DECAF: Deep Extreme Classification with Label Features

Anshul Mittal
Kunal Dahiya
me@anshulmittal.org
kunalsdahiya@gmail.com
IIT Delhi
India

Sheshansh Agrawal
Deepak Saini
sheshansh.agrawal@microsoft.com
desaini@microsoft.com
Microsoft Research
India

Sumeet Agarwal
sumeet@iitd.ac.in
IIT Delhi
India

Purushottam Kar
purushot@cse.iitk.ac.in
IIT Kanpur
Microsoft Research
India

Manik Varma
manik@microsoft.com
Microsoft Research
IIT Delhi
India

ABSTRACT

Extreme multi-label classification (XML) involves tagging a data point with its most relevant subset of labels from an extremely large label set, with several applications such as product-to-product recommendation with millions of products. Although leading XML algorithms scale to millions of labels, they largely ignore label metadata such as textual descriptions of the labels. On the other hand, classical techniques that can utilize label metadata via representation learning using deep networks struggle in extreme settings. This paper develops the DECAF algorithm that addresses these challenges by learning models enriched by label metadata that jointly learn model parameters and feature representations using deep networks and offer accurate classification at the scale of millions of labels. DECAF makes specific contributions to model architecture design, initialization, and training, enabling it to offer up to 2-6% more accurate prediction than leading extreme classifiers on publicly available benchmark product-to-product recommendation datasets, such as LF-AmazonTitles-1.3M. At the same time, DECAF was found to be up to 22× faster at inference than leading deep extreme classifiers, which makes it suitable for real-time applications that require predictions within a few milliseconds. The code for DECAF is available at the following URL
https://github.com/Extreme-classification/DECAF.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; *Supervised learning by classification.*

KEYWORDS

Extreme multi-label classification; product to product recommendation; label features; label metadata; large-scale learning

WSDM '21, March 8–12, 2021, Virtual Event, Israel

ACM ISBN 978-1-4503-8297-7/21/03...$15.00
https://doi.org/10.1145/3437963.3441807

ACM Reference Format:
Anshul Mittal, Kunal Dahiya, Sheshansh Agrawal, Deepak Saini, Sumeet Agarwal, Purushottam Kar, and Manik Varma. 2021. DECAF: Deep Extreme Classification with Label Features. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441807

1 INTRODUCTION

Objective: Extreme multi-label classification (XML) refers to the task of tagging data points with a relevant subset of labels from an extremely large label set. This paper demonstrates that XML algorithms stand to gain significantly by incorporating label metadata. The DECAF algorithm is proposed, which could be up to 2-6% more accurate than leading XML methods such as Astec [8], MACH [23], Bonsai [20], AttentionXML [38], *etc*, while offering predictions within a fraction of a millisecond, which makes it suitable for high-volume and time-critical applications.

Short-text applications: Applications such as predicting related products given a retail product's name [23], or predicting related webpages given a webpage title, or related searches [14], all involve short texts, with the product name, webpage title, or search query having just 3-10 words on average. In addition to the statistical and computational challenges posed by a large set of labels, short-text tasks are particularly challenging as only a few words are available per data point. This paper focuses on short-text applications such as related product and webpage recommendation.

Label metadata: Metadata for labels can be available in various forms: textual representations, label hierarchies, label taxonomies [19, 24, 31], or label correlation graphs, and can capture semantic relations between labels. For instance, the Amazon products (that serve as labels in a product-to-product recommendation task) "Panzer Dragoon", and "Panzer Dragoon Orta" do not share any common training point but are semantically related. Label metadata can allow collaborative learning, which especially benefits *tail* labels. Tail labels are those for which very few training points are available and form the majority of labels in XML applications [2, 3, 15]. For instance, just 14 documents are tagged with the label "Panzer Dragoon Orta" while 23 documents are tagged with the label "Panzer Dragoon" in the LF-AmazonTitles-131K dataset. In this paper, we will focus on utilizing label text as a form of label metadata.

DECAF: DECAF learns a separate linear classifier per label based on the 1-vs-All approach. These classifiers critically utilize label metadata and require careful initialization since random initialization [10] leads to inferior performance at extreme scales. DECAF proposes using a *shortlister* with large fanout to cut down training and prediction time drastically. Specifically, given a training set of N examples, L labels, and D dimensional embeddings being learnt, the use of the shortlister brings training time down from $\mathcal{O}(NDL)$ to $\mathcal{O}(ND \log L)$ (by training only on the $\mathcal{O}(\log L)$ most confusing negative labels for every training point), and prediction time down from $\mathcal{O}(DL)$ to $\mathcal{O}(D \log L)$ (by evaluating classifiers corresponding to only the $\mathcal{O}(\log L)$ most likely labels). An efficient and scalable two-stage strategy is proposed to train the shortlister.

Comparison with state-of-the-art: Experiments conducted on publicly available benchmark datasets revealed that DECAF could be 5% more accurate than the leading approaches such as DiSMEC [2], Parabel [29], Bonsai [20] AnnexML [33], *etc*, which utilize pre-computed features. DECAF was also found to be 2-6% more accurate than leading deep learning-based approaches such as Astec [8], AttentionXML [38] and MACH [23] that jointly learn feature representations and classifiers. Furthermore, DECAF could be up to 22× faster at prediction than deep learning methods such as MACH and AttentionXML.

Contributions: This paper presents DECAF, a scalable deep learning architecture for XML applications that effectively utilize label metadata. Specific contributions are made in designing a shortlister with a large fanout and a two-stage training strategy. DECAF also introduces a novel initialization strategy for classifiers that leads to accuracy gains, more prominently on data-scarce tail labels. DECAF scales to XML tasks with millions of labels and makes predictions significantly more accurate than state-of-the-art XML methods. Even on datasets with more than a million labels, DECAF can make predictions in a fraction of a millisecond, thereby making it suitable for real-time applications.

2 RELATED WORK

Summary: XML techniques can be categorized into 1-vs-All, tree, and embedding methods. Of these, one-vs-all methods such as Slice [14] and Parabel [29] offer the most accurate solutions. Recent advances have introduced the use of deep-learning-based representations. However, these techniques mostly do not use label metadata. Techniques such as the X-Transformer [7] that do use label text either do not scale well with millions of labels or else do not offer state-of-the-art accuracies. The DECAF method presented in this paper effectively uses label metadata to offer state-of-the-art accuracies and scale to tasks with millions of labels.

1-vs-All classifiers: 1-vs-All classifiers PPDSparse [36], DiSMEC [2], ProXML [3] are known to offer accurate predictions but risk incurring training and prediction costs that are linear in the number of labels, which is prohibitive at extreme scales. Approaches such as negative sampling, PLTs, and learned label hierarchies have been proposed to speed up training [14, 20, 29, 37], and predictions [17, 27] for 1-vs-All methods. However, they rely on sub-linear search structures such as nearest-neighbor structures or label-trees that are well suited for fixed or pre-trained features such as bag-of-words or FastText [18] but not support jointly learning deep representations since it is expensive to repeatedly update these search structures as deep-learned representations keep getting updated across learning epochs. Thus, these approaches are unable to utilize deep-learned features, which leads to inaccurate solutions. DECAF avoids these issues by its use of the *shortlister* which offers a high recall filtering of labels allowing training and prediction costs that are logarithmic in the number of labels.

Tree classifiers: Tree-based classifiers typically partition the label space to achieve logarithmic prediction complexity. In particular, MLRF [1], FastXML [30], PfastreXML [15] learn an ensemble of trees where each node in a tree is partitioned by optimizing an objective based on the Gini index or nDCG. CRAFTML [32] deploys random partitioning of features and labels to learn an ensemble of trees. However, such algorithms can be expensive in terms of training time and model size.

Deep feature representations: Recent works MACH [23], X-Transformer [7], XML-CNN [22], and AttentionXML [22] have graduated from using fixed or pre-learned features to using task-specific feature representations that can be significantly more accurate. However, CNN and attention-based mechanisms were found to be inaccurate on short-text applications (as shown in [8]) where scant information is available (3-10 tokens) for a data point. Furthermore, approaches like X-Transformer and AttentionXML that learn label-specific document representations do not scale well.

Using label metadata: Techniques that use label metadata e.g. label text include SwiftXML [28] which uses a pre-trained Word2Vec [25] model to compute label representations. However, SwiftXML is designed for *warm-start* settings where a subset of ground-truth labels for each test point is already available. This is a non-standard scenario that is beyond the scope of this paper. [11] demonstrated, using the GlaS regularizer, that modeling label correlations could lead to gains on tail labels. Siamese networks [34] are a popular framework that can learn representations so that documents and their associated labels get embedded together. Unfortunately, Siamese networks were found to be inaccurate at extreme scales. The X-Transformer method [7] uses label text to generate shortlists to speed up training and prediction. DECAF, on the other hand, makes much more direct use of label text to train the 1-vs-All label classifiers themselves and offers greater accuracy compared to X-Transformer and other XML techniques that also use label text.

3 DECAF: DEEP EXTREME CLASSIFICATION WITH LABEL FEATURES

Summary: DECAF consists of three components 1) a lightweight text embedding block suitable for short-text applications, 2) 1-vs-All classifiers per label that incorporate label text, and 3) a shortlister that offers a high recall label shortlists for data points, allowing DECAF to offer sub-millisecond prediction times even with millions of labels. This section details these components, and an approximate likelihood model with provable recovery guarantees, using which DECAF offers a highly scalable yet accurate pipeline for jointly training text embeddings and classifier parameters.

Notation: Let L be the number of labels and V be the dictionary size. Each of the N training points is presented as $(\mathbf{x}_i, \mathbf{y}_i)$. $\mathbf{x}_i \in \mathbb{R}^V$ is a bag-of-tokens representation for the i^{th} document i.e. x_{it} is the TF-IDF weight of token $t \in [V]$ in the i^{th} document. $\mathbf{y}_i \in \{-1, +1\}^L$

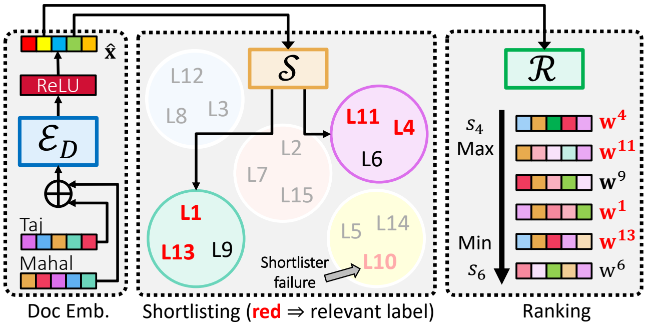

Figure 1: DECAF's frugal prediction pipeline scales to millions of labels. Given a document x, its text embedding $\hat{\mathbf{x}}$ (see Fig 3 (Left)) is first used by the shortlister $\mathcal{S}$ to shortlist the most probable $O(\log L)$ labels while maintaining high recall. The ranker $\mathcal{R}$ then uses label classifiers (see Fig 3 (Right)) of only the shortlisted labels to produce the final ranking.

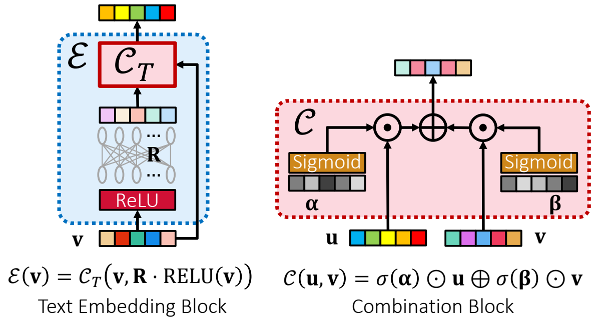
$\mathcal{E}$
$\mathcal{C}_T$
$\mathbf{R}$
ReLU
$\mathbf{v}$
$\mathcal{E}(\mathbf{v}) = \mathcal{C}_T(\mathbf{v}, \mathbf{R} \cdot \text{RELU}(\mathbf{v}))$
Text Embedding Block
$\mathcal{C}$
Sigmoid
$\boldsymbol{\alpha}$
Sigmoid
$\boldsymbol{\beta}$
$\mathbf{u}$
$\mathbf{v}$
$\mathcal{C}(\mathbf{u}, \mathbf{v}) = \sigma(\boldsymbol{\alpha}) \odot \mathbf{u} \oplus \sigma(\boldsymbol{\beta}) \odot \mathbf{v}$
Combination Block

Figure 2: (Left) DECAF uses a lightweight architecture with a residual layer to embed both document and label text (see Fig. 3). (Right) Combination blocks are used to combine various representations (separate instances are used in the text embedding blocks ($\mathcal{E}_D, \mathcal{E}_L$) and in label classifiers ($\mathcal{C}_L$)).

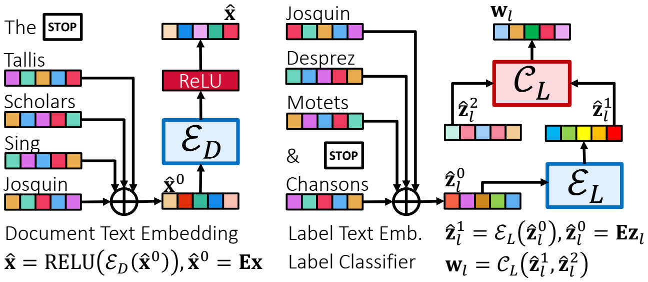
The
STOP
Tallis
Scholars
Sing
Josquin
$\hat{\mathbf{x}}$
ReLU
$\mathcal{E}_D$
$\hat{\mathbf{x}}^0$
Document Text Embedding
$\hat{\mathbf{x}} = \text{RELU}(\mathcal{E}_D(\hat{\mathbf{x}}^0)), \hat{\mathbf{x}}^0 = \mathbf{E}\mathbf{x}$
Josquin
Desprez
Motets
&
STOP
Chansons
Label Text Emb.
Label Classifier
$\mathbf{w}_l$
$\mathcal{C}_L$
$\hat{\mathbf{z}}_l^2$
$\hat{\mathbf{z}}_l^1$
$\hat{\mathbf{z}}_l^0$
$\mathcal{E}_L$
$\hat{\mathbf{z}}_l^1 = \mathcal{E}_L(\hat{\mathbf{z}}_l^0), \hat{\mathbf{z}}_l^0 = \mathbf{E}\mathbf{z}_l$
$\mathbf{w}_l = \mathcal{C}_L(\hat{\mathbf{z}}_l^1, \hat{\mathbf{z}}_l^2)$

Figure 3: (Left) Document text is embedded using an instance $\mathcal{E}_D$ of the text embedding block (see Fig. 2). Stop words (e.g. *and, the*) are discarded. (Right) DECAF critically incorporates label text into classifier learning. For each label $l \in [L]$, a one-vs-all classifier $\mathbf{w}_l$ is learnt by combining label text embedding $\hat{\mathbf{z}}_l^1$ (using a separate instance $\mathcal{E}_L$ of the text embedding block) and a refinement vector $\hat{\mathbf{z}}_l^2$. Note that $\mathcal{E}_D, \mathcal{E}_L, \mathcal{C}_L$ use separate parameters. However, all labels share the blocks $\mathcal{E}_L, \mathcal{C}_L$ and all documents share the block $\mathcal{E}_D$.

is the ground truth label vector with $y_{il} = +1$ if label $l \in [L]$ is relevant to the i^{th} document and $y_{il} = -1$ otherwise. For each label $l \in [L]$, its label text is similarly represented as $\mathbf{z}_l \in \mathbb{R}^V$.

Document and label-text embedding: DECAF learns D-dim embeddings for each vocabulary token i.e. $\mathbf{E} = [\mathbf{e}_1, \ldots, \mathbf{e}_V] \in \mathbb{R}^{D \times V}$ and uses a light-weight embedding block (see Fig 3) to encode label and document texts. The embedding block $\mathcal{E} = \{\mathbf{R}, \boldsymbol{\alpha}, \boldsymbol{\beta}\}$ is parameterized by a residual block $\mathbf{R} \in \mathbb{R}^{d \times d}$ and scaling constants $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{R}^D$ for the combination block (see Fig 2). The embedding for a bag-of-tokens vector, say $\mathbf{r} \in \mathbb{R}^V$, is $\mathcal{E}(\mathbf{r}) = \sigma(\boldsymbol{\alpha}) \odot \hat{\mathbf{r}}^0 + \sigma(\boldsymbol{\beta}) \odot (\mathbf{R} \cdot \text{ReLU}(\hat{\mathbf{r}}^0)) \in \mathbb{R}^D$ where $\hat{\mathbf{r}}^0 = \mathbf{E}\mathbf{r}$, $\odot$ denotes component-wise multiplication, and σ is the sigmoid function. Document embeddings, denoted by $\hat{\mathbf{x}}_i$, are computed as $\hat{\mathbf{x}}_i = \text{ReLU}(\mathcal{E}_D(\mathbf{x}_i))$. Label-text embeddings, denoted by $\hat{\mathbf{z}}_l^1$ are computed as $\hat{\mathbf{z}}_l^1 = \mathcal{E}_L(\mathbf{z}_l)$. Note that document and labels use separate instantiations $\mathcal{E}_D, \mathcal{E}_L$ of the embedding block. We note that DECAF could also be made to use alternate text representations such as BERT [9], attention [38], LSTM [13] or convolution [22]. However, such elaborate architectures negatively impact prediction time and moreover, DECAF outperforms BERT, CNN and attention based XML techniques on all our benchmark datasets indicating the suitability of DECAF's frugal architecture to short-text applications.

1-vs-All Label Classifiers: DECAF uses high capacity 1-vs-All (OvA) classifiers $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_L] \in \mathbb{R}^{D \times L}$ that outperform tree- and embedding-based classifiers [2, 3, 7, 14, 29, 36]. However, DECAF distinguishes itself from previous OvA works (even those such as [7] that do use label text) by directly incorporating label text into the OvA classifiers. For each label $l \in [L]$, the label-text embedding $\hat{\mathbf{z}}_l^1 = \mathcal{E}_L(\mathbf{z}_l)$ (see above) is combined with a *refinement* vector $\hat{\mathbf{z}}_l^2$ that is learnt separately per label, to produce the label classifier $\mathbf{w}_l = \sigma(\boldsymbol{\alpha}_L) \odot \hat{\mathbf{z}}_l^1 + \sigma(\boldsymbol{\beta}_L) \odot \hat{\mathbf{z}}_l^2 \in \mathbb{R}^D$ where $\boldsymbol{\alpha}_L, \boldsymbol{\beta}_L \in \mathbb{R}^D$ are shared across labels (see Fig 3). Incorporating $\hat{\mathbf{z}}_l^1$ into the label classifier $\mathbf{w}_l$ allows labels that never co-occur, but nevertheless share tokens, to perform learning in a collaborative manner since if two labels, say $l, m \in [L]$ share some token $t \in [V]$ in their respective texts, then $\mathbf{e}_t$ contributes to both $\hat{\mathbf{z}}_l^1$ and $\hat{\mathbf{z}}_m^1$. In particular, this allows rare labels to share classifier information with popular labels with which they share a token. Ablation studies (Tab 4,5,6) show that incorporating label text into classifier learning offers DECAF significant gains of over 2-6% compared to methods that do not use label text. Incorporating other forms of label metadata, such as label hierarchies, could also lead to further gains.

Shortlister: OvA training and prediction can be prohibitive, $\Omega(NDL)$ and $\Omega(DL)$ resp., if done naively. A popular way to accelerate training is to, for every data point $i \in [N]$, use only a *shortlist* containing all positive labels (that are relatively fewer around $O(\log L)$) and a small subset of the, say again $O(\log L)$, most challenging negative labels [5, 7, 14, 20, 29, 36]. This allows training to be performed in $O(ND \log L)$ time instead of $O(NDL)$ time. DECAF learns a *shortlister* $\mathcal{S}$ that offers a label-clustering based shortlisting. We have $\mathcal{S} = \{\mathcal{C}, \mathbf{H}\}$ where $\mathcal{C} = \{C_1, \ldots, C_K\}$ is a balanced clustering of the L labels and $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_K] \in \mathbb{R}^{D \times K}$ are OvA classifiers, one for each cluster. Given the embedding $\hat{\mathbf{x}}$ of a document and *beam-size* B, the top B clusters with the highest scores, say $\langle \mathbf{h}_{m_1}, \hat{\mathbf{x}} \rangle \geq \langle \mathbf{h}_{m_2}, \hat{\mathbf{x}} \rangle \geq \ldots$ are taken and labels present therein are shortlisted i.e. $\mathcal{S}(\hat{\mathbf{x}}) := \{m_1, \ldots, m_B\}$. As clusters are balanced, we get, for every datapoint, LB/K shortlisted labels in the clusters returned. DECAF uses $K = 2^{17}$ clusters for large datasets.

Prediction Pipeline: Fig 1 shows the frugal prediction pipeline adopted by DECAF. Given a document $\mathbf{x} \in \mathbb{R}^V$, its embedding $\hat{\mathbf{x}} = \text{ReLU}(\mathcal{E}_D(\mathbf{x}))$ is used by the shortlister to obtain a shortlist of B label clusters $\mathcal{S}(\hat{\mathbf{x}}) = \{m_1, \ldots, m_B\}$. Label scores are computed for every shortlisted label i.e. $l \in C_m, m \in \mathcal{S}(\hat{\mathbf{x}})$ by combining shortlister and OvA classifier scores as $s_l := \sigma(\langle \mathbf{w}_l, \hat{\mathbf{x}} \rangle) \cdot \sigma(\langle \mathbf{h}_m, \hat{\mathbf{x}} \rangle)$. These scores are sorted to make the final prediction. In practice, even on a dataset with 1.3 million labels, DECAF could make predictions within 0.2 ms using a GPU and 2 ms using a CPU.

3.1 Efficient Training: the DeepXML Pipeline

Summary: DECAF adopts the scalable DeepXML pipeline [8] that splits training into 4 *modules*. In summary, Module I jointly learns the token embeddings $\mathbf{E}$, the embedding modules $\mathcal{E}_D, \mathcal{E}_L$ and shortlister $\mathcal{S}$. Module II fine-tunes $\mathcal{E}_D, \mathcal{E}_L, \mathcal{S}$, and retrieves label shortlists for all data points. After performing initialization in Module III, Module IV jointly learns the OvA classifiers $\mathbf{W}$ and fine-tunes $\mathcal{E}_D, \mathcal{E}_L$ using the shortlists generated in Module II. Due to lack of space some details are provided in the supplementary material[1]

Module I: Token embeddings $\mathbf{E} \in \mathbb{R}^{D \times V}$ are randomly initialized using [12], residual blocks within the blocks $\mathcal{E}_D, \mathcal{E}_L$ are initialized to identity, and label *centroids* are created by aggregating document information for each label $l \in [L]$ as $\mathbf{c}_l = \sum_{i:y_{il}=+1} \mathbf{x}_i$. Balanced hierarchical binary clustering [29] is now done on these label centroids for 17 levels to generate K label clusters. Clustering labels using label centroids gave superior performance than using other representations such as label text $\mathbf{z}_l$. This is because the label centroid carries information from multiple documents and thus, a diverse set of tokens whereas $\mathbf{z}_l$ contains information from only a handful of tokens. The hierarchy itself is discarded and each resulting cluster is now treated as a *meta-label* that gives us a *meta* multi-label classification problem on the same training points, but with K meta-labels instead of the original L labels. Each meta label $m \in [K]$ is granted meta-label text as $\mathbf{u}_m = \sum_{l \in C_m} \mathbf{z}_l$. Each datapoint $i \in [N]$ is assigned a meta-label vector $\tilde{\mathbf{y}}_i \in \{-1, +1\}^K$ such that $\tilde{y}_{im} = +1$ if $y_{il} = +1$ for any $l \in C_m$ and $\tilde{y}_{im} = -1$ if $y_{il} = -1$ for all $l \in C_m$. OvA meta-classifiers $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_K] \in \mathbb{R}^{D \times K}$ are learnt to solve this meta multi-label problem but are constrained in Module I to be of the form $\mathbf{h}_m = \mathcal{E}_L(\mathbf{u}_m)$. This constrained form of the meta-classifier forces good token embeddings $\mathbf{E}$ to be learnt that allow meta-classification without the assistance of powerful refinement vectors. However, this form continues to allow collaborative learning among meta classifiers based on shared tokens. Module I solves the meta multi-label classification problem while jointly training $\mathcal{E}_D, \mathcal{E}_L, \mathbf{E}$ (implicitly learning $\mathbf{H}$ in the process).

Module II: The shortlister is fine-tuned in this module. Label centroids are recomputed as $\mathbf{c}_l = \sum_{i:\mathbf{y}^i_l=+1} \mathbf{E}\mathbf{x}_i$ where $\mathbf{E}$ are the task-specific token embeddings learnt in Module I. The meta multi-label classification problem is recreated using these new centroids by following the same steps outlined in Module I. Module II uses OvA meta-classifiers that are more powerful and resemble those used by DECAF. Specifically, we now have $\mathbf{h}_m = \sigma(\tilde{\boldsymbol{\alpha}}_P) \odot \hat{\mathbf{u}}^2_m + \sigma(\tilde{\boldsymbol{\beta}}_P) \odot \hat{\mathbf{u}}^1_m$ where $\hat{\mathbf{u}}^1_m = \sum_{l \in C_m} \mathcal{E}_L(\mathbf{z}_l)$ is the meta label-text embedding, $\hat{\mathbf{u}}^2_m$ are meta label-specific refinement vectors, and $C_P = \left\{\tilde{\boldsymbol{\alpha}}_P, \tilde{\boldsymbol{\beta}}_P\right\}$ is a fresh instantiating of the combination block. Module II solves the (new) meta multi-label classification problem, jointly learning $C_P, \hat{\mathbf{u}}^2_m$ (implicitly updating $\mathbf{H}$ in the process) and fine-tuning $\mathcal{E}_D, \mathcal{E}_L, \mathbf{E}$. The shortlister $\mathcal{S}$ so learnt is now used to retrieve shortlists $\mathcal{S}(\mathbf{x}_i)$ for each data point $i \in [N]$.

Module III: Residual blocks within $\mathcal{E}_D, \mathcal{E}_L$ are re-initialized to identity, $\mathcal{S}$ is frozen and combination block parameters for the OvA classifiers are initialized to $\boldsymbol{\alpha}_L = \boldsymbol{\beta}_L = \mathbf{0}$ (note that $\sigma(\mathbf{0}) = 0.5 \cdot \mathbf{1}$ where $\mathbf{1}$ is the all-ones vector). Refinement vectors for all L labels are initialized to $\hat{\mathbf{z}}^2_l = \mathbf{E}\mathbf{z}_l$. Ablation studies (see Tab 6) show that this refinement vector initialization offers performance boosts of up to 5-10% compared to random initialization as is used by existing methods such as AttentionXML [38] and the X-Transformer [7].

Module IV: This module performs learning using an approximate likelihood model. Let $\Theta = \{\mathbf{E}, \mathcal{E}_D, \mathcal{E}_L, C_L, \mathbf{W}\}$ be the model parameters in the DECAF architecture. We recall that C_L are combination blocks used to construct the OvA classifiers and meta classifiers, and $\mathbf{E}$ are the token embeddings. OvA approaches assume a likelihood decomposition such as $\mathbb{P}[\mathbf{y}_i \,|\, \mathbf{x}_i, \Theta] = \prod_{l=1}^{L} \mathbb{P}[y_{il} \,|\, \hat{\mathbf{x}}_i, \mathbf{w}_l] = \prod_{l=1}^{L} (1 + \exp(-y_{il} \cdot \langle \hat{\mathbf{x}}_i, \mathbf{w}_l \rangle))^{-1}$. Here $\hat{\mathbf{x}}_i = \text{ReLU}(\mathcal{E}_D(\mathbf{x}_i))$ is the document-text embedding and $\mathbf{w}_l$ are the OvA classifiers as shown in Fig 3. Let us abbreviate $\ell_{il}(\Theta) = \ln(1 + \exp(-y_{il} \cdot \langle \hat{\mathbf{x}}_i, \mathbf{w}_l \rangle))$. Then, our objective is to optimize $\arg\min_\Theta \mathcal{L}(\Theta)$ where

$$\mathcal{L}(\Theta) = \frac{1}{NL} \sum_{i \in [N]} \sum_{l \in [L]} \ell_{il}(\Theta)$$

However, performing the above optimization exactly is intractable and takes $\Omega(NDL)$ time. DECAF's solves this problem by instead optimizing $\arg\min_\Theta \tilde{\mathcal{L}}(\Theta \,|\, \mathcal{S})$ where

$$\tilde{\mathcal{L}}(\Theta \,|\, \mathcal{S}) = \frac{K}{NLB} \sum_{i \in [N]} \sum_{l \in \mathcal{S}(\hat{\mathbf{x}}_i)} \ell_{il}(\Theta)$$

Recall that for any document, $\mathcal{S}(\hat{\mathbf{x}}_i)$ is a shortlist of B label clusters (that give us a total of LB/K labels). Thus, the above expression contains only $NLB/K \ll NL$ terms as DECAF uses a large fanout of $K \approx$ 130K and $B \approx 100$. The result below assures us that model parameters and embeddings obtained by optimizing $\tilde{\mathcal{L}}(\Theta \,|\, \mathcal{S})$ perform well w.r.t. the original likelihood $\mathcal{L}(\Theta)$ if the dataset exhibits label sparsity, and the shortlister assures high recall.

THEOREM 3.1. *Suppose the training data has label sparsity at rate s i.e. $\sum_{i \in [N]} \sum_{l \in [L]} \mathbb{I}\{y_{il} = +1\} = s \cdot NL$ and the shortlister offers a recall rate of r on the training set i.e. $\sum_{i \in [N]} \sum_{l \in \mathcal{S}(\hat{\mathbf{x}}_i)} \mathbb{I}\{y_{il} = +1\} = rs \cdot NL$. Then if $\hat{\Theta}$ is obtained by optimizing the approximate likelihood function $\tilde{\mathcal{L}}(\Theta \,|\, \mathcal{S})$, then the following always holds*

$$\mathcal{L}(\hat{\Theta}) \leq \min_\Theta \mathcal{L}(\Theta) + \mathcal{O}\left(s(1-r)\ln(1/(s(1-r)))\right).$$

Please refer to Appendix A.1 in the supplementary material for the proof. As $s \to 0$ and $r \to 1$, the excess error term vanishes at rate at least $\sqrt{s(1-r)}$. Our XML datasets do exhibit label sparsity at rate $s \approx 10^{-5}$ and Fig 6 shows that DECAF's shortlister does offer high recall with small shortlists (80% recall with ≈ 50-sized shortlist and 85% recall with ≈ 100-sized shortlist). Since Thm 3.1 holds in the completely agnostic setting, it establishes the utility

[1] Supplementary Material Link: http://manikvarma.org/pubs/mittal21.pdf

of learning when likelihood maximization is performed only on label shortlists with high-recall. Module IV uses these shortlists to jointly learn the L OvA classifiers $\mathbf{W}$ and C_L, as well as fine-tune the embedding blocks $\mathcal{E}_D, \mathcal{E}_L$ and token embeddings $\mathbf{E}$.

Loss Function and Regularization: Modules I, II, IV use the logistic loss and the Adam [21] optimizer to train the model parameters and various refinement vectors. Residual layers used in the text embedding blocks $\mathcal{E}_D, \mathcal{E}_L$ were subjected to spectral regularization [26]. All ReLU layers were followed by a dropout layer with 50% drop-rate in Module-I and 20% for the rest of the modules.

Ensemble Learning: DECAF learns an inexpensive ensemble of 3 instances (see Figure 5). The three instances share Module I training to promote scalability i.e. they inherit the same token embeddings. However, they carry out training Module II onwards independently. Thus, the shortlister and embedding modules get fine-tuned for each instance.

Time Complexity: Appendix A.2 in the supplementary material presents time complexity analysis for the DECAF modules.

4 EXPERIMENTS

Datasets: Experiments were conducted on product-to-product and related-webpage recommendation datasets. These were short-text tasks with only the product/webpage titles being used to perform prediction. Of these, LF-AmazonTitles-131K, LF-AmazonTitles-1.3M, and LF-WikiSeeAlsoTitles-320K are publicly available at The Extreme Classification Repository [4]. Results are also reported on two proprietary product-to-product recommendation datasets (LF-P2PTitles-300K and LF-P2PTitles-2M) mined from click logs of the Bing search engine, where a pair of products was considered similar if the Jaccard index of the set of queries which led to a click on them was found to be more than a certain threshold. We also considered some datasets' long text counterparts, namely LF-Amazon-131K and LF-WikiSeeAlso-320K, which contained the entire product/webpage descriptions. Note that LF-AmazonTitles-131K and LF-AmazonTitles-1.3M (as well as their long-text counterparts) are subsets of the standard AmazonTitles-670K and AmazonTitles-3M datasets respectively, and were created by restricting the label set to labels for which label-text was available. Please refer to Appendix A.3 and Table 7 in the supplementary material for dataset preparation details and dataset statistics.

Baseline algorithms: DECAF was compared to leading deep extreme classifiers including the X-Transformer [7], Astec [8], XT [35], AttentionXML [38], and MACH [23], as well as standard extreme classifiers based on fixed or sparse BoW features including Bonsai [20], DiSMEC [2], Parabel [29], AnnexML [33]. Slice [14]. Slice was trained with fixed FastText [6] features, while other methods used sparse BoW features. Unfortunately, GLaS [11] could not be included in the experiments as their code was not publicly available. Each baseline deep learning method was given a 12-core Intel Skylake 2.4 GHz machine with 4 Nvidia V100 GPUs. However, DECAF was offered a 6-core Intel Skylake 2.4 GHz machine with a single Nvidia V100 GPU. A training timeout of 1 week was set for every method. Please refer to Table 9 in the supplementary material for more details.

Table 1: Results on publicly available short-text datasets. DECAF was found to be 2–6% more accurate, as well as an order of magnitude faster at prediction compared to other deep learning based approaches. Algorithms marked with a '-' were unable to scale on the given dataset within available resources and timeout period. Prediction times for DECAF within parenthesis indicate those obtained on a CPU whereas those outside parentheses are times on a GPU.

Method	PSP@1	PSP@5	P@1	P@5	Prediction Time (ms)
LF-AmazonTitles-131K					
DECAF	**30.85**	**41.42**	**38.4**	**18.65**	**0.1** (1.15)
Astec	29.22	39.49	37.12	18.24	2.34
AttentionXML	23.97	32.57	32.25	15.61	5.19
MACH	24.97	34.72	33.49	16.45	0.23
X-Transformer	21.72	27.09	29.95	13.07	15.38
Siamese	13.3	13.36	13.81	5.81	0.2
Parabel	23.27	32.14	32.6	15.61	0.69
Bonsai	24.75	34.86	34.11	16.63	7.49
DiSMEC	25.86	36.97	35.14	17.24	5.53
PfastreXML	26.81	34.24	32.56	16.05	2.32
XT	22.37	31.64	31.41	15.48	9.12
Slice	23.08	31.89	30.43	14.84	1.58
AnneXML	19.23	32.26	30.05	16.02	0.11
LF-WikiSeeAlsoTitles-320K					
DECAF	**16.73**	**21.01**	**25.14**	**12.86**	**0.09** (0.97)
Astec	13.69	17.5	22.72	11.43	2.67
AttentionXML	9.45	11.73	17.56	8.52	7.08
MACH	9.68	12.53	18.06	8.99	0.52
X-Transformer	-	-	-	-	-
Siamese	10.1	9.59	10.69	4.51	0.17
Parabel	9.24	11.8	17.68	8.59	0.8
Bonsai	10.69	13.79	19.31	9.55	14.82
DiSMEC	10.56	14.82	19.12	9.87	11.02
PfastreXML	12.15	13.26	17.1	8.35	2.59
XT	8.99	11.82	17.04	8.6	12.86
Slice	11.24	15.2	18.55	9.68	1.85
AnneXML	7.24	11.75	16.3	8.84	0.13
LF-AmazonTitles-1.3M					
DECAF	22.07	29.3	**50.67**	**40.35**	0.16 (1.73)
Astec	21.47	27.86	48.82	38.44	2.61
AttentionXML	15.97	22.54	45.04	36.25	29.53
MACH	9.32	13.26	35.68	28.35	2.09
X-Transformer	-	-	-	-	-
Siamese	-	-	-	-	-
Parabel	16.94	24.13	46.79	37.65	0.89
Bonsai	18.48	25.95	47.87	38.34	39.03
DiSMEC	-	-	-	-	-
PfastreXML	**28.71**	**32.51**	37.08	31.43	23.64
XT	13.67	19.06	40.6	32.01	5.94
Slice	13.8	18.89	34.8	27.71	1.45
AnneXML	15.42	21.91	47.79	36.91	**0.12**

Table 2: Results on proprietary product-to-product (P2P) recommendation datasets. C@20 denotes label coverage offered by the top 20 predictions of each method. DECAF offers significantly better accuracies than all competing methods. Other methods such as AnnexML and DiSMEC did not scale with available resources within the timeout period.

Method	PSP@1	PSP@5	P@1	P@5	C@20
LF-P2PTitles-300K					
DECAF	**42.43**	**62.3**	**47.17**	22.69	95.32
Astec	39.44	57.83	44.30	21.56	95.61
Parabel	37.26	55.32	43.14	20.99	95.59
PfastreXML	35.79	49.9	39.4	18.77	87.91
Slice	27.03	34.95	31.27	**25.19**	95.06
LF-P2PTitles-2M					
DECAF	**36.65**	**45.15**	**40.27**	**31.45**	93.08
Astec	32.75	41	36.34	28.74	95.3
Parabel	30.21	38.46	35.26	28.06	92.82
PfastreXML	28.84	35.65	30.52	24.6	88.05
Slice	27.03	34.95	31.27	25.19	93.43

Evaluation: Standard extreme classification metrics [3, 22, 29, 30, 38], namely Precision (P@k) and propensity scored precision (PSP@k) for $k = 1, 3, 5$ were used and are detailed in Appendix A.4 in the supplementary material.

Hyperparameters: DECAF has two tuneable hyperparameters a) beam-width B which determines the shortlist length LB/K and b) token embedding dimension D. B was chosen after concluding Module II training by setting a value that ensured a recall of $> 85\%$ on the training set (note that choosing $B = K$ trivially ensures 100% recall). Doing so did not require DECAF to re-train Module II yet ensured a high quality shortlisting. Token embedding dimension D was kept at 512 for larger datasets to improve the network capacity for large output spaces. For the small dataset LF-AmazonTitles-131K, clusters size K was kept at 2^{15} and for other datasets it was kept at 2^{17}. All other hyperparameters including learning rate, number of epochs were set to their default values across all datasets. Please refer to Table 8 in the supplementary material for details.

Results on public datasets: Table 1 compares DECAF with leading XML algorithms on short-text product-to-product and related-webpage tasks. For details as well as results on long-text versions of these datasets, please refer to Table 9 in the supplementary material. Furthermore, although DECAF focuses on product-to-product applications, results on product-to-category style datasets such as product-to-category prediction on Amazon or article-to-category prediction on Wikipedia are reported in Table 10 in the supplementary material. Parabel [28], Bonsai [20], AttentionXML [38] and X-Transformer [7] are the most relevant methods to DECAF as they shortlist labels based on a tree learned in the label centroid space. DECAF was found to be 4 − 10% more accurate than methods such as Slice [14], PfastreXML [16], DiSMEC [2], and AnnexML [33] that use fixed or pre-learnt features. This demonstrates that learning tasks-specific features can lead to significantly more accurate predictions. DECAF was also compared with other leading deep learning based approaches like MACH [23], and XT [35]. DECAF could be up to 7% more accurate while being more than 150× faster at prediction as compared to attention based models like X-Transformer and AttentionXML. DECAF was also compared to Siamese networks that had similar access to label metadata as DECAF. However, DECAF could be up to 15% more accurate than a Siamese network at an extreme scale. DECAF was also compared to Astec [8] that was specifically designed for short-text applications but does not utilize label metadata. DECAF could be up to 3% more accurate than Astec. This further supports DECAF's claim of using label meta-data for improving prediction accuracy. Even on long-text tasks such as the LF-WikiSeeAlso-320K dataset (please refer to Table 9 in the supplementary material), DECAF can be more

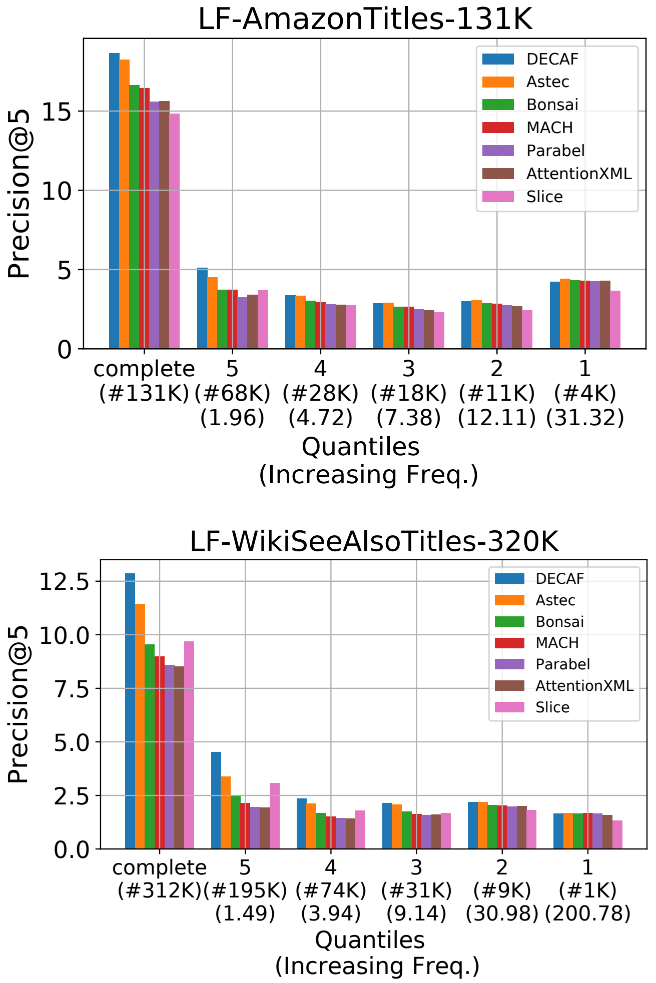

Figure 4: Quantile analysis of gains offered by DECAF in terms of contribution to P@5. The label set was divided into 5 equi-voluminous bins with increasing label frequency. Quantiles increase in mean label frequency from left to right. DECAF consistently outperforms other methods on all bins with the difference in accuracy being more prominent on bins containing data-scarce tail labels (e.g. bin 5).

Table 3: DECAF's predictions on selected test points. Document and label names ending in "…" were abbreviated due to lack of space. Please refer to Table 12 in the supplementary material for the complete table. Predictions in black and a non-bold/non-italic font were a part of the ground truth. Those in bold italics were part of the ground truth but never seen with other the ground truth labels in the training set i.e. had no common training points. Predictions in light gray were not a part of the ground truth. DECAF's exploits label metadata to discover semantically correlated labels.

Document	Top 5 predictions by DECAF
Panzer Dragoon Zwei	Panzer Dragoon, Action Replay Plus, Sega Saturn System - Video Game Console, The Legend of Dragoon , ***Panzer Dragoon Orta***
Wagner - Die Walkure …	Wagner - Siegfried …, Wagner - Gotterdammerung …, Wagner - Der Fliegende Holländer (1986), ***Wagner - Gotterdammerung*** …, Seligpreisung
New Zealand dollar	***Economy of New Zealand***, Cook Islands dollar, Politics of New Zealand , Pitcairn Islands dollar, ***Australian dollar***

accurate in propensity scored metrics compared to the second best method AttentionXML, in addition to being vastly superior in terms of prediction time. This indicates the suitability of DECAF's frugal architecture to product-to-product scenarios. The frugal architecture also allows DECAF to make predictions on a CPU within a few milliseconds even for large datasets such as LF-AmazonTitles-1.3M while other deep extreme classifiers can take an order of magnitude longer time even on a GPU. DECAF's prediction times on a CPU are reported within parentheses in Table 1.

Results on proprietary datasets: Table 2 presents results on proprietary product-to-product recommendation tasks (with details presented in Table 11 in the supplementary material). DECAF could easily scale to the LF-P2PTitles-2M dataset and be upto 2% more accurate than leading XML algorithms including Bonsai, Slice and Parabel. Unfortunately, leading deep learning algorithms such as X-Transformer could not scale to this dataset within the timeout. DECAF offers label coverage similar to state-of-the-art XML methods yet offers the best accuracy in terms of P@1. Thus, DECAF's superior predictions do not come at a cost of coverage.

Analysis: Table 3 shows specific examples of DECAF predictions. DECAF encourages collaborative learning among labels which allows it to predict the labels "Australian dollar" and "Economy of New Zealand" for the document "New Zealand dollar" when other methods failed to do so. This example was taken from the LF-WikiseeAlsoTitles-320K dataset (please refer to Table 12 in the supplementary material for details). It is notable that these labels do not share any common training instances with other ground truth labels but are semantically related nevertheless. DECAF similarly predicted a rare label "Panzer Dragoon Orta" for the (video game) product "Panzer Dragoon Zwei' whereas other algorithms failed to do so. To better understand the nature of DECAF's gains, the label set was divided into five uniform bins (quantiles) based on frequency of occurrence in the training set. DECAF's collaborative approach using label text in classifier learning led to gains in every quantile, the gains were more prominent on the data-scarce tail-labels, as demonstrated in Figure 4.

Table 4: Augmenting existing BoW-based XML methods by incorporating label metadata leads to 1.5% increase in the accuracy as compared to base method. However, DECAF could be up to 7% more accurate compared to even these.

Method	PSP@1	PSP@5	P@1	P@5
LF-AmazonTitles-131K				
DECAF	**30.85**	**41.42**	**38.4**	**18.65**
Parabel	23.27	32.14	32.6	15.61
Parabel + metadata	25.89	34.83	33.6	15.84
Bonsai	24.75	34.86	34.11	16.63
Bonsai + metadata	26.82	36.63	34.83	16.67
DiSMEC	26.25	37.15	35.14	17.24
DiSMEC + metadata	27.19	38.17	35.52	17.52
LF-WikiSeeAlsoTitles-320K				
DECAF	**16.73**	**21.01**	**25.14**	**12.86**
Parabel	9.24	11.8	17.68	8.59
Parabel + metadata	12.96	16.77	20.69	10.24
Bonsai	10.69	13.79	19.31	9.55
Bonsai + metadata	13.63	17.54	21.61	10.72
DiSMEC	10.56	14.82	19.12	9.87
DiSMEC + metadata	12.46	15.9	20.74	10.29

Incorporating metadata into baseline XML algorithms: In principle, DECAF's formulation could be deployed with existing XML algorithms wherever collaborative learning is feasible. Table 4 shows that introducing label text embeddings to the DiSMEC, Parabel, and Bonsai classifiers led to upto 1.5% gain as compared to their vanilla counterparts that do not use label text. Details of these augmentations are given in Appendix A.5 in the supplementary material. Thus, label text inclusion can lead to gains for existing methods as well. However, DECAF continues to be upto 7% more accurate than even these augmented versions. This shows that DECAF is more efficient at utilizing available label text.

Shortlister: DECAF's shortlister distinguishes itself from previous shortlisting strategies [7, 20, 29, 38] in two critical ways. Firstly, DECAF uses a massive fanout of $K = 2^{17} \approx 130$K clusters whereas existing approaches either use much fewer (upto 8K) clusters [5, 7] or use hierarchical clustering with a small fanout (upto 100) at each node [20, 38]. Secondly, in contrast to other methods that create shortlists from generic embeddings (e.g. bag-of-words or FastText [18]), DECAF fine-tunes its shortlister in Module II using task-specific embeddings learnt in Module I. Tables 5 and 6 show that DECAF's shortlister offers much better performance than shortlists computed using a small fanout or else computed using ANNS-based negative sampling [14]. Fig 6 shows that a large fanout offers much better recall even with small shortlist lengths than if using even moderate fanouts e.g. $K = 8$K.

Table 5: Using strategies used by existing XML algorithms for shortlisting labels instead of $\mathcal{S}$ hurts both both shortlist recall (R@20) and final prediction accuracy (P@k, PSP@k).

Method	PSP@1	PSP@5	P@1	P@5	R@20
			LF-AmazonTitles-131K		
DECAF	**30.85**	**41.42**	**38.4**	**18.65**	**55.86**
+ HNSW Shortlist	29.55	39.17	36.7	17.78	48.82
+ Parabel Shortlist	24.88	31.21	32.13	14.73	39.36
			LF-WikiSeeAlsoTitles-320K		
DECAF	**16.73**	**21.01**	**25.14**	**12.86**	**37.53**
+ HNSW Shortlist	15.68	19.38	23.84	12.11	30.26
+ Parabel Shortlist	13.17	15.09	21.18	10.05	23.91

Table 6: Analyzing the impact for alternative design and algorithmic choices for DECAF's components.

Component	PSP@1	PSP@5	P@1	P@5	R@20
			LF-AmazonTitles-131K		
DECAF	**30.85**	**41.42**	**38.4**	**18.65**	**55.86**
DECAF-FFT	25.5	33.38	32.42	15.43	47.23
DECAF-8K	29.07	38.7	36.29	17.52	51.65
DECAF-no-init	29.86	41.04	37.79	18.57	55.75
DECAF-$\hat{\mathbf{z}}^1$	28.02	38.38	33.5	17.09	53.83
DECAF-$\hat{\mathbf{z}}^2$	27.32	38.05	36	17.65	52.2
DECAF-lite	29.75	40.36	37.26	18.29	55.25
			LF-WikiSeeAlsoTitles-320K		
DECAF	16.73	21.01	**25.14**	**12.86**	**37.53**
DECAF-FFT	13.91	17.3	21.72	11	32.58
DECAF-8K	14.55	17.38	22.41	10.96	30.21
DECAF-no-init	15.09	19.47	23.81	12.25	36.18
DECAF-$\hat{\mathbf{z}}^1$	**18.04**	**21.48**	24.54	12.55	37.33
DECAF-$\hat{\mathbf{z}}^2$	11.55	15.24	20.82	10.53	29.72
DECAF-lite	16.59	20.84	24.87	12.78	37.24

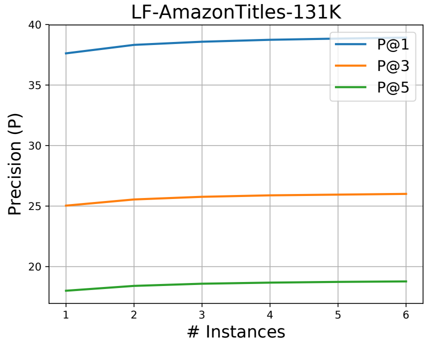

Figure 5: Impact of the number of instances in DECAF's ensemble on performance on the LF-AmazonTitles-131K dataset. DECAF offers maximum benefits using a small ensemble of 3 instances after which benefits taper off.

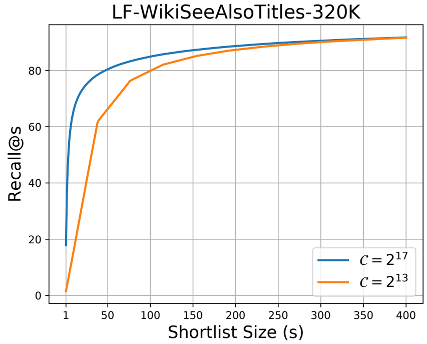

Figure 6: A comparison of recall when using moderate or large fanout on the LF-WikiSeeAlso-320K dataset. The x-axis represents various values of beam-width B and training recall offered by each. A large fanout offers superior recall with small beam width, and hence small shortlists lengths.

Ablation: As described in Section 3, the training pipeline for DECAF is divided into 4 modules mirroring the DeepXML pipeline [8]. Table 6 presents the results of extensive experiments conducted to analyze the optimality of algorithmic and design choices made in these modules. We refer to Appendix A.5 in the supplementary material for details. **a)** To assess the utility of learning task-specific token embeddings in Module I, a variant DECAF-FFT was devised that replaced these with pre-trained FastText embeddings: DECAF outperforms DECAF-FFT by 6% in PSP@1 and 3.5% in P@1. **b)** To assess the impact of a large fanout while learning the shortlister, a variant DECAF-8K was trained with a smaller fanout of $K = 2^{13} \approx$ 8K clusters that is used by methods such as AttentionXML and X-Transformer. Restricting fanout was found to hurt accuracy by 3%. This can be attributed to the fact that the classifier's final accuracy depends on the recall of the shortlister (see Theorem 3.1). Fig. 6 indicates that using $K = 2^{13}$ results in significantly larger shortlist lengths (upto 2× larger) being required to achieve the same recall as compared to using $K = 2^{17}$. Large shortlists make Module IV training and prediction more challenging, especially for large datasets involving millions of labels, thereby making a large fan-out K more beneficial. **c)** Approaches other than DECAF's shortlister $\mathcal{S}$ were considered for shortlisting labels, such as nearest neighbor search using HNSW [14] or PLTs with small fanout such as Parabel [29] learnt over dense document embeddings. Table 5 shows that both alternatives lead to significant loss, upto 15% in recall, as compared to that offered by $\mathcal{S}$. These sub-optimal shortlists eventually hurt final prediction which could be 2% less accurate as compared to DECAF. **d)** To assess the importance of label classifier initialization in Module III, a variant DECAF-no-init was tested which initialized $\hat{\mathbf{z}}_l^2$ randomly instead of with $\mathbf{E}\mathbf{z}_l$. DECAF-no-init was found to offer 1-1.5% less PSP@1 than DECAF, therefore indicating importance of proper initialization in Module III. **e)** Modules II and IV learn OvA classifiers as a combination of the label embedding vector and a refinement vector. To investigate the need for both components, Table 6 considers two DECAF variants: the first variant, named DECAF-$\hat{\mathbf{z}}^1$, discards the refinement vector in both modules i.e. using $\mathbf{w}_l = \hat{\mathbf{z}}_l^1$ and $\mathbf{h}_m = \hat{\mathbf{u}}_m^1$ whereas the second variant, named DECAF-$\hat{\mathbf{z}}^2$, rejects the label embedding component altogether and learns the OvA classifers from scratch using only the refinement vector i.e. using $\mathbf{w}_l = \hat{\mathbf{z}}_l^2$ and $\mathbf{h}_m = \hat{\mathbf{u}}_m^2$. Both variants take a hit of up to 5% in prediction accuracy as compared to DECAF. Incorporating label-text in the classifier is critical to achieve superior accuracies. **f)** Finally, to assess the utility of fine-tuning token embeddings in each successive module, a frugal version DECAF-lite was considered which freezes token embeddings after Module I and shares token embeddings among the three instances in its ensemble. DECAF-lite offers 0.5-1% loss in performance as compared to DECAF but is noticeably faster at training.

5 CONCLUSION

This paper demonstrated the impact of incorporating label metadata in the form of label text in offering significant performance gains on several product-to-product recommendation tasks. It proposed the DECAF algorithm that uses a frugal architecture, as well as a scalable prediction pipeline, to offer predictions that are up to 2-6% more accurate, as well as an order of magnitude faster, as compared to leading deep learning-based XML algorithms. DECAF offers millisecond-level prediction times on a CPU making it suitable for real-time applications such as product-to-product recommendation tasks. Future directions of work include incorporating other forms of label metadata such as label-correlation graphs, as well as diverse embedding architectures.

ACKNOWLEDGMENTS

The authors thank the IIT Delhi HPC facility for computational resources. AM is supported by a Google PhD Fellowship.

REFERENCES

[1] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. 2013. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*.

[2] R. Babbar and B. Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-label Classification. In *WSDM*.

[3] R. Babbar and B. Schölkopf. 2019. Data scarcity, robustness and extreme multi-label classification. *Machine Learning* 108 (2019), 1329–1351.

[4] K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[5] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. 2015. Sparse Local Embeddings for Extreme Multi-label Classification. In *NIPS*.

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* (2017).

[7] W-C. Chang, H.-F. Yu, K. Zhong, Y. Yang, and I. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *KDD*.

[8] K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. 2021. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *WSDM*.

[9] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

[10] X. Glorot and X. Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

[11] C. Guo, A. Mousavi, X. Wu, Daniel N. Holtmann-Rice, S. Kale, S. Reddi, and S. Kumar. 2019. Breaking the Glass Ceiling for Embedding-Based Classifiers for Large Output Spaces. In *Neurips*.

[12] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[13] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[14] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. 2019. Slice: Scalable Linear Extreme Classifiers trained on 100 Million Labels for Related Searches. In *WSDM*.

[15] H. Jain, Y. Prabhu, and M. Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking and Other Missing Label Applications. In *KDD*.

[16] V. Jain, N. Modhe, and P. Rai. 2017. Scalable Generative Models for Multi-label Learning with Missing Labels. In *ICML*.

[17] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. 2016. Extreme F-measure Maximization using Sparse Probability Estimates. In *ICML*.

[18] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the European Chapter of the Association for Computational Linguistics*.

[19] B. Kanagal, A. Ahmed, S. Pandey, V. Josifovski, J. Yuan, and L. Garcia-Pueyo. 2012. Supercharging Recommender Systems Using Taxonomies for Learning User Purchase Behavior. *VLDB* (June 2012).

[20] S. Khandagale, H. Xiao, and R. Babbar. 2020. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning* 109, 11 (2020), 2099–2119.

[21] P. D. Kingma and J. Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* (2014).

[22] J. Liu, W. Chang, Y. Wu, and Y. Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *SIGIR*.

[23] T. K. R. Medini, Q. Huang, Y. Wang, V. Mohan, and A. Shrivastava. 2019. Extreme Classification in Log Memory using Count-Min Sketch: A Case Study of Amazon Search with 50M Products. In *Neurips*.

[24] A. K. Menon, K.P. Chitrapura, S. Garg, D. Agarwal, and N. Kota. 2011. Response Prediction Using Collaborative Filtering with Hierarchies and Side-Information. In *KDD*.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NIPS*.

[26] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. 2018. Spectral Normalization for Generative Adversarial Networks. In *ICLR*.

[27] A. Niculescu-Mizil and E. Abbasnejad. 2017. Label Filters for Large Scale Multilabel Classification. In *AISTATS*.

[28] Y. Prabhu, A. Kag, S. Gopinath, K. Dahiya, S. Harsola, R. Agrawal, and M. Varma. 2018. Extreme multi-label learning with label features for warm-start tagging, ranking and recommendation. In *WSDM*.

[29] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *WWW*.

[30] Y. Prabhu and M. Varma. 2014. FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning. In *KDD*.

[31] N. Sachdeva, K. Gupta, and V. Pudi. 2018. Attentive Neural Architecture Incorporating Song Features for Music Recommendation. In *RecSys*.

[32] W. Siblini, P. Kuntz, and F. Meyer. 2018. CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning. In *ICML*.

[33] Y. Tagami. 2017. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *KDD*.

[34] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston. 2017. StarSpace: Embed All The Things! *CoRR* (2017).

[35] M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski. 2018. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *NIPS*.

[36] E.H. I. Yen, X. Huang, W. Dai, I. Ravikumar, P.and Dhillon, and E. Xing. 2017. PPDSparse: A Parallel Primal-Dual Sparse Method for Extreme Classification. In *KDD*.

[37] I. Yen, S. Kale, F. Yu, D. Holtmann R., S. Kumar, and P. Ravikumar. 2018. Loss Decomposition for Fast Learning in Large Output Spaces. In *ICML*.

[38] R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. 2019. Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *Neurips*.