KAGGLE COMPETITION

Team Wiski

Murielle Mardenli
Matricule: 2155882
murielle.mardenli@polymtl.ca

Zarine Amy Ardekani-Djoneidi
Matricule: 2139186
zarine-amy.ardekani-djoneidi@polymtl.ca

Lucas Bouchard
Matricule: 2143675
lucas-2.bouchard@polymtl.ca

1 FEATURE DESIGN

1.1 META-MODEL

Firstly, the dataset underwent several pre-processing methods before being passed into the Bernoulli Naive Bayes algorithm. Categorical features, such as `BMI_Category`, `Education_Level`, and `Income_Group`, contain non-continuous features. These were transformed into numerical representations to make them usable for machine learning models, as BNB expects numerical inputs.

As for missing values, through data exploration, it was determined that there weren't any missing values in the feature set nor in the labels set. Therefore, there wasn't a need to handle this use-case.

Through data exploration, it was observed that continuous features had different ranges of values (see figure 1). This can be penalizing for the models used in the meta-model such as Logistic Regression, since it's more sensible to outliers. Therefore, the examples of the dataset were normalized using a StandardScaler to ensure features contributed equally to the learning of the model and to improve its convergence.

Through more exploration of the features, it was decided that the feature `Age_Group` was to be removed from the dataset, since it contained the same value for all the examples, making it a redundant feature which would slow down the learning process if kept. The remaining categorical features `BMI_Category`, `Education_Level` and `Income_Group` were one-hot encoded since the neural network cannot have categorical values as inputs. This method was chosen in order to eliminate any ordinal relationship the values would have with regular encoding as these categories don't have an ordinal relationship within their values.

1.2 BERNOULLI NAIVE BAYES

For the Bernoulli Naive Bayes, a feature analysis was used, which focuses on the relationship between features and the target variable `Diabetes_binary`. The analysis is split into univariate (single feature) and multivariate (combinations of features) analyses. This preprocessing method was chosen for this particular project because the data is high dimensional and therefore it's vital to determine the relative importance of each feature to improve the model's performance.

Univariate analysis explores the relationship between a single feature and the target variable, which helps analyze the central tendency and dispersion (Dremio, 2024) . In the code, this analysis allows the calculation of the importance of each feature for predicting the target variable based on its distribution across the two classes.

On the other hand, multivariate analysis explores the interactions between features and its impact on the target variable. It seeks to draw conclusions from the complex relationships between variables and is very useful in the case of the project, where a large amount of features are given in the data set (Dremio, 2024). In the code, this analysis helps therefore to calculate the importance of each combination of features for the prediction of the target variable. These importance scores are used to guide the BNB model interpretation of the data and the combination of features are then used to have a more informative training set.

2 ALGORITHMS

2.1 PREVIOUS MODEL

The first model implemented for this task was a Random Forest tree classifier. This algorithm works by training multiple decision trees and taking their outputs to make a prediction (GeeksForGeeks, 2024). It was chosen because of its high accuracy and resistance to noise and outliers all due to its ensemble approach. However, the model had a low f1-score of 0.2. Due to the difficulty of interpretation, it was very challenging to debug and understand the lack of precision and accuracy of the predictions. Through more model exploration, it was observed that a neural network had a better performance than the Random Forest. We infer that this is due to its ability to learn complex non-linear relations between the data, as the data is itself complex. Therefore, the Random Forest model was discarded from the process.

2.2 FINAL MODEL

For our final model, 3 models were designed and chosen in order to make predictions due to their high f1-scores. These models were later combined using a stacking method.

2.2.1 COMBINATION OF BNB AND LOGISTIC REGRESSION MODEL

The first base learner combines **Bernoulli Naive Bayes (BNB)** and **Logistic Regression (LR)**.

BNB works by applying Bayes' theorem under the assumption of conditional independence among features. It was chosen due to its high suitability for binary predictions and categorical data and its compuational efficiency (GeeksForGeeks, 2023). It is used in this problem to generate initial probabilistic predictions for features engineered from the dataset. This feature engineering process analyzes the relations between features and the binary target prediction in order to calculate the importance of each feature on the prediction. More than that, it only considers the combinations of features that have a presence of at least 0.1 percent in the training data, to prevent rare combinations of features to flood the engineered features pool. These probabilities are then combined with the continuous features, which are after passed to the logistic regression model as inputs.

Logistic regression is a linear model optimized for binary classification tasks. It works by separating data points using a linear boundary. The output of the BNB model is passed in this logistic regression model to attempt to extrapolate as much meaning as possible from the data, with a model fit for discrete features and another for continuous features.

Therefore, the combination of the two models improves the feature representation for either types (continuous or discrete). The new engineered features attempts to maximize the usefulness of BNB by already combining features that are highly correlated when it comes to predicting the target class. Thus, the BNB model will assume independence between combinations of features instead of independence between each individual features, which reduces the inter-dependance of the features. After BNB makes a continuous prediction for each sample, these predictions originating from discrete features are combined with the continuous features.

By combining a probabilistic model like BNB that draws probabilistic insights using feature dependencies modeling, and Logistic Regression, a discriminative model that optimizes decision boundaries for classification tasks, the logistic regression model can make better estimated predictions Mitchell (1997). This combination is particularly interesting in this project, where the data contains a mix of continuous and binary features.

2.2.2 ARTIFICIAL NEURAL NETWORK

The second base learner is a fully connected **Artificial Neural Network** designed for this classification task. Its architecture features three dense layers with large neuron counts (2000, 1000, 1000), along with Leaky ReLU activation functions to avoid vanishing gradients. Dropout layers are incorporated to prevent over-fitting with a value of 0.3. Then, a final sigmoid activation normalizes the output between 0 and 1. This model is trained using a custom F1-score loss function, which optimizes the evaluation metric. The neural network's flexibility allows it to model highly complex and non-linear relationships within the data.

2.2.3 STACKING METHODOLOGY

The stacking ensemble method was chosen in order to combine the predictions of the previous models to obtain a better prediction as well as reduce both bias and variance. Stratified k-fold cross-validation was used in order to prevent overfitting. Therefore, the dataset was split into training and validation folds, with a balanced representation of classes in each fold. For each fold:

1. The two base models are trained independently on the training portion of the fold.
2. Then, the predictions are generated on the validation fold. These predictions form the input features for the meta-model.

After all folds are processed, the predictions from the base models on each validation fold are concatenated to form a new dataset, which includes the original target labels. This dataset is used to train a **meta-model**, which learns to integrate the outputs of the base learners.

2.2.4 FINAL MODEL TRAINING AND EVALUATION

Once the meta-model is trained, the base models are re-trained on the full training dataset to ensure they use all the available data. Their predictions are then passed to the meta-model, which makes the final predictions. A hold-out validation set or additional cross-validation is used to evaluate the overall performance of the stacking ensemble. This ensures that the results are reliable and free from data leakage.

3 METHODOLOGY

3.1 NEURAL NETWORK

Before stacking models, we tried to optimize our base neural network as much as possible. We started by creating a loss function `f1_loss()` that would help us maximize the f1 score of our model. Since the dataset contains an imbalanced number of positive and negative labels, an increase in the accuracy score does not necessarily result in an increase of the f1 score. Indeed, the f1 score differentiates between wrongfully classifying positive labels as negative and wrongly classifying negative labels as positive, but the accuracy metric does not. We cannot use the f1 score itself as a loss function because it is not differentiable, but we can make it differentiable by using probabilities as input instead of 0 and 1 predictions. The loss function then consists in minimizing (1 – modified_f1) (Haltuf, 2018).

To help us get a sense of what problems our model encountered during training, we plotted the loss and f1 score on the training set and validation set at each epoch. Thanks to these plots, we noticed our initial version of the neural network was overfitting the training data. The training loss is consistently decreasing but the validation loss quickly stops decreasing and increases instead, as we can see in figure 2.

To reduce the issue, we added regularization using dropout layers and weight decay. Dropout layers randomly deactivate some neurons at each epoch of training, which creates some noise and helps reduce overfitting. Weight decay penalizes high values for weights (L2 penaly).

Using weight decay alone was not enough to limit overfitting: the validation loss almost stops decreasing after 50 epochs (figure 3).

Using dropout layers alone is not enough either: again, the validation loss almost stops decreasing after 50 epochs (figure 4).

We were able to get better results by using both weight decay and drop out layers: the validation loss is still decreasing at a good rate after 50 epochs (figure 5).

We used an Adam optimizer because it gave us good results. To determine the best learning rate, we tested different values and tried to get a plot where the loss is decreasing at a good rate and is not too "shaky". After a process of trial and error, we concluded that the following set of hyperparameters gets the best results:

- Learning rate = 2e-5
- Weight decay strength =1e-3
- Dropout probability = 0.3

We can see that the model with the best validation f1 score is not necessarily the model obtained after the last epoch of training. The best model may be in the middle of the training process. Because of this observation, we decided to save the current model in a file each time it gets a new best validation f1 score, so that we can train our model for a long time without worrying about losing the best model.

3.2 Stacking

After finding the best hyperparameters for our base models, we used them in a stacked model. We were hoping to build on the strengths of both base models and get a final model that would perform better than both of them. We trained the meta model on a dataset containing predictions of both base models on the training data. In order to train the meta model on the whole training set and not only a small validation portion, we used k-fold cross validation: we divided our training set into ten folds and, for each fold, we trained our base models on nine training folds, then made predictions on the validation fold. We used stratified k-fold to make sure the positive and negative labels are equally distributed in the folds. We also held out a small validation set (20% of the initial training dataset) and never used it to train any model, so that we can use it to evaluate our final model at the very end of training.

We used a neural network for the meta model, but we reduced the number of neurons and did not use dropout layers because the meta model does not need to be as complex as the base models. The hyperparameters for the meta-model were chosen in order to have a slow and steady convergence (thus lower values):

- Learning rate = 5e-4
- Weight decay = 1e-3

Finally, we compared the validation f1 scores of the base models and the stacked model and noticed stacking achieves a slight improvement. The scores of the models are as follows:

- Bernoulli x logistic regression: 0.4364
- Neural network: 0.4667
- Stacked model: 0.4670

4 Results

Having now gone trough the complete machine learning pipeline and explained our choices for each step, it is time to see the results:

- Final Validation F1 score: 0.467

But looking back at the scores for each sub-model before combining them using a stacking method:

- Bernoulli-logistic F1 score: 0.436
- Neural Network F1 score: 0.467

It can be observed that the neural network does most of the heavy lifting. This would mean that the Bernoulli-logistic approach adds little improvement to the overall results. Through further examination, we have concluded that this happens since the neural network extrapolates relations between all inputs whereas the Bernoulli-logistic approach limits itself to only a few combinations of binary and continuous features.

Let us focus on a few of the most important hyper-parameters and methods we applied to the neural network. Let us start with the most important hyper-parameter, the learning rate for the neural network. We chose a value of 2e-5 for the final model, which can be seen in the figure 6, as we had only tested the model with learning rate values ranging from 1e-02 to 2e-05 and this was the best value for the model. We can see that the curve is noisier for both the loss and the f1-score, but the model learns at a good rate as the loss decreases a lot for the training set. When you increase the learning rate to 2e-04 in figure 8, the curve becomes even noisier and less stable for both the f1-scores and the loss, which doesn't make a smooth convergence. From this base learning rate, when you decrease it from 2e-05 to 2e-6 in figure 6, the learning curve and the f1-score curves are much smoother, allowing for a smoother convergence of the loss and a smoother value of accuracy and precision. In retrospect, lower values of learning rates could've been tested for better performance.

And for the number of hidden layers, we chose 3 of size (in order) 2000, 1000 and 1000. We found that adding more neurons did not help increase the model accuracy. Finally, a dropout rate of .3 for a few of the hidden layers greatly improved the ability of our model to generalize. So, in conclusion, we did our best to train a neural-network and finally ended up 15th in the Kaggle rankings with a final f1-score of 0.465. But looking at the top score of 0.830!!!!, which is almost double, we still have a long way to go.

5 Discussion

So, in the end, while the model performed relatively well on the training set, it showed much less promise when applied to validation sets during cross-validation. This indicates that while the model fit relatively well to the training data, its ability to generalize to unseen data was limited by some factors.

Our method provided some advantages in this problem. Firstly, one of the main advantages of our approach was the neural network's ability to automatically perform feature selection and adapt to complex patterns in the data. This is particularely important in this problem, where the data has complex relations and is very large. Additionally, our approach uses stacking, which in theory is supposed to improve the accuracy of the model. We also used a k-fold method in order to prevent the favorism of one model if it overfits too much to the data.

However, our method had a few faults due to its lack of generalization. One hypothesis of this is probably due to insufficient data preprocessing. The lack of data preprocessing is a very likely answer since it would explain our highly flexible model's inability to generalize properly on unseen data. And so, more time should have been dedicated to preprocessing steps such as handling outliers and addressing class imbalances which would likely have reduced the noise in the dataset and therefore helped the model to generalize better. Also, the one-hot encoding method could have not been applicable to all features to scale them, as the values for the `BMI_Category` do have an ordinal relationship, which was erased by the use of one-hot encoding instead of regular encoding.

In future areas of work, it would be interesting to consider better pre-processing methods in order to thoroughly analyze the data. A more thorough hyperparameter search could've been used in order to find other parameters which can improve the performance of the model, such as the framework Optuna, specifically designed for Machine Learning models. Another interesting approach to the problem would've been to solely focus on a Logistic Regression model, while preprocessing the data well in order to eliminate the effect of outliers, which is penalizing in a linear boundary context.

6 Statement of contribution

Defining the problem and methodology:

- All members contributed equally to developing the initial models and assessing the project requirements

Coding the solution:

- Murielle: Code a Random Forest model
- Lucas: Code a BNB and Logistic Regression model
- Zarine and Murielle: Code a Neural Network

Writing the report:

- Murielle: Feature design and Algorithms
- Lucas: Results and Discussion
- Zarine: Methodology and citations

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

Dremio. Univariate and multivariate analysis. *Dremio*, 2024.

GeeksForGeeks. Bernoulli naive bayes. *Geeks For Geeks*, 2023.

GeeksForGeeks. What are the advantages and disadvantages of random forest? *Geeks For Geeks*, 2024.

Michal Haltuf. Best loss function for f1-score metric. *Kaggle*, 2018.

Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

A Appendix

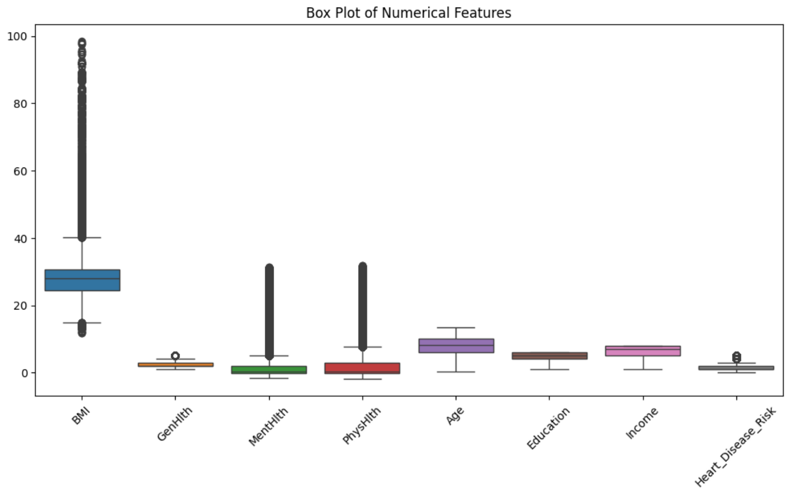

Figure 1: Range of values of numeric features of dataset

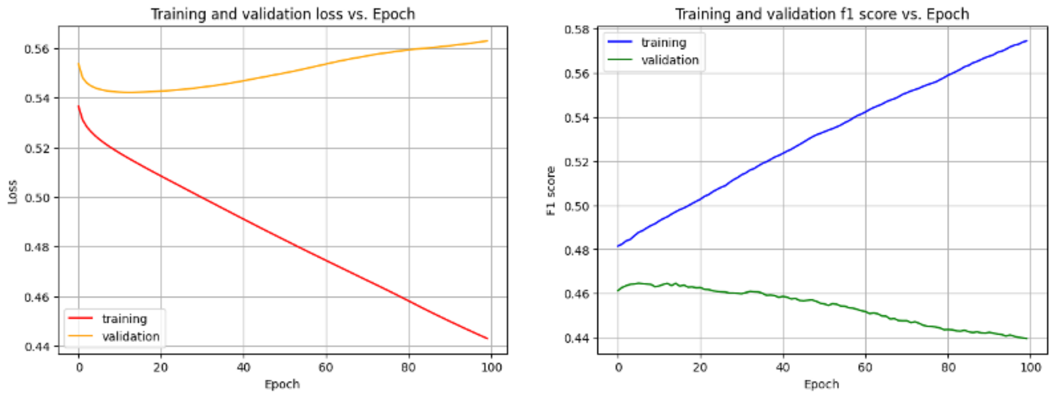

Figure 2: Training and validation loss and f1-score for initial model

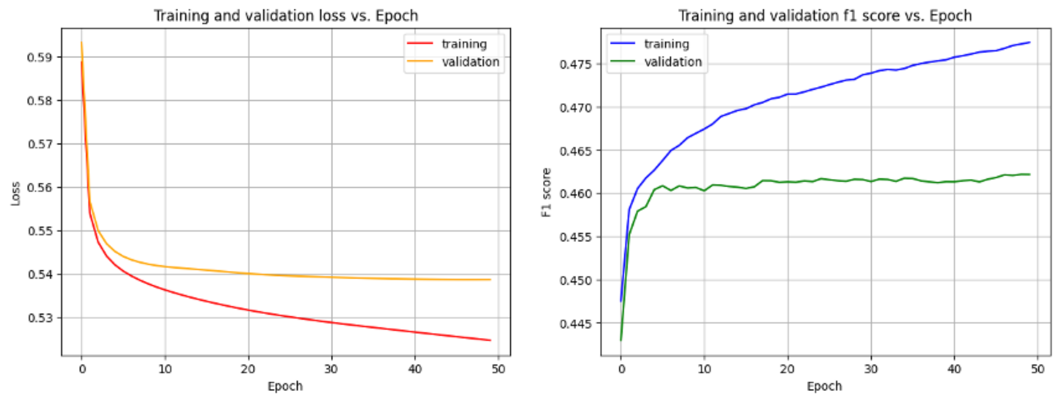

Figure 3: Training and validation loss and f1-score with learning rate = 1e-5 and weight decay strength = 1e-6

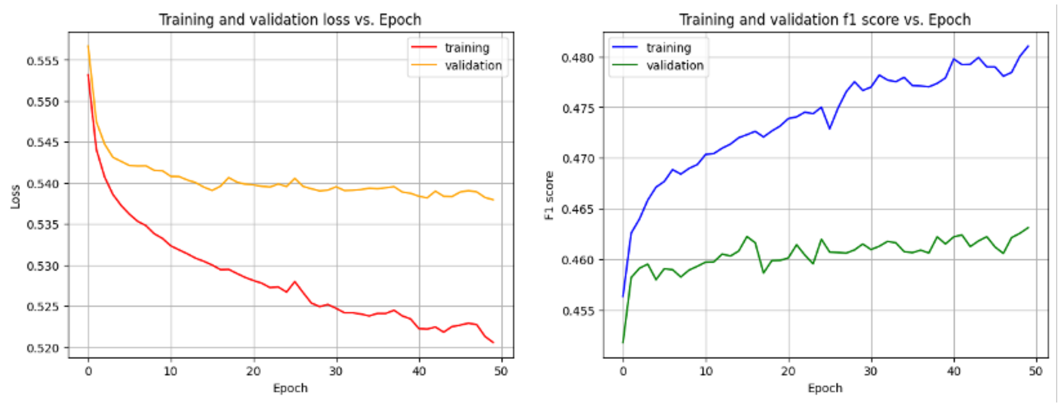

Figure 4: Training and validation loss and f1-score with learning rate = 1e-5 and dropout layers

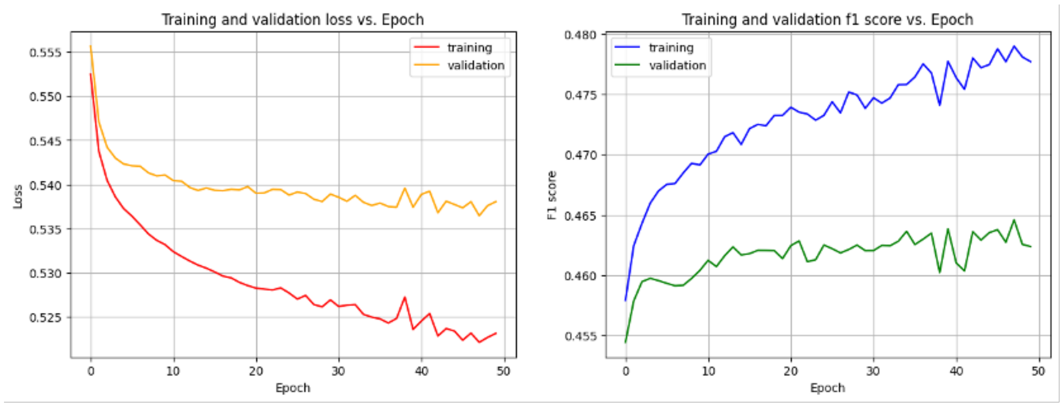

Figure 5: Training and validation loss and f1-score with learning rate = 1e-5, weight decay strength = 1e-6 and dropout layers

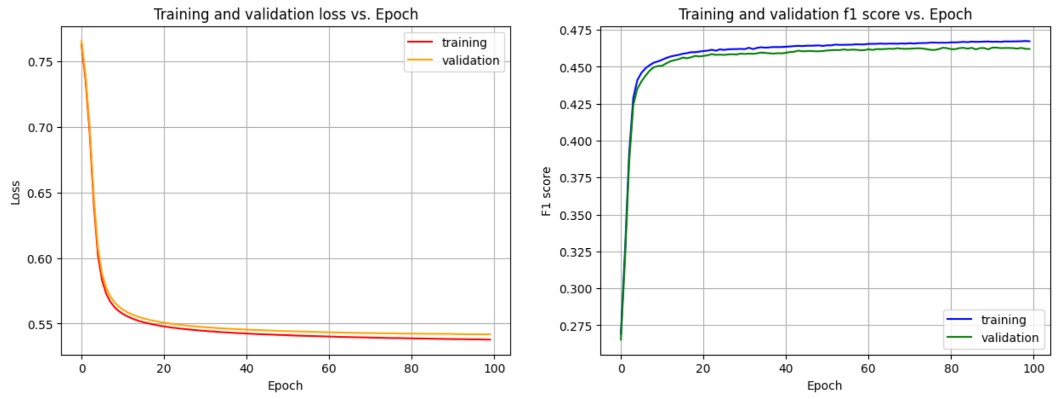

Figure 6: Training and validation loss and f1-score with lr = 2e-6 (f1 validation upper bound at 0.462)

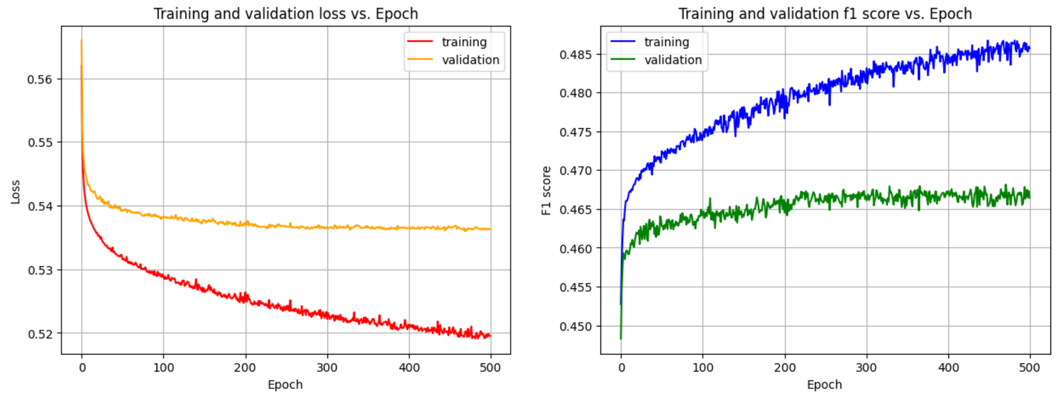

Figure 7: Training and validation loss and f1-score with lr = 2e-5

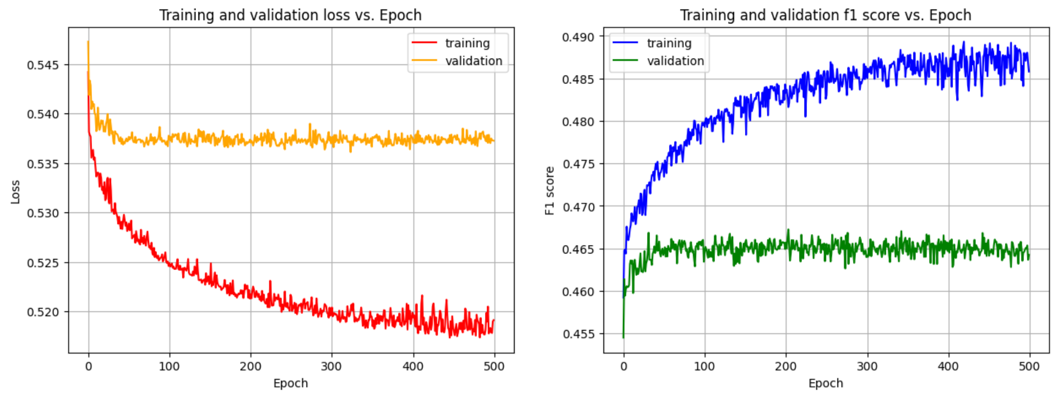

Figure 8: Training and validation loss and f1-score with lr = 2e-4