arXiv:1809.06004v1 [cs.CL] 17 Sep 2018

Learning to Accept New Classes without Training

Hu Xu[1], Bing Liu[1], Lei Shu[1], Philip S. Yu[1,2]
[1]Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA
[2]Institute for Data Science, Tsinghua University, Beijing, China
{hxu48, liub, lshu3, psyu}@uic.edu

Abstract

Classic supervised learning makes the *closed-world assumption*, meaning that classes seen in testing must have been seen in training. However, in the dynamic world, new or unseen class examples may appear constantly. A model working in such an environment must be able to reject unseen classes (not seen or used in training). If enough data is collected for the unseen classes, the system should incrementally learn to accept/classify them. This learning paradigm is called *open-world learning* (OWL). Existing OWL methods all need some form of re-training to accept or include the new classes in the overall model. In this paper, we propose a meta-learning approach to the problem. Its key novelty is that it only needs to train a meta-classifier, which can then continually accept new classes when they have enough labeled data for the meta-classifier to use, and also detect/reject future unseen classes. No re-training of the meta-classifier or a new overall classifier covering all old and new classes is needed. In testing, the method only uses the examples of the seen classes (including the newly added classes) on-the-fly for classification and rejection. Experimental results demonstrate the effectiveness of the new approach [1].

1 Introduction

An AI agent working in a real-life open environment must be able to recognize the classes of things that it has seen/learned previously and detect things that it has not seen before and learn to accept the new things. This learning paradigm is called *open-world learning* (OWL) (or open-world recognition in computer vision) [1, 2]. This is in contrast to the classic learning paradigm which assumes that the classes seen in testing or real-life applications have been seen in training. This is called the *close-world assumption*. Most existing supervised learning methods solve this closed-world learning or classification problem. With the increased popularity of AI agents such as intelligent personal assistants, self-driving cars, and other robots that need to work in real-life open environments and interact with humans and other systems, the open-world learning capability is becoming critical.

For example, the very first interface for many intelligent personal assistants (such as Amazon Alexa, Google Assistant, and Microsoft Cortana, etc.) is to classify user utterances into existing known domain/intent classes (e.g., Alexa's skills, Google's actions and Cortana's skills) and also detect or reject utterances from unknown domain/intent classes. Most existing solutions to open-world learning are built on top of closed-world classification models [1, 3, 2, 4], e.g., by setting thresholds on the logits before the softmax function where unseen classes tend to mix with existing seen classes. Further, these models cannot easily add new/unseen classes to the set of seen classes without re-training or incremental training. For example, Alexa allows 3rd-party developers to add new skills (new apps), i.e., new domain or intent classes. This presents a major challenge to the maintenance of the deployed model and training data for the new classes. The existing solution, in this case, is simply to re-train the

[1]The code and data will be available on the first author's website.

Preprint. Work in progress.

whole model periodically [5]. As a result, the new skills added by the 3rd-parties may not be effective until the next scheduled re-training by Amazon. Several incrementally learning techniques (such as iCaRL [6] or DEN [7]) have been proposed to incrementally adding new classes. However, they are incapable of rejecting examples from unseen classes as existing open-world learning systems can (e.g., OSDN [3] and DOC [4]). Moreover, these incremental learning methods still need re-training or tuning the old model.

This paper proposes to solve the open-world learning problem in an entirely new way via meta-learning. Before going further, let us state what we want to achieve for OWL.

Problem Statement: At any point in time, our learning system is aware of a set of seen classes $S = \{c_1, \ldots, c_m\}$ and has an OWL model/classifier for S but is unaware of a set of unseen classes $U = \{c_{m+1}, \ldots\}$ (any class not in S can be in U) that the model may encounter. Our goal is two-fold: (1) our OWL model should classify examples (text documents in this paper) from classes in S and reject examples from all classes in U, and (2) when a new class c_{m+1} (without loss of generality) is removed from U (now $U = \{c_{m+2}, \ldots\}$) and added to S (now $S = \{c_1, \ldots, c_m, c_{m+1}\}$, our OWL model can still perform (1) with no additional training.

There are two main challenges in solving the problem. (1) How to classify examples of seen classes into their respective classes and also detect/reject examples of unseen classes. (2) How to incrementally include/accept the new/unseen classes when they have enough training data.

As indicated above, existing open-world learning methods basically adapted some existing closed-world learning techniques by setting thresholds in classification for unseen class detection. For incremental learning, they all need some form of re-training, either full re-training from scratch by using the training data of both the old and new/unseen classes [4], or partial re-training (without training from scratch) [1, 2].

Our new meta-learning based approach addresses the above challenges. The proposed framework is called learning to accept class (L2AC) that learns to build a meta-classifier to accept/classify or reject a test example by comparing it with its nearest examples from each seen class in S. Based on the comparison results, it determines whether the test example belongs to the seen class or not. If the test example is not classified as any seen class in S, it is rejected. Unlike closed-world models, the parameters of the meta-classifier are not trained from the set of seen classes. That is why the meta-classifier can potentially work with any class (seen or unseen) without being re-trained.

We can see that the proposed framework works like a nearest neighbor classifier (e.g., kNN). However, the key difference is that we train a meta-classifier to make the classification and rejection decision based on a learned metric and a learned voting mechanism.

The major advantage of the proposed approach is that it makes open-world learning a problem of maintaining the seen class set S and the (labeled) examples in each class in S. Once the meta-classifier is trained, the user/system can simply add any new class with its data to the seen class set S without re-training the meta-classifier. The system can still perform classification and rejection simply based on the updated S using the meta-classifier.

The main contributions of this paper are as follows.

1. It proposes a novel approach to open-world learning based on meta-learning, which is entirely different from existing approaches.
2. The key advantage of the approach is that with the meta-classifier, the open-world learning problem becomes simply maintaining the seen class set S because both classification and unseen class example rejection/detection are based on comparing the test example with the examples of each class in S. To be able to accept/classify any new class, we only need to put the class and its examples in S.
3. The proposed approach has been experimentally evaluated and the results show its competitive performance.

2 L2AC Framework

The overview of the L2AC framework is shown in Fig. 1, which depicts how L2AC classifies a test example into an existing seen class or rejects it as from an unseen class. Note that the training

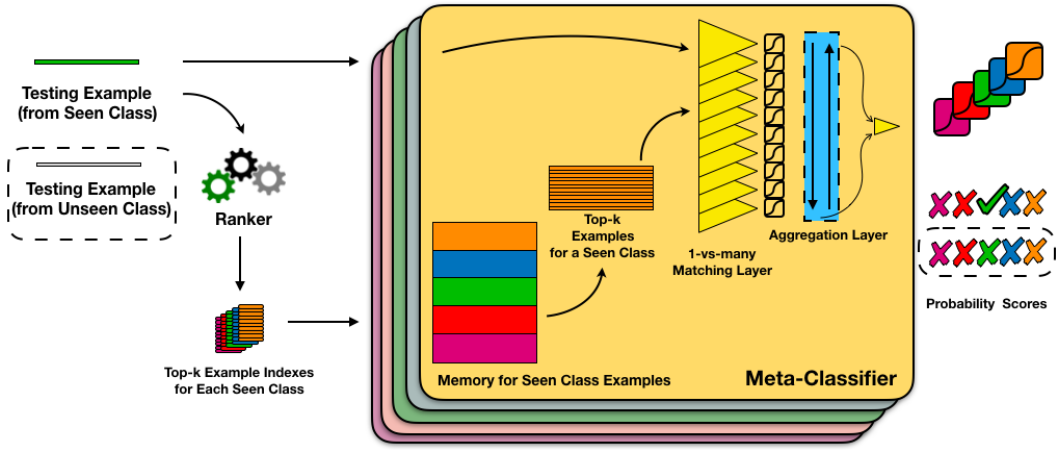


Figure 1: Overview of the L2AC framework (best viewed in colors). Assume the seen class set S has 5 classes and each class' examples are indicated by one color. L2AC has two components: a ranker and a meta-classifier. Given a (green) testing example from a seen class, the ranker first retrieves the top-k nearest examples (memory indexes) from each seen class. Then the meta-classifier takes both the test example/example and the top-k nearest examples for a seen class to produce a probability score for that class. The meta-classifier is applied 5 times (indicated by 5 rounded rectangles) over these 5 seen classes and yields 5 probability scores, where the 3rd (green) class attends the maximum score as the final class (green) prediction. However, if the test example (grey) is from an unseen class (as indicated by the dashed box), none of those probability scores from the seen classes will predict positive, which leads rejection.

process for the meta-classifier is not shown, which will be discussed in the next section. The L2AC framework has two major components: a ranker and a meta-classifier. The ranker is used to retrieve some examples from a seen class that are similar/near to the test example. The meta-classifier performs classification after it reads the retrieved examples from the seen classes. The two components work together as follows.

Assume we have a set of seen classes S. Given a test example x_t that may come from either a seen class or an unseen class, the ranker finds a list of top-k nearest examples to x_t from each seen class $c \in S$, denoted as $x_{a_{1:k}|c}$. The meta-classifier produces the probability $p(c = 1|x_t, x_{a_{1:k}|x_t,c})$ that the test x_t belongs to the seen class c based on c's top-k examples (most similar to x_t). If none of these probabilities from the seen classes S exceeds a threshold (e.g., 0.5 for the sigmoid function), the L2AC framework decides that x_t is from an unseen class (rejection); otherwise, it predicts x_t as from the seen class with the highest probability (for classification). Note that unless necessary, we denote $p(c = 1|x_t, x_{a_{1:k}|x_t,c})$ simply as $p(c|x_t, x_{a_{1:k}})$. Note also that although we also use a threshold, our threshold is on the meta-classifier that directly learns to reject rather than on an existing closed-world classifier. More importantly, our threshold is pre-fixed (not empirically setting via hyper-parameter tuning) and the meta-classifier is trained based on this fixed threshold.

As we can see, the proposed framework works like a supervised lazy learning model, such as k-nearest neighbor (kNN). Such a lazy learning model allows dynamic maintenance of a set of seen classes, where an unseen class can be easily added to the seen class set. However, the key difference is that we train a meta-classifier to make the classification and rejection decision based on a learned metric space and a learned voting mechanism for nearest examples.

Retrieving top-k nearest examples $x_{a_{1:k}}$ for a given test example x_t needs a ranking model (the ranker). We will detail an sample implementation of the ranker in Experiments. and discuss the details of the meta-classifier in the next section.

3 Meta-Classifier

Meta-classifier serves as the core component of the L2AC framework. It is a binary classifier on a seen class. It takes the top-k nearest examples (to the test example x_t) of the seen class as the input and determines whether x_t belongs to that seen class or not. In this section, we first describe how to represent examples of a seen class. Then we describe how the meta-classifier processes these examples together with the test example into an overall probability score (via a voting mechanism) for deciding whether the test example should belong to the seen class (classification) or not (rejection). Along with that we also describe how a joint decision is made for open-world classification over a set of seen classes. Finally, we describe how to train the meta-classifier via another set of meta-training classes and their examples.

3.1 Example Representation and Memory

Representation learning is a crucial part of neural networks. Following the success of using pre-trained weights from large-scale image datasets (such as ImageNet [8]) as feature encoders, we assume there is an encoder that captures almost all features for text classification.

Given an example x representing a text document (a sequence of tokens), we obtain its continuous representation (a vector) via an encoder $h = g(x)$, where the encoder $g(\cdot)$ is typically a neural network (e.g., CNN or LSTM). We will detail a simple encoder implementation in Experiments.

We save the continuous representations of examples into a memory of the meta-classifier. So later, the top-k examples can be efficiently retrieved via the index (address) in the memory. The memory is essentially a matrix $E \in \mathbb{R}^{n \times |h|}$, where n is the number of all examples from seen classes and $|h|$ is the size of the hidden dimension. Note that we will still use x instead of h to refer to an example when it is not necessary to detail the specific form of its representation. Given the test example x_t, the meta-classifier first looks up the actual continuous representations $x_{a_{1:k}}$ of the top-k examples for a seen class. Then the meta-classifier computes the similarity score between x_t and each x_{a_i} ($1 \leq i \leq k$) individually via a 1-vs-many matching layer as described next.

3.2 1-vs-many Matching Layer

To compute the overall probability score between a test example and a seen class, a 1-vs-many matching layer in the meta-classifier first computes the individual similarity score between the test example and each of the top-k retrieved examples of the seen class. The 1-vs-many matching layer essentially consists of k shared matching networks as indicated by big yellow triangles in Fig. 1. We denote each matching network as f and compute similarity scores $r_{1:k}$ for all top-k examples $r_{1:k} = f(x_t, x_{a_{1:k}})$.

The matching network first transforms the test example x_t and x_{a_i} from the continuous representation space to a single example in a similarity space. We leverage two similarity functions to obtain the similarity space. The first function is the absolute values of the element-wise subtraction: $f_{\text{abssub}}(h_t, h_{a_i}) = |h_q - h_{a_i}|$. The second one is the element-wise summation: $f_{\text{sum}}(h_t, h_{a_i}) = h_q + h_{a_i}$. Then the final similarity space is the concatenation of these two functions' results: $f_{\text{sim}}(h_t, h_{a_i}) = f_{\text{abssub}}(x_t, x_{a_i}) \oplus f_{\text{sum}}(x_t, x_{a_i})$, where $\oplus$ denotes the concatenation operation. We then pass the result to two fully-connected layers and a sigmoid function $\sigma(x) = 1/(1 + e^{-x})$. Since there are k nearest examples, we have k similarity scores denoted as $r_{1:k}$. The hyperparameters are detailed in the Experiments section.

3.3 Open-world Learning via Aggregation Layer

After getting the individual similarity scores, an aggregation layer in the meta-classifier merges the k similarity scores into a single probability score. By having the aggregation layer, the meta-classifier essentially has a *parametric voting mechanism* so that it can learn how to vote on multiple nearest examples from a seen class instead of a single example when deciding whether a test example belongs to that seen class or not. So the meta-classifier has more reliable predictions, which can be seen in the Experiments section.

One obvious choice for the aggregation layer is a (many-to-one) BiLSTM [9, 10] that can read k similarity scores and make a single prediction. We set the output size of BiLSTM to 2 (1 per direction

of LSTM). Then the output of BiLSTM is connected to a fully-connected layer followed by a sigmoid function that outputs the probability score. The computation of the meta-classifier for a given test example x_t and $x_{a_{1:k}}$ for a seen class c can be summarized as:

$$p(c|x_t, x_{a_{1:k}}) = \sigma\Big(W \text{ BiLSTM}\big(f(x_t, x_{a_{1:k}})\big) + b\Big). \tag{1}$$

Lastly, for each class $c \in S$, we evaluate Eq. 1 as:

$$\hat{y} = \begin{cases} \textit{reject}, \text{ if } \max_{c \in S} p(c|x_t, x_{a_{1:k}}) \leq 0.5; \\ \\ \arg\max_{c \in S} p(c|x_t, x_{a_{1:k}}), \text{ otherwise.} \end{cases} \tag{2}$$

If none of existing seen classes S gives a probability score above 0.5, we *reject* x_t as an example from some unseen class. To make L2AC an easily accessible approach, we use 0.5 as the threshold naturally and do not introduce an extra hyper-parameter that needs to be artificially tuned. Note that as discussed earlier, the seen class set S and its examples can be dynamically maintained (e.g., one can add to or remove any class from S). So the meta-classifier simply performs open-world classification over the current seen class set S.

3.4 Training of Meta-Classifier

Since the meta-classifier is a general classifier that is supposed to work for any class, training the meta-classifier $p_\theta(c|x_t, x_{a_{1:k}|x_t,c})$ requires examples from another set M of classes called *meta-training classes*. Note $|M|$ is typically very large to have a good coverage of different classes. This is similar to few-shot learning [11]. We also enforce $(S \cup U) \cap M = \varnothing$, so that all seen and unseen classes are totally unknown to the meta-classifier.

Next, we formulate the meta-training examples from M, which consist of a set of pairs (with positive and negative labels). The first component of a pair is a training document x_q from a class in M, and the second component is a sequence of top-k nearest examples also from a class in M.

We assume every example (document) of a class in M can be a training document x_q. Assuming x_q is from class $c \in M$, a positive training pair is $(x_q, x_{a_{1:k}|x_q,c})$, where $x_{a_{1:k}|x_q,c}$ are top-k examples from class c that are most similar or nearest to x_q; a negative training pair is $(x_q, x_{a_{1:k}|x_q,c'})$, where $c' \in M$, $c \neq c'$ and $x_{a_{1:k}|x_q,c'}$ are top-k examples from class c' that are nearest to x_q. We call c' one *negative class* for x_q. Since there are many negative classes $c' \in M \backslash c$ for x_q, we keep top-n negative classes for each training example x_q. Note that each x_q has one positive training pair and n negative training pairs. To balance the classes in the training loss, we give a weight ratio $n : 1$ for a positive and a negative pair, respectively. We detail the finding of the top-n negative classes in Experiments.

Training the meta-classifier also requires validation classes for model selection (during optimization) and hyperparameters (k and n) tuning (as detailed in Experiments). Since the classes tested by the meta-classifier are unexpected, we further use a set of *validation classes* M', where $M' \cap M = \varnothing$ (also $M' \cap (S \cup U) = \varnothing$), to ensure generalization on the seen/unseen classes.

Note that the meta-training can also leverage the example indexes and memory for efficient training (to avoid loading concrete examples every time). But the memory must be swapped to the examples of seen classes after meta-training.

4 Experiments

We want to address the following Research Questions(RQs).
RQ1: What is an appropriate public dataset in the domain of text classification for open-world learning using meta-learning?
RQ2: How is the performance of the meta-classifier with different settings of top-k examples and n negative classes?
RQ3: How is the performance of L2AC compared with state-of-the-art text classifiers for open-world learning?

4.1 Dataset

Two datasets were used in [4] (with the state-of-the-art text classifier for open-world learning): 20-Newsgroup (20 classes) and reviews (50 classes). They both have small numbers of classes. We also noticed that the review dataset has a large overlapping of classes, which explains the weak result of only 0.666 in the F1 score. As training a meta-classifier requires an extra meta-training set with a large number of classes, so we decide to adopt a dataset with a large number of classes.

To answer RQ1, we leverage the product descriptions in the Amazon Datasets [12]. Amazon.com maintains a tree-structured category system. We consider each leaf node (product type) in the category system as a class. We formulate a product type classification problem based on product descriptions. We removed products belonging to multiple classes to ensure the classes have no overlapping. This gives us 2598 classes, where 1018 classes have more than 400 products per class. We randomly choose 1000 classes from the 1018 classes with 400 randomly selected products per class as the *encoder training set*; 100 classes with 150 products per class are used as the (classification) *test set*, including both seen classes S and unseen classes U; another 1000 classes with 100 products per class are used as the *meta-training set* (including both M and M'). For the 100 classes of the test set, we further hold out 50 examples (products) from each class as test examples. The rest 100 examples are training data for baselines, or seen classes examples to be read by the meta-classifier (which only reads those examples but is not trained on those examples). To train the meta-classifier, we further split the meta-training set as 900 *meta-training classes* (M) and 100 *validation classes* (M').[2]

4.2 Preprocessing

For all datasets, we use NLTK[3] as the tokenizer, and regard all words that appear more than once as the vocabulary. This gives us 17,526 unique words. We take the maximum length of each document as 120 since the majority of product descriptions are under 100 words.

4.3 Ranker

Since a high-performance ranker is not our focus, we simply use cosine similarity to rank the examples in each seen (or meta-training) class for a given test (or meta-training) example x_t (or x_q). We apply cosine directly on the hidden representations of the encoder as $cosine(h_*, h_{a_i}) = \frac{h_* \cdot h_{a_i}}{|h_*|_2 |h_{a_i}|_2}$, where $*$ can be either t or q, $|\cdot|_2$ denotes the l-2 norm and $\cdot$ denotes the dot product of two examples. There is clearly room to improve the ranker, which we leave to future work.

Training the meta-classifier also requires a ranking of negative classes for a meta-training example x_q. We first compute a *class vector* for each meta-training class. This class vector is averaged over all encoded representations of examples of that class. Then we rank classes by computing cosine similarity between the class vectors and the meta-training example x_q. The top-n (defined in the previous section) classes are selected as negative classes for x_q. We explore different settings of n later.

4.4 Evaluation

Similar to [4], we choose 25, 50, and 75 classes from the (classification) test set of 100 classes as the seen classes for three (3) experiments. Note that each class in the test set has 150 examples, where 100 examples are for baseline training (or seen class examples for L2AC) and 50 examples are for testing both baselines and L2AC. We evaluate the results on all 100 classes for those three (3) experiments. For example, when there are 25 seen classes, testing examples from the rest 75 unseen classes are taken as from a *rejection class* c_{rej}, as in [4].

Besides using macro F1 as used in [4], we also use weighted F1 score over all classes (including seen and the rejection class) as the evaluation metric. Weighted F1 is computed as $\sum_{c \in S \cup \{c_{\text{rej}}\}} \frac{N_c}{\sum_{c \in S \cup \{c_{\text{rej}}\}} N_c} \cdot \text{F1}_c$, where N_c is the number of examples for class c and F1_c is the F1 score of that class. We use this metric because macro F1 has a bias on the importance of rejection when the seen class set is small (macro F1 treats the rejection class as equally important as one seen

[2] We will release all selections of datasets for future research.

[3] https://www.nltk.org/

Layers	Out Dims	Params
Embedding	300	-
Dropout	300	0.5
BiLSTM	512	-
ReLU	512	-
Dropout	512	0.5
FC	1000	-
Softmax	1000	-

Layers	Out Dims	Params
Memory	512	-
AbsSub	512	-
Sum	512	-
FC	512	-
Dropout	512	0.5
Sigmoid	1	-

Table 1: The hyperparameters for text encoder and matching network.

Methods	$\|S\| = 25$ (WF1)	$\|S\| = 25$ (MF1)	$\|S\| = 50$ (WF1)	$\|S\| = 50$ (MF1)	$\|S\| = 75$ (WF1)	$\|S\| = 75$ (MF1)
DOC-CNN	53.25(1.0)	55.04(0.39)	70.57(0.46)	76.91(0.27)	81.16(0.47)	86.96(0.2)
DOC-LSTM	57.87(1.26)	57.6(1.18)	69.49(1.58)	75.68(0.78)	77.74(0.48)	84.48(0.33)
DOC-Enc	82.92(0.37)	75.09(0.33)	82.53(0.25)	84.34(0.23)	83.84(0.36)	88.33(0.19)
DOC-CNN-Gaus	85.72(0.43)	76.79(0.41)	83.33(0.31)	83.75(0.26)	84.21(0.12)	87.86(0.21)
DOC-LSTM-Gaus	80.31(1.73)	70.49(1.55)	77.49(0.74)	79.45(0.59)	80.65(0.51)	85.46(0.25)
DOC-Enc-Gaus	88.54(0.22)	80.77(0.22)	84.75(0.21)	85.26(0.2)	83.85(0.37)	87.92(0.22)
L2AC-$n9$-NoVote	91.1(0.17)	82.51(0.39)	84.91(0.16)	83.71(0.29)	81.41(0.54)	85.03(0.62)
L2AC-$n9$-Vote3	91.54(0.55)	82.42(1.29)	84.57(0.61)	82.7(0.95)	80.18(1.03)	83.52(1.14)
L2AC-$k5$-$n9$-AbsSub	92.37(0.28)	84.8(0.54)	85.61(0.36)	84.54(0.42)	83.18(0.38)	86.38(0.36)
L2AC-$k5$-$n9$-Sum	83.95(0.52)	70.85(0.91)	76.09(0.36)	75.25(0.42)	74.12(0.51)	78.75(0.57)
L2AC-$k5$-$n9$	93.07(0.33)	86.48(0.54)	86.5(0.46)	85.99(0.33)	84.68(0.27)	88.05(0.18)
L2AC-$k5$-$n14$	**93.19**(0.19)	86.91(0.33)	**86.63**(0.28)	86.42(0.2)	85.32(0.35)	88.72(0.23)
L2AC-$k5$-$n19$	93.15(0.24)	86.9(0.45)	86.62(0.49)	86.48(0.43)	**85.36**(0.66)	88.79(0.52)

Table 2: Weighted F1 (WF1) and macro F1 (MF1) scores on test set: all results are evaluated on the same whole testing data with 3 settings (25, 50, and 75 seen classes). The results are the averages over 10 runs with standard deviations in parenthesis.

class). For example, when the number of seen classes is small, the rejection class should have a higher weight as a classifier on a small seen set is more likely challenged by examples from unseen classes. Further, to stabilize the results, we train all models with 10 different initializations and average the results.

4.5 Hyperparameters

For simplicity, we leverage a BiLSTM [9, 10] on top of a GloVe.840b.300d [13] embedding layer as the encoder (other choices are also possible). Similar to feature encoders trained from ImageNet [8], we train classification over the encoder training set with 1000 classes and use 5% of the encoding training data as encoder validation data. The hyperparameters of the encoder are detailed in Table 1. and the classification accuracy of the encoder on validation data is **81.76%**. The matching network (the shared network within the 1-vs-many matching layer) has two fully-connected layers. Its hyperparameters are also given in Table 1. We set the batch size of meta-training as 256.

To answer RQ2 on two hyperparameters k (number of nearest examples from each class) and n (number of negative classes), we use the 100 validation classes to determine these two hyperparameters. We formulate the validation data similar to the testing experiment on 50 seen classes. For each validation class, we select 50 examples for validation. The rest 50 examples from each validation seen class are used to find top-k nearest examples. We perform grid search of averaged weighted F1 over 10 runs for $k \in \{1, 3, 5, 10, 15, 20\}$ and $n \in \{1, 3, 5, 9\}$, where $k = 5$ and $n = 9$ reach the highest weighted F1 (87.60%). Further increasing n gives limited improvements (e.g., 87.69% for $n = 14$ and 87.68% for $n = 19$, when $k = 5$). But a large n significantly increases the number of training examples (e.g., $n = 14$ ended with more than 1 million meta-training examples). So we decide to select $k = 5$ and $n = 9$ for all ablation studies later. Note the validation classes are also used to compute (formulated in a way similar to the meta-training classes) the validation loss for selecting the best model during Adam [14] optimization.

4.6 Compared Methods

To the best of our knowledge, DOC [4] is the only state-of-the-art baseline for open-world learning (with rejection) for text classification. It has been shown in [4] that DOC significantly outperforms the methods CL-cbsSVM and cbsSVM in [2] and OpenMax in [3]. OpenMax is the state-of-the-art method for image classification with the rejection capability.

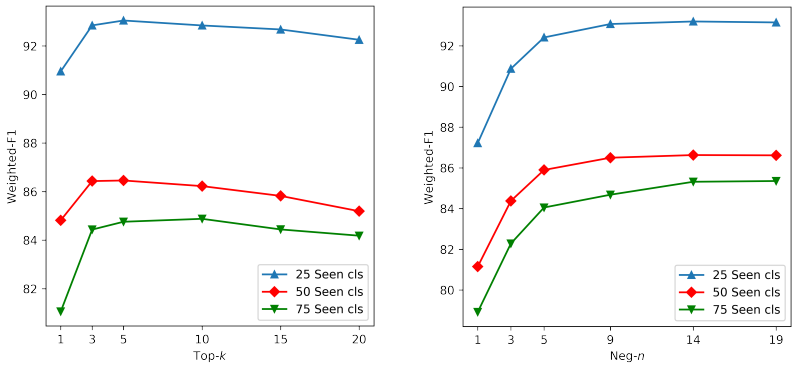


Figure 2: Weighted F1 scores for different k's ($n = 9$) and different n's ($k = 5$).

To answer RQ3, we use DOC and its variants to show that the proposed method has comparable performance with the best open-world learning method with re-training. Note that DOC cannot incrementally add new classes. So we re-train DOC over different sets of seen classes from scratch every time new classes are added to that set. Although it is unfair to compare our method against DOC since DOC is trained on the actual training examples of all classes, our method still performs better in general. We obtained the code of DOC from its authors and created six (6) variants of it.

DOC-CNN: CNN implementation as in the original DOC paper without Gaussian fitting (using 0.5 as the threshold for rejection). It operates directly on a sequence of tokens.

DOC-LSTM: a variant of DOC-CNN, where we replace CNN with BiLSTM to encode the input sequence for fair comparison. Note the BiLSTM is trainable and the input is still a sequence of tokens.

DOC-Enc: this is adapted from DOC-CNN, where we remove the feature learning part of DOC-CNN and feed the hidden representation from our encoder directly to the fully-connected layers of DOC for fair comparison with L2AC.

DOC-*-Gaus: By applying Gaussian fitting proposed in [4] on the above three baselines, we have 3 more DOC baselines. Note that these 3 baselines have exactly the same model as above. They only differ in the thresholds used for rejection. We use these baselines to show that the Gaussian fitted threshold improves the rejection performance of DOC a lot but may lower the performance of classification. The original DOC is **DOC-CNN-Gaus** here.

The following baselines are variants of L2AC.

L2AC-n9-NoVote: this is a variant of the proposed L2AC that only takes one most similar example (from each class), i.e., $k = 1$, with one positive class paired with $n = 9$ negative classes in meta-training ($n = 9$ has the best performance as indicated in answering RQ2 above). We use this baseline to show that the performance of taking only one sample may not be good enough. This baseline clearly does not have/need the aggregation layer and only has a single matching network in the 1-vs-many layer.

L2AC-n9-Vote3: this baseline uses exactly the same model as L2AC-n9-NoVote. But during evaluation, we allow a non-parametric voting process (like kNN) for prediction. We report the results of voting over top-3 examples per seen class as it has the best result (ranging from 3 to 10). If the average of top-3 similar examples in a seen class has example scores with more than 0.5, L2AC believes the testing example belongs to that class. We use this baseline to show that the aggregation layer is effective in learning to vote and L2AC can read more similar examples and get better performance.

L2AC-k5-n9-AbsSub/Sum: To show that using two similarity functions ($f_{\text{abssub}}(\cdot,\cdot)$ and $f_{\text{sum}}(\cdot,\cdot)$) gives better results, we further perform ablation study by only using one of those similarity functions as two baselines.

L2AC-k5-n9/14/19: this baseline has the best $k = 5$ and $n = 9$ on the validation classes, as indicated in the previous subsection. Interestingly, further increasing k may reduce the performance as L2AC may focus on not-so-similar examples. We further report results on $n = 14$ or 19 to show that the results do not get much better.

4.7 Results Analysis

From Table 2, we can see that L2AC outperforms DOC, especially when the number of seen classes is small. First, from Figure 2 we can see that $k = 5$ and $n = 9$ gets reasonably good results. Increasing k may harm the performance as taking in more examples from a class may let L2AC focus on not-so-similar examples, which is bad for classification. More negative classes give L2AC better performance in general but further increasing n beyond 9 has little impact.

Note that testing on 25 seen classes is more about testing a model's rejection capability while testing on 75 seen classes is more about the classification performance. From Table 2, we notice that L2AC can effectively leverage multiple nearest examples and negative classes. The non-parametric voting of L2AC-n9-Vote3 over top-3 examples may not improve the performance but introduce higher variances. But our best $k = 5$, indicating the meta-classifier can dynamically leverage more nearest examples. Running L2AC on a single similarity function gives poorer results as in L2AC-k5-n9-AbsSub or L2AC-k5-n9-Sum.

DOC without encoder (DOC-CNN or DOC-LSTM) is bad when the number of seen classes is small. Without Gaussian fitting, DOC's (DOC-CNN, DOC-LSTM or DOC-Enc) performance increases as more seen classes are available (closer to closed-world classification). Gaussian fitting (DOC-*-Gaus) is important to improve DOC's performance.

5 Related Work

5.1 Open-world Learning

Open-world learning has been studied in text mining and computer vision (where it is called open-set recognition) [1, 2]. Most existing approaches focus on building a classifier that can predict examples from unseen classes into a (hidden) *rejection class*. These solutions are built on top of closed-world classification models [1, 3, 4]. Since a closed-world classifier cannot predict examples from unseen classes (they will be classified into some seen classes), some thresholds are used so that these closed-world models can also be used to do rejection. As discussed earlier, when incrementally learning new classes, they also need some form of re-training, either full re-training from scratch [3, 4] or partial re-training in an incremental manner [1, 2]. This is because these models are not originally trained for rejection, but purely trained for seen classes and they empirically reject unseen examples based on the predictions on seen classes using some thresholds. However, our meta-classifier is trained for rejection.

Our work is also related to class incremental learning [6, 15, 7], where new classes can be added dynamically to the classifier. For example, iCaRL [6] maintains some exemplary data for each class and incrementally tunes the classifier to support more new classes. However, our work is quite different from such incremental learning methods because they do not do rejection of unseen classes as we do. They also require some training with each new class added.

5.2 Meta-learning

Our work is clearly related to meta-learning (or learning to learn) [16], which has been successfully applied to many machine learning tasks lately. For example, it has been used to learn an optimizer [17], to learn network configurations [18], to learn initial and easy-to-tune weights for few-shot learning [19, 20], to learn a teacher model that can guide training sample selection [21] and to learn a domain training corpus for word embeddings [22]. Our proposed framework focuses on learning the similarity between an example and an arbitrary class via reading that class' examples. We are not aware of open-world learning work based on meta-learning.

5.3 Zero-shot Learning

The proposed framework is also related to zero-shot learning [23, 24, 25] in that we do not require training data for classes in testing. However, existing zero-shot learning methods mostly focus on learning a mapping from the input space to an attribute or embedding space and then using some external knowledge to make the class prediction from the attribute or embedding space. We focus on

learning a generalized mapping of the input space to a binary space, so unexpected classes can also benefit from such a mapping without the requirement for training data.

5.4 kNN and Metric Learning

The proposed model is related to k-nearest neighbors (kNN) as well, which also does not require any training but only requires training examples for each seen class during testing. However, kNN is a non-parametric model that leverages a pre-defined metric to compare similarities of testing examples with training examples and uses a non-parametric voting process for classification. Also, kNN is only used for closed-world classification and does not perform rejection. In contrast, our framework has a parametric model and it learns a similarity metric, a voting mechanism, and a rejection capability.

Learning a similarity metric is related to metric learning [26] and deep convolutional siamese networks [27, 28] for a pair of examples. Their learned metric is commonly used for few-shot classification [28, 29], which is again a closed-world classification task without rejection.

The proposed method is also related to computer vision applications such as face recognition [30, 31]. However, face recognition is a controlled application, where the type (face) of all classes is pre-defined (so the variance among classes is limited and different classes share a significant amount of features (e.g., glasses, shirts, etc.)). The training data for face recognition is close to few-shot learning where the number of classes is huge but each class has only 2 or a few examples. The open-world learning problem is more challenging given no restrictions on the definition of a class.

5.5 Memory Augmented Neural Networks

Since the proposed meta-classifier reads examples from a seen class, it is thus related to memory augmented neural networks, such as Neural Turing Machine [32] and Memory Networks [33]. But we focus on building a meta-classifier that reads seen class examples to accept or reject a class.

6 Conclusions

In this paper, we proposed a meta-learning framework called L2AC to support flexible class incremental learning and open-world learning for text classification. Compared to traditional closed-world classifiers, our meta-classifier can incrementally accept new classes by simply adding new class examples without re-training. Compared to other open-world learning methods, the rejection capability of L2AC is trained rather than realized using some empirically set thresholds. Our experiments showed superior performances to state-of-the-art baselines.

References

[1] Bendale, A., T. Boult. Towards open world recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1893–1902. 2015.

[2] Fei, G., S. Wang, B. Liu. Learning cumulatively to become more knowledgeable. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1565–1574. ACM, 2016.

[3] Bendale, A., T. E. Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572. 2016.

[4] Shu, L., H. Xu, B. Liu. Doc: Deep open classification of text documents. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2911–2916. Association for Computational Linguistics, Copenhagen, Denmark, 2017.

[5] Kim, Y.-B., D. Kim, A. Kumar, et al. Efficient large-scale domain classification with personalized attention. *arXiv preprint arXiv:1804.08065*, 2018.

[6] Rebuffi, S.-A., A. Kolesnikov, G. Sperl, et al. icarl: Incremental classifier and representation learning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5533–5542. IEEE, 2017.

[7] Lee, J., J. Yun, S. Hwang, et al. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[8] Russakovsky, O., J. Deng, H. Su, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

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

[10] Schuster, M., K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[11] Lake, B., R. Salakhutdinov, J. Gross, et al. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33. 2011.

[12] He, R., J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517. International World Wide Web Conferences Steering Committee, 2016.

[13] Pennington, J., R. Socher, C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. 2014.

[14] Kingma, D. P., J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Rusu, A. A., N. C. Rabinowitz, G. Desjardins, et al. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[16] Thrun, S., L. Pratt. *Learning to learn*. Springer, 2012.

[17] Andrychowicz, M., M. Denil, S. Gomez, et al. Learning to learn by gradient descent by gradient descent. In *NIPS*, pages 3981–3989. 2016.

[18] Fernando, C., D. Banarse, C. Blundell, et al. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[19] Finn, C., P. Abbeel, S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. 2017.

[20] Finn, C., K. Xu, S. Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

[21] Fan, Y., F. Tian, T. Qin, et al. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018.

[22] Xu, H., B. Liu, L. Shu, et al. Lifelong domain word embedding via meta-learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. AAAI Press, 2018.

[23] Lampert, C. H., H. Nickisch, S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009.

[24] Palatucci, M., D. Pomerleau, G. E. Hinton, et al. Zero-shot learning with semantic output codes. In *NIPS*, pages 1410–1418. 2009.

[25] Socher, R., M. Ganjoo, C. D. Manning, et al. Zero-shot learning through cross-modal transfer. In *NIPS*, pages 935–943. 2013.

[26] Xing, E. P., M. I. Jordan, S. J. Russell, et al. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528. 2003.

[27] Bromley, J., I. Guyon, Y. LeCun, et al. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744. 1994.

[28] Koch, G., R. Zemel, R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, vol. 2. 2015.

[29] Vinyals, O., C. Blundell, T. Lillicrap, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638. 2016.

[30] Taigman, Y., M. Yang, M. Ranzato, et al. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708. 2014.

[31] Schroff, F., D. Kalenichenko, J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823. 2015.

[32] Graves, A., G. Wayne, I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[33] Sukhbaatar, S., J. Weston, R. Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448. 2015.