Assignment 1 - Predict Medical Issues

Administrative details

Deadline: March 26th 2025, 23:59

Scoring: 2 points. After the deadline, there is a 0.25 points penalty per day, for 4 days. If you delay your submission by more than 4 days, the maximum score for the assignment is 1 point. You will personally present the assignment to a teaching assistant in the assignments evaluation week (week 9).

Questions: Need help? Use the *#Teme* Teams channel.

Submission: Upload link: `https://forms.gle/3QsxEscKS9Gr1Vag7`. The assignment should be a Python Notebook that contains the code and presents the results of the experiments.

Assignment

Task According to the World Health Organization stroke is the second leading cause of death globally, responsible for approximately 11% of total deaths. We propose you a dataset to predict whether a patient is likely to get stroke based on the input parameters like gender, age, and various health and social information.

What will you learn? The purpose of this assignment is to familiarize you with the training and evaluation process of a basic Neural Network in PyTorch. It also introduces to you some notions necessary when working with imbalanced datasets (which is very common in practice).

Dataset Description The dataset contains anonymized data for 5110 patients. In Fig. 1 you can see the 11 columns available in the dataset. We have already separated the dataset into a train and test splits (see the extra `split` column in the dataset). **Reminder**: Do not tune your hyper-parameters on the test split.

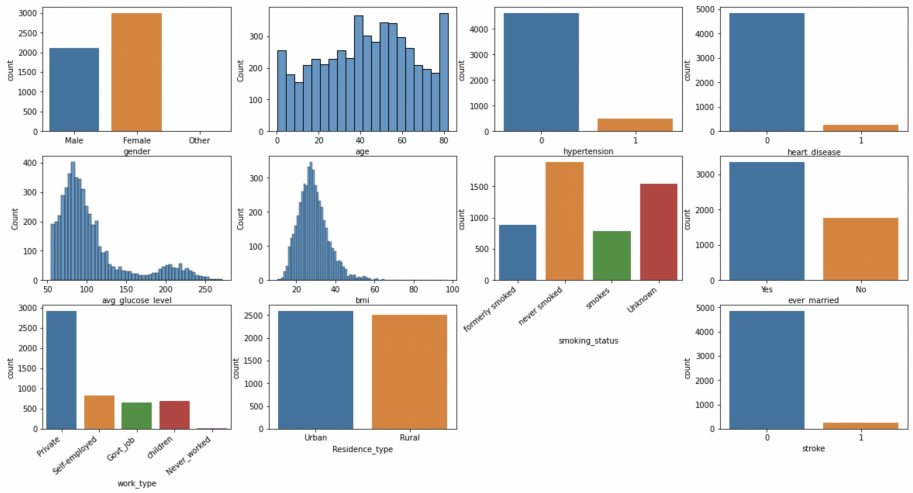

Figure 1: Original Dataset Source: Healthcare Dataset Stroke Data from Kaggle. Please use the dataset from the csv that we provided you, because we already preprocessed the data for you.

TODOs

We will guide you step by step through the necessary steps for solving each subtask, providing you also details on how to verify that things work as expected. In this assignment, you should train a neural network to predict for each entry (each patient) the stroke label from the dataset.

0. Make a new notebook in `https://colab.research.google.com`, mount your google drive and copy the `pmi-data.csv` dataset there. Here you have the detailed steps for it. Load the csv dataset in the notebook using the pandas package and apply `describe` to the pandas dataframe to see its structure.

1. (0.1 points) Datasets and Dataloaders Start by splitting the existing train dataset into a train and a validation one (you can use `train_test_split` from `sklearn.model_selection` package). Make sure you shuffle the data and use 25% of the train dataset for validation. Briefly explain why do you think that those choices are considered good practices. Build next a dataset and a dataloader for both training and validation sets (read more about it here).

2. (0.2 points) Model Build a Neural Net model class that inherits the `nn.Module` and overwrites the `__init__` and `forward` functions. Use several linear layers followed by non-linearities (e.g.`nn.Linear` and `F.relu`). Make sure that the default `parameters` function works as expected (do not overwrite it!). This function is used by the optimizer to keep track of the trainable parameters.

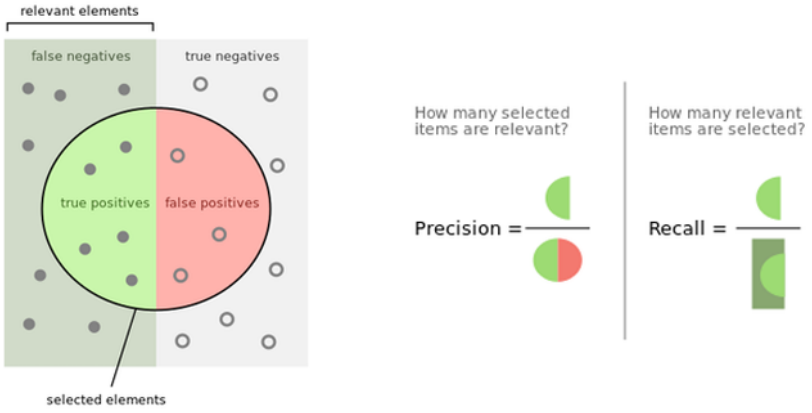

Figure 2: Precision and recall.

```
# this should print all the parameters from your model
for param in model.parameters(): #named_parameters???
    print(param)
```

3. (0.1 points) Optimizer and Loss function Initialize an optimizer of your choice (e.g. Adam, SGD, etc.) and a proper loss function (also called criterion) for binary classification (we recommend `BCEWithLogitsLoss`).

4. (0.3 points) Training By now you collected all the elements needed for training your PyTorch neural net model from the previous steps. Iterate through the training Dataloader and update your parameters. Make sure you use the GPU from the colab environment. Keep track of your performance for the training and validation set and plot them at the end of the training (both the loss value and the accuracy). Save the model's parameters with the best accuracy on the validation set. For more details on training a model in PyTorch see Lab 3 or this tutorial. Observe how the training and validation curves look.

5. (0.1 points) Evaluation We will now take a closer look at your predictions. As you saw in Fig. 1, the proposed dataset is highly imbalanced for the stroke column (with way more entries on 0 rather than 1). This might lead our neural network into learning that it is better to always predict 0 all the time (since the average loss will be close to zero). Check this by showing the confusion matrix for your predictions on the validation set. Enrich your evaluation by also computing the precision, recall and the F1 scores on the validation set (e.g.`precision_recall_fscore_support` from `sklearn.metrics` package). As the F1 score is the harmonic mean between the precision and recall scores, it is largely affected if either the precision or the recall is small (Fiq. 2).

6. (0.2 points) Balance the training data distribution Use the `pos_weight` parameter for the `BCEWithLogitsLoss` to make your positive (1) samples weigh more in the expected loss. We just saw that accuracy is not an adequate metric in a scenario with severe class imbalance. Keep track of the precision, recall and F_1 scores as well and save the model's parameters based on the validation set F_1 score. **Remark**: Any F_1 above 0.2 is good enough on this dataset.

7. (0.3 points) Focal Loss We will now look at a generalization of the standard Cross Entropy loss. The Focal Loss was introduced for the task of Object Detection, where a severe imbalance between positive (foreground) and negative (background) classes is common (refer to the original paper for more details and experiments). The loss introduces 2 new hyper-parameters:

- $\alpha \in (0, 1)$ - a positive class weight parameter (similar to the one in `BCEWithLogitLoss`). For the negative class, the weight is set to $1 - \alpha$.
- γ - it reduces the importance of making confident predictions, focusing only on making correct ones. Follow the next formulas and figures to better understand its role in the loss.

Using the notations from the original paper:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases} \qquad \alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{if } y = 0 \end{cases}$$

, where p the positive class probability predicted by the model, the Focal Loss (FL) is defined as:

$$\text{FL}(p, y) = -\alpha_t (1 - p_t)^\gamma log(p_t) \tag{1}$$

We present in Figure 3 a graph from the original article for the Cross Entropy and Focal losses. Notice how setting γ to higher values reduces the loss on samples that are predicted correctly ($p_t > 0.5$), especially when the model is also confident (e.g. $p_t > 0.75$).

Use the Focal Loss to train you network. You can implement the loss by yourself based on the given formulas, or you can use any existing implementation. We recommend starting with $\alpha = 0.7$ and $\gamma = 2$ - these values should work decently well, but you can obtain even better result by adjusting them. As in the previous task, keep track of the precision, recall and F_1 scores during training and save the model's parameters with the best validation F_1 score.

8. (0.1 points) Inspecting model outputs. Plot histograms for the positive-class probabilities predicted on the validation set by the three previously trained networks. What do you observe based on these histograms? How do the predictions of the model trained with the Focal Loss differ from those of the model trained with the `BCEWithLogitsLoss` with balancing class weights?

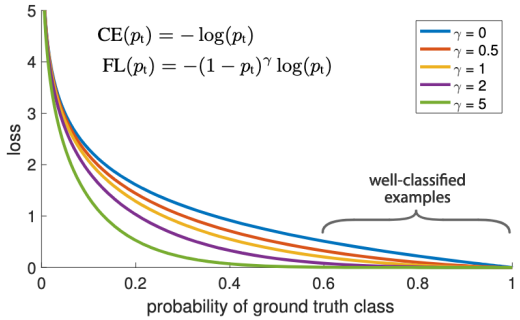

Figure 3: Focal Loss for different values of γ, without the class weighting parameter α. Note that setting $\gamma = 0$ makes the Focal Loss equivalent to the standard Cross Entropy loss.

9. (0.2 points) Beyond default thresholds As you have seen at the previous point, the distribution of probabilities produced by each model can greatly differ. With this in mind, the default threshold of 0.5 used to separate negative and positive samples may not be the optimal one for all models.

Iterate through threshold values in the range [0.25, 0.75] with a step of 0.01, compute the predictions on the validation set based on this threshold and determine the best validation F_1 score that can be attained.

10. (0.1 points) Threshold sensitivity Plot the validation set F_1 score for each threshold from before. Which of the three models is the most sensitive to changes in the threshold's value?

11. (0.2 points) Threshold agnostic metric In this section we will take a look at a threshold agnostic metric that can be used to compare different models. Complete the steps below for the 3 models that you have trained so far:

- plot the precision-recall curve on the validation set - check the `precision_recall_curve` function from the `sklearn.metrics` package
- compute the area under the curve (AUC) metric for the precision-recall curve from the previous point. You can use the `auc` function from the `sklearn.metrics` package.

12. (0.1) Test performance In this final step you must make predictions on the test set with the 3 models that you have trained. Use both the default threshold and the one you selected based on the validation set F_1 score - you will have 6 sets of predictions in total. Compute the accuracy, precision, recall and F_1 scores for these predictions and arrange all of them in a table. Highlight the best results for each metric.