A F_1-Based Losses

This section contains the formulas for the F_1-based loss functions used in the experiments. In the following, $\hat{y}_{t,k}$ denotes the model output for sample t corresponding to the k-th class, c_k, after application of an activation function. In the binary single-label setting, where $y_t \in \{0, 1\}$, the positive class is assumed to be denoted by label 1. A small stabilizer ϵ was added to the denominator of the F_1 formula, inspired by past research on F_1-based loss functions[11]; $\epsilon = 1e-05$ was used in the experiments.

Binary-F_1 Loss for Single-Label Settings The F_1 loss geared towards binary classification tasks aims to maximize the F_1 score of the positive class. Given a batch $\mathcal{B}$, the loss can be computed by the following formula.

$$\mathcal{L}_{F_1}(\mathcal{B}) = 1 - \frac{2\widehat{tp}}{2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon},$$

where

$$\begin{aligned}
\widehat{tp} &= \sum_{t=1}^{M} \hat{y}_{t,1} \times y_t, \\
\widehat{fp} &= \sum_{t=1}^{M} \hat{y}_{t,1} \times (1 - y_t), \\
\widehat{fn} &= \sum_{t=1}^{M} (1 - \hat{y}_{t,1}) \times y_t,
\end{aligned}$$

$y_t \in \{0, 1\}$ and $\hat{y}_{t,1}$ denotes the predicted probability for the positive class for sample t after application of the softmax.

Macro-F_1 Loss for Multi-Label Settings In the multi-label case, we use a macro-averaged F_1 loss, which averages the differentiable F_1 scores of all K classes.

$$\mathcal{L}_{F_{1_M}}(\mathcal{B}) = 1 - \frac{1}{K} \sum_{k=1}^{K} \frac{2\widehat{tp}_k}{2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon},$$

where

[11] The stabilizer was also used in https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d, accessed: 2022-08-11

$$\widehat{tp}_k = \sum_{t=1}^{M} \hat{y}_{t,k} \times y_{t,k},$$
$$\widehat{fp}_k = \sum_{t=1}^{M} \hat{y}_{t,k} \times (1 - y_{t,k}),$$
$$\widehat{fn}_k = \sum_{t=1}^{M} (1 - \hat{y}_{t,k}) \times y_{t,k},$$

$y_{t,k} \in \{0, 1\}$ and $\hat{y}_{t,k}$ denotes the predicted probability of class k for sample t after application of the sigmoid function.

B AGRA Multi-Label Setting

In the multi-label setting, the annotation for each data point is given by a binary vector of size K, where K is the number of classes. The AGRA multi-label setting allows to ignore individual entries of this vector for the update. For the following formulation of the multi-label variant of AGRA, the label-specific gradients are denoted by $\left(\nabla\widetilde{\mathcal{L}}\left(x_t, y_t\right)\right)_k$ and $\left(\nabla\widetilde{\mathcal{L}}_{com}\right)_k$, where $k \in \{1, ..., K\}$, and the corresponding similarity score is given by $sim^k_{y_t} = sim\left(\left(\nabla\widetilde{\mathcal{L}}\left(x_t, y_t\right)\right)_k, \left(\nabla\widetilde{\mathcal{L}}_{com}\right)_k\right)$. If a particular label $y_{t,k}$ should not be considered for the update, due to $sim^k_{y_t}$ being non-positive, it is replaced with the label i. For the model update, we use a masked version of the BCE loss that ensures that the replaced labels do not influence the weight update. Namely,

$$\widetilde{\mathcal{L}}_{BCE}(\mathcal{B}) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{\widetilde{M}_k} \sum_{t=1}^{M} -[\log(\hat{y}_{t,k}) y_{t,k} + \log(1 - \hat{y}_{t,k})(1 - y_{t,k})] \mathbb{1}(\widetilde{y}_{t,k} \neq i)$$

where $\widetilde{M}_k$ denotes the number of retained labels for class k and $\widetilde{y}$ denotes the masked label vector. Pseudocode for the AGRA multi-label setting can be found in Algorithm 2.

Algorithm 2: AGRA Algorithm for Multi-Label Datasets

Input: training set $\mathcal{X}$, initial model $f(\cdot;\theta)$, number of epochs E, batch size M, number of classes K, label assigned to ignored entries i
Output: trained model $f(\cdot;\theta^*)$
for *epoch = 1,..., E* **do**
  for *batch* $\mathcal{B}$ **do**
    Sample a comparison batch $\widetilde{\mathcal{B}}$, $\widetilde{\mathcal{B}} \subset \mathcal{X}$, $|\widetilde{\mathcal{B}}| = M$
    Compute $\nabla\widetilde{\mathcal{L}}_{com}$ on $\widetilde{\mathcal{B}}$
    for $(x_t, y_t) \in \mathcal{B}$ **do**
      Compute $\nabla\widetilde{\mathcal{L}}(x_t, y_t)$
      Set up corrected label vector $\widetilde{y}_t \leftarrow y_t$
      for *k=1, ..., K* **do**
        $sim^k_{y_t} = sim\left(\left(\nabla\widetilde{\mathcal{L}}(x_t, y_t)\right)_k, \left(\nabla\widetilde{\mathcal{L}}_{com}\right)_k\right)$ (Eq.1)
        if $sim^k_{y_t} \leq 0$ **then**
          $\widetilde{y}_{t,k} \leftarrow i$
      $\mathcal{B} \leftarrow \mathcal{B} \setminus \{(x_t, y_t)\} \cup \{(x_t, \widetilde{y}_t)\}$
    $\theta \leftarrow Optim(\theta, \mathcal{B}, \mathcal{L})$

C Dataset Preprocessing & Statistics

We used already preprocessed versions of text datasets in the Wrench framework [51]. For CheXpert, where each class corresponds to one pathology that might be present in a training sample, we needed to additionally adapt the labels so that they can be used for our experiments. In the original dataset, the label assigned to each class equals either -1 (uncertain), blank (not mentioned), 0 (negative), or 1 (positive). We binarized the label vector by mapping -1 to 1 (potentially increasing the noise ratio, but keeping more positive training samples), and blank to 0. The assignment of the value 0 to the blank labels is somewhat intuitive: not every radiographic report will mention all 12 pathologies, and if an observation is not even mentioned in the study report, there likely are no indications for it in the corresponding images [15,23]. Among the possible ways of handling the uncertainty labels (turning them to either 0 or 1, ignoring them, or keeping them as a separate class), we decided to choose the reassignment to label 1 as it was the best performing method for most of the classes in [23]. We focus on predicting the 12 pathologies in the CheXpert dataset, omitting the additional "Support Devices" and "No Finding" labels.

Table 4 provides information about the number of samples belonging to each class for all datasets used in the experiments. Note that in the multi-class multi-label CheXpert dataset, one sample can belong to multiple classes.

Dataset	Class	%Samples
YouTube	SPAM	52.3%
	HAM	47.7%
SMS	SPAM	63.8%
	HAM	36.2%
TREC	DESC	56.6%
	ENTY	8.0%
	HUM	12.8%
	ABBR	1.0%
	LOC	8.0%
	NUM	13.6%
Yorùbá	Africa	6.1%
	Entertainment	25.2%
	Health	6.5%
	Nigeria	10.3%
	Politics	26.2%
	Sport	6.8%
	World	18.9%
Hausa	Africa	19.6%
	Health	10.1%
	Nigeria	8.6%
	Politics	7.9%
	World	53.7%
CheXpert	Enlarged Cardiomediastinum	10.4%
	Cardiomegaly	15.8%
	Lung Opacity	49.8%
	Lung Lesion	4.8%
	Edema	29.1%
	Consolidation	19.1%
	Pneumonia	11.1%
	Atelectasis	30.1%
	Pneumothorax	10.1%
	Pleural Effusion	43.7%
	Pleural Other	2, 8%
	Fracture	4.3%

Table 4: Percentage of samples belonging to each class in all data sets used for the experiments. Note that CheXpert dataset in designed for a multi-label classification, meaning more than one class can be assigned to one sample.

D Data Encoding

For the text datasets, we used feature-based TF-IDF vector representations. The samples were encoded with the `TfidfVectorizer` tool available in Sklearn library[12].

For the CheXpert dataset, we used a fine-tuned convolutional neural network (CNN) as a feature extractor, following Giacomello et al. [17]. In our experiments, the data samples were encoded using a fine-tuned EfficientNet-B0 [45]. The `torchvision` implementation [22] of EfficientNet-B0 used for our experiments was pre-trained on ImageNet. Despite the fact that the images contained in ImageNet greatly differ from chest radiographs, the pre-training still gives good intialization, as was shown by Ke et al. [25]. The chest radiographs were rescaled to a size of 224x224 and normalized with the mean and standard deviation from ImageNet. Then, the EfficientNet was fine-tuned on the training data for two epochs using the Adam optimizer with learning rate 0.0001 and batch size 16. Once the fine-tuning was completed, 1280-dimensional feature vectors for each image were extracted from the penultimate layer of the network and used as the data representations.

For CIFAR-10, we fine-tune a pre-trained ResNet-50 for 100 epochs with learning rate 0.001, batch size 64 and no weight decay and retrieve emeddings from the penultimate layer.

[12] https://scikit-learn.org/stable/

E The Hyper-Parameters and Search Space

The hyper-parameters for our algorithm and baselines were retrieved with a grid search on the validation sets; each trial was performed 3 times with different initialization. The search space is provided in Table 6, and the exact selected values for our method AGRA can be found in Table 5. All AGRA results presented in the paper are reproducible with the seed value 0.

The experiments are performed on a single Tesla V100 GPU on Nvidia DGX-1. The parameter grid search takes between 24 and 96 hours; the model run with the selected parameters takes between 1 and 5 minutes depending on parameter values and dataset size.

Dataset	Learning Rate η	Batch Size M	Weight Decay λ
YouTube	$1e-2$	32	$1e-3$
SMS	$1e-3$	128	$1e-3$
TREC	$1e-1$	32	$1e-4$
Yorùbá	$1e-1$	512	$1e-4$
Hausa	$1e-1$	512	$1e-4$
CIFAR-10	$1e-3$	512	$1e-4$
CheXpert	$1e-3$	128	$1e-3$

Table 5: Selected hyper-parameters with grid search for the best AGRA configurations, which are:

- for YouTube: $F1$ comparison loss, no weighted sampling, no alternative label,
- for SMS: CE comparison loss, weighted sampling, no alternative label,
- for TREC: $F1$ comparison loss, weighted sampling,
- for Yorùbá: $F1$ comparison loss, weighted sampling,
- for Hausa: $F1$ comparison loss, weighted sampling,
- for CIFAR-10: $F1$ comparison loss, weighted sampling,
- for CheXpert: $F1$ comparison loss, weighted sampling not implemented.

For CheXpert, the hyperparameters were fixed to learning rate $1e-3$, weight decay $1e-3$ and batch size 128 without grid search for AGRA and Cleanlab due to energy considerations and resource constraints. The Cleanlab hyperparameter n_folds was set to 2 for CheXpert. For CORES^2, we choose $\beta = 0.2 \cdot K$, where K is the number of classes, as suggested by [10].

Method	Hyper-parameter	Search Space
AGRA	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 128, 512
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
No denoising	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 64, 128
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
DP	DP learning rate	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	DP batch size	32, 64, 128
	DP num epochs	5, 10, 50, 100, 200
	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 64, 128
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
MeTal	MeTal learning rate	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	MeTal weight decay	32, 64, 128
	MeTal num epochs	5, 10, 50, 100, 200
	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 64, 128
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
FlyingSquid	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 64, 128
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
Cleanlab	CL n_folds	2, 5, 7, 9, 12
	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 128, 512
	weight decay λ	$1e-05$, $1e-04$, $1e-03$
CORES^2	learning rate η	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$
	batch size M	32, 128, 512
	weight decay λ	$1e-05$, $1e-04$, $1e-03$, $1e-02$, $1e-01$

Table 6: Hyper-parameters and search space used for a grid search. The search spaces for DP, MeTaL, and FS methods were inherited from Wrench [51]. The search spaces for Cleanlab and AGRA were selected empirically.