Multi-label Classification through Single-Objective and Multi-Objective Optimizations

Implementation Track
4 Team Members

Abstract

The F_β-measure is a popular performance measure for multi-label classification (MLC) problems, where multiple labels can be active simultaneously . In particular, the F_β-measure handles the inter-label dependencies by balancing precision and recall, both of which are important in evaluating the performance of a classifier (specially when the dataset has label-imbalance). Since the domain of F_β-measure is discrete and also exponentially-large with respect to (w.r.t) the number of labels, directly optimizing it is computationally hard. Recently, [1] has proposed a *single surrogate-risk minimization* framework that can perform almost as good as the *Bayes-optimal classifier (BOC)* for the F_β-measure. However, the computational complexity of the resulting learning-algorithm, namely *Zhang's Algorithm (ZA)*, is still high. On the other hand, [2] has viewed the MLC problem as a multi-task learning (MTL) problem and proposed a learning-algorithm that tries to minimize multiple surrogate-risks together (one for each label). In other words, the learning-algorithm of [2], namely *Sener's Algorithm (SA)*, handles the inter-label dependencies by considering one classifier for each label and then training all of them together through *multi-objective optimization*. This is in contrast to the *Binary Relevance (BR)* method, where the classifier for each label is trained separately. An interesting feature of SA is that it offers a much more flexible (tunable) computational complexity compared to ZA. Given this attractive feature, we wish to find out whether SA can be used as a reliable heuristic classifier for the F_β-measure. This report summarizes our experiments and their results performed in this regard.

1 Introduction

In supervised learning, multi-label classification (MLC) is a type of classification problem in which multiple labels (or 'tags') can be (simultaneously) assigned to the same instance. Consider, for example, a scene image where more than one tags such as `tree`, `cloud`, `sun` are active simultaneously. Such problems arise in prominent applications, including semantic scene understanding [3], multi-category text classification [4], gene function prediction in bioinformatics [5] etc. In principle, any MLC problem can be cast as a multi-class classification (MCC) problem. However, the resulting MCC problem cannot be solved efficiently when the number of tags is even moderately large. This hurdle arises because the label-space of the MLC problem is exponential in the number of tags, a phenomenon generally referred to as the "*curse-of-dimensionality*" (COD).

A number of works have appeared in an effort to efficiently solve the MLC problem w.r.t different performance metrics, such as accuracy, selectivity, G-mean, F_β-measure, balanced-accuracy, one-error, coverage, and ranking-loss [6, 7]. In this work, we focus on efficiently solving the MLC problem w.r.t the F_β-measure. To this end, we implement three methods namely ZA, SA, and BR. Of these three algorithms, we respectively use ZA and BR as baselines for maximal and minimal F_β performance. We attempt to gauge how SA performs in comparison to these two baselines.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Solving the MLC problem w.r.t the F_β -measure is both important and challenging. Firstly, as mentioned earlier, the F_β-measure has been observed to be reliable in evaluating the performance of multi-label classifiers, [8]. Secondly, besides the COD issue that is inherent in any MLC problem, solving it w.r.t the F_β-measure presents an optimization challenge also, namely, the F_β-measure is defined on a discrete and exponentially large domain.

This report summarizes our project's work and is organized as follows. In Section 2, we set up our notation and formulate the general MLC problem. Section 3 provides a gist of important related works. In Section 4, we provide the descriptions of ZA and SA algorithms leaving the auxiliary details for Appendices A and B. Section 5 describes our experiments and the qualitative assessment of the algorithms based on the observed results. Supplementary implementation details and results are provided in Appendices E, G and F. Finally, we present our concluding remarks in Section 6.

2 Problem Setup

In this section, we formalize the MLC problem in the supervised learning framework.

MLC Problem: Let $\mathcal{X}$ denote the *feature-space*, and let $\mathcal{L} = [s] \coloneqq \{1, 2, \ldots, s\}$ be the set of all labels that can be associated with each such instance. We are provided with training examples that constitute the *training-set* $S = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$. Here, n denotes the size of the training set, each $\mathbf{x}_i$ belongs to $\mathcal{X}$, and each $\mathbf{y}_i$ is a binary-vector of length s that reveals which of the tags are active and/or inactive in the instance $\mathbf{x}_i$. Specifically for each $(i, j) \in [n] \times [s]$, $y_{ij} = 1$ means that tag-j is active in instance $\mathbf{x}_i$ and $y_{ij} = 0$ indicates otherwise. The goal of the MLC problem is to devise a *learning-algorithm* that uses the training-set S to output a (multi-label) classifier of the form $h_S : \mathcal{X} \to \{0, 1\}^s$. Here we note that the output label-space of the (multi-label) classifier h_S is $\{0, 1\}^s$, which is of size 2^s. Thus, solving the MLC problem as an MCC problem is of no avail when the number of tags is even moderately large, say 40.

MLC Problem w.r.t F_β-measure: For a binary vector $\mathbf{y} \in \{0, 1\}^s$, let us denote the number of 1's in $\mathbf{y}$ by $\|\mathbf{y}\|_1$, i.e., $\|\mathbf{y}\|_1 = \sum_{j=1}^{s} y_j$. Then, given a true labeling $\mathbf{y} \in \{0, 1\}^s$ and a predicted labeling $\widehat{\mathbf{y}} \in \{0, 1\}^s$, the F_β-measure, denoted by $F_\beta : \{0, 1\}^s \times \{0, 1\}^s \to [0, 1]$, is the weighted harmonic mean of the *recall* and *precision* computed on the pair $(\mathbf{y}, \widehat{\mathbf{y}})$, i.e.,

$$F_\beta(\mathbf{y}, \widehat{\mathbf{y}}) = \left(\left(\frac{\beta^2}{1+\beta^2} \right) \frac{1}{\mathrm{rec}(\mathbf{y}, \widehat{\mathbf{y}})} + \left(\frac{1}{1+\beta^2} \right) \frac{1}{\mathrm{prec}(\mathbf{y}, \widehat{\mathbf{y}})} \right)^{-1} = \frac{\left(1+\beta^2\right) \sum_{j=1}^{s} y_j \widehat{y}_j}{\beta^2 \|\mathbf{y}\|_1 + \|\widehat{\mathbf{y}}\|_1}. \quad (1)$$

Here, $\beta > 0$ is the weight for the harmonic-mean, $\mathrm{rec}(\mathbf{y}, \widehat{\mathbf{y}}) = (\sum_{j=1}^{s} y_j \widehat{y}_j)/\|\mathbf{y}\|_1$ denotes *recall* (fraction of active tags that are predicted correctly), and $\mathrm{prec}(\mathbf{y}, \widehat{\mathbf{y}}) = (\sum_{j=1}^{s} y_j \widehat{y}_j)/\|\widehat{\mathbf{y}}\|_1$ denotes *precision* (fraction of tags predicted as active that are actually so). In defining, F_β, we use the convention $0/0 = 1$ so that when $\mathbf{y} = \widehat{\mathbf{y}} = \mathbf{0}$, we have $F_\beta(\mathbf{0}, \mathbf{0}) = 1$.

Since both recall and precision assume values between 0 and 1, and F_β is the weighted harmonic mean of the two, it is clear that $0 \leqslant F_\beta(\mathbf{y}, \widehat{\mathbf{y}}) \leqslant 1$. Therefore, equivalently, one can define F_β-loss, $L^{F_\beta}(\mathbf{y}, \widehat{\mathbf{y}}) = 1 - F_\beta(\mathbf{y}, \widehat{\mathbf{y}})$. We may note that F_β (and consequently L^{F_β}) is defined over a discrete and exponentially large space (of size 2^{2s}). Therefore, directly minimizing its corresponding empirical-risk function, $\frac{1}{n} \sum_{i=1}^{n} L^{F_\beta}(\mathbf{y}_i, h_S(\mathbf{x}_i))$, is computationally hard. Consequently, one must settle for some approximation. One baseline approach is the BR method, which simply ignores the inter-label dependencies and trains a binary classifier for each label separately.

3 Related Work

There has been extensive work on the MLC problem itself and on MTL problems. For a thorough survey, we suggest [7, 9, 10] (for MLC) and [11] (for MTL).

In general, for the MLC problem, there are two main algorithmic approaches, namely, *method-adaptation* and *ensemble-of-classifiers* [12]. The method-adaptation approach is based on adapting existent machine learning algorithms for the MLC problem. For example, casting MLC as an MCC problem is one method-adaptation approach. Importantly, both ZA and SA algorithms are examples of method-adaptation approaches. On the other hand, in the ensemble-of-classifiers approach, different classifiers are combined in some heuristic way (such as weighted-averaging) to produce a multi-label

prediction (e.g., Random-K labels algorithm [13]). Such approaches are usually computationally-intensive and lack theoretical guarantees.

In the F_β line of works, [8] was the first work to describe the form of the Bayes-optimal classifier (BOC) for the F_β-measure. Specifically, [8] showed that in an s-label MLC problem, given an observed feature-vector $\mathbf{x} \in \mathcal{X}$, one can produce a Bayes-optimal prediction by knowing only s^2+1 conditional probabilities (instead of knowing all the 2^s conditional probabilties $\mathbb{P}(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x})$ for all $\mathbf{y} \in \{0,1\}^s$). These s^2+1 conditional probabilities are $\mathbb{P}(\mathbf{Y}=\mathbf{0}|\mathbf{X}=\mathbf{x})$ and $\mathbb{P}(Y_j=1, \|\mathbf{Y}\|=k|\mathbf{X}=\mathbf{x})$ for $j,k \in [s]$. [8] also shows that if these s^2+1 conditional probabilities are known, one can use them in a $O(s^3)$-time algorithm to generate the Bayes-optimal prediction. Following these two results of [8], [14] attempts to estimate these s^2+1 conditional probabilities by solving s MCC problems each with $s+1$ classes. Once these probabilities are estimated, [14] uses them in the $O(s^3)$-time algorithm of [8] to generate the final multi-label prediction. This overall algorithm of [14] is known as *Exact F-measure plug-in (EFP)* algorithm. However, one issue is that it solves s MCC problems separately leading to a high complexity in hyper-parameter tuning.

Relatively fewer works have considered MLC problem from an MTL viewpoint. However, the MTL literature itself is ripe with a number of works. In MTL, the learning is usually performed via hard or soft parameter sharing. In *hard parameter sharing* [15, 16], a subset of the parameters are shared between tasks while other parameters are task-specific. In *soft parameter sharing* [17, 18], all parameters are task-specific but they are jointly constrained via some conditions. As mentioned in 4.2, [2] uses hard parameter sharing which is also the norm in most MTL applications. An additional novelty that [2] brings is the idea of multi-objective optimization; interested readers can refer to [19] and [20] for surveys on multi-objective optimization methods.

4 Methods

In this section, we provide the background materials as well as the complete descriptions of [1] and [2]. For readability, we present the results of both [1] and [2] in a unifying notation.

4.1 Zhang's Algorithm (ZA) - Single-Objective Minimization

As mentioned earlier, minimizing L^{F_β} is computationally hard. A well-known approach to this issue is to instead minimize a w-dimensional[1] convex surrogate-loss $\psi : \{0,1\}^s \times \mathbb{R}^w \to \mathbb{R}_{\geqslant 0}$ by learning a w-dimensional score-function $f_S : \mathcal{X} \to \mathbb{R}^w$. Importantly, the dimension w is chosen large enough to ensure an approximately-accurate MLC classifier but also small enough so that the learning process is computationally-efficient. For example, if we cast our MLC problem for L^{F_β} as an MCC problem with 2^s classes, we can choose $w=2^s$ to ensure that each possible class in $\{0,1\}^s$ is given a separate score. This of course is computationally infeasible. An important question then is how small can w be made while still ensuring a reasonably accurate learning. If w is chosen strictly smaller than 2^s, we must also know how to decode the low-dimensional score $f_S(\mathbf{x})$ into the final prediction $h_S(\mathbf{x})$ which lives in the larger space $\{0,1\}^s$. We generally denote the mapping that does this conversion of $f_S(\mathbf{x})$ into $h_S(\mathbf{x})$ by $\texttt{decode}: \mathbb{R}^w \to \{0,1\}^s$. Thus, in such approaches, the resulting MLC classifier is specified as $h_S(\mathbf{x}) = \texttt{decode}(f_S^*(\mathbf{x}))$, where f_S^* minimizes the empirical ψ-surrogate risk,

$$\widehat{R}_\psi(f_S) = \frac{1}{n}\sum_{i=1}^{n} \psi\left(\mathbf{y}_i\, f_S(\mathbf{x}_i)\right). \tag{2}$$

In practice, f_S^* is found by restricting it to a certain (parametrized) function-class $\mathcal{F}$. Since f_S^* is found by minimizing $\widehat{R}_\psi$, which in turn depends on ψ, one can specify the learning-algorithm completely by the pair (ψ,decode). The key ingenuity in such an approach is to design the pair (ψ, decode) such that, in the limit of large training-data ($n \to \infty$), the classifier h_S learnt by using (ψ, decode) is approximately as good as the Bayes-optimal classifier for L^{F_β}. This property of the pair (ψ, decode) is called L^{F_β}*-calibration* (see Appendix A.1 for a formal definition).

Now that the overall approach is clear, we reveal that [1] has found a family of (s^2+1)-dimensional (convex) surrogate losses, denoted by $\Psi = \{\psi : \{0,1\}^s \times \mathbb{R}^{s^2+1} \to \mathbb{R}_{\geqslant 0}\}$, and a decode mapping, namely "decode-z", such that the pair (ψ, decode-z) is L^{F_β}-calibrated for any $\psi \in \Psi$. Each

[1] The dimension of a surrogate loss is defined as the dimension of its second argument.

member ψ of the family Ψ is derived from some *strictly-proper composite binary loss (CBL)* which also has a corresponding (invertible) link function $\gamma : [0,1] \to \mathbb{R}$ (that converts a class-probability estimate into a score). The exact definition of a strictly-proper CBL and its corresponding link-function is provided in Appendix A.2. Intuitively, a strictly-proper CBL is one whose minimization leads to accurate class probability estimates in a binary class probability estimation (CPE) problem. We note that the well-known logistic-loss is a strictly-proper CBL (see (7)).

The key idea behind the above result of [1] is that even though the domain of L^{F_β} is exponentially large, its range is still small. That is, if we view the L^{F_β} function as $2^s \times 2^s$-matrix $\mathbf{L}^{F_\beta}$ with $\mathbf{L}^{F_\beta}_{\mathbf{y},\widehat{\mathbf{y}}} = L^{F_\beta}(\mathbf{y}, \widehat{\mathbf{y}})$, then the rank of $\mathbf{L}^{F_\beta}$ is at most $s^2 + 1$. This fact along with its proof is provided in Proposition 1 in Appendix A.3. Now, viewing the s-label MLC problem as a MCC problem with 2^s classes and the low-rank of $\mathbf{L}^{F_\beta}$, ZA uses the already existing result of [21]. Specifically, [21] shows that a MCC problem with a loss-matrix of rank-w can be converted into a $(\psi, \texttt{decode})$-type learning-algorithm in which the dimension of ψ is the same as the rank w. This result of [21] is formally stated in Theorem 1 in Appendix A.4 and directly applying it to $\mathbf{L}^{F_\beta}$ with the binary logistic-loss ϕ_{log} and its link-function γ_{log} gives the below forms for ψ and `decode-z`.

$$\begin{aligned}\psi_{log}(\mathbf{y}, \mathbf{u}) = a_0(\mathbf{y}) \cdot \phi_{log}\left(+1, u_0\right) + \left(1 - a_0(\mathbf{y})\right) \cdot \phi_{log}\left(-1, u_0\right) \\ + \sum_{j=1}^{s} \sum_{k=1}^{s} a_{j,k}(\mathbf{y}) \cdot \phi_{log}\left(+1, u_{j,k}\right) + \left(1 - a_{j,k}(\mathbf{y})\right) \cdot \phi_{log}\left(-1, u_{j,k}\right), \end{aligned} \tag{3}$$

$$\texttt{decode-z}(\mathbf{u}) \in \operatorname*{arg\,min}_{\widehat{\mathbf{y}} \in \{0,1\}^s} \; b_0(\widehat{\mathbf{y}}) \cdot \gamma_{log}^{-1}(u_0) + \sum_{j=1}^{s} \sum_{k=1}^{s} b_{j,k}(\widehat{\mathbf{y}}) \cdot \gamma_{log}^{-1}(u_{j,k}). \tag{4}$$

Here, importantly, we note that the vector $\mathbf{u}$ represents the score-function $f_S(\mathbf{x})$ and is indexed as $\mathbf{u} = (u_0, u_{1,1}, \ldots, u_{1,s}, \ldots, u_{s,1}, \ldots, u_{s,s})$, where u_0 is the score assigned to the class corresponding to the event $\{\mathbf{Y} = \mathbf{0}\}$ and $u_{j,k}$ is the score for the class corresponding to the event $\{Y_j = 1, \|\mathbf{Y}\|_1 = k\}$. Furthermore, the functions $a_0(\mathbf{y}), a_{j,k}(\mathbf{y}), b_0(\widehat{\mathbf{y}}), b_{j,k}(\widehat{\mathbf{y}}), \phi_{log}, \gamma_{log}$ are given as follows:

$$a_0(\mathbf{y}) = \mathbf{1}\left(\|\mathbf{y}\|_1 = 0\right); \qquad a_{j,k}(\mathbf{y}) = \mathbf{1}\left(\|\mathbf{y}\|_1 = k\right) \cdot y_j; \tag{5}$$

$$b_0(\widehat{\mathbf{y}}) = -\mathbf{1}\left(\|\widehat{\mathbf{y}}\|_1 = 0\right); \qquad b_{j,k}(\widehat{\mathbf{y}}) = -\frac{\left(1 + \beta^2\right) \cdot \widehat{y}_j}{\beta^2 k + \|\widehat{\mathbf{y}}\|_1}; \tag{6}$$

$$\phi_{\log}(y, u) = \ln(1 + e^{yu}); \qquad \gamma_{\log}(p) = \ln\left(\frac{p}{1-p}\right). \tag{7}$$

Looking at (4), we note that implementing `decode-z` requires iterating over 2^s candidate predictions. ZA circumvents this issue by using the already existing result of [14] which, as mentioned earlier, can decode the above $s^2 + 1$ probability estimates into the final prediction $\texttt{decode-z}(f_S(\mathbf{x}))$ in $O(s^3)$-time. For details of this $O(s^3)$-time implementation of `decode-z`, please see Appendix A.5.

Importantly, since the co-domain of f_S has dimension $s^2 + 1$, the number of parameters needed to search over a richer function-class $\mathcal{F}$ is high. For example, the number of parameters needed to learn just a linear score-function is $O(s^2)$. This completes our description of ZA whose pseudocode is presented in Appendix C.

4.2 Sener's Algorithm (SA) - Multi-Objective Minimization

In contrast to ZA, which uses L^{F_β} to incorporate inter-label dependencies, SA considers s separate binary classifiers (one for each label) and trains them together through *multi-objective optimization* and *hard parameter sharing*. Specifically, for each $j \in [s]$, label-j's classifier learns a parameterized score-function $f_S^j(\cdot|\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) : \mathcal{X} \to \mathbb{R}^2$. (The co-domain of f_S^j can be $\mathbb{R}^1$ also if one-hot encoding is not to be used). Here, $\boldsymbol{\theta}^{sh}$ denotes the vector of all parameters that are shared across the different labels, and $\boldsymbol{\theta}^j$ denotes parameters for label-j only. This score-function is then used in label-j's (convex) surrogate-loss $\phi^j : \{0,1\} \times \mathbb{R}^2 \to \mathbb{R}_{\geqslant 0}$ (such as the cross-entropy/logistic loss), which yields the empirical ϕ^j-surrogate-risk, $\widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) = \frac{1}{n}\sum_{i=1}^{n} \phi^j(y_{ij}, f_S^j(\mathbf{x}_i|\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j))$. With this label-specific setup in place, the MLC problem is then cast as a multi-objective minimization of the form

$$\min_{\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s} \widehat{\mathbf{R}}_{\boldsymbol{\phi}}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s\right) = \min_{\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s} \left(\widehat{R}^1_{\phi^1}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1), \ldots, \widehat{R}^s_{\phi^s}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^s)\right) \tag{8}$$

Solving optimization problems with multiple objective functions (that may possibly conflict with each other) has been studied in multi-optimization theory, where instead of the usual notions of local/global extrema, the concept of *Pareto-optimality* is used.

Definition 1. *(Pareto optimality) A solution $\bar{\boldsymbol{\theta}} = (\bar{\boldsymbol{\theta}}^{sh}, \bar{\boldsymbol{\theta}}^1, \ldots, \bar{\boldsymbol{\theta}}^s)$ to (8) is called Pareto-optimal if there exists no solution that dominates $\bar{\boldsymbol{\theta}}$. That is, there does not exist any $\boldsymbol{\theta} = (\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s)$ such that $\widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) \leqslant \widehat{R}^j_{\phi^j}(\bar{\boldsymbol{\theta}}^{sh}, \bar{\boldsymbol{\theta}}^j)$ for every $j \in [s]$, and $\widehat{\mathbf{R}}_{\boldsymbol{\phi}}(\boldsymbol{\theta}) \neq \widehat{\mathbf{R}}_{\boldsymbol{\phi}}(\bar{\boldsymbol{\theta}})$.*

Intuitively speaking, a Pareto-optimal solution to (8) is one in which, if we further minimize a certain objective function, it increases the value of some other objective function. To find a Pareto-optimal solution for (8), a theorem similar to the Karush-Kuhn-Tucker (*KKT*) theorem has been derived which states that if $\boldsymbol{\theta} = (\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s)$ is a Pareto-optimal solution to (8), then we have: *i*) $\nabla_{\boldsymbol{\theta}^j} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = \mathbf{0}$ for every $j \in [s]$; and *ii*) there exist $\alpha^1, \alpha^2 \cdots, \alpha^s \geqslant 0$ s.t. $\sum_{j=1}^s \alpha^j = 1$ and $\sum_{j=1}^s \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = \mathbf{0}$. Although these conditions are necessary, in practice, one usually hopes that they are also sufficient. Thus, we wish to find $\boldsymbol{\theta}$ that satisfies the above conditions. We note that for the label-specific parameters $\{\boldsymbol{\theta}^j\}_{j\in[s]}$, the condition for optimality is the familiar zero-gradient condition. However, the optimality condition for $\boldsymbol{\theta}^{sh}$ cannot be solved using the gradient-descent scheme directly. [22] has handled this issue by showing that if we consider the optimization problem,

$$\text{minimize } \| \sum_{j\in[s]} \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) \|^2 \text{ s.t. } \sum_{j\in[s]} \alpha^j = 1 \text{ and } \alpha^j \geqslant 0 \text{ for every } j \in [s], \quad \text{(MGDA)}$$

then its solution $\{\alpha^j\}_{j\in[s]}$ either satisfies *KKT* condition 1 (mentioned above) or it gives a direction for $\boldsymbol{\theta}^{sh}$, namely $\sum_{j=1}^s \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)$, along which each objective function $(\widehat{R}^j_{\phi^j})$ decreases. Here, $\|\cdot\|$ represents the Euclidean norm.

With the aforementioned setup, we have the following outline for solving (8): *i*) Perform gradient descent on each label-specific parameter $\boldsymbol{\theta}^j$; *ii*) Solve the optimization problem (MGDA) to find a descent direction for $\boldsymbol{\theta}^{sh}$; *iii*) Use the descent direction found in step-*ii*) to perform gradient descent over $\boldsymbol{\theta}^{sh}$; *iv*) Repeat *i-iii*) till convergence or maximum iteration count is reached. This outline is formally presented in Algorithm 4. Amongst steps *i-iv*), step *ii*) is the novel step and requires solving the optimization problem (MGDA) which we note is a convex quadratic program with linear constraints, and can be solved by the well-known Frank-Wolfe method (see Appendix B.1).

The main computational issue so far is that in deep neural network (DNN) implementations, the dimension of $\boldsymbol{\theta}^{sh}$ is very high. The algorithm outline we have described so far requires computing the gradient $\nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right)$ for every label-j, which means that in the training process, we have to perform s backward propagations. Considering the fact that backward propagation is computationally expensive in a DNN, the training time will scale with the number of labels. Addressing this issue is another novelty of [2] which shows that (under a reasonable assumption), we can compute all the descent directions, $\{\sum_{j=1}^s \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)\}_{j\in[s]}$ in a single backward pass. The key idea behind this is the simple chain-rule of differentiation. For most DNN models, the $\boldsymbol{\theta}^{sh}$ parameter is used to extract the most meaningful representation that is shared by all the labels. In such models, for every label-j, and every training-input pair $(\mathbf{x}, \mathbf{y})$ we can write, $f^j_S(\mathbf{x}|\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = f^j_S(g(\mathbf{x}|\boldsymbol{\theta}^{sh})|\boldsymbol{\theta}^j)$, where the common representation of $\mathbf{x}$ is denoted by $g(\mathbf{x}|\boldsymbol{\theta}^{sh})$. This, in turn, implies that the empirical risk function $\widehat{R}^j_{\phi^j}$ depends on $\boldsymbol{\theta}^{sh}$ only through the common representations $\{g(\mathbf{x}_i|\boldsymbol{\theta}^{sh})\}_{i\in[n]}$. If we denote the common representation for each input $\mathbf{x}_i$ by $\mathbf{z}_i = g(\mathbf{x}_i|\boldsymbol{\theta}^{sh})$, and let $\mathbf{Z}$ be the vector $(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$, then, we can write $\widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = \widehat{R}^j_{\phi^j}(\mathbf{Z}(\boldsymbol{\theta}^{sh}), \boldsymbol{\theta}^j)$. By the chain-rule of differentiation, this gives

$$\sum_{j\in[s]} \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) = \left(\partial \mathbf{Z}/\partial \boldsymbol{\theta}^{sh}\right) \sum_{j\in[s]} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right),$$

which, by the sub-multiplicative property of matrix-norms, gives

$$\| \sum_{j\in[s]} \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) \|^2 \leqslant \|\partial \mathbf{Z}/\partial \boldsymbol{\theta}^{sh}\|^2 \| \sum_{j\in[s]} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) \|^2. \tag{9}$$

Thus, to approximately solve (MGDA), we may minimize its upper-bound given by the right-hand-side of 9. This has two advantages. Firstly, the gradients $\nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right)$ can be computed in a single backward pass. Secondly since $\|\partial \mathbf{Z}/\partial \boldsymbol{\theta}^{sh}\|^2$ does not depend on α^j's, it can be removed from the objective function of (MGDA). This gives us the below-mentioned approximate version of (MGDA), which we refer to as *Multiple Gradient Descent Algorithm - Upper-Bound* (MGDA-UB).

$$\text{minimize } \| \sum_{j\in[s]} \alpha^j \nabla_{\boldsymbol{Z}} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right) \|^2 \text{ s.t. } \sum_{j\in[s]} \alpha^j = 1,\ \alpha^j \geqslant 0 \text{ for every } j \in [s]. \quad \text{(MGDA-UB)}$$

Briefly speaking, (MGDA-UB) is similar to the (MGDA), except now the gradient of $\widehat{R}^j_{\phi_j}$ is computed w.r.t $\mathbf{Z}(\theta^{sh})$ instead of $\boldsymbol{\theta}^{sh}$. Thus, similar to the case of (MGDA), (MGDA-UB) can be solved by the Frank-Wolfe method (Appendix B.1). An important question then is, does solving (MGDA-UB) produce a Pareto-optimal solution. [2] shows that, under a reasonable assumption (namely full-rank of $\partial \boldsymbol{Z}/\partial \boldsymbol{\theta}^{sh}$), this is indeed true. This fact is formally presented in Theorem 2 in Appendix B.2. This completes our description of Sener's Algorithm whose pseudocode is presented in Appendix D.

5 Experiments and Results

To evaluate SA in comparison to ZA and BR, we conducted experiments on three distinct datasets namely Synthetic-1, Synthetic-2, and Emotions. Important statistics of these datasets are provided in Table 1. When evaluating each algorithm on a certain dataset, we computed different performance metrics, namely, *F_β-score, precision, recall, (multi-label) hamming loss, test-time, and the per-label hamming losses (one for each label)*. Importantly, to understand the influence of SA's hard-parameter sharing ($\boldsymbol{\theta^{sh}}$) on its learning ability for F_β-score, we simulated three distinct neural-networks (NNs) of SA for each dataset. Specifically, for each dataset-k ($k \in [3]$), we denote the corresponding three neural-networks of SA by SA-kl where $l = 1$ is a very shallow NN, $l = 3$ is a sufficiently deep NN, and $l = 2$ is NN with depth that is midstream of $l = 1$ and $l = 3$. The architecture for $l = 3$ was found out through several training runs and then finding out a saturation point for F_β-score. The architecture for $l = 2$ was then simply chosen as a midpoint between the architectures of $l = 1$ and $l = 3$. For BR, as we intended to use it as a baseline for minimal performance, we implemented it by training linear binary classifiers for each label using the logistic-loss.

5.1 Datasets

Synthetic-1 and Synthetic-2: Our first two datasets, i.e., Synthetic,-1 and Synthetic-2, were generated using the method prescribed in [1]. These two datasets served two important purposes. Firstly, as mentioned in the project proposal, either dataset can be used for correctness-check of our implementation of ZA. Secondly, due to their *generative* nature, both datasets serve as good instruments for qualitatively assessing SA w.r.t the F_β-score. Below, we describe the generation process of these datasets in a step-wise manner. It is worth noting that the specific joint distribution from which these datasets are generated is such that the $s^2 + 1$ statistics (namely $\mathbb{P}(\|\mathbf{Y}\| = 0|\mathbf{X} = \mathbf{x})$, and $\mathbb{P}(Y_j = 1, \|\mathbf{Y}\| = k|\mathbf{X} = \mathbf{x})$ for every $j, k \in [s]$) required for the Bayes-optimal (multi-label) classifier are always a linear-function of $\mathbf{x}$.

Generation of Synthetic-1 and Synthetic-2: Choose some suitable d and s such that 2^s is not too large enough. (This is required to implement BOC which has access to all the 2^s conditional probabilities). Generate a full-rank matrix $\mathbf{W} \in [0, 1]^{(s^2+1)\times d}$ where each entry is drawn randomly from $[0, 1]$. Now, generate a vector $\boldsymbol{\beta} \in [0.1, 1]^{2^s}$ whose entries are drawn randomly from $[0.1, 1]$. Once $\mathbf{W}$ and $\boldsymbol{\beta}$ are generated, they remain fixed and the following steps are repeated to generate the datapoints $(\mathbf{x}, \mathbf{y})$ and the corresponding conditional probabilities $\mathbb{P}(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$. *i*) Sample a probability vector $\mathbf{p} \in \Delta_{\{0,1\}^s}$ from the Dirichlet($\boldsymbol{\beta}$) distribution (see Appendix A.6). (This distribution $\mathbf{p}$ serves as the conditional probability of $\mathbf{Y}$ for the generated feature-vector $\mathbf{x}$, i.e., $\mathbf{p}_\mathbf{y} = \mathbb{P}(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$). *ii*) Set $\mathbf{q} = \mathbb{E}_\mathbf{p}\left[a_0(\mathbf{Y}), a_{1,1}(\mathbf{Y}), \ldots, a_{1,s}(\mathbf{Y}), \ldots, a_{s,1}(\mathbf{Y}), \ldots, a_{s,1}(\mathbf{Y})\right]^T$, where a_0 and $a_{j,k}$ are as defined in (5). (By definition, $\mathbf{q}$ is the vector of the true $s^2 + 1$ conditional probabilities that are sufficient to know for producing a Bayes-optimal prediction). *iii*) Set $\mathbf{x} = \mathbf{W}^\dagger \gamma_{\log}(\mathbf{q})$ and generate $\mathbf{y} \sim \mathbf{p}$. Here, $\gamma_{\log}$ is applied to $\mathbf{q}$ element-wise and $\mathbf{W}^\dagger$ denotes the pseudo-inverse of $\mathbf{W}$. (Since $\gamma_{\log}$ is invertible, we can see that $\mathbf{q}(\mathbf{x}) = \gamma^{-1}_{\log}(\mathbf{W}\mathbf{x})$).

Emotions: Our third dataset, i.e., Emotions, is a small multi-label dataset drawn from the Mulan repository [23] that is not used in either [1] or [2]. The dataset belongs to the music domain and has 72-dimensional feature-vectors and six labels. The six labels indicate different emotions namely `amazed`, `happy`, `relaxing`, `quiet`, `sad`, and `angry`. The training and testing data are available in the form of `arff` files, and a separate `Python` code was written to convert the data into `numpy` files. Of the available 391 training points, 79 were used for validation during the training process.

Table 1: Multi-label datasets used in experiments

Dataset Index	**Dataset name**	**# train**	**# validation**	**# test**	**# labels** (s)	**# features** (d)
1	Synthetic-1	5000	1000	3000	6	100
2	Synthetic-2	7000	1000	3000	10	200
3	Emotions	312	79	202	6	72

5.2 Training Details

First, we briefly describe the architecture of the training-models we implemented for each algorithm. *i*) As mentioned above, we intended to use BR as a baseline for minimal performance. Therefore, in each dataset, we implemented BR by training s linear binary classifiers (one for each label). *ii*) The score-function learning models for ZA in Synthetic-1 and Synthetic-2 were restricted to linear functions from $\mathcal{X}$ to $\mathbb{R}^{s^2+1}$ (as mentioned above, this suffices for an almost Bayes-optimal performance). For dataset-3, we implemented ZA's score-function learning-model with a 1-hidden layer neural-network. *iii*) For Synthetic-k ($k \in [2]$), SA-$k1$ had 1 hidden layer; SA-$k2$ had 2 hidden layers; and SA-$k3$ had 3 hidden layers. For $k = 3$ (Emotions dataset), SA-31 had 1 hidden layer; SA-32 had 2 hidden layers; and SA-33 had 4 hidden layers. For exact sizes of $\boldsymbol{\theta}^{sh}$ and $\boldsymbol{\theta}^{j}$'s, and the visualization of the training models of ZA and SA, please see Appendix E.

Next, we describe the training process which was implemented in `Python` language using the `PyTorch` library. Unless otherwise noted, the described process applies to each dataset.

Experiment-1: Given a dataset-k and a training-model (BR, SA-kl, ZA), we trained the model's score-function/s using the Adam-Optimizer with a weight-decay of 10^{-4} (for regularization). The loss-function used with the optimizer was in accordance with the algorithm, i.e., (3) for ZA, and cross-entropy/logistic-loss for each label in SA and BR. (The loss-function for ZA was implemented using the custom-loss function feature of PyTorch). The number of epochs and batch-size were set to 100 and 128 respectively, and the learning-rates were varied. Specifically, for each learning rate in the set $\{10^{-4}, 10^{-3}, 10^{-2}\}$, the trained models were saved after every 9^{th}-epoch along with their F_β-score on the validation data. This was done to ensure that the model does not overfit on the training data. (Choosing the right number of epochs is a critical step in training; too many epochs can lead to overfitting, whereas too few may result in underfitting. By saving the model every 9^{th}-epoch, we gave ourselves the flexibility of choosing the right number of epochs). Of the last three saved models, the best model in terms of validation F_β-score was chosen for testing on the test data. We call this set of simulations as Experiment-1.

Experiment-2: Supplementary to the aforementioned experiment, we used the (generative) Synthetic-1 and Synthetic-2 datasets to obtain a rough understanding of the learning ability of each algorithm w.r.t the training-size. We call this set of simulations as Experiement-2. For space reasons, we provide them in Appendix F.

5.3 Test Results and Qualitative Assessment

Here, we provide and assess the test results for Experiment-1 which are summarized in Table 2.

Overall, we observe that ZA is the most competitive w.r.t F_β-score and recall, and BR performs better w.r.t precision and (multi-label) hamming-loss. This is not surprising because ZA is specifically designed for F_β-loss and thus operates at an optimal trade-off between precision and recall. BR, on the other hand, trains separate binary classifiers and ignores the inter-label dependencies. Therefore, while each label's classifier outputs higher number of correct predictions on the average (leading to

Table 2: Comparison of ZA, SA, and BR, on datasets 1-3. [The computed BOC (Bayes-optimal classifier) F_β-scores for Dataset-1 and Dataset-2 were 0.6737 and 0.6450 respectively].

Dataset	Model	Performance Metrics				Efficiency Metrics
		F_β score	Precision	Recall	Hamming loss	Test-time (sec)
1	ZA	0.6719	0.5403	0.9736	0.4553	0.8359
	SA-11	**0.6724**	0.5327	**1.0000**	0.4673	0.1721
	SA-12	**0.6724**	0.5327	**1.0000**	0.4673	0.2587
	SA-13	0.6632	0.5348	0.9688	0.4655	0.2016
	BR	0.6360	**0.5612**	0.8097	**0.4385**	**0.0080**
2	ZA	**0.6450**	0.4939	**1.0000**	0.5061	3.8299
	SA-21	0.4783	**0.5009**	0.4988	**0.4930**	0.3660
	SA-22	0.4684	0.4931	0.4999	0.5008	0.5396
	SA-23	0.4701	0.4985	0.4966	0.4954	0.4807
	BR	0.5950	0.4954	0.8042	0.5013	**0.0598**
3	ZA	**0.6166**	0.5250	**0.7895**	0.3044	0.067
	SA-31	0.3259	0.6709	0.2657	0.2847	0.0140
	SA-32	0.2832	0.2973	0.3032	0.4653	0.0140
	SA-33	0.5416	0.6667	0.5313	0.2417	0.0350
	BR	0.5228	**0.6718**	0.5438	**0.2376**	**0.0608**

higher precision and lesser hamming-loss), the individual label predictions when concatenated in the final prediction are usually not consistent with each other, leading to a lesser F_β-score.

Surprisingly, the performance of SA varies across the training-models as well as the datasets. In Synthetic-1, all the three models of SA perform competitively whereas in Synthetic-2 and Emotions datasets, SA underperforms when compared to BR. There can be quite a few reasons for these two observations. We note that SA can handle the inter-label dependencies through multi-objective optimization only, which generally works well when the different objective functions do not compete with each other. This is perhaps the potential reason for its bad performance on the Emotions dataset. That is, the labels in the Emotions dataset do not compete with each other (the presence of one emotion most often negates the presence of the other). For example, it is not likely to have both `happy` and `sad` emotions in the same feature-vector. Same applies to `relaxing` and `angry`, etc. As concerns the Synthetic-2 dataset, we first note that SA does perform competitively on Synthetic-1, and the difference between Synthetic-1 and Synthetic-2 is only in the dimensionality of the feature-vector (d) and the number of labels (s) (they are produced by the same generation method). Indeed, in comparison to Synthetic-1, Synthetic-2 has twice as many features and almost twice the number of labels. We believe the potential reason for bad performance of SA on Synthetic-2 is that both d and s were too high, so much so that our chosen sizes of $\boldsymbol{\theta}^{sh}$ and $\boldsymbol{\theta}^j$'s were not large enough for a deep hard-parameter sharing and a rich score-function class for each label.

Finally, we note that the test time of ZA is significantly higher than those of SA and BR. This is not surprising because ZA needs to implement the `decode-z` mapping whose execution time is cubic in the number of labels ($O(s^3)$).

6 Conclusion

In this project, we focused on the MLC problem with special emphasis on the F_β-measure. We implemented three MLC algorithms namely ZA (single-objective minimization designed for F_β), SA (multi-objective minimization), and BR (assuming labels are independent), with the motivation of finding out whether SA, which is a computationally flexible algorithm, can perform accurately on the F_β-measure. To this end, we used BR and ZA as baselines for minimal and maximal F_β performance. We learnt that while performing MLC through multi-objective optimization as done in SA is computationally flexible, it is vulnerable to produce unstable results that can vary a lot with the choice of the training model as well as the subject dataset. In light of our experiments and results, we conclude that *SA cannot be used as a reliable heuristic classifier for the F_β-measure*.

7 Acknowledgements and Contributions

We express our sincere gratitude to Professor Alfred Hero for helping us devise a solid research agenda and a grounded plan of experiments. We also thank Professor Clayton Scott for motivating us to go beyond the nominal expectation and implement both ZA and SA algorithms. While this did increase our targets beyond the course requirements, we are glad to share that our project's stated goals were completed successfully and we also went further in extending the scope of our project, i.e., using BR as a baseline for minimal performance. We are glad to see our project come to fruition along with the detailed learning that we have acquired both theoretically and empirically.

We also sincerely thank all the GSIs for their valuable suggestions, discussion during office hours, and on Piazza. We also thank the reviewers for their initial comments which proved valuable in shaping this report.

Contributions: Overall, A_1 led the complete project, collaborated and worked very closely with all of the authors A_2, A_3 and A_4 in their respective tasks.

1. Literature Review: A_1 and A_2 looked for potential topics for the project. $A_1 - A_4$ read and understood the two reference papers [1, 2], participated in discussions with the course instructor to shape the deliverables of the project. A_1 also encouraged other authors to go through the related works [8, 24, 25].

2. Project Proposal: A_1 contributed in developing the problem formalism and led the theory of both ZA and SA. A_2 and A_1 worked on the theory of ZA and A_3 worked with A_1 in the theory of SA. $A_1 - A_4$ collectively devised the project motivation as well as different contingency plans. All the authors provided critical feedback which helped to structure a camera-ready project proposal.

3. Data-set Collection/Generation and Pre-processing: $A_1 - A_4$ collectively finalized the datasets. A_4 contributed in converting the Emotions dataset from `.arff` to `.npy` format, and then splitting the data into training, validation and test sets. A_2 and A_1 performed the generation of the synthetic datasets.

4. Python Implementation of ZA and SA: For ZA, A_1 implemented the `decode-z` mapping and A_3 implemented the surrogate-risk minimization using custom-loss function feature of `PyTorch`. A_1, A_2, A_3 performed the code-checking and debugging of surrogate-risk minimization of ZA. A_3 implemented the SA algorithm. A_1 contributed some help to A_3 during the process. Moreover, A_4 implemented the training regime of ZA and A_3, A_2, A_1 implemented the scripts for neural-network models, dataset-loaders, and training and testing regimes across both ZA and SA.

5. Correctness-Check Steps: A_1, A_2, and A_4 contributed in debugging the surrogate-risk minimization implementation of ZA. A_1 and A_2 performed the correctness-check of ZA's implementation using the synthetic datasets. A_3 performed the correctness-check of SA's implementation using the MultiMNIST dataset.

6. Simulations: A_1 and A_3 chalked out the plan of experiments. A_2 and A_4 provided feedback which helped revise some simulations. A_3 implemented the core framework of SA including scripts for the core algorithm, custom-defined gradient scheme, data-loaders, architecture-designs for SA-$k1$, SA-$k2$, and SA-$k3$ (for all the datasets $k \in [3]$), and the training and testing scripts. A_3 and A_1 conducted experiments for ZA, SA and BR on all the datasets. Codes for evaluation of best models were written by A_3 and A_1.

7. Final Project Report: A_2 and A_1 did the qualitative assessment and proof reading of the final report, worked together in writing the introduction, related work, theory of ZA and SA, conclusion, supplementary materials for ZA. A_1 contributed in writing the experiments and results, writing the 'implementation of `decode-z`' mapping in the appendix. A_2 contributed in writing the supplementary materials for SA, Algorithm-2 and Algorithm-4 in appendix, tabulating results, creating the figures of NN architectures for ZA and SA w.r.t synthetic-1, synthetic-2 and emotions datasets. A_3 contributed in writing implementations details of ZA and SA. A_4 contributed in writing the Algorithm-1 and Algorithm-3 in appendix.

References

[1] M. Zhang, H. G. Ramaswamy, and S. Agarwal, "Convex calibrated surrogates for the multi-label f-measure," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 11 246–11 255. [Online]. Available: http://proceedings.mlr.press/v119/zhang20w.html

[2] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 525–536.

[3] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern Recognition*, vol. 37, no. 9, pp. 1757–1771, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320304001074

[4] R. Aly, S. Remus, and C. Biemann, "Hierarchical multi-label classification of text with capsule networks," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 323–330. [Online]. Available: https://aclanthology.org/P19-2045

[5] S. Feng, P. Fu, and W. Zheng, "A hierarchical multi-label classification algorithm for gene function prediction," *Algorithms*, vol. 10, 12 2017.

[6] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[7] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.

[8] K. Dembczynski, W. Waegeman, W. Cheng, and E. Hüllermeier, "An exact algorithm for f-measure maximization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/71ad16ad2c4d81f348082ff6c4b20768-Paper.pdf

[9] E. Gibaja and S. Ventura, "Multi-label learning: a review of the state of the art and ongoing research," *WIREs Data Mining and Knowledge Discovery*, vol. 4, no. 6, pp. 411–444, 2014. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1139

[10] I. Pillai, G. Fumera, and F. Roli, "Designing multi-label classifiers that maximize f measures: State of the art," *Pattern Recognition*, vol. 61, pp. 394–404, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320316302217

[11] Y. Zhang and Q. Yang, "A survey on multi-task learning," 2021.

[12] F. Herrera, F. Charte, A. J. Rivera, and M. J. Del Jesus, "Multilabel classification," in *Multilabel Classification*. Springer, 2016, pp. 17–31.

[13] G. Tsoumakas and I. P. Vlahavas, "Random k -labelsets: An ensemble method for multilabel classification," in *ECML*, 2007.

[14] K. Dembczynski, A. Jachnik, W. Kotlowski, W. Waegeman, and E. Huellermeier, "Optimizing the f-measure in multi-label classification: Plug-in rule approach versus structured loss minimization," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1130–1138.

[15] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3994–4003.

[16] E. M. Rudd, M. Günther, and T. E. Boult, "Moon: A mixed objective optimization network for the recognition of facial attributes," in *European Conference on Computer Vision*. Springer, 2016, pp. 19–35.

[17] B. Bakker and T. Heskes, "Task clustering and gating for bayesian multitask learning," 2003.

[18] Y. Xue, X. Liao, L. Carin, and B. Krishnapuram, "Multi-task learning for classification with dirichlet process priors." *Journal of Machine Learning Research*, vol. 8, no. 1, 2007.

[19] K. Miettinen, "Concepts," in *Nonlinear multiobjective optimization*. Springer, 1998, pp. 5–36.

[20] M. Ehrgott, *Multicriteria optimization*. Springer Science & Business Media, 2005, vol. 491.

[21] H. G. Ramaswamy, B. Srinivasan Babu, S. Agarwal, and R. C. Williamson, "On the consistency of output code based learning algorithms for multiclass learning problems," in *Proceedings of The 27th Conference on Learning Theory*, ser. Proceedings of Machine Learning Research, M. F. Balcan, V. Feldman, and C. Szepesvári, Eds., vol. 35. Barcelona, Spain: PMLR, 13–15 Jun 2014, pp. 885–902. [Online]. Available: https://proceedings.mlr.press/v35/ramaswamy14.html

[22] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multiobjective optimization," *Comptes Rendus Mathematique*, vol. 350, no. 5-6, pp. 313–318, 2012.

[23] "Mulan repository," http://mulan.sourceforge.net/datasets-mlc.html.

[24] M. D. Reid and R. C. Williamson, "Composite binary losses," *Journal of Machine Learning Research*, vol. 11, no. 83, pp. 2387–2422, 2010. [Online]. Available: http://jmlr.org/papers/v11/reid10a.html

[25] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 427–435. [Online]. Available: https://proceedings.mlr.press/v28/jaggi13.html

[26] M. Frank, P. Wolfe *et al.*, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[27] J. C. Dunn and S. Harshbarger, "Conditional gradient algorithms with open loop step size rules," *Journal of Mathematical Analysis and Applications*, vol. 62, no. 2, pp. 432–444, 1978.

A Supplementary Theory for Zhang's Algorithm

A.1 L^{F_β}-calibration

Definition 2. *Let D be the joint probability distribution from which the training samples, $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, are generated in an IID (independent and identically distributed) fashion. Then the pair $(\psi, \texttt{decode})$ is said to be L^{F_β}-calibrated if*[2]

$$\mathbb{E}_D\left[\psi(\mathbf{Y}, f_S(\mathbf{X}))\right] \xrightarrow{n\to\infty} \inf_{f'_S} \mathbb{E}_D\left[\psi(\mathbf{Y}, f'_S(\mathbf{X}))\right]$$
$$\implies \mathbb{E}_D\left[L^{F_\beta}(\mathbf{Y}, h_S(\mathbf{X}))\right] \xrightarrow{n\to\infty} \mathbb{E}_D\left[L^{F_\beta}(\mathbf{Y}, h^{Bayes}(\mathbf{X}))\right].$$

where h^{Bayes} is the Bayes-optimal classifier that has access to all the conditional probabilities of the form $\mathbb{P}(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$ for all $\mathbf{y} \in \{0,1\}^s$ and all $\mathbf{x} \in \mathcal{X}$.

A.2 Strictly Proper Composite Binary Losses

A *binary loss* $\phi : \{\pm 1\} \times \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ is said to be *strictly proper composite binary loss* with underlying (invertible) link function $\gamma : [0,1] \to \mathbb{R}$ if for all $q \in [0,1]$ and $u \neq \gamma(q) \in \mathbb{R}$:

$$\mathbb{E}_{y\sim Bin^{\pm 1}(q)}[\phi(y,u) - \phi(y,\gamma(q))] > 0,$$

where $y \sim Bin^{\pm 1}(q)$ denotes a $\{\pm 1\}$-valued random variable that takes value $+1$ with probability q and -1 with probability $1-q$. The intuition behind the above definition is that if we minimize a binary loss that satisfies the above equation, it allows us to recover accurate class probability estimates for binary CPE problems (because whenever the score is not equal to $\gamma(q)$, the average loss difference is strictly larger). Once the real-valued score function is learned, it is simply inverted by the inverse of the link-function, i.e., γ^{-1}. For example, if we consider the well-known binary logistic-loss $\phi_{log} : \{+1,-1\} \times \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ given by

$$\phi_{\log}(y,u) = \ln(1 + e^{yu}), \tag{10}$$

[2]Note that when n is large, minimizing $\mathbb{E}_D\left[\psi(\mathbf{Y}, f_S(\mathbf{X}))\right]$ is approximately equivalent to minimizing a suitable regularization of $\widehat{R}_\psi$ over a suitably rich function-class $\mathcal{F}$ [1].

then we can show that it is strictly proper composite with the link-function $\gamma_{log} : [0,1] \to \mathbb{R}$,

$$\gamma_{\log}(p) = \ln\left(\frac{p}{1-p}\right). \tag{11}$$

See [24] for the proof as well as an excellent introduction to strictly proper composite binary losses.

A.3 Low Rank of $\mathbf{L}^{F_\beta}$

Proposition 1. $rank(\mathbf{L}^{F_\beta} - 1) \leqslant s^2 + 1$.

Proof. From the definition of $\mathbf{L}^{F_\beta}_{\mathbf{y},\widehat{\mathbf{y}}}$, we have,

$$\begin{aligned}
\mathbf{L}^{F_\beta}_{\mathbf{y},\widehat{\mathbf{y}}} - 1 = -F_\beta(\mathbf{y},\widehat{\mathbf{y}}) &= -\frac{\left(1+\beta^2\right)\sum_{j=1}^{s} y_j \widehat{y}_j}{\beta^2\|\mathbf{y}\|_1 + \|\widehat{\mathbf{y}}\|_1} \\
&= -\mathbf{1}\left(\|\mathbf{y}\|_1 = 0\right)\cdot\mathbf{1}\left(\|\widehat{\mathbf{y}}\|_1 = 0\right) - \sum_{k=1}^{s}\mathbf{1}\left(\|\mathbf{y}\|_1 = k\right)\cdot\frac{\left(1+\beta^2\right)\sum_{j=1}^{s} y_j\widehat{y}_j}{\beta^2 k + \|\widehat{\mathbf{y}}\|_1} \\
&= a_0(\mathbf{y})\cdot b_0(\widehat{\mathbf{y}}) + \sum_{j=1}^{s}\sum_{k=1}^{s} a_{j,k}(\mathbf{y})\cdot b_{j,k}(\widehat{\mathbf{y}}),
\end{aligned}$$

where,

$$\begin{aligned}
a_0(\mathbf{y}) &= \mathbf{1}\left(\|\mathbf{y}\|_1 = 0\right); \qquad & a_{j,k}(\mathbf{y}) &= \mathbf{1}\left(\|\mathbf{y}\|_1 = k\right)\cdot y_j; \\
b_0(\widehat{\mathbf{y}}) &= -\mathbf{1}\left(\|\widehat{\mathbf{y}}\|_1 = 0\right); \qquad & b_{j,k}(\widehat{\mathbf{y}}) &= -\frac{\left(1+\beta^2\right)\cdot\widehat{y}_j}{\beta^2 k + \|\widehat{\mathbf{y}}\|_1}.
\end{aligned}$$

For every $\mathbf{y}, \widehat{\mathbf{y}} \in \{0,1\}^s$, let us define

$$\begin{aligned}
\mathbf{a}_{\mathbf{y}} &\coloneqq \left(a_0(\mathbf{y}), a_{1,1}(\mathbf{y}), a_{1,2}(\mathbf{y}), \ldots, a_{1,s}(\mathbf{y}), \ldots, a_{s,1}(\mathbf{y}), \ldots, a_{s,s}(\mathbf{y})\right); \\
\mathbf{b}_{\widehat{\mathbf{y}}} &\coloneqq \left(b_0(\widehat{\mathbf{y}}), b_{1,1}(\widehat{\mathbf{y}}), b_{1,2}(\widehat{\mathbf{y}}), \ldots, b_{1,s}(\widehat{\mathbf{y}}), \ldots, b_{s,1}(\widehat{\mathbf{y}}), \ldots, b_{s,s}(\widehat{\mathbf{y}})\right)
\end{aligned}$$

Then, we can write

$$\mathbf{L}^{F_\beta}_{\mathbf{y},\widehat{\mathbf{y}}} - 1 = \mathbf{a}_{\mathbf{y}}^T\mathbf{b}_{\widehat{\mathbf{y}}}.$$

This gives us the below rank-s^2+1 decomposition of $\mathbf{L}^{F_\beta}$.

$$\mathbf{L}^{F_\beta} - \mathbf{1}\mathbf{1}^T = \begin{bmatrix}\mathbf{a_0} & \ldots & \mathbf{a_1}\end{bmatrix}^T\begin{bmatrix}\mathbf{b_0} & \ldots & \mathbf{b_1}\end{bmatrix}.$$

□

A.4 Result of Ramaswamy et al. for MCC Problems with Low-rank Discrete Losses

Theorem 1. *Consider a MCC problem with p classes where each feature vector $\mathbf{x}$ belongs to some class $y \in [p]$. Let $\mathbf{L} \in \mathbb{R}^{p\times p}_{\geqslant 0}$ be the discrete loss-matrix for this problem that has rank-w and let the rank decomposition of $\mathbf{L}$ be given by $\begin{bmatrix}\mathbf{a}_0 & \ldots & \mathbf{a}_p\end{bmatrix}^T\begin{bmatrix}\mathbf{b}_0 & \ldots & \mathbf{b}_p\end{bmatrix}$., where $\mathbf{a}_i, \mathbf{b}_i \in \mathbb{R}^w$ for all $i \in [p]$. Then for any strictly-proper composite binary loss $\phi : \{\pm 1\} \times \mathbb{R} \to \mathbb{R}_{\geqslant 0}$ (see Appendix A.2) with its corresponding link-function $\gamma : [0,1] \to \mathbb{R}$, there exists a pair $(\psi, \texttt{decode})$ such that the pair $(\psi, \texttt{decode})$ is $\mathbb{L}$-calibrated. The form of ψ and $\texttt{decode}$ is given as follows:*

$$\psi(y, \mathbf{u}) = \sum_{j=1}^{w}\left[\widetilde{a}_{yj}\,\phi\left(+1, u_j\right) + \left(1 - \widetilde{a}_{yj}\right)\phi\left(-1, u_j\right)\right],$$

$$decode(\mathbf{u}) \in \operatorname*{arg\,min}_{\widehat{y}\in[p]} \sum_{j=1}^{w}\widetilde{b}_{\widehat{y}j}\gamma^{-1}\left(u_j\right) + c_{\widehat{y}},$$

where

$$\tilde{a}_{yj} = \frac{a_{yj} - a_{\min}}{a_{\max} - a_{\min}} (\in [0,1]),$$

$$\tilde{b}_{\hat{y}j} = (a_{\max} - a_{\min})\, b_{\hat{y}j},$$
$$c_{\hat{y}} = a_{\min} \sum_{j=1}^{w} b_{\hat{y}j},$$
$$a_{\min} = \min_{y,j} a_{yj},$$
$$a_{\max} = \max_{y,j} a_{yj}.$$

Proof. For the proof of this Theorem, see [21]. □

A.5 Implementation of '`decode-z`' mapping

Here, we describe how to efficiently implement the mapping `decode-z` : $\mathbb{R}^{s^2+1} \to \{0,1\}^s$. Recalling (4), we want to solve

$$\arg\min_{\widehat{\mathbf{y}} \in \{0,1\}^s} \; b_0(\widehat{\mathbf{y}}) \cdot \mathbb{P}(\mathbf{Y} = \mathbf{0} | \mathbf{X} = \mathbf{x}) + \sum_{j=1}^{s} \sum_{k=1}^{s} b_{j,k}(\widehat{\mathbf{y}}) \cdot \mathbb{P}(Y_j = 1, \|\mathbf{Y}\| = k | \mathbf{X} = \mathbf{x}), \tag{12}$$

where $b_0(\widehat{\mathbf{y}}) = -\mathbf{1}\left(\|\widehat{\mathbf{y}}\|_1 = 0\right)$ and $b_{j,k}(\widehat{\mathbf{y}}) = -\frac{\left(1+\beta^2\right)\cdot \widehat{y}_j}{\beta^2 k + \|\widehat{\mathbf{y}}\|_1}$. Let us partition the prediction space $\{0,1\}^s$ into $s+1$ sets of the form $\mathcal{H}_l := \{\widehat{\mathbf{y}} \in \{0,1\}^s : \|\widehat{\mathbf{y}}\|_1 = l\}$ where $l = 0, 1, \ldots, s$. Then, we can solve (12) by solving $s+1$ inner-minimizations, namely,

$$\widehat{\mathbf{y}}^{l,*}(\mathbf{x}) \in \arg\min_{\widehat{\mathbf{y}} \in \mathcal{H}_l} z(\widehat{\mathbf{y}}|\mathbf{x}) = \sum_{j=1}^{s} \widehat{y}_j \sum_{k=1}^{s} \frac{-(1+\beta^2)}{\beta^2 k + l} \mathbb{P}(Y_j = 1, \|Y\| = k | \mathbf{X} = \mathbf{x}), \tag{13}$$
$$\widehat{\mathbf{y}}^{0,*}(\mathbf{x}) = \mathbf{0} \qquad \text{(trivially solved)},$$

that are followed by an outer-maximization,

$$\widehat{\mathbf{y}}^{*}(\mathbf{x}) \in \arg\min_{\widehat{\mathbf{y}} \in \{\widehat{\mathbf{y}}^{0,*}, \widehat{\mathbf{y}}^{1,*}, \ldots, \widehat{\mathbf{y}}^{s,*}\}} z(\widehat{\mathbf{y}}|\mathbf{x}). \tag{14}$$

Here, we can note that the outer-minimization (14) can be done by simply checking all the $s+1$ possibilities. The main effort is then required for solving the inner-minimization (13). For any $l \in [s]$, let us define,

$$T_{jl} = \sum_{k=1}^{s} \frac{-(1+\beta^2)}{\beta^2 k + l} \mathbb{P}(Y_j = 1, \|Y\| = k | \mathbf{X} = \mathbf{x}).$$

Then, we can write

$$\widehat{\mathbf{y}}^{l,*}(\mathbf{x}) \in \arg\min_{\widehat{\mathbf{y}} \in \mathcal{H}_l} z(\widehat{\mathbf{y}}|\mathbf{x}) = \sum_{j=1}^{s} \widehat{y}_j T_{jl}. \tag{15}$$

Now, since $\widehat{\mathbf{y}}^{l,*} \in \mathcal{H}_l$, it must be that $\widehat{y}_j^{l,*} = 1$ for exactly l number of 1s. Looking at (15), we can see that it must be the cast that $\widehat{y}_j^{l,*} = 1$ for smallest l values of T_{jl} and 0 otherwise. Therefore, the task is to find all the s^2 values of T_{jl} for $j, k \in [s]$. If we define a matrix $\mathbf{T}$ such that $\mathbf{T}_{j,l} = T_{jl}$. Then, we can write

$$\mathbf{T} = \mathbf{P}\mathbf{V}. \tag{16}$$

where $\mathbf{T}_{j,l} = (\mathbf{P}\mathbf{V})_{j,l} = \sum_{k=1}^{s} P_{jk} V_{kl}$ with $P_{jk} = \mathbb{P}(Y_j = 1, \|Y\| = k | \mathbf{X} = \mathbf{x})$ and $V_{kl} = \frac{-(1+\beta^2)}{\beta^2 k + l}$.

Thus, we see that if we knew $\mathbb{P}(\mathbf{Y} = \mathbf{0} | \mathbf{X} = \mathbf{x})$ and the s^2 probabitlies $\mathbb{P}(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x})$, we could implement the decode mapping in $O(s^3)$-time (required for computing equation (16)). Since these are exactly the probabilities estimated by ZA's score function, one can use the above method for `decode-z` by replacing $\mathbb{P}(\mathbf{Y} = \mathbf{0} | \mathbf{X} = \mathbf{x})$ and $\mathbf{P}$ by their corresponding estimates obtained

Algorithm 1 decode-z

Input: Vector $\mathbf{u} = \left(u_0, (u_{jk})_{j,k=1}^{s}\right)^{\top} \in \mathbb{R}^{s^2+1}$.
Parameters: Link function $\gamma : [0,1] \to \mathbb{R}$.
Define matrices $\mathbf{Q} \in [0,1]^{s\times s}$ and $\mathbf{V} \in \mathbb{R}^{s\times s}$ as follows:

$$Q_{jk} = \gamma^{-1}(u_{jk}); \qquad V_{kl} = \frac{-(1+\beta^2)}{\beta^2 k + l}.$$

Compute $\mathbf{T} = \mathbf{QV}$. ▷ Matrix multiplication, $O(s^3)$ time
for $l = 1 \dots s$: **do** ▷ For loop takes total $O(s^2 \ln(s))$ time
    Find the l smallest numbers among $\{T_{jl} : j \in [s]\}$; let their indices be $j_1^l, \dots, j_l^l$.
    Define $\widehat{\mathbf{y}}^{l,*} \in \{0,1\}^s$ as follows:

$$\widehat{y}_j^{l,*} = \begin{cases} 1 & \text{if } j \in \left\{j_1^l, \dots, j_l^l\right\}, \\ 0 & \text{otherwise.} \end{cases} \quad \text{// this solves } \widehat{\mathbf{y}}^{l,*} \in \operatorname{argmin}_{\widehat{\mathbf{y}}\in\widehat{\mathcal{Y}}_l} \sum_{j=1}^{s} \widehat{y}_j T_{jl}$$

    Set $z_l^* = \sum_{j=1}^{s} \widehat{y}_j^* T_{jl}$.
end for
Pick $\widehat{\mathbf{y}}^* \in \{0,1\}^s$ as follows:

$$\widehat{\mathbf{y}}^* \in \operatorname*{argmin}_{\widehat{\mathbf{y}}\in\{\mathbf{0},\widehat{\mathbf{y}}^{1,*},\dots,\widehat{\mathbf{y}}^{s,*}\}} -\mathbf{1}(\widehat{\mathbf{y}} = \mathbf{0}) \cdot \gamma^{-1}(u_0) + \mathbf{1}(\widehat{\mathbf{y}} \neq \mathbf{0}) \cdot z^*_{\|\widehat{\mathbf{y}}\|_1}.$$

Output: $\widehat{\mathbf{y}}^* \in \{0,1\}^s$.

by ZA. Specifically, let $\mathbf{Q}$ be the matrix of s^2 probabilities estimates obtained by ZA, i.e., $Q_{jk} = (\gamma^{-1}(f_S(\mathbf{x})))_{jk}$ and let us denote the ZA's estimate for $\mathbb{P}(\mathbf{Y} = \mathbf{0}|\mathbf{X} = \mathbf{x})$ by $q_0 = \gamma^{-1}(f_S(\mathbf{x}))$. Then, we can write

$$\mathbf{T} = \mathbf{QV}.$$

The pseudocode of this complete procedure is presented in Algorithm 1.

A.6 Dirichlet Distribution

The Dirichlet distribution is a family of continuous multivariate probability distributions parameterized by a (finite) k-length vector of positive real numbers $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_k)$, and denoted as $\mathrm{Dir}(\boldsymbol{\beta})$. The infinite-dimensional generalization of the Dirichlet distribution is known as Dirichlet process, and it is commonly used in non-parametric Bayesian inference as prior distribution.

Definition 3. *Let $\boldsymbol{X} = [X_1, \dots, X_k]^T$ be a vector with non-negative values, i.e. $X_i \geqslant 0$ for $i = 1, 2, \dots, k$ and $\sum_{i=1}^{k} X_i = 1$. Also, let $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_k)$, where $\beta i > 0$ for each i. The Dirichlet distribution of order $k \geqslant 2$ with parameters $(\beta_1, \cdots, \beta_k)$ has a probability density function on the Euclidean space $\mathbb{R}^{k-1}$ given by*

$$f(x_1, \cdots, x_k; \boldsymbol{\beta}) = \left(\frac{\Gamma(\beta_0)}{\prod_{i=1}^{k}\Gamma(\beta_i)}\right) \prod_{i=1}^{k} x_i^{\beta_i - 1}.$$

where $\beta_0 = \sum_{i=1}^{k} \beta_i$, and $x_i \geqslant 0$, $\sum x_{i=1}^{k} = 1$.

B Supplementary Theory for Sener's Algorithm

B.1 Solving optimization problem (MGDA) using Frank-Wolfe method

Here, we show how to solve the (MGDA) optimization problem using the Frank-Wolfe optimization method. We start by briefly discussing about the Frank-Wolfe optimization algorithm [26]. Then, we show the connection of (MGDA) optimization problem with finding the min-norm point in the

convex hull of set of input points problem. After that, we show how to solve the min-norm point problem for two points (or labels, i.e. s = 2) case. Finally, to solve a general case, we use the two points case as a subroutine inside Frank-Wolfe optimization method.

Frank-Wolfe optimization algorithm is an iterative first-order optimization algorithm for constrained convex optimization problems (CCOP).

Definition 4. *(Constrained Convex Optimization Problem) Suppose $\mathcal{D}$ is a compact convex set in a vector space and $f : \mathcal{D} \to \mathbb{R}$ is a convex and continuously differentiable real-valued function.*

$$\text{CCOP} : \min_{x \in \mathcal{D}} f(x)$$

where $\mathcal{D}$ and $f(x)$ are known as domain and objective function respectively.

In each iteration, the Frank–Wolfe algorithm considers a *linearization* of the objective function (linear approximation by considering the first-order *Taylor approximation* of f around $x^{(k)}$) , and moves towards a minimizer of this linear function, taken over the same domain, this is known as *linear search*. It is known [26,27] that every update of the Frank-Wolfe algorithm satisfy $f(x^{(k)}) - f(x^*) \leqslant O(\frac{1}{k})$, where x^* is an optimal solution of CCOP. Thus, the worst-case convergence of the Frank-Wolfe method is $O(\frac{1}{k})$, but the faster convergence rate can be achieved for specific problem classes, such as strongly convex optimization problem.

Now, we discuss how to use Frank-Wolfe optimization algorithm to solve (MGDA) (or (MGDA-UB)). Let $\mathbf{v}^j = \nabla_{\boldsymbol{\theta}^{sh}/\boldsymbol{Z}(\boldsymbol{\theta}^{sh})} \widehat{R}^j_{\phi^j}\left(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j\right)$.

Then it can be easily be observe that the problem of solving (MGDA) (or (MGDA-UB)) is similar to finding a minimum-norm point in the convex hull of the set of input points problem, i.e. $\min_{\boldsymbol{\alpha}} \left\|\alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_s \boldsymbol{v}_s\right\|^2$ s.t $\sum_{i=1}^s \alpha_i = 1,\ \alpha_i \geqslant 0$. In the computational geometry literature, finding min-norm point is a well-studied problem, and many algorithms have been proposed to find a minimum norm points in the convex hull of a large number of points in a low-dimensional space Since min-norm point problem is a CCOP with linear constraints. Therefore, we use Frank-Wolfe optimization method to solve min-norm problem (so (MGDA) or (MGDA-UB)). Let's first, consider the case of two points, i.e. $s = 2$. Thus, the min-norm point optimization problem can be written as:

$$\min_{\alpha \in [0,1]} \|\alpha \boldsymbol{v}_1 + (1-\alpha)\boldsymbol{v}_2\|_2^2$$

The above optimization problem is a one-dimensional quadratic function of α with an analytical solution,

$$\hat{\alpha} = \left[\frac{\boldsymbol{v}_2^T(\boldsymbol{v}_2 - \boldsymbol{v}_1)}{\|\boldsymbol{v}_1 - \boldsymbol{v}_2\|_2^2}\right]_*$$

where $[a]_* := \max(\min(a, 1), 0)$. Now, to solve the general case, we use the above two-points case as a subroutine for *linear search* inside the Frank-Wolfe optimizer. The detailed description is presented in Algorithm 2.

B.2 Pareto-Stationarity of (MGDA-UB)

Theorem 2. *Assume $\partial \boldsymbol{Z}/\partial \boldsymbol{\theta}^{sh}$ is full-rank. If $(\alpha^1, \cdots, \alpha^s)$ is the solution of (MGDA-UB), then one of the following is true:*

i) $\sum_{j=1}^s \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = 0$ *and the current parameters are Pareto-stationary, i.e.*

- $\nabla_{\boldsymbol{\theta}^j} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = \mathbf{0}$ *for every $j \in [s]$; and*
- $\exists \alpha^1, \alpha^2 \cdots, \alpha^s \geqslant 0$ *s.t.* $\sum_{j=1}^s \alpha^j = 1$ *and* $\sum_{j=1}^s \alpha^j \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j) = \mathbf{0}$.

ii) $\sum_{j=1}^s \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)$ *is a descent direction that decreases $\widehat{R}^j_{\phi^j}$ for every $j \in [s]$.*

[3] Let $\mathbf{H}$ be a symmetric matrix. The Mahalanobis norm of a vector $\mathbf{v}$ with respect to $\mathbf{H}$ is defined as $\|\mathbf{v}\|_{\mathbf{H}} = \mathbf{v}^T\mathbf{H}\mathbf{v}$.

Algorithm 2 FrankWolfe Optimizer

function FRANKWOLFESOLVER($\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^s$)
  Initialize $\boldsymbol{\alpha} = (\alpha^1, \cdots, \alpha^s) = \frac{1}{s}\mathbf{1}_{1\times s}$.
  Pre-compute $\mathbf{H}$ s.t. $\mathbf{H}_{i,j} = \boldsymbol{v}_i^T \boldsymbol{v}_j$.
  repeat
    $\hat{j} \leftarrow \arg\min_i \sum_j \alpha^j \mathbf{H}_{i,j} (= \arg\min_j \sum_i \alpha^i \mathbf{H}_{i,j})$.
    $\hat{\gamma} \leftarrow$ MinNormPoint_ConvexHull($\boldsymbol{e}_{\hat{j}}, \boldsymbol{\alpha}, \mathbf{H}$).[3]
    $\boldsymbol{\alpha} \leftarrow (1-\hat{\gamma})\boldsymbol{\alpha} + \hat{\gamma}\boldsymbol{e}_{\hat{j}}$.
  until $\hat{\gamma} \sim 0$ **or** Number of Iterations Limit
  return $\boldsymbol{\alpha} = (\alpha^1, \cdots, \alpha^s)$.
end function

// two-point case as 'line-search' subroutine
function MINNORMPOINT_CONVEXHULL($a, b, \mathbf{H}$) $= \arg\min_\gamma \|\gamma a + (1-\gamma)b\|_{\mathbf{H}}^2$
  if $a^{\mathbf{T}}b \geqslant \|a\|_2^2$ **then**
    return $\gamma = 1$
  else if $a^{\mathbf{T}}b \geqslant \|a\|_2^2$ **then**
    return $\gamma = 0$
  else
    return $\gamma = \frac{(b-a)^{\mathbf{T}}b}{\|a-b\|_2^2}$
  end if
end function

Proof. We begin the proof by showing the first case of Theorem 2, that if the optimum value of (MGDA-UB) is 0, so the optimum value of (MGDA).

Let $(\alpha^1, \cdots, \alpha^s)$ is the solution of (MGDA) and the optimum value of (MGDA-UB) is 0, i.e. $\left\|\sum_{j=1}^s \alpha^j \nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^j\right\|_2^2 = 0$, and by the *positive definiteness* property of matrix norm, it implies $\left(\sum_{j=1}^s \alpha^j \nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^j\right) = 0$. Thus,

$$\left(\sum_{j=1}^s \alpha^j \nabla_{\theta^{sh}} \widehat{R}_{\phi^j}^j\right) = \left(\frac{\partial \mathbf{Z}}{\partial \boldsymbol{\theta}^{sh}}\right)\left(\sum_{j=1}^s \alpha^j \nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^j\right) = 0$$

Hence, $(\alpha^1, \cdots, \alpha^s)$ is the solution of (MGDA) and the optimal value of (MGDA) is 0. Since $\partial\mathbf{Z}/\partial\boldsymbol{\theta}^{sh}$ is full rank, and $(\alpha^1, \cdots, \alpha^s)$ is the solution to both (MGDA) and (MGDA-UB), thus from KKT conditions mentioned in section-4.2, the current parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^s)$ are Pareto stationary. This completes the first case of Theorem 2.

Now consider the second case of Theorem 2, where we need to show that the descent direction calculated by the (MGDA-UB) decreases all the loss functions. Mathematically, it suffice to show that

$$\left(\sum_{j=1}^s \alpha^j \nabla_{\theta^{sh}} \widehat{R}_{\phi^j}^j\right)^{\mathrm{T}} \left(\nabla_{\theta^{sh}} \widehat{R}_{\phi^j}^{j'}\right) \geqslant 0 \quad \forall j' \in [s],$$

which is equivalent to showing

$$\left(\sum_{j=1}^s \alpha^j \nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^j\right)^{\mathrm{T}} \left(\frac{\partial \mathbf{Z}}{\partial \theta^{sh}}\right)^{\mathrm{T}} \left(\frac{\partial \mathbf{Z}}{\partial \theta^{sh}}\right) \left(\nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^{j'}\right) \geqslant 0 \quad \forall j' \in [s].$$

Since $\left(\frac{\partial \mathbf{Z}}{\partial \theta^{sh}}\right)$ is full rank, $\left(\frac{\partial \mathbf{Z}}{\partial \theta^{sh}}\right)^T \left(\frac{\partial \mathbf{Z}}{\partial \theta^{sh}}\right)$ is positive definite. Thus, it further reduces to show

$$\left(\sum_{j=1}^s \alpha^j \nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^j\right)^{\mathrm{T}} \left(\nabla_{\mathbf{Z}} \widehat{R}_{\phi^j}^{j'}\right) \geqslant 0 \quad \forall j' \in [s]. \tag{17}$$

We will show the inequality 17 by constructing the Lagrangian of (MGDA-UB), which is

$$L(\alpha^1, \cdots, \alpha^s, \lambda) = \left(\sum_{j=1}^{s} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\right)^{\mathrm{T}} \left(\sum_{j=1}^{s} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\right) - \lambda \left(\sum_{j=1}^{s} \alpha^j - 1\right) \text{ where } \lambda \geqslant 0.$$

Now, if we consider the KKT conditions for the above Lagrangian, we get the inequality 17,

$$\left(\sum_{j=1}^{s} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}\right)^{\mathrm{T}} \left(\nabla_{\mathbf{Z}} \widehat{R}^{j'}_{\phi^j}\right) = \frac{\lambda}{2} \geqslant 0.$$

This completes the proof of the second case of Theorem 2. □

C Pseudocode of ZA

Here we present the pseudocode for the ZA.

Algorithm 3 Zhang's Algorithm (ZA)

1: **Input:** Training sample $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$.
2: **Parameters:** *i*) Strictly-proper composite binary loss $\phi : \{\pm 1\} \times \mathbb{R} \to \mathbb{R}_{\geqslant 0}$; *ii*) Search-space $\mathcal{F}$ for score-functions $f_S : \mathcal{X} \to \mathbb{R}^{s^2+1}$.
3: Find $f_S \in \arg\min_{f \in \mathcal{F}} \widehat{R}_\psi(f)$ where $\widehat{R}_\psi$ is as defined in (2) and ψ is as defined in (3).
4: **Output:** Multi-label classifier $h_S = \texttt{decode} \circ f_S$, where `decode` is as defined in (4) and is implemented using Algorithm 1.

D Pseudocode of SA

Here we present the pseudocode for the SA.

Algorithm 4 SA's Update procedure for $\boldsymbol{\theta} = (\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots, \boldsymbol{\theta}^s)$

for $j = 1$ to s **do**
  $\boldsymbol{\theta}^j \leftarrow \boldsymbol{\theta}^j - \eta \nabla_{\boldsymbol{\theta}^j} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)$ ▷ Gradient descent on label-specific parameters
end for

$\boldsymbol{v}_j \leftarrow \nabla_{\boldsymbol{\theta}^{sh}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)$ for $j \in [s]$.
$(\alpha^1, \cdots, \alpha^s) = \textsc{FrankWolfeSolver}(\boldsymbol{v}^1, \boldsymbol{v}^2, \cdots, \boldsymbol{v}^s)$ ▷ Algorithm 2 in Appendix B.1
$\boldsymbol{\theta}^{sh} \leftarrow \boldsymbol{\theta}^{sh} - \eta \sum_{j=1}^{s} \alpha^j \nabla_{\mathbf{Z}} \widehat{R}^j_{\phi^j}(\boldsymbol{\theta}^{sh}, \boldsymbol{\theta}^j)$ ▷ Gradient descent on shared parameters

E Implementation Details of ZA and SA

In this section, we describe architectures detail for ZA and SA. As mentioned before, we implemented three architectures for SA in order to understand the effect of hard parameter sharing. A pictorial representation for each of the architecture used in the experiments is described in Fig 1. We designed a linear architecture for ZA for synthetic-1 and synthetic-2, whereas, for emotions dataset we increase the model complexity by introducing a hidden layer, in order to give ZA, the flexibility to learn non-linear functions. The NN-based ZA classifier takes d dimension input and give $s^2 + 1$ output where s is the number of classes, d is 100, 200 and 72 for Synthetic-1, Synthetic-2 and Emotions respectively. These $s^2 + 1$ scores then passed to the decode function 1 and we obtained s class prediction. For SA, in order to understand the influence of θ^{sh} on each task. Specifically, we designed 3 architecture namely, SA-$k1$, SA-$k2$ and SA-$k3$. All hidden layers are ReLU activated to overcome any potential gradient vanishing problem. The specifics about the number of layers and number of neurons in each layer is given in Fig 1. SA also takes the same d dimension input as ZA and give s

outputs which then passed to *soft-max* function to give a probability interpretation. The parameters involved denoted with θ^j in Fig 1 in each SA variant referred to as label-specific parameters, and they are responsible for learning task specific features, whereas the rest of parameters are responsible for learning correlation between different label (shared representation) and are referred to as shared parameters. Details about the size of parameters in θ^{sh} and θ^t are described in 2 here DNN-1 refers to SA-1, DNN-2 refers to SA-2 and DNN-3 refers to SA-3 Architecture.

ZA is trained on custom defined loss function described in (3) with Adam optimizer. whereas, SA is trained on cross-entropy loss with custom defined gradients described in Algorithm 4 with same Adam optimizer. Both the algorithm are trained for 100 epochs with batch size 128.

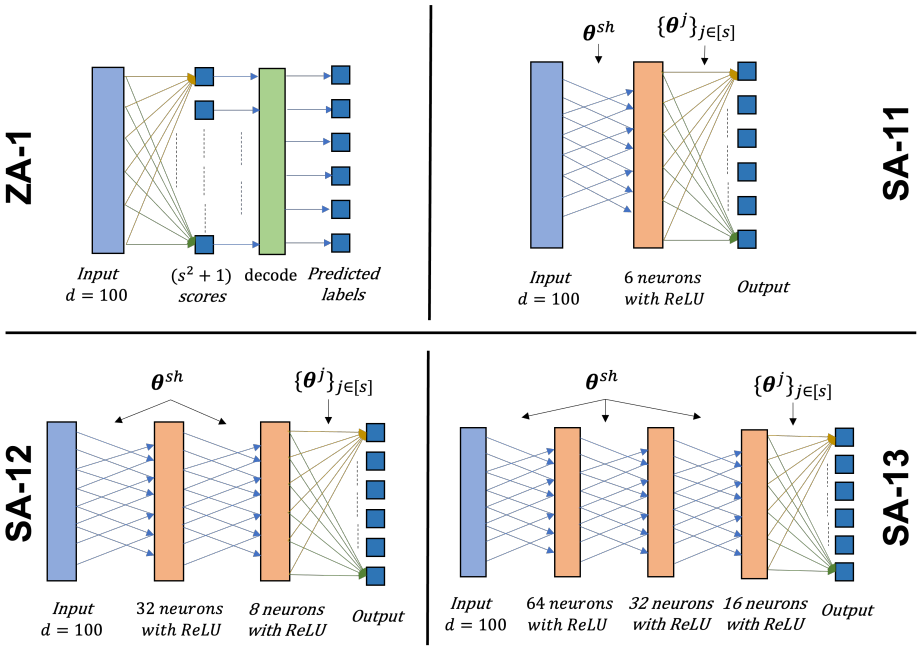

Figure 1: NN Architectures for Dataset-2 (Synthetic 1).

Table 3: Comparison of ZA, SA, and BR, on datasets 1-3

Dataset	Model	Size of parameters	
		$\lvert\boldsymbol{\theta}^{sh}\rvert$	$\lvert\boldsymbol{\theta}^{j}\rvert$
1	DNN1	600	72
	DNN2	3456	96
	DNN3	8960	192
2	DNN1	2000	200
	DNN2	14848	640
	DNN3	35840	832
3	DNN1	432	72
	DNN2	4992	72
	DNN3	5632	224

F Details of Experiment-2

Since both Synthetic-1 and Synthetic-2 datasets are generated from a probability distribution, they can serve as good instruments to to obtain a rough understanding of the learning ability of each algorithm w.r.t the training-size. Here, we describe our set of simulations that were performed over different learning-rates and different training-sizes for each algorithm. Specifically, first, a learning-rate was

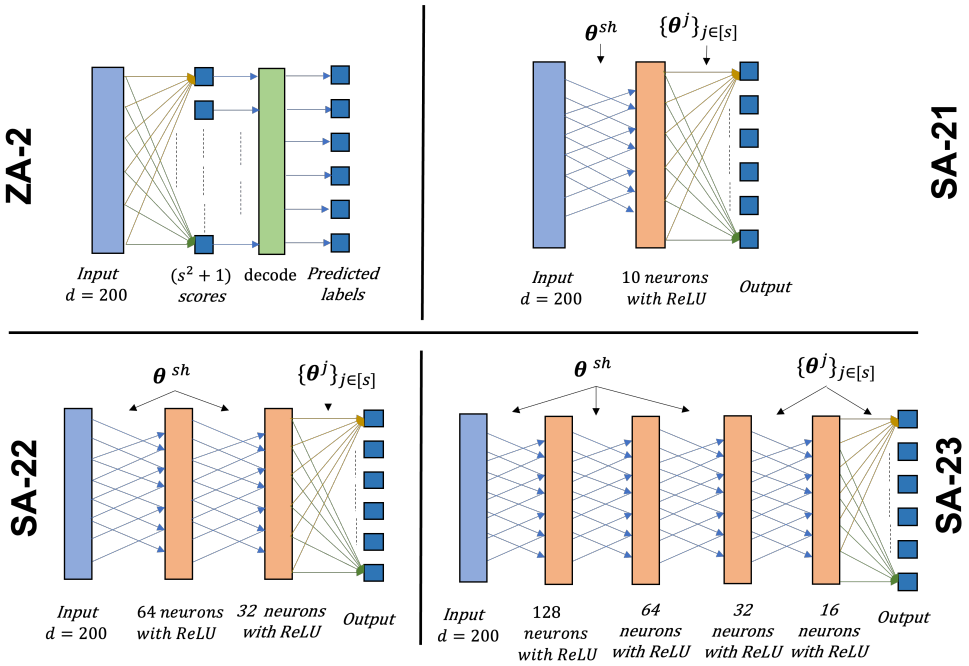

Figure 2: NN Architectures for Dataset-2 (Synthetic 2).

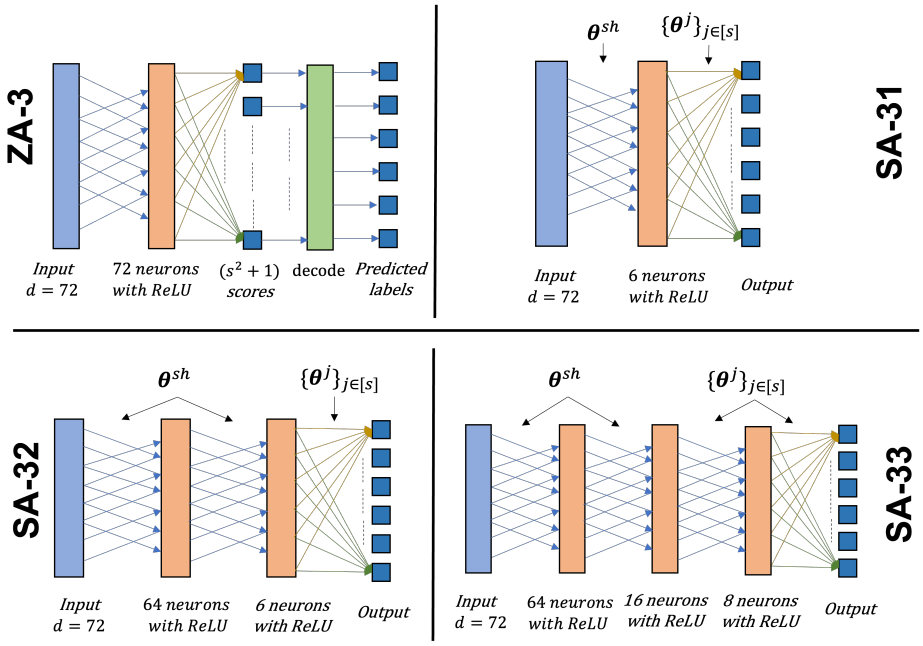

Figure 3: NN Architectures for Dataset-3 (Emotions).

picked from the set $\{10^{-4}, 10^{-3}, 10^{-2}\}$ using a for-loop within which the training was carried out for 50 different training-sizes evenly spaced between 10 and the dataset's total number of training-points (#-train).[4] At the end of the training, the learned model was applied to the test-data to produce the test F_β-score for that specific learning-rate and training-size. We call this set of simulations as Experiment-2.

[4]Each training was performed with 100 number of epochs and a batch-size of 128.

Here, we provide the results of Experiment-2 that was performed on Synthetic-1 and Synthetic-2. Importantly, in Synthetic-1, we can observe that ZA approaches the Bayes-optimal F_β-accuracy across all learning-rates whereas SA, when used with learning-rate of 0.01, is also able to achieve a near Bayes-optimal F_β-score. In Synthetic-2, again we can observe the L^{F_β}-consistency of ZA; however, this time we observe underperformance of SA.

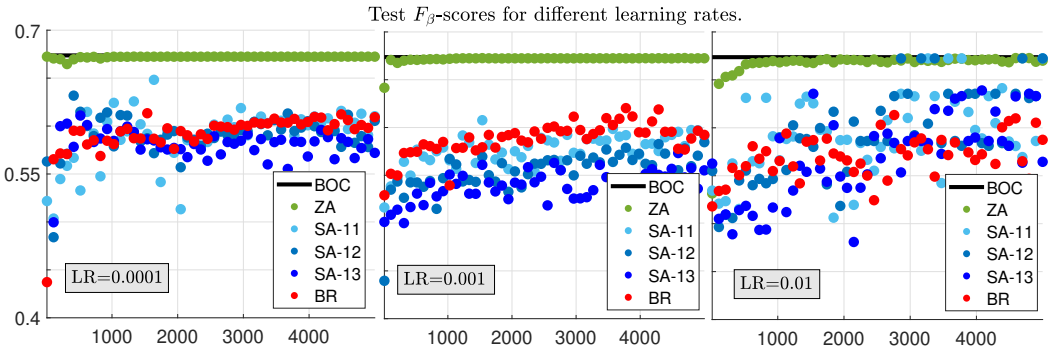

Figure 4: Test-F_β-score vs Training-size plots for ZA, SA, and BR across different learning-rates on Synthetic-1.

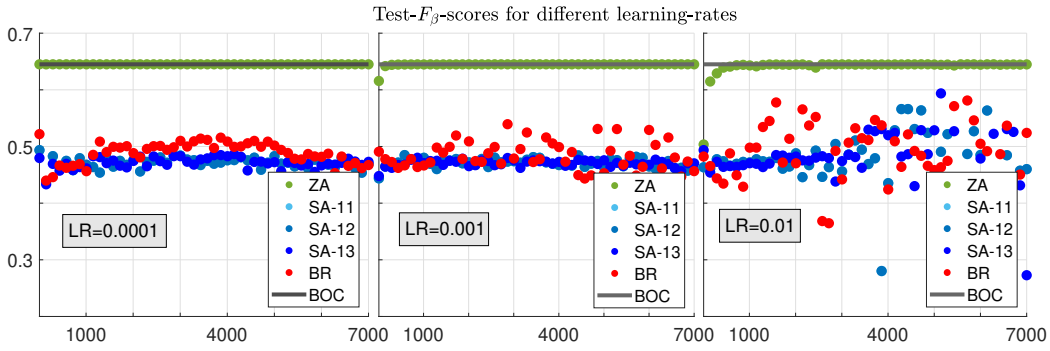

Figure 5: Test-F_β-score vs Training-size plots for ZA, SA, and BR across different learning-rates on Synthetic-2.

G Additional Results of Experiment-1

In the main text, we listed the F_β-scores, precision, recall, multi-label hamming loss, and test-time. However, we did not tabulate the per-label hamming-losses due to space limitations. Here, we list the per-label hamming-loss for each individual label in datasets-1 through 3.

Table 4: Per-Label Hamming Losses on dataset-1 through 3

Dataset-index	Label-Index	Model-wise Per-label Hamming losses				
(k)	(j)	ZA	SA-$k1$	SA-$k2$	SA-$k3$	BR
1	1	0.4620	0.4730	0.4730	0.4730	**0.4423**
	2	**0.4217**	0.4333	0.4335	0.4335	0.4310
	3	0.4733	0.4920	0.4920	0.4913	**0.4510**
	4	0.4190	0.4250	0.4250	0.4250	**0.4040**
	5	0.4727	0.4800	0.4800	0.4770	**0.4413**
	6	0.4830	0.5007	0.5007	0.4936	**0.4617**
2	1	0.5263	**0.4817**	0.4790	0.5010	0.5167
	2	0.5017	**0.4973**	0.5147	0.5100	0.5057
	3	**0.4946**	0.4957	0.4990	0.4997	0.4947
	4	0.5127	0.5023	0.5000	**0.4930**	0.5040
	5	**0.4923**	0.4947	0.5150	0.5043	0.4857
	6	0.5110	**0.4987**	0.5060	0.5080	0.5110
	7	0.5127	0.4957	**0.4947**	0.4977	0.5043
	8	0.4927	**0.4917**	0.4890	0.4773	0.4897
	9	0.5023	**0.4803**	0.5000	**0.4803**	0.4907
	10	0.5147	0.4920	0.5110	**0.4823**	0.5107
3	1	0.3119	0.2426	0.2673	0.2376	**0.2129**
	2	0.3416	**0.2921**	0.7079	0.2970	0.3119
	3	**0.3218**	0.3317	0.4752	0.3416	0.3366
	4	0.2030	0.2921	0.7079	0.1337	**0.1535**
	5	0.3762	0.3614	0.3614	**0.1931**	0.2228
	6	0.2723	**0.1881**	0.2723	0.2475	**0.1881**