Handling Class Imbalance in Multi-Class Classification

(Medical Misinformation Detection in LLM Responses Example)

By: Adeel Ahmed

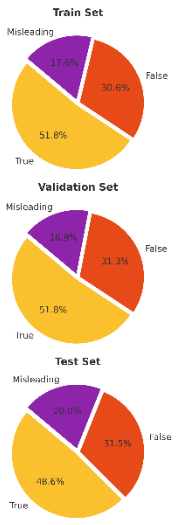
Why Class Imbalance Hurts Model Performance

In imbalanced datasets, models often focus on the majority class and overlook minority classes. In our dataset (3 labels: true, false, misleading), the training set is moderately imbalanced:

```
HealthFact Train label distribution:
label
true          0.517952
false         0.306100
MISLEADING    0.175949
Name: proportion, dtype: float64

HealthFact Validation label distribution:
label
true          0.518122
false         0.313015
MISLEADING    0.168863
Name: proportion, dtype: float64

HealthFact Test label distribution:
label
true          0.485807
false         0.314680
MISLEADING    0.199513
Name: proportion, dtype: float64
```



This imbalance can severely hurt performance on the minority class. A standard model may predict "true" most often to achieve high accuracy, failing to detect misleading claims. This leads to poor performance on the misleading class. Since the model sees few misleading examples during training, it struggles to recognize them. The result is often high overall accuracy but low recall for misleading. To measure this properly, metrics like **Macro F1** are preferred, as they penalize poor performance on minority classes. Accuracy or Weighted F1 may still appear high if the model performs well on true/false and ignores misleading.

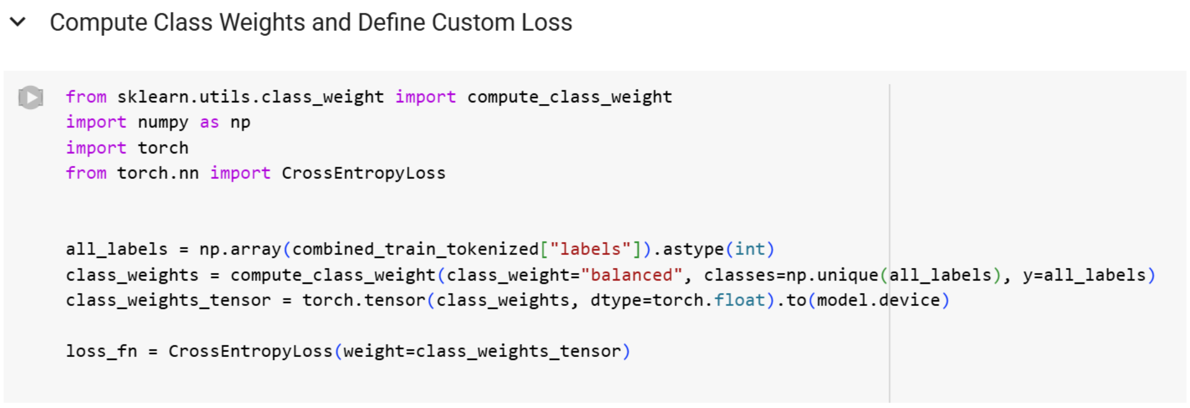
Key point: Class imbalance (especially when a class forms only ~10–20% or less of the data) can lead to a model that almost never predicts the minority class. We need strategies to ensure the misleading class is not overlooked.

Strategies to Address Class Imbalance

To handle the imbalance and improve detection of the misleading class, we can use a combination of **model level** and **data-level** approaches, as well as **robust evaluation** techniques:

1. Use Class Weights in the Loss Function

One effective approach is to **weight the loss function** to pay more attention to minority classes. By assigning a higher weight to the misleading class, the training loss for misclassifying a misleading example will be larger, forcing the model to give it more consideration. In PyTorch's cross-entropy loss, this is supported via a *weight* parameter. With Hugging Face's Trainer, we can implement this by **computing class weights** and then **using a custom Trainer** that applies a weighted loss.

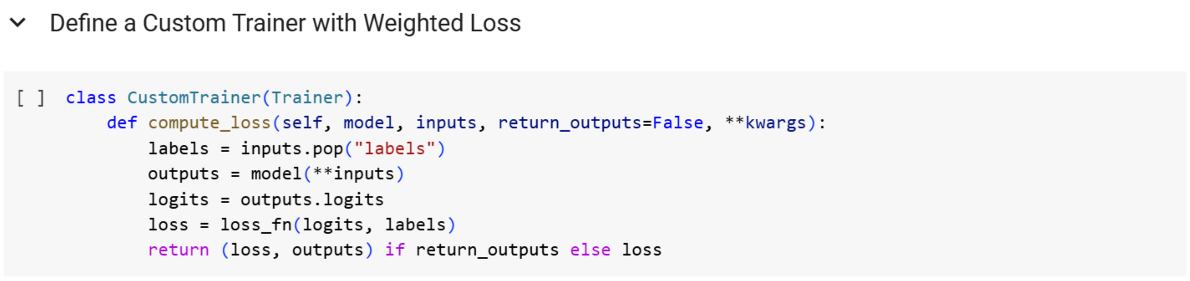


Compute Class Weights and Define Custom Loss

```python
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
import torch
from torch.nn import CrossEntropyLoss


all_labels = np.array(combined_train_tokenized["labels"]).astype(int)
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(all_labels), y=all_labels)
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(model.device)

loss_fn = CrossEntropyLoss(weight=class_weights_tensor)
```



Define a Custom Trainer with Weighted Loss

```python
[ ] class CustomTrainer(Trainer):
        def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
            labels = inputs.pop("labels")
            outputs = model(**inputs)
            logits = outputs.logits
            loss = loss_fn(logits, labels)
            return (loss, outputs) if return_outputs else loss
```

To effectively handle class imbalance, class weights are calculated based on the distribution of labels in the training data using scikit-learn's **compute_class_weight** with the "balanced" option. This ensures that less frequent classes, such as "misleading," are given higher importance during training. The computed weights are then converted into a PyTorch tensor and used to define a **weighted CrossEntropyLoss**, enabling the loss function to penalize misclassifications of minority classes more heavily.

To apply this weighted loss in training, a **custom Trainer** class was defined by subclassing Hugging Face's Trainer and overriding its compute_loss method. This modified trainer extracts labels from inputs, computes the model's logits, and calculates the loss using the predefined weighted loss function. This setup encourages the model to pay more attention to underrepresented classes, leading to more balanced and fair learning across all categories.

2. Use Appropriate Evaluation Metrics (Macro vs. Weighted F1)

When classes are imbalanced, don't rely only on overall accuracy, it can be misleading. Instead, evaluate with metrics that consider each class:

- **Macro F1:** This metric averages the F1-score of each class giving each class equal weight, regardless of size. It is a fair indicator for imbalanced data because it will be low if misleading-class performance is poor. In our case, improving the model's detection of misleading will greatly increase Macro F1, even if the overall accuracy changes little.

- **Weighted F1:** This is the average F1 weighted by support (frequency of each class). It's also useful it will be closer to accuracy for imbalanced data, since it down-weights the minority class. Ideally, we want to see improvements in both Macro F1 and Weighted F1 (the former indicating better minority class handling, the latter indicating overall performance).

You should compute and monitor both. With Hugging Face Trainer, you can supply a function. For example:

```python
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return {
        "eval_accuracy": accuracy_score(labels, preds),
        "eval_f1": f1_score(labels, preds, average="macro"),
        "macro_f1": f1_score(labels, preds, average="macro"),
        "weighted_f1": f1_score(labels, preds, average="weighted"),
    }
```

These metrics help track the impact of imbalance and improvements after applying techniques like class weighting.

3. Data-Level Techniques: Undersampling/Oversampling

In some cases, modifying the dataset directly can help address class imbalance:

- **Oversampling**: This involves duplicating samples from the minority class ("misleading") to increase its presence in the training data. To avoid overfitting, augmentation tools like *nlpaug* can be used to create slightly varied text versions instead of exact duplicates.

- **Undersampling**: This technique reduces the number of majority class examples by randomly removing them, helping balance the dataset. However, this can lead to information loss if not done carefully.

- **Combination**: A balanced approach often applies mild oversampling to minority classes and slight undersampling to majority classes. This can yield better results than using either method alone, especially in moderately imbalanced datasets.

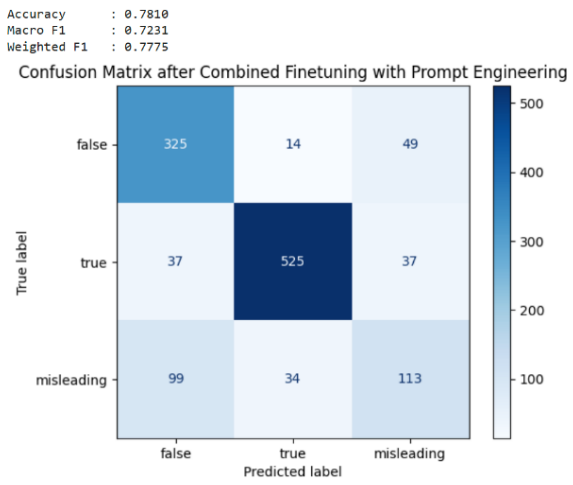
For example, in a dataset with a 52% true, 30% false, and 18% misleading class distribution, class weighting may be enough. But if the minority class becomes much smaller (below 5%), oversampling becomes more important.

Important: Sampling should only be applied to the training set. Applying it to validation or test sets can bias evaluation and lead to inaccurate metrics.

4. Examine Confusion Matrix and Per-Class Metrics

To truly understand model performance on each class, it's useful to look at the confusion matrix and detailed metrics after training:

- **Confusion Matrix:**



```
Accuracy     : 0.7810
Macro F1     : 0.7231
Weighted F1  : 0.7775
```

  A balanced model will have most counts on the diagonal (correct predictions) for each class.

- **Per-class Precision/Recall/F1:** A classification report breaks down precision, recall, and F1 for each class. This is crucial if you want to check that the misleading class has decent recall and precision and not near 0%.

Conclusion

Addressing class imbalance using weighted loss functions, appropriate metrics, and sampling techniques can significantly improve model performance on underrepresented classes like "misleading." These methods ensure more balanced performance, especially crucial in domains like health misinformation detection.

References

- Google for Developers. *Imbalanced datasets | Machine Learning*. https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets
- Stack Overflow. (2019, April 17). *Macro VS Micro VS Weighted VS Samples F1 Score*. https://stackoverflow.com/questions/55740220/macro-vs-micro-vs-weighted-vs-samples-f1-score
- Stack Overflow. (2022, February 2). *Using weights with transformers huggingface*. https://stackoverflow.com/questions/70979844/using-weights-with-transformers-huggingface
- Hugging Face Forums. (2020, September 14). *How can I use class_weights when training?* https://discuss.huggingface.co/t/how-can-i-use-class-weights-when-training/1067