NLP Data Scientist - Quantexa

Ana Lucia Diaz Leppe

The challenge: character-level input

```
Original Tweet: @VirginAmerica it was amazing, and arrived an hour early. You're too good to me.
Cleaned Tweet:  it was amazing and arrived an hour early youre too good to me
Encoded Tweet: [1, 71, 82, 1, 85, 63, 81, 1, 63, 75, 63, 88, 71, 76, 69, 1, 63, 76, 66, 1, 63, 80, 80, 71, 84, 67, 66, 1, 63, 76, 1, 70, 77, 83, 80,
1, 67, 63, 80, 74, 87, 1, 87, 77, 83, 80, 67, 1, 82, 77, 77, 1, 69, 77, 77, 66, 1, 82, 77, 1, 75, 67]
Padded Tweet Shape: torch.Size([1, 100])
Padded Tweet: tensor([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  1, 71, 82,  1, 85, 63, 81,  1, 63, 75, 63, 88, 71, 76, 69,  1,
         63, 76, 66,  1, 63, 80, 80, 71, 84, 67, 66,  1, 63, 76,  1, 70, 77, 83,
         80,  1, 67, 63, 80, 74, 87,  1, 87, 77, 83, 80, 67,  1, 82, 77, 77,  1,
         69, 77, 77, 66,  1, 82, 77,  1, 75, 67]], dtype=torch.int32)
```

Data Loading and Preparation:

- Reads tweet data from files.
- Processes each tweet, labels it, and prepares it for model training.

Text Preprocessing:

- Cleans the text by removing URLs, usernames, and special characters.
- Builds a character-level vocabulary and encodes the text.

Encoding and Padding:

- Encodes tweets into sequences of character indexes
- Pads sequences to ensure uniform input size for the neural network.

Data Conversion:

- Converts lists of encoded and padded tweets into tensors for use in PyTorch.

Training, Validation and Testing

Due to short time:
it reads the num of rows of the minority class and all have the same amount of data.

Data Splitting: Divides the data into training, validation, and test sets while maintaining class proportions.
Tensor Conversion: Converts numpy arrays to PyTorch tensors for use in a neural network.
DataLoaders: Creates DataLoaders for efficiently managing and iterating over the data in batches.

```
Training set label distribution: {0: 1656, 1: 1656, 2: 1656}
Validation set label distribution: {2: 184, 1: 184, 0: 184}
Test set label distribution: {1: 460, 0: 460, 2: 460}
```

What I would have done?
1.SMOTE (big fan): creates new instances for the minority class
2. Undersampling

In my model I have tested:
class weights: assign different weights to different classes.

my “hybrid” neural network model

input:

sequence of characters

embedding layer converts characters to vectors

CNN

CNN

CNN

detecting patterns through convolutional operations

Self-Attention Layers: Captures dependencies between characters in the input sequence.

Bidirectional LSTM Layer: Allows the model to capture information from both directions in the sequence.

Fully Connected Layer: Linear Transformation: Maps the LSTM output to the classification space.

Output:

(0,1,2)

Special Considerations

Optimizers and Learning Rate:
Adam Optimizer: adjusts the learning rate for each parameter.
Learning Rate Schedule: Helps in fine-tuning by reducing the learning rate as training progresses.

Regularization: Includes dropout in the attention mechanism and LSTM.

Training and evaluation

Epochs: Number of training iterations.
Optimizer: Updates model parameters.
Loss Calculation: Computes the loss for each batch and accumulates it.
Backpropagation: Computes gradients.
Scheduler Step: Adjusts the learning rate after each epoch.

```
Epoch 1/8, Train Loss: 1.0509, Validation Loss: 1.0445
Epoch 2/8, Train Loss: 0.9781, Validation Loss: 0.9139
Epoch 3/8, Train Loss: 0.8662, Validation Loss: 0.8403
Epoch 4/8, Train Loss: 0.8104, Validation Loss: 0.8073
Epoch 5/8, Train Loss: 0.8033, Validation Loss: 0.8149
Epoch 6/8, Train Loss: 0.6908, Validation Loss: 0.8106
Epoch 7/8, Train Loss: 0.6361, Validation Loss: 0.8326
Epoch 8/8, Train Loss: 0.6029, Validation Loss: 0.8671
```

Final Results

```
Test Loss: 0.877
Test Accuracy: 0.640
Proportion of predictions for class 0: 0.365
Proportion of predictions for class 1: 0.243
Proportion of predictions for class 2: 0.392
Class 0: Precision = 0.662, Recall = 0.725, F1-Score = 0.692
Class 1: Precision = 0.760, Recall = 0.555, F1-Score = 0.641
Class 2: Precision = 0.545, Recall = 0.641, F1-Score = 0.589
```

1. Test Loss: The average loss over all test samples.
2. Test Accuracy: The proportion of correctly classified samples out of the total number of samples.
3. Precision, Recall and F1-score (key)
4. Proportions of Predictions: I wanted to know if i was really capturing different types of classes.

Other metrics I could have used:
ROC-AUC , CrossValidation (to much for my computer)