Unifying Heterogeneous Classifiers with Distillation
Supplementary Material

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

Contents
This supplementary material contains the following contents.

- Sec. 1 Cross-Entropy Method and Geometric Program
- Sec. 2 Alternating Least Squares (ALS) for Matrix Factorisation Methods
- Sec. 3 Computation cost
- Sec. 4 Complete results for sensitivity analysis

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 \cite{2}. First, we rewrite $J(q)$ as follows:

\[
J(q) = -\sum_{i} \sum_{l} p_i(X = l) \log \frac{q(X = l)}{\sum_{k} q(X = k)} \quad (1)
\]

\[
= \log \frac{\prod_{l} \prod_{k} q(X = k)}{\prod_{l} \prod_{k} q(X = l) p_i(X = l)} \quad (2)
\]

Then, we can transform the following problem

\[
\text{minimise} \quad J(q) = \log \frac{\prod_{l} \prod_{k} q(X = k)}{\prod_{l} \prod_{k} q(X = l) p_i(X = l)}
\]

into

\[
\text{minimise} \quad \log \frac{\prod_{l} \prod_{k} q(X = k)}{\prod_{l} \prod_{k} q(X = l) p_i(X = l)} \quad (3)
\]

\[
\text{subject to} \quad \sum_{k} q(X = k) \leq t_i, \quad i = 1, \ldots, N, \quad (4)
\]

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

\[
\text{minimise} \quad \frac{\prod_{l} \prod_{k} q(X = l) p_i(X = l)}{\prod_{l} \prod_{k} q(X = l) p_i(X = l)} \quad (6)
\]

\[
\text{subject to} \quad \sum_{k} q(X = k) \leq t_i, \quad i = 1, \ldots, N. \quad (7)
\]

which is a geometric program with variables $q$ and $t_i$ \cite{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 $l \in L_U$ 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

\[
\hat{J}([u_l]) = -\sum_{i} \sum_{l} p_i(X = l) \left( u_l - \log \left( \sum_{k} \exp(u_k) \right) \right), \quad (8)
\]

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) \cite{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):

\[
\text{minimise} \quad \|M \odot (P - uv^\top)\|_F^2 \quad (9)
\]

\[
\text{subject to} \quad u^\top 1_L = 1 \quad (10)
\]

\[
v \geq 0_N, \quad u \geq 0_L, \quad (11)
\]
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 \( u \) and \( v \) in the cost function, resp. Steps 5 and 13 project \( u \) and \( v \) to the nonnegative orthants to satisfy the constraints in (11). Steps 7 to 10 are for normalising \( u \) 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 nonnegative numbers. For termination criteria, we terminate the algorithm if the RMSE between different iterations of \( u \) and \( v \) 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 probability space

**Input:** \( M, P \)

**Output:** \( u, v \)

1: Initialise \( v := 1_N \)
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:     \( u_j := \text{max}(0, u_j) \)
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:   \( v_i := \text{max}(0, v_i) \)
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):

\[
\text{minimise } ||M \odot (Z - uv^T - 1_L c^T)||_F^2 + \lambda(||u||_2^2 + ||v||_2^2)
\]

subject to \( v \geq 0_N \), \( \forall i \)

(12) \tag{12}

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 solution of each variable, followed by suitable projection.

Algorithm 2 Matrix factorisation in logit space

**Input:** \( M, Z, \lambda \)

**Output:** \( u, v, c \)

1: Initialise \( c_i := \left( \sum_{j=1}^{L} M_{ji} Z_{ji} \right) / \left( \sum_{j=1}^{L} M_{ji} \right), \forall i \)
2: Initialise \( v := 1_N \)
3: while not converged do
4:   for \( j := 1, \ldots, L \) do
5:     \( u_j := \left( \sum_{i=1}^{N} M_{ji} (Z_{ji} - c_i) v_i \right) / \left( \lambda + \sum_{i=1}^{N} M_{ji} v_i^2 \right) \)
6:   end for
7:   for \( i := 1, \ldots, N \) do
8:     \( v_i := \left( \sum_{j=1}^{L} M_{ji} (Z_{ji} - c_i) u_j \right) / \left( \lambda + \sum_{j=1}^{L} M_{ji} u_j^2 \right) \)
9:     \( v_i := \text{max}(0, v_i) \)
10: end for
11: for \( i := 1, \ldots, N \) do
12:   \( c_i := \left( \sum_{j=1}^{L} M_{ji} (Z_{ji} - u_j v_i) \right) / \left( \sum_{j=1}^{L} M_{ji} \right) \)
13: end for
14: 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\}, \( i \in U \) and \{C_i\}, (ii) estimating \( q \) from \{p_i\}, and (iii) training \( C_U \) from \( x \) 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\}, 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 \( U \); 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
