# Unifying Heterogeneous Classifiers with Distillation
# Supplementary Material

Jayakorn Vongkulbhisal[1], Phongtharin Vinayavekhin[1], Marco Visentini-Scarzanella[2]
[1]IBM Research, Tokyo, Japan
[2]Amazon, Tokyo, Japan
jayakornv@ibm.com, pvmilk@jp.ibm.com, marcovs@amazon.com

## Contents

This supplementary material contains the following contents.



## 1. Cross-Entropy Method and Geometric Program

In this section, we show how to transform Eq. (7) in the main paper into a geometric program [2]. First, we rewrite $J(q)$ as follows:

$$J(q) = -\sum_i \sum_{l\in\mathcal{L}_i} p_i(X=l)\log\frac{q(X=l)}{\sum_{k\in\mathcal{L}_i} q(X=k)} \tag{1}$$

$$= \log\frac{\prod_i\prod_{l\in\mathcal{L}_i}\left(\sum_{k\in\mathcal{L}_i} q(X=k)\right)^{p_i(X=l)}}{\prod_i\prod_{l\in\mathcal{L}_i} q(X=l)^{p_i(X=l)}}. \tag{2}$$

Then, we can transform the following problem

$$\underset{q}{\text{minimise}} \quad J(q) = \log\frac{\prod_i\prod_{l\in\mathcal{L}_i}\left(\sum_{k\in\mathcal{L}_i} q(X=k)\right)^{p_i(X=l)}}{\prod_i\prod_{l\in\mathcal{L}_i} q(X=l)^{p_i(X=l)}} \tag{3}$$

into

$$\underset{q,\{t_i\}_i}{\text{minimise}} \quad \log\frac{\prod_i\prod_{l\in\mathcal{L}_i} t_i^{p_i(X=l)}}{\prod_i\prod_{l\in\mathcal{L}_i} q(X=l)^{p_i(X=l)}} \tag{4}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{L}_i} q(X=k) \le t_i, i=1,\ldots,N, \tag{5}$$

where we add new variables $t_i, i=1,\ldots,N$, to upper-bound each posynomial term in the numerator of the objective function. This turns the objective into a log of a monomial and adds inequality constraints to the formulation. Since log is an increasing function and its argument in the objective function is a monomial, removing log from the objective does not affect the minimum. This leads us to

$$\underset{q,\{t_i\}_i}{\text{minimise}} \quad \frac{\prod_i\prod_{l\in\mathcal{L}_i} t_i^{p_i(X=l)}}{\prod_i\prod_{l\in\mathcal{L}_i} q(X=l)^{p_i(X=l)}} \tag{6}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{L}_i} q(X=k) \le t_i, i=1,\ldots,N. \tag{7}$$

which is a geometric program with variables $q$ and $t_i$ [2]. With this formulation, we can further transform it into a convex problem with a change of variable. Here, we define $u_l \in \mathbb{R}$ for $l \in \mathcal{L}_U$ as $q(X=l) = \exp(u_l)$ (*i.e.*, $u_l = \log q(X=l)$). Instead of changing $q$ in (6), we directly change $q$ in (1). This transforms $J(q)$ to

$$\hat{J}(\{u_l\}_l) = -\sum_i\sum_{l\in\mathcal{L}_i} p_i(X=l)\left(u_l - \log\left(\sum_{k\in\mathcal{L}_i}\exp(u_k)\right)\right), \tag{8}$$

which is Eq. (9) in the main paper.

## 2. Alternating Least Squares (ALS) for Matrix Factorisation Methods

In this section, we detail the Alternative Least Squares (ALS) [1] algorithms used for matrix factorisation in the main paper.

### 2.1. ALS for matrix factorisation in probability space

First, let us recall the formulation (Eq. (12) in the main paper):

$$\underset{\mathbf{u},\mathbf{v}}{\text{minimise}} \quad \|\mathbf{M}\odot(\mathbf{P}-\mathbf{u}\mathbf{v}^\top)\|_F^2 \tag{9}$$

$$\text{subject to} \quad \mathbf{u}^\top\mathbf{1}_L = 1 \tag{10}$$

$$\mathbf{v}\ge\mathbf{0}_N, \mathbf{u}\ge\mathbf{0}_L, \tag{11}$$

The ALS algorithm for solving the above formulation is shown in Alg. 1. Steps 4 and 12 are derived from the closed-form solution of $\mathbf{u}$ and $\mathbf{v}$ in the cost function, *resp*. Steps 5 and 13 project $\mathbf{u}$ and $\mathbf{v}$ to the nonnegative orthants to satisfy the constraints in (11). Steps 7 to 10 are for normalising $\mathbf{u}$ to sum to 1 per constraint (10). In fact, for this algorithm, steps 5 and 13 are actually not necessary. This is because all $u_j$'s from step 4 and $v_i$'s from step 12 are already nonnegative since they are the results of division between nonnagative numbers. For termination criteria, we terminate the algorithm if the RMSE between different iterations of $\mathbf{u}$ and $\mathbf{v}$ is less than $10^{-3}$. We also use the maximum number of iterations of 3000 as a termination criteria.

In terms of implementation, each for-loop can be computed with vector operations (*e.g.*, in MATLAB or with Numpy in Python) instead of using for-loops. In addition, the factorisation of different samples can be performed in parallel on GPUs. These techniques allow a significant speed up compared with the naive implementation.

**Algorithm 1** Matrix factorisation in probabilty space

**Input:** $\mathbf{M}$, $\mathbf{P}$
**Output:** $\mathbf{u}$, $\mathbf{v}$
1: Initialise $\mathbf{v} := \mathbf{1}_N$
2: **while** not converged **do**
3: **for** $j := 1, \ldots, L$ **do**
4: $u_j := \left(\sum_{i=1}^N M_{ji}P_{ji}v_i\right) / \left(\sum_{i=1}^N M_{ji}v_i^2\right)$
5: $u_j := \max(0, u_j)$
6: **end for**
7: $\bar{u} := \sum_{j=1}^L u_j$
8: **for** $j := 1, \ldots, L$ **do**
9: $u_j := u_j / \bar{u}$
10: **end for**
11: **for** $i := 1, \ldots, N$ **do**
12: $v_i := \left(\sum_{j=1}^L M_{ji}P_{ji}u_j\right) / \left(\sum_{j=1}^L M_{ji}u_j^2\right)$
13: $v_i := \max(0, v_i)$
14: **end for**
15: **end while**

### 2.2. ALS for matrix factorisation in logit space

Again, let us recall the formulation (Eq. (15) in the main paper):

$$\underset{\mathbf{u},\mathbf{v},\mathbf{c}}{\text{minimise}} \; \|\mathbf{M} \odot (\mathbf{Z} - \mathbf{u}\mathbf{v}^\top - \mathbf{1}_L\mathbf{c}^\top)\|_F^2 + \lambda(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2) \tag{12}$$

$$\text{subject to } \mathbf{v} \geq \mathbf{0}_N, \tag{13}$$

The ALS for solving the above formulation is shown in Alg. 2. The derivation is similar to that in Sec. 2.1. That is, each step is derived via the closed-form solution of each variable, followed by appropriate projection steps. We use the same termination criteria as in previous section.

**Algorithm 2** Matrix factorisation in logit space

**Input:** $\mathbf{M}$, $\mathbf{Z}$, $\lambda$
**Output:** $\mathbf{u}$, $\mathbf{v}$, $\mathbf{c}$
Initialise $c_i := \left(\sum_{j=1}^L M_{ji}Z_{ji}\right) / \left(\sum_{j=1}^L M_{ji}\right), \forall i$
Initialise $\mathbf{v} := \mathbf{1}_N$
**while** not converged **do**
**for** $j := 1, \ldots, L$ **do**
$u_j := \left(\sum_{i=1}^N M_{ji}(Z_{ji} - c_i)v_i\right) / \left(\lambda + \sum_{i=1}^N M_{ji}v_i^2\right)$
**end for**
**for** $i := 1, \ldots, N$ **do**
$v_i := \left(\sum_{j=1}^L M_{ji}(Z_{ji} - c_i)u_j\right) / \left(\lambda + \sum_{j=1}^L M_{ji}u_j^2\right)$
$v_i := \max(0, v_i)$
**end for**
**for** $i := 1, \ldots, N$ **do**
$c_i := \left(\sum_{j=1}^L M_{ji}(Z_{ji} - u_jv_i)\right) / \left(\sum_{j=1}^L M_{ji}\right)$
**end for**
**end while**

## 3. Computation cost

Recall that to tackle UHC, our approach comprises three steps (Sec. 3 in the main paper, second paragraph): (*i*) obtaining $\{p_i\}_i$ from $\mathbf{x} \in \mathcal{U}$ and $\{C_i\}_i$, (*ii*) estimating $q$ from $\{p_i\}_i$, and (*iii*) training $C_U$ from $\mathbf{x}$ and $q$. The computation cost of different methods in the main paper differs only in step (*ii*), while it is the same for all methods in steps (*i*) and (*iii*). Focusing on (*ii*), standard distillation (Sec. 3.1) needs $\mathcal{O}(NL)$ to compute $q$ from $\{p_i\}_i$, while cross-entropy (Sec. 3.3) and matrix factorisation (Sec. 3.4) methods need to solve an optimisation problem, incurring much higher cost of $\mathcal{O}(tNL)$, where $t$ is the number of optimisation iterations. However, step (*ii*) is parallelisable for both cross-entropy and matrix factorisation methods, and it is a fixed cost irrelevant of classifier models. In contrast, the cost of training neural networks in step (*iii*) significantly overwhelms this fixed cost, thus in practice the difference is almost negligible.

## 4. Complete results for sensitivity analysis

In this section, we provide the results of sensitivity analysis of all methods. Fig. 1 shows the sensitivity result for size of transfer set $\mathcal{U}$; Fig. 2 shows that of temperature $T$; and Fig. 3 shows that of accuracy of $C_i$'s. Note that we use different legend style from the main paper to account for more methods.

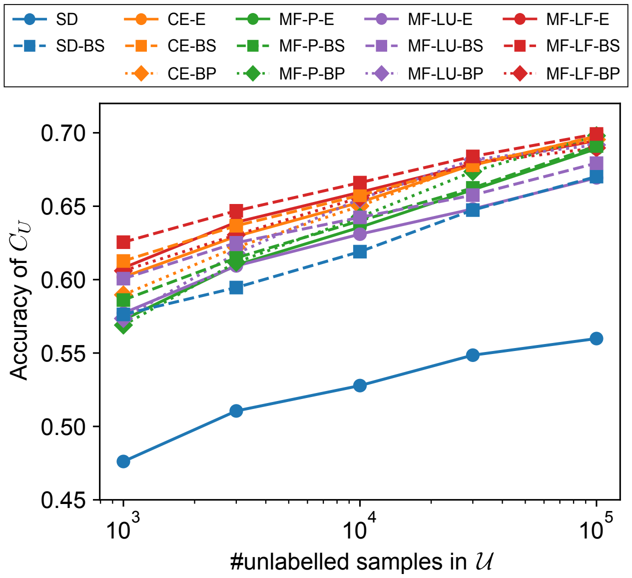


Figure 1. Sensitivity results on the size of the unlabelled set $\mathcal{U}$.

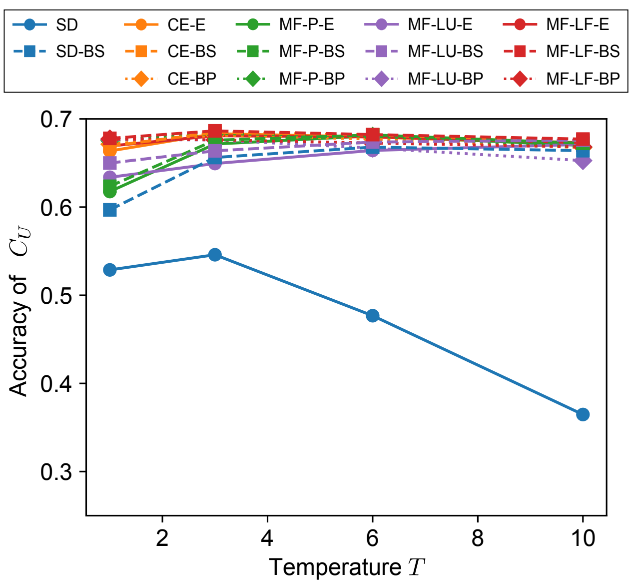


Figure 2. Sensitivity results on the temperature $T$.

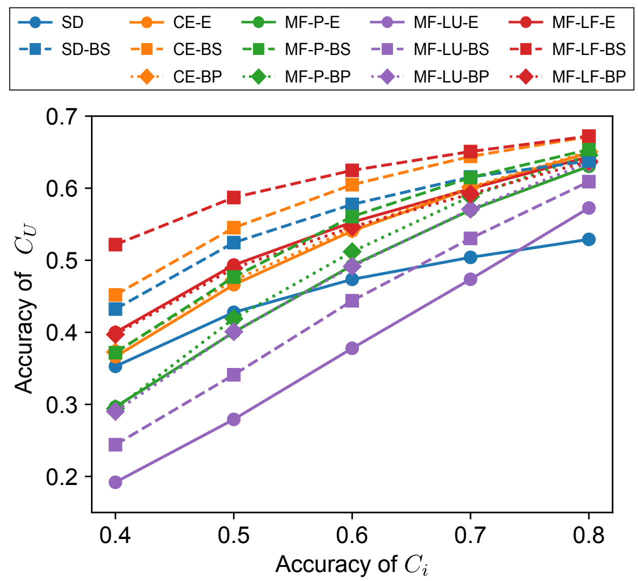


Figure 3. Sensitivity results on the accuracy of $C_i$.

## References

[1] Michael W. Berry, Murray Browne, Amy N. Langville, Paul V. Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007. 1

[2] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67, 2007. 1