Node Package Manager) is installed on your machine.

Configuration

- Download the `.env` file from this link.
- Create a `.env` file in the `app` folder. Copy the contents of the downloaded `env` file to `.env`.
- `.env` should now contain a username and password. The final path for `.env` should be `app/.env`.

Installation

- Navigate to the `app` folder in the project directory using the command line or terminal: `cd app`.
- Install the required dependencies by running the following command: `npm install`.

Running the App

1. Once the dependencies are installed, you can start the app by running the following command: `npm start` or `node server.js`.
2. The app will now be running and hosted on `http://localhost:3000` in your web browser.

Terminate

- To stop the app, simply terminate the Node.js process by pressing `Ctrl + C` in the command line or terminal.

5.2 Features

This web application serves as a valuable tool in the field of medicine and healthcare by providing users with a prediction of their mental health support needs. By collecting essential information such as age, gender, and other relevant details, the app utilizes a pre-trained deep learning model to make accurate predictions. The results can then be emailed to the users, which can serve as proof of their mental health support needs and can be verified by employers or other relevant parties.

Early identification and intervention are crucial in managing mental health conditions effectively and preventing them from worsening. This can result in improved health outcomes for individuals and reduce the burden on healthcare systems. Moreover, the option to email the results to users allows them to share verifiable evidence of their mental health support needs with employers or other relevant parties. This can facilitate better communication helping them to advocate for their mental health needs in workplace settings or other relevant contexts. This can lead to increased awareness and understanding of mental health in various settings and promote a supportive environment.

5.3 Usage

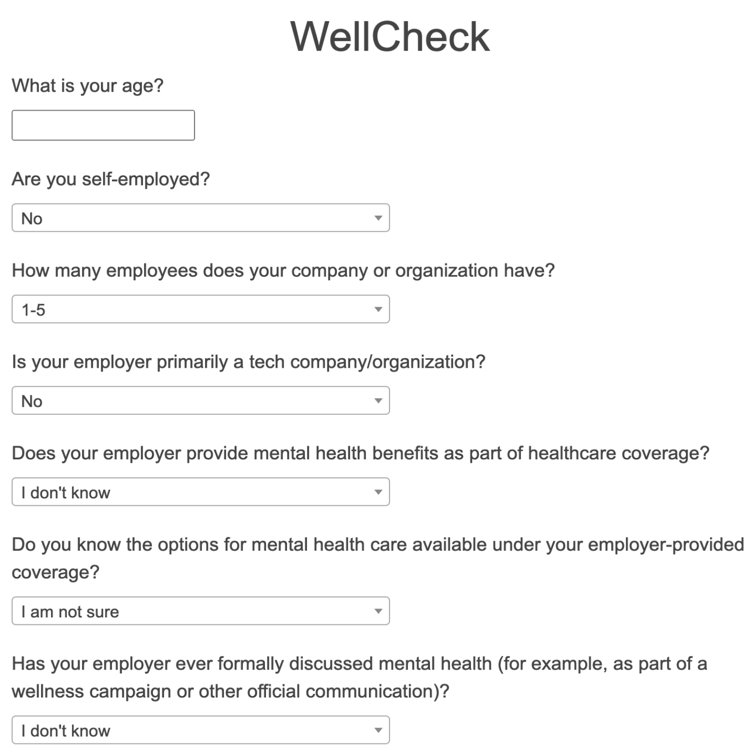

Figure 5.1: Home Page

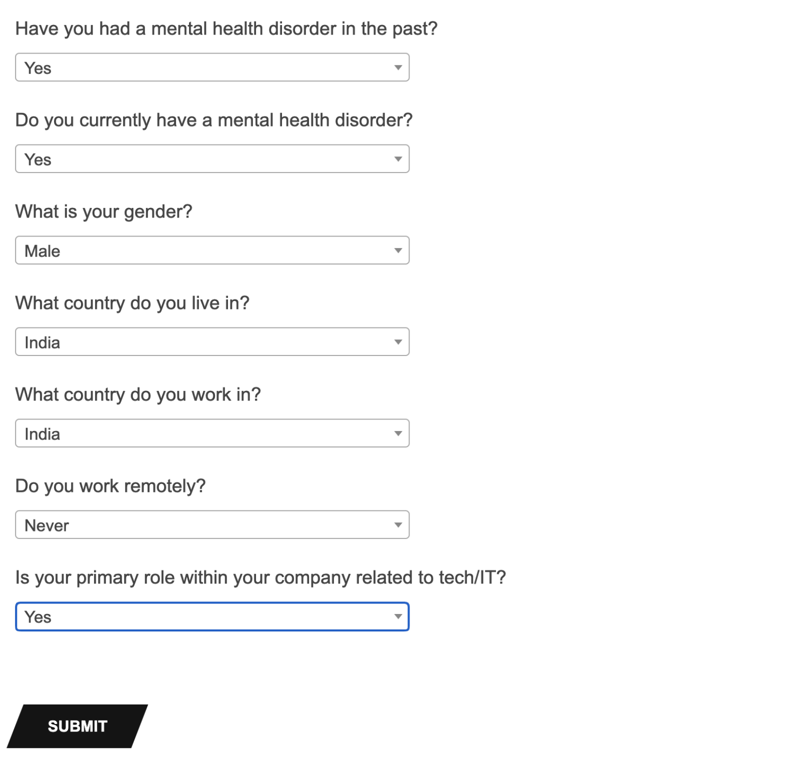

Figure 5.2: Submit form

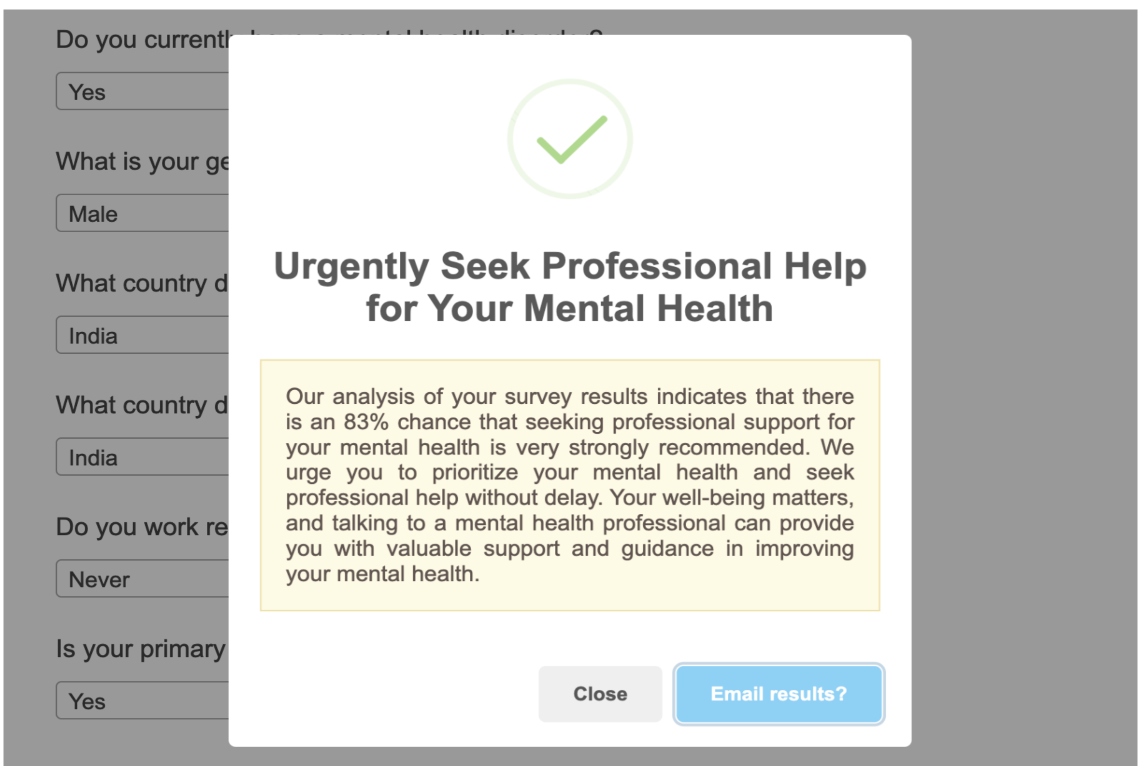

Figure 5.3: Prediction

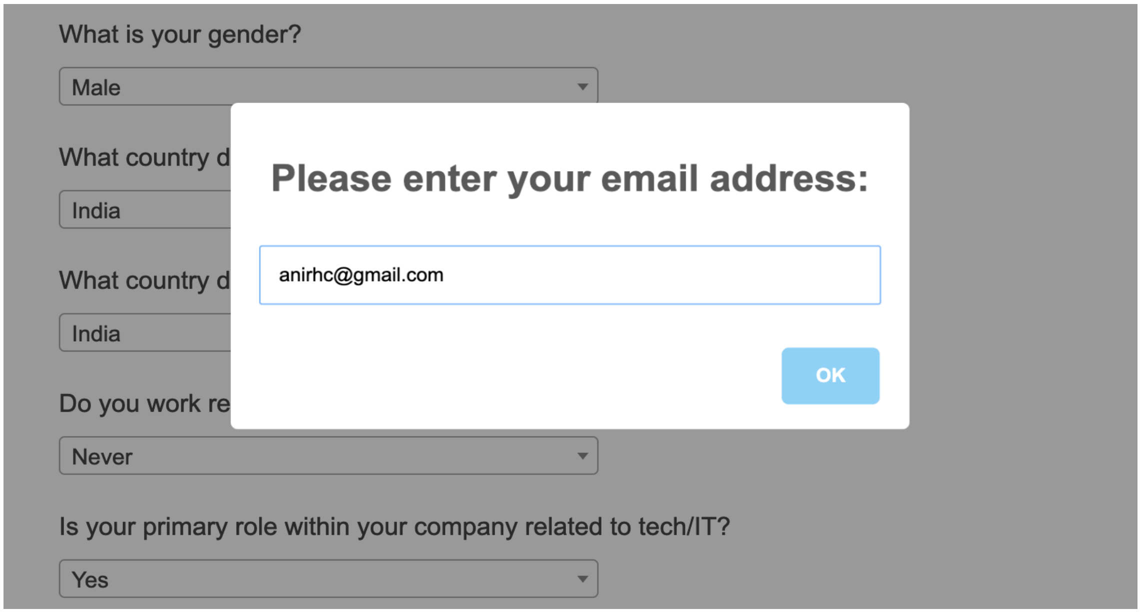

Figure 5.4: Enter email address

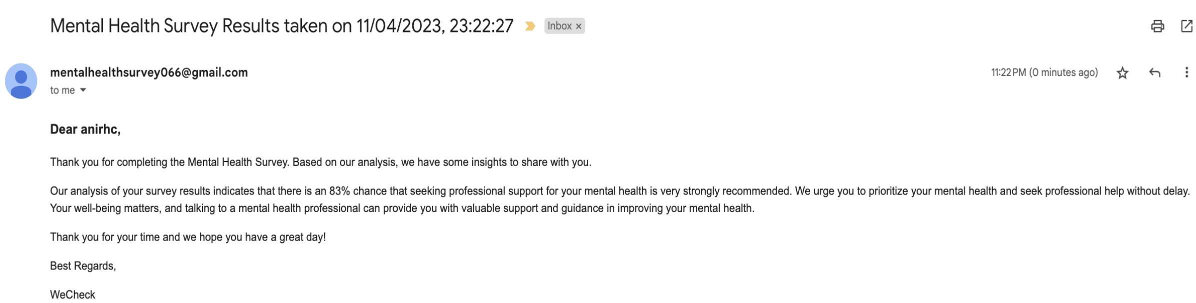

Figure 5.5: Email received

Chapter 6

Model Malfunction Examples

Mental health is a sensitive topic and also one that we still have so much to learn about. It is not extremely straightforward to build a classification model for it. This highlights the complexity and nuance of mental health, and the need for caution when developing models to predict mental health disorders/diseases. There are many factors that go into assessing one's mental health that go beyond the scope of the dataset used in this project and so many more that are actively being studied. Despite these challenges, this project can be seen as a first step towards developing a more sophisticated and accurate model that can better assist individuals struggling with mental health issues. It emphasizes the importance of ongoing efforts to improve our understanding of mental health and develop effective approaches to support those in need.

Taking all of this into account, this project's deep learning model, while promising, comes with its potential limitations. These come in the form of false positives, false negatives or even ethical concerns. These are part and parcel of every deep learning project and although it is not ideal, it is key that we acknowledge and understand these shortcomings to only make the model even better and work towards a more complex and comprehensive model as mentioned earlier.

- **False Positives**: The model may incorrectly classify users as needing mental health support when they actually do not require it. This is mainly because there are many other factors apart from the ones covered in the dataset and this project that constitute the decision making if one is in need of mental health support.
- **False Negatives**: The model may fail to identify users who actually require mental health support. This is also because there are many other factors apart from the ones covered in the dataset and this project that constitute the decision making if one is in need of mental health support.
- **Bias**: The model could also exhibit bias, leading to inaccurate predictions. For example, it may be biased against certain genders, age groups or employment types. This is not always the case in the real world where there is one universal formula to decide based on one factor if someone is in need of mental health support or not. A No Free Lunch Theorem of its own sorts for mental health.
- **Ethical concerns**: Our model may raise ethical concerns related to privacy and transparency. Since our model uses sensitive information keyed in by the user, such as gender or employer details, it may raise privacy concerns.

Chapter 7

Contributions by Team Members

7.1 Contributions

Name	**Responsibilities**
Harikrishnan Chalapathy Anirudh	Train Function, Web Application, Report
Swastik Majumdar	Model, Train Function, Hyper-parameter tuning
Aditya Vishwanath	Data Visualization, Video, Report

Table 7.1: Team Members and Contributions

Chapter 8

Bibliography

8.1 References

- Sethi, A. (2022, December 1). One-hot encoding vs. label encoding using Scikit-Learn. Analytics Vidhya. Retrieved April 15, 2023, from https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/
- ML Cheatsheet documentation. (n.d.). Activation Functions. Retrieved April 15, 2023, from https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html