Supplementary : Extreme Multi-label Learning with Label Features for Warm-start Tagging, Ranking & Recommendation

Yashoteja Prabhu*
yashoteja.prabhu@gmail.com

Anil Kag†
anilkagak2@gmail.com

Shilpa Gopinath‡
shilpagopitvm@gmail.com

Kunal Dahiya*
kunalsdahiya@gmail.com

Shrutendra Harsola†
shharsol@microsoft.com

Rahul Agrawal†
Rahul.Agrawal@microsoft.com

Manik Varma*†
manik@microsoft.com

ACM Reference format:
Yashoteja Prabhu, Anil Kag, Shilpa Gopinath, Kunal Dahiya, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Supplementary : Extreme Multi-label Learning with Label Features for Warm-start Tagging, Ranking & Recommendation. In *Proceedings of ACM WSDM Conference, Los Angeles, CA, USA, February 2018 (WSDM'2018),* 14 pages.
https://doi.org/10.475/123_4

Section 1 presents the pseudocodes for SwiftXML training and prediction algorithms. Section 2 reports complete set of experimental results comparing SwiftXML to various baselines in terms of both propensity-scored precisions ($PSP1, PSP3, PSP5$) as well as standard precisions ($P1, P3, P5$). Section 3 shows the derivations for individual steps of the alternating minimization algorithm used for node partitioning, as well as derivations of approximations for base classifier optimizations.

1 ALGORITHMS

Algorithm 1 SwiftXML-PREDICT($\{\mathcal{T}_1, ..\mathcal{T}_T\}, \{\boldsymbol{\mu}_1, .., \boldsymbol{\mu}_L\}, \mathbf{x}, \mathbf{z}$)

for $i = 1, .., T$ **do**
  $n \leftarrow \mathcal{T}_i$.root
  while n.isleaf $\neq 1$ **do**
    $\mathbf{w}_x \leftarrow n.\mathbf{w}_x$
    $\mathbf{w}_z \leftarrow n.\mathbf{w}_z$
    if $C_x \mathbf{w}_x^\top \mathbf{x} + C_z \mathbf{w}_z^\top \mathbf{z} > 0$ **then**
      $n \leftarrow n$.left_child
    else
      $n \leftarrow n$.right_child
    end if
  end while
  $\mathbf{P}_i \leftarrow n.\mathbf{P}$
end for
$\mathbf{P}_{\text{pf}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{P}_i$
$P_{\text{tail},l} = 1/(1 + \exp(\frac{\gamma}{2} \|\mathbf{x} - \boldsymbol{\mu}_l\|^2))\ \forall l \in \{1..L\}$
$\mathbf{r} = \text{rank}_k \left(\alpha \log(\mathbf{P}_{\text{pf}}) + (1 - \alpha) \log(\mathbf{P}_{\text{tail}}) \right)$
return r

*Indian Institute of Technology Delhi
†Microsoft Research and AI
‡Samsung Research

WSDM'2018, February 2018, Los Angeles, CA, USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

Algorithm 2 SwiftXML-TRAIN($\{\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i\}_{i=1}^N, \mathbf{p}, T$)

for $i = 1, \ldots, N$ **do**
  for $l = 1, \ldots, L$ **do**
    $y_{il}^p = y_{il}/p_{il}$
  end for
end for
parallel-for $i = 1, .., T$ **do**
  $n^{root} \leftarrow$ new node
  $n^{root}.Id \leftarrow \{1, .., N\}$ # Root contains all instances
  GROW-NODE-RECURSIVE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n^{root}$)
  $\mathcal{T}_i \leftarrow$ new tree
  $\mathcal{T}_i$.root $\leftarrow n^{root}$
end parallel-for
for $l = 1, .., L$ **do**
  $\boldsymbol{\mu}_l = \frac{\sum_{i=1}^N y_{il}\mathbf{x}_i}{\sum_{i=1}^N y_{il}}$
end for
return $\{\mathcal{T}_1, .., \mathcal{T}_T\}, \{\boldsymbol{\mu}_1, .., \boldsymbol{\mu}_L\}$

procedure GROW-NODE-RECURSIVE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n$)
  if $|n.Id| \leq$ MaxLeaf **then** # Make n a leaf
    n.isleaf $\leftarrow 1$
    $n.\mathbf{P} \leftarrow$PROCESS-LEAF($\{\mathbf{y}_i^p\}_{i=1}^N, n$)
  else # Split node and grow child nodes recursively
    $\{n.\mathbf{w}_x, n.\mathbf{w}_z, n.$left_child$, n.$right_child$\}$
      $\leftarrow$ SPLIT-NODE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n$)
    GROW-NODE-RECURSIVE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n.$left_child)
    GROW-NODE-RECURSIVE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n.$right_child)
  end if
end procedure

procedure PROCESS-LEAF($\{\mathbf{y}_i^p\}_{i=1}^N, n$)
  $\mathbf{P} \leftarrow \text{top-k}\left(\frac{\sum_{i \in n.Id} \mathbf{y}_i^p}{|n.Id|}\right)$
return $\mathbf{P}$ # Return scores for top k labels
end procedure

Algorithm 3 SPLIT-NODE($\{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i\}_{i=1}^N, n$)

$Id \leftarrow n.Id$
$\delta_i[0] \sim \{-1, 1\}, \forall i \in Id$ # Random coin tosses
$\mathbf{w}_x[0] \leftarrow \mathbf{0}, \mathbf{w}_z[0] \leftarrow \mathbf{0}, t \leftarrow 0$ # Various counters
repeat
  $\mathbf{r}^{\pm}[t+1] \leftarrow \text{rank}_L\left(\sum_{i \in Id} \frac{1}{2}(1 \pm \delta_i[t]) I_L(\mathbf{y}_i^p)\mathbf{y}_i^p\right)$
  $\boldsymbol{\delta}[t+1] \leftarrow \text{FDELTA}(\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i, \delta_i[t]\}_{i=1}^N, Id)$
  $\mathbf{w}_x[t+1] \leftarrow \underset{\mathbf{w}_x}{\text{argmin}} \|\mathbf{w}_x\|_1 + C_x \sum_{i \in Id} \log(1 + e^{-\delta_i[t]\mathbf{w}_x{}^\top \mathbf{x}_i})$
  $\boldsymbol{\delta}[t+1] \leftarrow \text{FDELTA}(\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i, \delta_i[t+1]\}_{i=1}^N, Id)$
  $\mathbf{w}_z[t+1] \leftarrow \underset{\mathbf{w}_z}{\text{argmin}} \|\mathbf{w}_z\|_1 + C_z \sum_{i \in Id} \log(1 + e^{-\delta_i[t]\mathbf{w}_z{}^\top \mathbf{z}_i})$
  $\boldsymbol{\delta}[t+1] \leftarrow \text{FDELTA}(\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i, \delta_i[t+1]\}_{i=1}^N, Id)$
  $t \leftarrow t + 1$
until $\boldsymbol{\delta}[t] \neq \boldsymbol{\delta}[t-1]$ # Convergence
$n^+ \leftarrow$ new node$, n^- \leftarrow$ new node
$n^{\pm}.Id \leftarrow \left\{i \in Id : \text{sign}\{C_x \mathbf{w}_x[t]^\top \mathbf{x}_i + C_z \mathbf{w}_z[t]^\top \mathbf{z}_i\} = \pm 1\right\}$
  return $\mathbf{w}_x[t], w_z[t], n^+, n^-$

procedure FDELTA($\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \{\mathbf{x}_i, \mathbf{y}_i^p, \mathbf{z}_i, \delta_i\}_{i=1}^N, Id$)
  for $i \in Id$ **do**
    $v^{\pm} \leftarrow C_x \log(1 + e^{\mp \mathbf{w}_x[t]^\top \mathbf{x}_i})$
      $+ C_z \log(1 + e^{\mp \mathbf{w}_z[t]^\top \mathbf{z}_i})$
      $-C_r I_L(\mathbf{y}_i^p) \sum_{l=1}^{L} \left(\frac{y^p_{ir_l^{\pm}[t+1]}}{\log(1+l)}\right)$
    if $v^+ = v^-$ **then**
      $\delta_i' = \delta_i$
    else
      $\delta_i' = \text{sign}(v^- - v^+)$
    end if
  end for
return δ'
end procedure

2 RESULTS

Table 1: The proposed SwiftXML makes significantly more accurate predictions as compared to both state-of-the-art extreme classifiers and classical recommendation algorithms. SwiftXML consistently improves as more and more test labels are revealed, and achieves accuracy gains of upto 14% as compared to the baselines. Performance is evaluated using unbiased propensity-scored Precision (PSP1,PSP3,PSP5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
WRMF	8.87	9.80	11.05	12.44	13.69	16.58	13.59	15.50	19.77	13.21	18.10	22.85
SVD++	0.17	0.31	0.41	0.17	0.29	0.51	0.18	0.34	0.61	0.14	0.29	0.60
BPR	1.17	1.23	1.13	1.18	0.89	1.01	1.06	0.72	0.86	1.09	1.65	2.24
PfastreXML	43.76	45.66	48.21	41.05	42.99	48.64	39.03	42.50	51.13	33.29	44.21	52.46
SLEEC	34.14	39.14	42.72	36.16	40.52	46.31	36.01	40.79	48.56	34.64	44.63	51.64
PDSparse	34.49	40.32	43.79	34.52	39.80	45.72	32.97	37.81	46.02	31.05	42.33	49.87
DiSMEC	35.15	42.85	47.03	35.77	42.39	48.30	35.87	42.93	50.54	34.44	45.11	51.63
IMC	10.28	10.73	11.23	9.26	9.90	11.45	7.94	9.02	11.45	6.11	9.30	11.72
Matchbox	0.25	0.48	0.50	–	–	–	0.59	0.65	1.00	0.60	0.79	1.09
SwiftXML	44.49	46.13	48.46	42.83	43.56	49.72	42.27	44.72	53.12	38.52	48.18	55.70

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
WRMF	5.93	5.40	5.27	6.80	6.10	6.01	6.82	6.34	6.34	5.74	5.70	6.33
PfastreXML	22.78	20.46	19.80	20.48	18.56	18.17	17.56	16.11	16.31	13.07	13.35	14.77
SLEEC	11.10	11.92	12.42	11.21	11.94	12.57	10.92	11.51	12.28	9.83	10.58	12.14
PDSparse	9.54	8.95	8.02	8.94	7.97	6.78	7.94	6.76	5.72	6.09	5.18	4.73
DiSMEC	11.99	14.10	15.47	11.87	13.81	15.19	11.43	13.01	14.53	10.23	11.83	13.87
IMC	2.57	2.38	2.36	3.63	3.40	3.42	4.04	3.80	3.87	3.10	3.45	3.98
SwiftXML	23.10	20.63	19.92	21.30	19.35	19.07	17.75	16.60	17.06	14.17	14.49	16.23

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML	70.36	73.92	76.32	70.30	73.22	75.80	69.23	72.39	75.17	66.80	71.55	76.30
PDSparse	50.65	62.57	65.25	53.52	64.27	61.61	55.90	61.18	58.37	58.17	57.41	57.47
SwiftXML	70.40	74.44	77.17	73.89	77.94	81.10	76.37	81.00	83.77	79.78	84.31	87.83

CitationNetwork-36K [N=62K, D=39K, L=36K]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML	11.10	13.31	15.39	10.26	12.75	15.19	9.04	12.59	15.30	7.70	12.46	15.24
SLEEC	7.41	9.43	11.35	7.65	10.08	12.43	7.37	10.80	13.36	6.40	10.91	13.79
PDSparse	10.14	12.71	14.65	9.31	12.27	14.48	8.36	12.02	14.05	7.18	12.06	14.21
DiSMEC	11.94	15.11	17.84	11.22	14.66	17.78	9.94	14.72	18.06	8.81	15.02	18.49
SwiftXML	11.84	14.57	16.92	11.50	14.86	17.84	11.48	16.12	19.44	9.97	15.79	19.34

Amazon-79K [$N = 490K, D = 136K, L = 79K$]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML	25.92	31.56	36.39	25.39	31.52	36.14	24.19	32.50	36.61	22.92	31.82	35.40
SLEEC	15.43	19.98	23.83	17.91	23.87	28.30	18.88	27.58	32.24	18.26	27.32	31.33
PDSparse	23.60	29.51	34.12	23.11	29.55	33.57	21.88	30.12	33.54	20.59	29.82	32.85
DiSMEC	27.13	35.15	41.89	26.67	35.34	41.94	25.53	36.54	42.54	24.43	36.31	41.86
SwiftXML	26.48	32.43	37.69	29.42	37.64	42.80	36.04	47.40	51.33	35.15	46.47	49.44

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Revealed Label Percentages

Algorithm	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML	33.76	32.00	33.34	32.17	31.19	33.35	29.38	30.27	33.22	26.26	31.39	35.22
SwiftXML	35.48	33.42	34.76	34.19	33.04	35.31	31.49	32.49	35.68	28.33	33.90	38.07

Table 2: The proposed SwiftXML makes significantly more accurate predictions as compared to both state-of-the-art extreme classifiers and classical recommendation algorithms. SwiftXML consistently improves as more and more test labels are revealed, and achieves accuracy gains of upto 14% as compared to the baselines. Performance is evaluated using unbiased propensity-scored nDCG (PSN1,PSN3,PSN5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	20% PSN1	20% PSN3	20% PSN5	40% PSN1	40% PSN3	40% PSN5	60% PSN1	60% PSN3	60% PSN5	80% PSN1	80% PSN3	80% PSN5
WRMF	8.87	9.54	10.29	12.44	13.31	14.90	13.59	14.81	16.98	13.21	16.16	18.30
SVD++	0.17	0.26	0.32	0.17	0.25	0.37	0.18	0.27	0.41	0.14	0.22	0.36
BPR	1.17	1.22	1.16	1.18	0.97	1.03	1.06	0.81	0.89	1.09	1.42	1.68
PfastreXML	43.76	45.16	46.67	41.05	42.43	45.53	39.03	41.19	45.57	33.29	39.81	43.54
SLEEC	34.14	37.82	40.05	36.16	39.31	42.54	36.01	39.16	43.11	34.64	40.63	43.82
PDSparse	34.49	38.73	40.93	34.52	38.34	41.65	32.97	36.16	40.33	31.05	37.89	41.29
DiSMEC	35.15	40.75	43.45	35.77	40.53	43.87	35.87	40.63	44.51	34.44	40.95	43.91
IMC	10.28	10.65	10.94	9.26	9.75	10.58	7.94	8.67	9.90	6.11	8.06	9.15
SwiftXML	44.49	45.67	47.04	42.83	43.40	46.75	42.27	43.77	48.02	38.52	44.33	47.73

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	20% PSN1	20% PSN3	20% PSN5	40% PSN1	40% PSN3	40% PSN5	60% PSN1	60% PSN3	60% PSN5	80% PSN1	80% PSN3	80% PSN5
WRMF	5.93	5.53	5.43	6.80	6.26	6.18	6.82	6.45	6.43	5.74	5.68	6.03
PfastreXML	22.78	21.04	20.50	20.48	19.05	18.72	17.56	16.49	16.54	13.07	13.23	14.01
SLEEC	11.10	11.73	12.08	11.21	11.76	12.18	10.92	11.34	11.83	9.83	10.33	11.20
PDSparse	9.54	9.11	8.50	8.94	8.23	7.46	7.94	7.08	6.42	6.09	5.45	5.18
DiSMEC	11.99	13.56	14.53	11.87	13.31	14.24	11.43	12.59	13.55	10.23	11.32	12.47
IMC	2.57	2.43	2.41	3.63	3.47	3.47	4.04	3.86	3.90	3.10	3.36	3.65
SwiftXML	23.10	21.23	20.66	21.30	19.84	19.57	17.75	16.87	17.10	14.17	14.36	15.32

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	20% PSN1	20% PSN3	20% PSN5	40% PSN1	40% PSN3	40% PSN5	60% PSN1	60% PSN3	60% PSN5	80% PSN1	80% PSN3	80% PSN5
PfastreXML	70.36	72.94	74.44	70.30	72.37	73.89	69.23	71.30	72.78	66.80	69.68	71.86
PDSparse	50.65	59.28	61.35	53.52	61.28	60.08	55.90	59.87	58.46	58.17	57.65	57.68
SwiftXML	70.40	73.35	75.05	73.89	76.78	78.65	76.37	79.48	80.98	79.78	82.60	84.22

CitationNetwork-36K [N=62K, D=39K, L=36K]

Algorithm	20% PSN1	20% PSN3	20% PSN5	40% PSN1	40% PSN3	40% PSN5	60% PSN1	60% PSN3	60% PSN5	80% PSN1	80% PSN3	80% PSN5
PfastreXML	11.10	12.61	13.76	10.26	11.89	13.16	9.04	11.29	12.59	7.70	10.54	11.74
SLEEC	7.41	8.78	9.85	7.65	9.25	10.48	7.37	9.55	10.78	6.40	9.07	10.32
PDSparse	10.14	11.89	12.98	9.31	11.26	12.27	8.36	10.67	11.66	7.18	10.08	11.02
DiSMEC	11.94	14.09	15.61	11.22	13.48	15.11	9.94	12.97	14.58	8.81	12.50	14.00
SwiftXML	11.84	13.71	15.02	11.50	13.71	15.27	11.48	14.42	16.02	9.97	13.45	14.99

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	20% PSN1	20% PSN3	20% PSN5	40% PSN1	40% PSN3	40% PSN5	60% PSN1	60% PSN3	60% PSN5	80% PSN1	80% PSN3	80% PSN5
PfastreXML	33.76	32.45	33.13	32.17	31.36	32.48	29.38	29.81	31.27	26.26	29.35	31.07
SwiftXML	35.48	33.95	34.63	34.19	33.25	34.43	31.49	31.98	33.57	28.33	31.69	33.55

Table 3: The proposed SwiftXML performs consistently better, across different revealed label percentages, as compared to baseline PfastreXML extensions: PfastreXML-early and PfastreXML-late. Performance is evaluated according to the unbiased propensity scored Precisions (PSP1,PSP3,PSP5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	43.67	45.11	47.29	41.77	43.54	49.82	39.30	43.08	52.04	35.31	46.10	54.40
PfastreXML-late	43.76	45.66	48.21	42.17	43.55	49.55	39.64	43.57	52.25	31.13	40.13	46.05
SwiftXML	43.03	44.49	46.65	42.83	43.56	49.72	42.27	44.72	53.12	38.52	48.18	55.70

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	22.98	20.40	19.59	20.29	18.52	18.17	17.50	16.15	16.31	13.27	13.52	14.94
PfastreXML-late	22.78	20.46	19.80	20.63	18.47	18.03	17.56	16.11	16.31	13.20	13.28	14.65
SwiftXML	23.10	20.63	19.92	21.30	19.35	19.07	17.75	16.60	17.06	14.17	14.49	16.23

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	67.95	71.42	74.56	71.39	75.30	78.53	72.87	76.98	79.77	71.57	76.93	81.52
PfastreXML-late	69.83	73.70	76.23	70.86	74.72	78.20	72.65	77.78	81.43	73.60	80.74	85.40
SwiftXML	70.40	74.44	77.17	73.89	77.94	81.10	76.37	81.00	83.77	79.78	84.31	87.83

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	10.74	12.98	15.04	10.08	12.55	15.08	9.14	12.65	15.43	7.88	12.60	15.54
PfastreXML-late	11.11	13.57	15.73	10.92	14.16	17.12	10.53	15.27	19.11	9.25	15.73	19.86
SwiftXML	11.84	14.57	16.92	11.50	14.86	17.84	11.48	16.12	19.44	9.97	15.79	19.34

Amazon-79K [$N = 490K, D = 136K, L = 79K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	25.83	31.45	36.45	25.32	31.47	36.18	24.08	32.42	36.64	22.76	31.75	35.43
PfastreXML-late	25.20	31.43	36.71	27.18	34.80	40.42	30.35	41.22	46.71	30.14	41.55	45.98
SwiftXML	26.48	32.43	37.69	29.42	37.64	42.80	36.04	47.40	51.33	35.15	46.47	49.44

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5	PSP1	PSP3	PSP5
PfastreXML-early	34.59	32.86	34.23	33.14	32.24	34.44	30.37	31.39	34.43	27.16	32.51	36.47
PfastreXML-late	33.88	32.35	33.78	32.30	31.83	34.26	29.90	31.49	34.88	27.42	33.32	37.66
SwiftXML	35.48	33.42	34.76	34.19	33.04	35.31	31.49	32.49	35.68	28.33	33.90	38.07

Table 4: The proposed SwiftXML performs consistently better, across different revealed label percentages, as compared to baseline PfastreXML extensions: PfastreXML-early and PfastreXML-late. Performance is evaluated according to the unbiased propensity scored nDCGs (PSN1,PSN3,PSN5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5
PfastreXML-early	43.67	44.70	45.99	41.77	43.05	46.50	39.30	41.70	46.25	35.31	44.03	45.49
PfastreXML-late	43.76	45.16	46.67	42.17	43.16	46.46	39.64	42.15	46.56	31.13	36.64	39.30
SwiftXML	44.49	45.67	47.04	42.83	43.40	46.75	42.27	43.77	48.02	38.52	44.33	47.73

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5
PfastreXML-early	22.98	21.03	20.39	20.29	18.95	18.66	17.50	16.48	16.53	13.27	13.41	14.19
PfastreXML-late	22.78	21.04	20.50	20.63	19.01	18.64	17.56	16.49	16.54	13.20	13.21	13.96
SwiftXML	23.10	21.23	20.66	21.30	19.84	19.57	17.75	16.87	17.10	14.17	14.36	15.32

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5
PfastreXML-early	67.95	70.45	72.38	71.39	74.16	76.07	72.87	75.62	77.12	71.57	76.08	76.97
PfastreXML-late	69.83	72.65	74.22	70.86	73.61	75.66	72.65	76.04	78.00	73.60	78.06	80.21
SwiftXML	70.40	73.35	75.05	73.89	76.78	78.65	76.37	79.48	80.98	79.78	82.60	84.22

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5
PfastreXML-early	10.74	12.27	13.41	10.08	11.71	13.03	9.14	11.38	12.71	7.88	10.72	11.98
PfastreXML-late	11.11	12.79	13.98	10.92	13.06	14.61	10.53	13.52	15.36	9.25	13.12	14.89
SwiftXML	11.84	13.71	15.02	11.50	13.71	15.27	11.48	14.42	16.02	9.97	13.45	14.99

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5	PSN1	PSN3	PSN5
PfastreXML-early	34.59	33.29	33.99	33.14	32.38	33.53	30.37	30.88	32.39	27.16	30.39	32.16
PfastreXML-late	33.88	32.35	32.72	32.30	31.84	33.12	29.90	30.79	32.48	27.42	30.99	32.92
SwiftXML	35.48	33.95	34.63	34.19	33.25	34.43	31.49	31.98	33.57	28.33	31.69	33.55

Table 5: The proposed SwiftXML makes significantly more accurate predictions as compared to both state-of-the-art extreme classifiers as well as classical recommendation algorithms. SwiftXML consistently improves as more and more test labels are revealed, and achieves accuracy gains of upto 3% as compared to the baselines. Performance is evaluated using standard precisions (P1,P3,P5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
WRMF	22.71	17.20	14.27	28.77	21.02	16.71	27.80	18.66	14.17	22.11	12.95	9.69
SVD++	0.42	0.56	0.55	0.39	0.49	0.56	0.37	0.44	0.49	0.24	0.23	0.30
BPR	3.87	2.83	1.93	3.57	1.81	1.34	2.79	1.13	0.79	2.23	1.49	1.20
PfastreXML	67.52	56.94	47.50	61.41	49.81	38.56	53.08	38.99	29.03	38.86	24.20	17.54
SLEEC	72.72	57.63	46.01	70.73	52.31	39.16	62.77	41.66	29.52	49.93	27.36	18.53
PDSparse	71.44	57.13	45.84	65.42	49.52	37.39	55.21	37.32	26.96	43.50	24.84	17.34
DiSMEC	78.10	64.27	51.17	72.59	55.81	40.99	64.19	44.05	30.39	50.85	27.72	18.42
IMC	29.38	20.59	15.86	24.26	16.78	12.64	18.54	12.21	9.14	11.76	7.52	5.54
Matchbox	0.58	0.78	0.59	–	–	–	1.26	0.85	0.73	1.08	0.60	0.49
SwiftXML	67.58	55.02	45.40	64.53	49.59	38.78	58.15	40.90	29.87	47.02	26.74	18.65

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
WRMF	35.50	25.82	21.03	37.94	26.26	20.86	34.57	23.08	17.67	23.78	14.72	11.10
PfastreXML	62.59	52.47	46.23	55.20	44.31	38.34	44.68	33.84	28.43	29.55	20.91	17.02
SLEEC	78.88	64.35	53.29	73.61	57.07	46.29	63.13	45.31	35.50	44.60	28.17	21.05
PDSparse	75.47	54.87	41.11	65.90	44.22	30.68	52.75	31.91	20.93	32.68	17.39	10.95
DiSMEC	80.52	68.38	58.62	73.34	58.91	49.05	62.14	45.87	36.59	42.99	28.17	21.31
IMC	5.65	4.98	4.65	6.18	5.15	4.57	6.03	4.65	3.95	3.72	2.95	2.46
SwiftXML	60.85	51.15	45.48	55.09	44.97	39.27	47.85	36.83	30.96	30.83	22.32	18.06

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
PfastreXML	85.42	74.82	60.20	82.93	68.60	49.56	78.37	53.00	36.23	71.02	33.53	21.88
PDSparse	87.91	72.01	54.23	84.46	64.59	42.27	78.18	47.19	29.39	67.21	28.22	17.25
SwiftXML	86.69	76.08	61.12	88.03	73.11	52.94	86.73	59.12	40.17	84.81	39.28	24.96

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
PfastreXML	18.61	13.92	11.12	16.59	11.93	9.35	13.55	9.30	7.13	10.60	6.75	5.05
SLEEC	15.61	10.56	8.32	15.25	9.95	7.64	13.22	8.22	6.11	10.19	6.00	4.41
PDSparse	19.02	13.27	10.16	16.78	11.32	8.31	13.62	8.63	6.18	10.55	6.27	4.39
SwiftXML	20.23	15.11	12.06	19.11	13.75	10.76	17.35	11.53	8.68	13.88	8.24	6.11

Amazon-79K [$N = 490K, D = 136K, L = 79K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
PfastreXML	32.32	22.70	16.55	31.19	20.39	14.15	28.44	15.96	10.77	25.99	12.26	8.18
SLEEC	20.85	14.72	10.89	23.43	15.62	11.02	23.22	13.52	9.32	21.33	10.43	7.07
PDSparse	30.06	20.95	15.17	28.78	18.78	12.83	25.87	14.50	9.65	23.39	11.24	7.40
DiSMEC	35.26	25.02	18.53	33.88	22.49	15.94	30.68	17.59	12.14	27.99	13.65	9.34
SwiftXML	33.21	27.70	17.05	35.88	23.86	16.35	39.27	21.61	14.06	36.90	16.52	10.56

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	Revealed Label Percentages 20%			40%			60%			80%		
	P1	P3	P5	P1	P3	P5	P1	P3	P5	P1	P3	P5
PfastreXML	57.78	38.14	28.62	52.20	32.48	23.70	43.53	24.59	17.49	33.40	16.69	11.48
SwiftXML	59.58	39.07	29.21	54.54	33.66	24.44	45.95	25.76	18.22	35.48	17.51	11.99

Table 6: The proposed SwiftXML makes significantly more accurate predictions as compared to both state-of-the-art extreme classifiers as well as classical recommendation algorithms. SwiftXML consistently improves as more and more test labels are revealed, and achieves accuracy gains of upto 3% as compared to the baselines. Performance is evaluated using standard nDCG metrics (N1,N3,N5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	N1	N3	N5	N1	N3	N5	N1	N3	N5	N1	N3	N5
WRMF	22.71	18.43	16.84	28.77	22.99	24.05	27.80	23.34	26.27	22.11	23.76	26.66
SVD++	0.42	0.52	0.56	0.39	0.45	0.65	0.37	0.47	0.72	0.24	0.36	0.61
BPR	3.87	3.07	2.47	3.57	2.20	2.17	2.79	1.64	1.74	2.23	2.49	2.95
PfastreXML	67.52	59.72	57.08	61.41	53.63	55.69	53.08	48.35	53.97	38.86	44.71	49.22
SLEEC	72.72	61.48	56.87	70.73	57.68	58.45	62.77	53.06	57.42	49.93	52.25	55.47
PDSparse	71.44	60.74	56.61	65.42	54.37	55.52	55.21	47.54	52.01	43.50	47.14	50.76
DiSMEC	78.10	67.83	62.94	72.59	60.93	61.30	64.19	55.75	59.57	50.85	53.32	56.06
IMC	29.38	22.71	20.40	24.26	18.80	18.99	18.54	15.47	17.28	11.76	13.57	15.20
SwiftXML	67.58	58.19	55.15	64.53	54.16	56.44	58.15	51.15	56.39	47.02	50.48	54.39

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	N1	N3	N5	N1	N3	N5	N1	N3	N5	N1	N3	N5
WRMF	35.50	27.98	24.08	37.94	28.81	24.41	34.57	25.59	21.45	23.78	17.00	15.60
PfastreXML	62.59	54.77	49.87	55.20	46.80	42.11	44.68	36.31	32.49	29.55	23.45	22.98
SLEEC	78.88	67.81	59.27	73.61	60.94	52.55	63.13	49.38	42.18	44.60	32.58	30.11
PDSparse	75.47	59.54	48.83	65.90	49.10	38.51	52.75	36.56	28.15	32.68	121.25	17.98
DiSMEC	80.52	71.22	63.78	73.34	62.27	54.68	62.14	49.65	42.94	42.99	32.27	30.02
IMC	5.65	5.15	4.88	6.18	5.41	4.94	6.03	4.97	4.45	3.72	3.20	3.10
SwiftXML	60.85	53.32	48.85	55.09	47.27	42.81	47.85	39.30	35.21	30.83	24.91	24.32

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	N1	N3	N5	N1	N3	N5	N1	N3	N5	N1	N3	N5
PfastreXML	85.42	82.35	82.49	82.93	79.65	81.12	78.37	77.59	79.41	71.02	75.06	77.26
PDSparse	87.91	80.72	77.94	84.46	76.75	74.74	78.18	72.28	71.49	67.21	67.90	67.97
SwiftXML	86.69	83.53	83.35	88.03	84.40	85.73	86.73	85.77	87.17	84.81	87.23	88.64

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	N1	N3	N5	N1	N3	N5	N1	N3	N5	N1	N3	N5
PfastreXML	18.61	18.39	19.33	16.59	16.95	18.41	13.55	15.64	17.36	10.60	14.57	16.34
SLEEC	15.61	13.94	14.25	15.25	14.00	14.84	13.22	13.85	15.05	10.19	13.04	14.43
PDSparse	19.02	17.84	18.35	16.78	16.40	17.30	13.62	14.97	16.12	10.55	13.89	15.08
SwiftXML	20.23	19.87	20.78	19.11	19.23	20.76	17.35	19.15	20.95	13.88	17.86	19.83

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	Revealed Label Percentages											
	20%			40%			60%			80%		
	N1	N3	N5	N1	N3	N5	N1	N3	N5	N1	N3	N5
PfastreXML	57.78	48.61	47.23	52.20	44.41	44.59	43.53	40.25	41.56	33.40	36.45	38.50
SwiftXML	59.58	49.75	48.11	54.54	45.93	45.91	45.95	41.95	43.17	35.48	38.19	40.23

Table 7: The proposed SwiftXML performs consistently better, across different revealed label percentages, as compared to baseline PfastreXML extensions which make use of label features. Performance is evaluated according to the standard Precisions (P1,P3,P5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	66.79	55.23	46.14	61.54	49.31	38.98	53.71	39.27	29.23	41.56	25.45	18.18
PfastreXML-late	67.52	56.94	47.50	62.09	49.51	38.78	53.66	39.52	29.16	37.36	22.09	15.42
SwiftXML	67.58	55.02	45.40	64.53	49.59	38.78	58.15	40.90	29.87	47.02	26.74	18.65

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	61.59	51.19	44.96	54.94	44.51	38.46	44.53	34.10	28.58	30.74	21.55	17.36
PfastreXML-late	62.59	52.47	46.23	53.69	43.13	37.16	44.68	33.84	28.43	30.18	20.93	17.01
SwiftXML	60.85	51.15	45.48	55.09	44.97	39.27	47.85	36.83	30.96	30.83	22.32	18.06

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	80.36	70.99	58.12	82.82	70.01	51.09	82.13	56.20	38.36	76.00	36.00	23.34
PfastreXML-late	85.92	75.21	60.43	85.39	70.50	51.30	83.61	56.97	39.14	78.50	37.67	24.31
SwiftXML	86.69	76.08	61.12	88.03	73.11	52.94	86.73	59.12	40.17	84.81	39.28	24.96

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	18.36	13.72	10.97	16.62	11.87	9.36	14.02	9.44	7.23	11.00	6.88	5.17
PfastreXML-late	19.31	14.12	11.14	17.31	12.54	9.85	14.79	10.24	8.02	11.87	7.66	5.84
SwiftXML	20.23	15.11	12.06	19.11	13.75	10.76	17.35	11.53	8.68	13.88	8.24	6.11

Amazon-79K [$N = 490K, D = 136K, L = 79K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	32.30	22.68	16.58	31.19	20.41	14.18	28.42	15.94	10.79	25.90	12.26	8.20
PfastreXML-late	31.44	22.48	16.54	32.92	22.08	15.47	34.08	19.33	13.10	32.46	15.19	10.07
SwiftXML	33.21	27.70	17.05	35.88	23.86	16.35	39.27	21.61	14.06	36.90	16.52	10.56

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	20% P1	20% P3	20% P5	40% P1	40% P3	40% P5	60% P1	60% P3	60% P5	80% P1	80% P3	80% P5
PfastreXML-early	57.81	38.63	29.06	52.62	33.19	24.22	44.25	25.26	17.95	34.11	17.13	11.78
PfastreXML-late	57.11	38.30	28.83	52.24	33.16	24.27	44.43	25.58	18.24	34.94	17.62	12.13
SwiftXML	59.58	39.07	29.21	54.54	33.66	24.44	45.95	25.76	18.22	35.48	17.51	11.99

Table 8: The proposed SwiftXML performs consistently better, across different revealed label percentages, as compared to baseline PfastreXML extensions which make use of label features. Performance is evaluated according to the standard nDCG metrics (N1,N3,N5).

EURLex-4K [$N = 15K, D = 5K, L = 4K$]

Algorithm	20% N1	20% N3	20% N5	40% N1	40% N3	40% N5	60% N1	60% N3	60% N5	80% N1	80% N3	80% N5
PfastreXML-early	66.79	58.15	55.64	61.54	53.28	56.05	53.71	48.70	54.37	41.56	49.62	51.35
PfastreXML-late	67.52	59.72	57.08	62.09	53.55	56.03	53.66	49.00	54.43	37.36	42.89	44.31
SwiftXML	67.58	58.19	55.15	64.53	54.16	56.44	58.15	51.15	56.39	47.02	50.48	54.39

Wiki10-31K [$N = 14K, D = 101K, L = 31K$]

Algorithm	20% N1	20% N3	20% N5	40% N1	40% N3	40% N5	60% N1	60% N3	60% N5	80% N1	80% N3	80% N5
PfastreXML-early	61.59	53.53	48.63	54.94	46.86	42.16	44.53	36.45	32.59	30.74	24.22	23.56
PfastreXML-late	62.59	54.77	49.87	53.69	45.52	40.86	44.68	36.31	32.49	30.18	23.58	23.05
SwiftXML	60.85	53.32	48.85	55.09	47.27	42.81	47.85	39.30	35.21	30.83	24.91	24.32

AmazonCat-13K [$N = 1.18M, D = 203K, L = 13K$]

Algorithm	20% N1	20% N3	20% N5	40% N1	40% N3	40% N5	60% N1	60% N3	60% N5	80% N1	80% N3	80% N5
PfastreXML-early	80.36	77.68	78.26	82.82	80.28	82.09	82.13	81.82	83.54	76.00	80.25	82.31
PfastreXML-late	85.92	82.76	82.78	85.39	81.72	83.26	83.61	82.66	84.41	78.50	82.57	84.48
SwiftXML	86.69	83.53	83.35	88.03	84.40	85.73	86.73	85.77	87.17	84.81	87.23	88.64

CitationNetwork-36K [$N = 62K, D = 39K, L = 36K$]

Algorithm	20% N1	20% N3	20% N5	40% N1	40% N3	40% N5	60% N1	60% N3	60% N5	80% N1	80% N3	80% N5
PfastreXML-early	18.36	18.03	18.88	16.62	16.78	18.23	14.02	15.89	17.60	11.00	14.90	16.73
PfastreXML-late	19.31	18.91	19.72	17.31	17.83	19.32	14.79	17.08	19.12	11.87	16.33	18.50
SwiftXML	20.23	19.87	20.78	19.11	19.23	20.76	17.35	19.15	20.95	13.88	17.86	19.83

Wikipedia-500K [$N = 1.81M, D = 2.38M, L = 501K$]

Algorithm	20% N1	20% N3	20% N5	40% N1	40% N3	40% N5	60% N1	60% N3	60% N5	80% N1	80% N3	80% N5
PfastreXML-early	57.81	49.98	47.62	52.62	45.08	45.27	44.25	41.11	42.46	34.11	37.32	39.41
PfastreXML-late	57.11	48.61	47.35	52.24	44.95	45.12	44.43	41.28	42.63	34.94	38.02	40.18
SwiftXML	59.58	49.75	48.11	54.54	45.93	45.91	45.95	41.95	43.17	35.48	38.19	40.23

3 DERIVATIONS OF OPTIMIZATION ALGORITHMS

3.1 Node Partitioning Objective

Objective: SwiftXML uses the following node partitioning objective:

$$\textbf{Min}\ \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\}|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$
$$= \textbf{Min}\ \|\mathbf{w}_x\|_1 + C_x \sum_i \mathcal{L}_{\text{reg}}(\delta_i \mathbf{w}_x^\top \mathbf{x}_i) + \|\mathbf{w}_z\|_1 + C_z \sum_i \mathcal{L}_{\text{reg}}(\delta_i \mathbf{w}_z^\top \mathbf{z}_i)$$
$$+ C_r \sum_i \left(\frac{1+\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r) + \frac{1-\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y}_i^r) \right)$$
$$\textbf{w.r.t. } \mathbf{w}_x \in \mathcal{R}^D, \mathbf{w}_z \in \mathcal{R}^{D'}, \boldsymbol{\delta} \in \{-1,+1\}^L, \mathbf{r}^+, \mathbf{r}^- \in \Pi(1,L)$$
$$\textbf{where } \mathcal{L}_{\text{reg}}(x) = \log(1+e^{-x})\ \textbf{,}\ \mathcal{L}_{\text{rank}}(\mathbf{r},\mathbf{y}) = -\frac{\sum_{l=1}^{L} \frac{y_l}{p_l \log(r_l+1)}}{\sum_{l=1}^{L} \frac{1}{\log(l+1)}} \tag{1}$$

where, i enumerates the training users; $\delta_i \in \{-1,+1\}$ indicates the user assignment to either negative (right) or positive (left) partition; $\mathbf{w}_x, \mathbf{w}_z$ represent the separating hyperplanes learned in the user and item-set feature spaces; $\mathbf{r}^+$ and $\mathbf{r}^-$ represent the item ranking variables for positive and negative partitions; $\Pi(1,L)$ denotes the space of all possible rankings over the L items; C_x, C_z, C_r are SwiftXML hyper-parameters; p_l are the item propensity scores.

The above objective is optimized through an alternating minimization algorithm which alternately optimizes over one of the four classes of variables $(\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$ at a time with the others held constant.

For the following discussions, let:

$$F(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i) = C_x \mathcal{L}_{\text{reg}}(\delta_i \mathbf{w}_x^\top \mathbf{x}_i) + C_z \mathcal{L}_{\text{reg}}(\delta_i \mathbf{w}_z^\top \mathbf{z}_i)$$
$$+ C_r \left(\frac{1+\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r) + \frac{1-\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y}_i^r) \right) \tag{2}$$

Hence:

$$\textbf{Min}_{\mathbf{w}_x, \mathbf{w}_z, \boldsymbol{\delta}, \mathbf{r}^{\pm}}\ \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\}|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$
$$= \textbf{Min}_{\mathbf{w}_x, \mathbf{w}_z, \boldsymbol{\delta}, \mathbf{r}^{\pm}}\ \|\mathbf{w}_x\|_1 + \|\mathbf{w}_z\|_1 + \sum_i F(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i) \tag{3}$$

Minimization w.r.t $\boldsymbol{\delta}$: Keeping $\mathbf{r}^{\pm}, \mathbf{w}_x, \mathbf{w}_z$ constant and optimizing w.r.t $\boldsymbol{\delta}$ reduces (3) to:

$$\boldsymbol{\delta}^* = \textbf{Argmin}_{\boldsymbol{\delta} \in \{-1,+1\}^L}\ \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\}|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$
$$\equiv \boldsymbol{\delta}^* = \textbf{Argmin}_{\boldsymbol{\delta} \in \{-1,+1\}^L} \sum_i F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i)$$

Since δ_i are separable:

$$\equiv \delta_i^* = \textbf{Argmin}_{\delta_i \in \{-1,+1\}}\ F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i)$$
$$\equiv \delta_i^* = \textbf{Sign}\left(F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, +1) - F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, -1) \right)$$
$$\equiv \delta_i^* = \textbf{Sign}\left(C_x \log\left(\frac{1+e^{-\mathbf{w}_x^\top \mathbf{x}_i}}{1+e^{+\mathbf{w}_x^\top \mathbf{x}_i}}\right) + C_z \log\left(\frac{1+e^{-\mathbf{w}_z^\top \mathbf{z}_i}}{1+e^{+\mathbf{w}_z^\top \mathbf{z}_i}}\right) \right.$$
$$\left. + C_r(\mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}) - \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y})) \right)$$
$$\equiv \delta_i^* = \textbf{Sign}\left(C_x \mathbf{w}_x^\top \mathbf{x}_i + C_z \mathbf{w}_z^\top \mathbf{z}_i + C_r(\mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}) - \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y})) \right)$$

where each δ_i^* can be derived by solving the above trivial equation.

Minimization w.r.t $\mathbf{r}^{\pm}$: Keeping $\boldsymbol{\delta}, \mathbf{w}_x, \mathbf{w}_z$ constant and optimizing w.r.t $\mathbf{r}^{\pm}$ reduces (3) to:

$$\mathbf{r}^{\pm *} = \textbf{Argmin}_{\mathbf{r}^{\pm} \in \Pi(1,L)}\ \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\}|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$
$$\equiv \mathbf{r}^{\pm *} = \textbf{Argmin}_{\mathbf{r}^{\pm} \in \Pi(1,L)} \sum_i F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i|\mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i)$$

After ignoring $\mathbf{r}^{\pm}$ independent terms:

$$\equiv \mathbf{r}^{\pm *} = \textbf{Argmin}_{\mathbf{r}^{\pm} \in \Pi(1,L)} \sum_i \left(\frac{1+\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r) + \frac{1-\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y}_i^r) \right)$$

Since $\mathbf{r}^+$ and $\mathbf{r}^-$ terms are separable:

$$\mathbf{r}^{+*} = \textbf{Argmin}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_i \left(\frac{1+\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r) \right)$$

and

$$\mathbf{r}^{-*} = \textbf{Argmin}_{\mathbf{r}^- \in \Pi(1,L)} \sum_i \left(\frac{1-\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^-, \mathbf{y}_i^r) \right)$$

Now,

$$\mathbf{r}^{+*} = \textbf{Argmin}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_i \left(\frac{1+\delta_i}{2} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r) \right)$$
$$\equiv \mathbf{r}^{+*} = \textbf{Argmin}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_{i:\delta_i=1} \mathcal{L}_{\text{rank}}(\mathbf{r}^+, \mathbf{y}_i^r)$$
$$\equiv \mathbf{r}^{+*} = \textbf{Argmin}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_{i:\delta_i=1} -\frac{\sum_{l=1}^{L} \frac{y_{il}^r}{p_l \log(r_l+1)}}{\sum_{l=1}^{L} \frac{1}{\log(l+1)}}$$
$$\equiv \mathbf{r}^{+*} = \textbf{Argmax}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_{i:\delta_i=1} \sum_{l=1}^{L} \frac{y_{il}^r}{p_l \log(r_l+1)}$$
$$\equiv \mathbf{r}^{+*} = \textbf{Argmax}_{\mathbf{r}^+ \in \Pi(1,L)} \sum_{l=1}^{L} \frac{\sum_{i:\delta_i=1} y_{il}^r}{p_l \log(r_l+1)}$$
$$\equiv \mathbf{r}^{+*} = \textbf{Argmax}_{\mathbf{r}^+ \in \Pi(1,L)} \left(\sum_{i:\delta_i=+1} \tilde{\mathbf{y}}_i^r \right)^\top \mathbf{d} \tag{4}$$

where $\tilde{y}_{il}^r = \frac{y_{il}^r}{p_l}$ and $\mathbf{d}$ is an L-vector such that $d_l = 1/\log(1+r_l^+)$. Since $\mathbf{r}^+$ are permutations of $1, 2, \ldots, L$ it is clear that (4) will be maximized if r_l is chosen as the index of the l^{th} largest value in the vector $\sum_{i:\delta_i=11} \tilde{\mathbf{y}}_i$. Thus:

$$\mathbf{r}^{+^*} = \text{rank}_L \left(\sum_{i:\delta_i=+1} \tilde{\mathbf{y}}_i^r \right)$$

and through similar derivations:

$$\mathbf{r}^{-^*} = \text{rank}_L \left(\sum_{i:\delta_i=-1} \tilde{\mathbf{y}}_i^r \right)$$

Minimization w.r.t $\mathbf{w}_x$: Keeping $\boldsymbol{\delta}, \mathbf{w}_z, \mathbf{r}^{\pm}$ constant and optimizing w.r.t $\mathbf{w}_x$ reduces (3) to:

$$\mathbf{w}_x^* = \mathbf{Argmin}_{\mathbf{w}_x} \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\} | \mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$

$$\equiv \mathbf{w}_x^* = \mathbf{Argmin}_{\mathbf{w}_x} \|\mathbf{w}_x\|_1 + \sum_i F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i | \mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i)$$

After ignoring terms independent of $\mathbf{w}_x$:

$$\equiv \mathbf{w}_x^* = \mathbf{Argmin}_{\mathbf{w}_x} \|\mathbf{w}_x\|_1 + \sum_i C_x \log(1 + e^{-\delta_i \mathbf{w}_x^\top \mathbf{x}_i}) \tag{5}$$

(5) is a standard L1 regularized logistic regression problem and can be efficiently solved using Liblinear package.

Minimization w.r.t $\mathbf{w}_z$: Keeping $\boldsymbol{\delta}, \mathbf{w}_x, \mathbf{r}^{\pm}$ constant and optimizing w.r.t $\mathbf{w}_z$ reduces (3) to:

$$\mathbf{w}_z^* = \mathbf{Argmin}_{\mathbf{w}_z} \mathcal{F}(\{\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i\} | \mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \boldsymbol{\delta})$$

$$\equiv \mathbf{w}_z^* = \mathbf{Argmin}_{\mathbf{w}_z} \|\mathbf{w}_z\|_1 + \sum_i F(\mathbf{x}_i, \mathbf{y}_i^r, \mathbf{z}_i | \mathbf{w}_x, \mathbf{w}_z, \mathbf{r}^{\pm}, \delta_i)$$

After ignoring terms independent of $\mathbf{w}_z$:

$$\equiv \mathbf{w}_z^* = \mathbf{Argmin}_{\mathbf{w}_z} \|\mathbf{w}_z\|_1 + \sum_i C_z \log(1 + e^{-\delta_i \mathbf{w}_z^\top \mathbf{z}_i}) \tag{6}$$

(6) is a standard L1 regularized logistic regression problem and can be efficiently solved using Liblinear package.

3.2 Base Classifiers Optimization and Approximation

We learn compact hyperspherical decision boundaries for each label j independently, according to:

$$\begin{aligned} B_j(\mathbf{x}_i) &= 1/\left(1 + v_{ij}^{2y_{ij}^r - 1}\right) \\ \text{where } v_{ij} &= e^{\left(\frac{\lambda_x}{2}\|\mathbf{x}_i - \boldsymbol{\mu}_j^x\|_2^2 + \frac{\lambda_z}{2}\|\mathbf{z}_i - \boldsymbol{\mu}_j^z\|_2^2\right)} \end{aligned} \tag{7}$$

where, $\boldsymbol{\mu}_j^x, \boldsymbol{\mu}_j^z$ are the centroids of the hyperspherical regressors and λ_x, λ_z are the algorithm's hyperparameters.

For jth label, the optimization problem is as follows:

$$\mathbf{Min}_{\mu_x, \mu_z} \prod_{i=1}^N B_j(\mathbf{x}_i)$$

$$\equiv \mathbf{Min}_{\mu_x, \mu_z} \prod_{i=1}^N 1/\left(1 + v_{ij}^{2y_{ij}^r - 1}\right)$$

By taking negative logarithm:

$$\equiv \mathbf{Max}_{\mu_x, \mu_z}\ O = \sum_{i=1}^N \log\left(1 + v_{ij}^{2y_{ij}^r - 1}\right) \tag{8}$$

Since (8) is continuous and unconstrained, at the optimum the following conditions hold:

$\Delta_{\mu_x} O = 0$ and $\Delta_{\mu_z} O = 0$

where

$$\Delta_{\boldsymbol{\mu}_j^x} O = \sum_{i: y_{ij}^r = 1} \frac{\lambda_x v_{ij}}{1 + v_{ij}} (\boldsymbol{\mu}_j^x - \mathbf{x}_i) - \sum_{i: y_{ij}^r = 0} \frac{\lambda_x}{1 + v_{ij}} (\boldsymbol{\mu}_j^x - \mathbf{x}_i) \tag{9}$$

and

$$\Delta_{\boldsymbol{\mu}_j^z} O = \sum_{i: y_{ij}^r = 1} \frac{\lambda_z v_{ij}}{1 + v_{ij}} (\boldsymbol{\mu}_j^z - \mathbf{z}_i) - \sum_{i: y_{ij}^r = 0} \frac{\lambda_z}{1 + v_{ij}} (\boldsymbol{\mu}_j^z - \mathbf{z}_i) \tag{10}$$

We assume the following:

$$\exists \Delta \in \mathcal{R},\ \|\mathbf{x}_i - \boldsymbol{\mu}_j^x\|, \|\mathbf{z}_i - \boldsymbol{\mu}_j^z\| \geq \Delta > 0\ \forall i \in \{1, .., N\} \tag{11}$$

and

$\lambda_x, \lambda_z \gg 0$

Above assumptions imply that:

$$\lambda_x \|\mathbf{x}_i - \boldsymbol{\mu}_j^x\|^2 \geq \lambda_x \Delta^2 \gg 0 \text{ and } \lambda_z \|\mathbf{z}_i - \boldsymbol{\mu}_j^z\|^2 \geq \lambda_z \Delta^2 \gg 0$$

$$\implies v_{ij} \gg 1$$

$$\implies \Delta_{\boldsymbol{\mu}_j^x} O \approx \sum_{i: y_{ij}^r = 1} \lambda_x (\boldsymbol{\mu}_j^x - \mathbf{x}_i) = 0 \tag{12}$$

and

$$\Delta_{\boldsymbol{\mu}_j^z} O \approx \sum_{i: y_{ij}^r = 1} \lambda_z (\boldsymbol{\mu}_j^z - \mathbf{z}_i) = 0 \tag{13}$$

$$\implies \boldsymbol{\mu}_j^x \approx \frac{\sum_{i=1}^N y_{ij}^r \mathbf{x}_i}{\sum_{i=1}^N y_{ij}^r} \text{ and } \boldsymbol{\mu}_j^z \approx \frac{\sum_{i=1}^N y_{ij}^r \mathbf{z}_i}{\sum_{i=1}^N y_{ij}^r} \tag{14}$$

The above approximate values of $\boldsymbol{\mu}_l^x$ and $\boldsymbol{\mu}_l^z$ are not only efficient to calculate, but also provide good prediction performance as observed experimentally.