DATA ANALYTICS LABORATORY DA5401

ML-Challenge Report

SUBMITTED BY

GLEN PHILIP SEQUEIRA
DA24C005

Contents

1. Exploratory Data Analysis

1.1. Visualizations. :

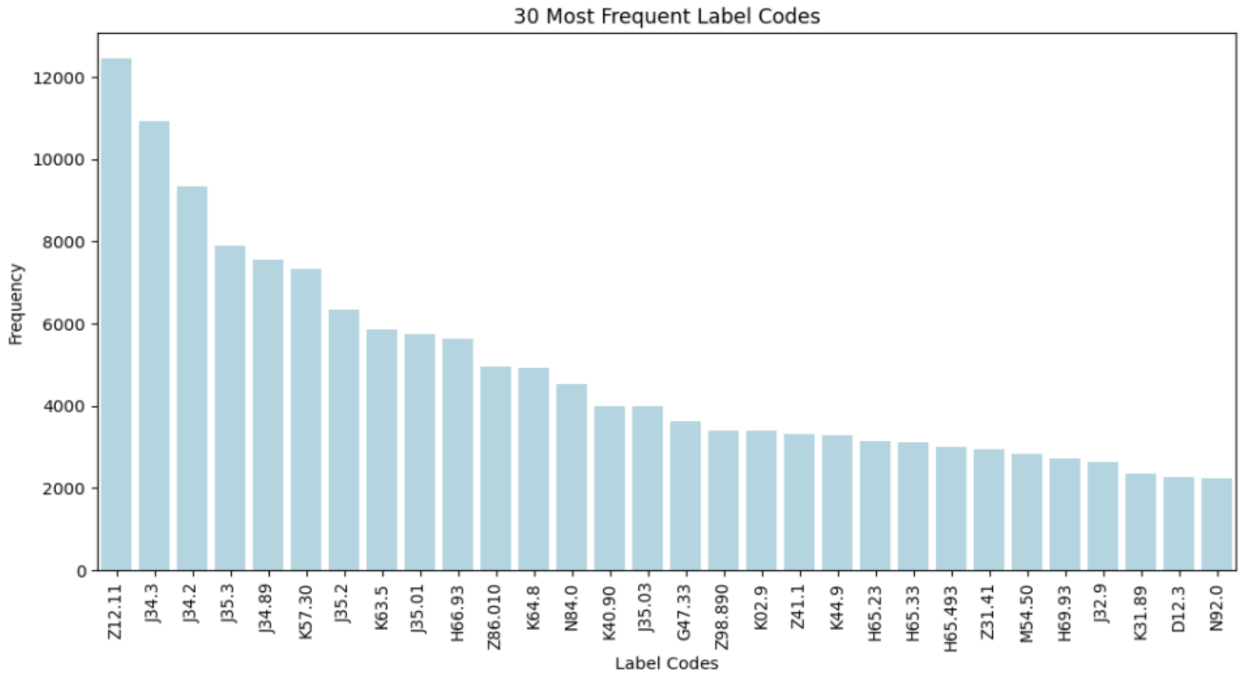


Figure 1.

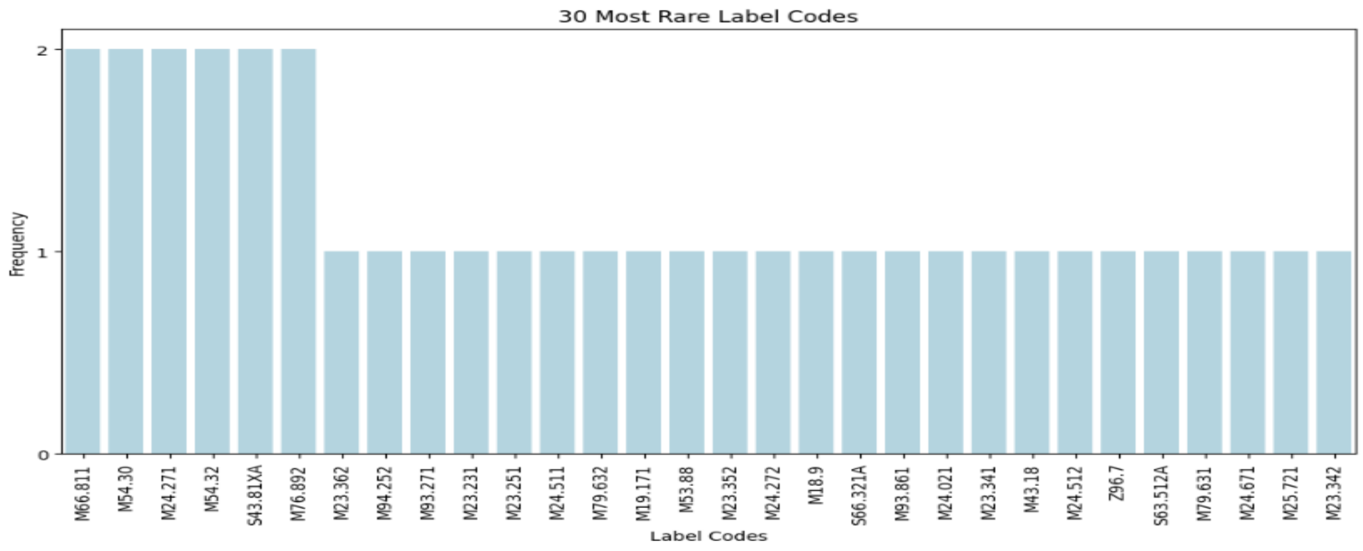
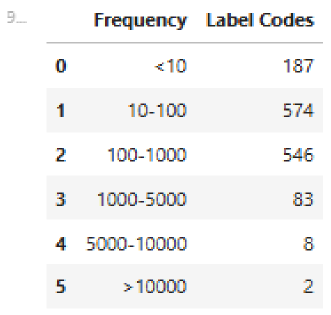


Figure 2.



	Frequency	Label Codes
0	<10	187
1	10-100	574
2	100-1000	546
3	1000-5000	83
4	5000-10000	8
5	>10000	2

Figure 3.

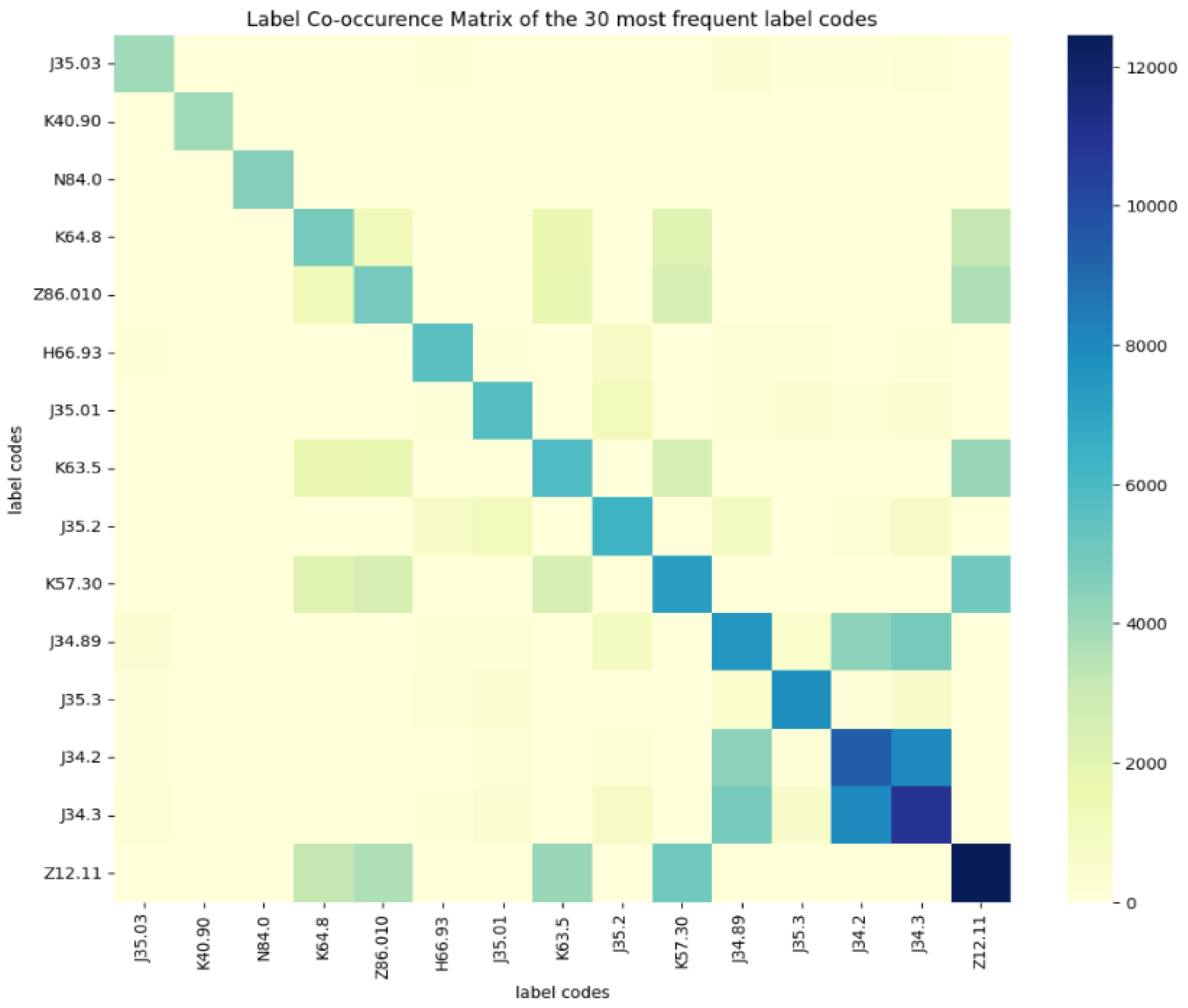


Figure 4.

1.2. Observations and Insights. : EDA is performed on the given dataset and these observations are made:

- From Figure 1,2 and 3, it is very clear that this dataset is highly imbalanced. Some labels appear only once or twice in the dataset, whereas some labels appear in more than thousands of datapoints.

- From figure 3, we can see that 187 label codes appear in less than 10 datasamples. Roughly 90+ codes appear in more than a 1000 datasamples. Whereas, most of the label codes i.e. 1120 out of 1400 appear 10-1000 times.

- From figure 4, we come to know the frequency relationship between the 15 most frequently occuring label codes. Using this visualisation, we observe that a strong correlation in frequency does not exist among the most frequently appearing label codes.

From the above observations, we can derive the following insights:

- Since we know the dataset is highly imbalanced, we can adjust the formula of our custom `f2_score` to balance precision and score appropriately. Also, we can choose suitable loss functions in our model which address class imbalance.

- Due to high class imbalance, the model will focus more on the frequently occurring label codes, overlooking the rare ones. To address this issue, we can do oversampling which increases the representation of the minority labels. By oversampling, we can duplicate the rare labels, which ensures that the model focuses on the rare labels as well.

We are dealing with a medical dataset. The label codes refer to the diagnosis conditions present in the outpatient (OP) medical chart.

- Addressing class imbalance is of high priority in a medical setup because usually we have very little data regarding the rare but dangerous disorders. By addressing class imbalance, we can reduce the number of false negatives (specially of rare diseases) in the diagnosis process.

- Understanding label co-occurrence is very important in the medical domain because it helps us to understand the patterns between diseases which are usually diagnosed together like diabetes and hypertension.

2. Data Preprocessing

2.1. Multi-Hot Encoding. :

`MultiLabelBinarizer()` converts multi-label data into a binary format, where each label is represented as a vector of 0s and 1s. This is called multi hot encoding. This is done because most machine learning algorithms require numerical input and cannot process raw text data, (in our case the ICD20 medical codes). This technique converts text labels into binary vectors, where each unique label is assigned an index, and the presence of each label is indicated by a 1 in the corresponding position of the vector. Also, multi-hot encoding ensures that each label is treated as an independent binary feature, which is crucial for tasks where labels are not mutually exclusive. In our case, we apply multi-hot encoding to `labels` to convert it from text into binary vector format.

2.2. Oversampling. :

Oversampling is done to address class imbalance. In the EDA section, we clearly observed that there is high class imbalance in this dataset. 761 out of 1400 class labels appear in less than 100 data-samples.

In my approach, I have firstly identifies those labels which appear less than 100 times. Then I have oversampled these label codes by duplicated the data-sample. This ensures that the an efficient model is learnt which focusses even on the minority labels.

After oversampling, we split the dataset into training and validation sets using `train_test_split()`. Since, the dataset has roughly 2,00,000 datapoints, we take the validation set as 0.1% of the total dataset. We take majority of the data in the test split so that a good model is learnt on majority of the data so that it can perform well on the actual test data.

3. Neural Network Model

3.1. Choice of Model. :

To solve this multi-label classification task, I have used Neural networks.

- For our model, the embeddings are used as input which are 1024-dimensional vectors. Neural networks can handle complex and non-linear relationships between high-dimensional data, which simpler models cannot capture.

- Since this dataset has 1400 unique labels, it is important to identify combinations of labels across a high-dimensional space. Neural networks can handle this suitably as they can manage a high number of outputs and are capable of learning the dependencies and co-occurrences between multiple labels.

- We have 2,00,000 data points for training. Neural networks usually require a lot of data so that it can learn the complex patterns in the training data so that it can perform well on the unseen test data.

- We can tune several hyperparameters in neural networks. We can also use custom loss functions and performance metrics. This flexibility in choice helps to tune the model according to the task at hand.

3.2. Working. :

Neural Networks:
I have used a neural network model using `TensorFlow` with various hyperparameters suitable for getting a high F2 score on the test data.

- The neural network model that I have used has an input layer, two hidden layers and an output layer.The input layers accepts vectors of size 1024 to match the dimensionality of the embeddings. The two hidden or dense layers each have 2900 neurons to capture complex relationships in the data. The output layer has 1400 neurons with a sigmoid activation functioin, corresponding to the 1400 ICD10 medical codes. Each neuron independently predicts the probability of one label.

- $\texttt{LeakyReLU}(\mathrm{x}) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}$
  where x is the input and α is the negative slope coefficient
  `LeakyReLu` activation function is applied to the 2900 neurons in the two hidden layers. It helps to avoid inactive neurons by maintaining gradient flow and boosting the learning process which is ideal for multi-label classification task.

- $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$
  where x is the input
  Sigmoid activation function outputs values between 0 and 1, with values close to zero for large negative inputs and values close to one for large positive inputs. Since there are 1400 neurons in the output layer with the sigmoid function. The presence of a label is indicated by each neuron independently if the sigmoid function outputs

a value above a threshold, say 0.5.

- `Dropout` layers and `BatchNormlaization` layers are added after each hidden layers. Dropout layers helps the model to learn generalizable patterns by reducing overfitting. Bacth Normalization is done to normalize the activations to stabilize and speed up the training.

- `dropout` rate is a regularization technique used to prevent overfitting. It's a value between 0 and 1, which indicates the percentage of neurons that are ignored or dropped out during training.

- The `negative_slope` value in `LeakyReLu` activation function controls the slope for negative inputs. The standard `ReLU` function setss negative inputs to zero. Using `negative_slope` we allow a small, non-zero gradient for negative values to prevent neurons from becoming inactive ("dead neurons") during training.

- In our model, we use the `binary_crossentropy` loss. It is commonly used in binary classification and multi-label classification tasks.It measures the difference between the predicted probabilities and the actual binary labels (0 or 1). It penalizes the model more when the predicted probability is far from the actual label, and less when the prediction is close to the true label.

- An optimizer is used to adjust the parameters of a model during training in order to minimize the loss function. For our model, we use `Adam` optimizer.`Adam` optimizer adjusts the learning rate for each parameter, helping the model converge faster and more efficiently. This is suitable for our task as we have a large dataset with 2,00,000 datapoints and 1400 labels.

`f2_score` function

We make a custom function which calculates the F2 score, which emphasizes recall more than precision. The following formulas are used in the function.

Predictions are converted into binary values (0 or 1) based on the threshold. If the predicted value is greater than the threshold (0.5), it becomes 1 (positive) otherwise, it becomes 0 (negative).

$$\text{True Positives (TP)} = \sum(\text{y\_actual} \cdot \text{y\_pred})$$
$$\text{False Positives (FP)} = \sum((1 - \text{y\_actual}) \cdot \text{y\_pred})$$
$$\text{False Negatives (FN)} = \sum(\text{y\_actual} \cdot (1 - \text{y\_pred}))$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP} + \epsilon}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN} + \epsilon}$$

where ϵ is a small constant to avoid division by zero.

Then, we calculate the F2 score using the formula:

$$\text{F2 Score} = \frac{5 \cdot \text{Precision} \cdot \text{Recall}}{4 \cdot \text{Precision} + \text{Recall} + \epsilon}$$

`predict` function

This function uses a trained neural network model with a specific value of `dropout` and `negative_slope` to make predictions on a test dataset. The model then generates predictions on the embeddings, which are probabilities between 0 and 1.The predictions are converted into binary values based on a threshold (0.5), which indicate the presence or absence of the label. The binary predictions are then converted back into the original labels using `inverse_transform` method of `MultiLabelBinarizer`. A DataFrame is created with the sample IDs and their corresponding predicted labels, and it is returned.

3.3. Hacks and Workarounds. :

The neural network model has several hyperparameters like `dropout`, `negative_slope`, `optimizer learning_rate`, `loss_function`, `batch_size`, `epochs`, `evaluation_metric`.

I tried different permutations and combinations of these hyperparameters, which resulted in 188 submissions. In the code I have submitted, I have depicted one way in which I tried different combinations:

The two nested for loops iterate over different combinations of `dropout` rates and `negative_slope` values for the `LeakyReLU` activation function. For each combination, a new neural network model is created and trained on the training data. The model's performance is evaluated on both the training and validation data using loss and F2 score.For each model, predictions are done on the test data and saved in a `.csv` file, whose name indicates the combination used to generate the prediction. For example: if predictions are done using a model trained on dropout = 0.5 and negative slope = 0.1, the prediction file will be saved as `dropout_0.5_negslope_0.1.csv`. This helped to keep track of the hyperparameters which would give a good F2 score on the test data.

I tried varying the number of neurons the hidden layers, tried increasing the number of hidden layers, varied the number of `epochs`,texttbath_size, `dropout` rates and `negative_slope`. I also tried different activation functions, optimizers and loss functions.

In the code I have submitted I have depicted how I did manual grid search over the hyperparameters using nested for loops over `dropout` rates and `negative_slope`. But actually, I tried different combinations of all the hyperparameters by varying their values.

In the next subsection *Results*, I have shown the performance of different combinations of `dropout` and `negative_slope` values.

3.4. Results. :

We train the neural network model for 65 epochs using different values of `dropout` and `negative_slope`. For every combination, we record the train and validation loss and F2 score of the last epoch. This gives us an idea of the performance of the model on the training data and the 'unseen' validation data.

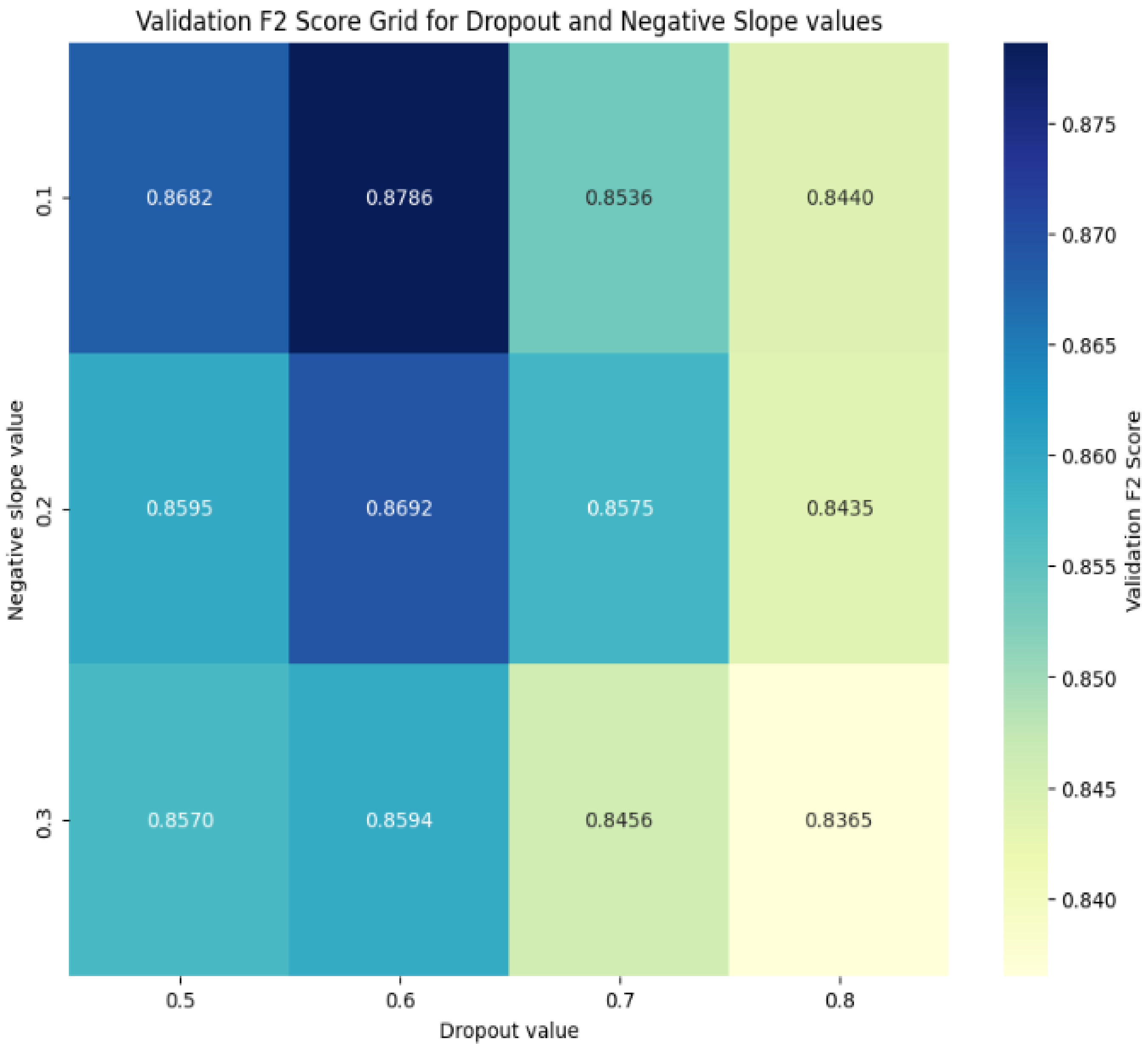
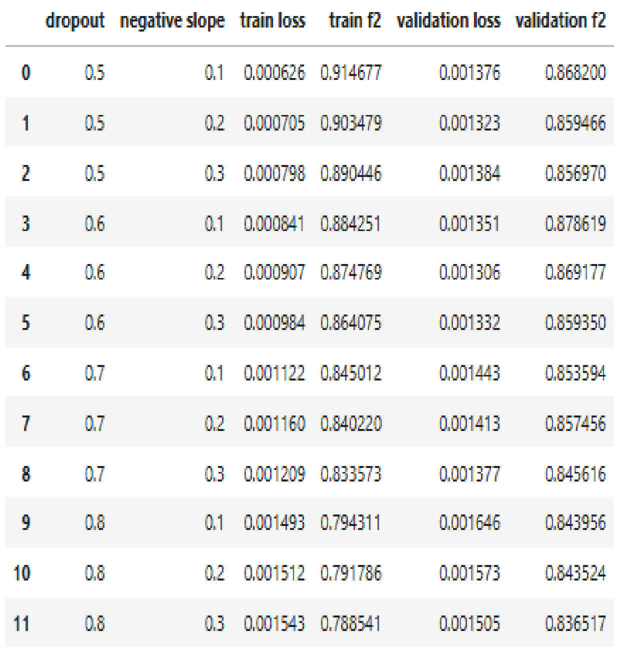


FIGURE 5.



	dropout	negative slope	train loss	train f2	validation loss	validation f2
0	0.5	0.1	0.000626	0.914677	0.001376	0.868200
1	0.5	0.2	0.000705	0.903479	0.001323	0.859466
2	0.5	0.3	0.000798	0.890446	0.001384	0.856970
3	0.6	0.1	0.000841	0.884251	0.001351	0.878619
4	0.6	0.2	0.000907	0.874769	0.001306	0.869177
5	0.6	0.3	0.000984	0.864075	0.001332	0.859350
6	0.7	0.1	0.001122	0.845012	0.001443	0.853594
7	0.7	0.2	0.001160	0.840220	0.001413	0.857456
8	0.7	0.3	0.001209	0.833573	0.001377	0.845616
9	0.8	0.1	0.001493	0.794311	0.001646	0.843956
10	0.8	0.2	0.001512	0.791786	0.001573	0.843524
11	0.8	0.3	0.001543	0.788541	0.001505	0.836517

FIGURE 6.

Figure 5 visualizes the f2 score (last epoch) of the validation data for different values of `dropout` and `negative_slope`. This gives us an idea of the relative performance of the combination on 'unseen' data. The visualisation shows that the combination of dropout = 0.6 and negative slope = 0.1 gives best validation performance but f2 score on the actual test data could be different. That's why I submitted all the prediction `.csv` files. I observed which gave the higher values of F2 score on 60% of the test data. I noted down those hyperparameter values and varied the combinations accordingly to get better F2 scores.

Figure 6 is a table which visualizes the loss and F2 score (last epoch) of the train and validation data for different hyperparameter values.

Finally, the hyperparameters which gave me the highest F2 score (0.489) on 60% of test data was:

- `dropout`: 0.72
- `negative_slope`: 0.3
- No of units in the hidden layer: 2900 in both hidden layers
- `optimizer`: Adam
- `loss_function`: `binary cross-entropy`
- `batch_size`: 128
- `epochs`: 65