Expert Systems With Applications 147 (2020) 113198

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Learn#: A Novel incremental learning method for text classification

Guangxu Shan, Shiyao Xu, Li Yang, Shengbin Jia, Yang Xiang*

College of Electronic and Information Engineering, Tongji University, Shanghai 201804, China

ARTICLE INFO

Article history:
Received 20 September 2019
Revised 20 December 2019
Accepted 8 January 2020
Available online 9 January 2020

Keywords:
Learn#
Incremental learning
Reinforcement learning

ABSTRACT

Deep learning is an effective method for extracting the underlying information in text. However, it performs better on closed datasets and is less effective in real-world scenarios for text classification. As the data is updated and the amount of data increases, the models need to be retrained, in what is often a long training process. Therefore, we propose a novel incremental learning strategy to solve these problems. Our method, called Learn#, includes four components: a *Student* model, a reinforcement learning (RL) module, a *Teacher* model, and a discriminator model. The *Student* models first extract the features from the texts, then the RL module filters the results of multiple *Student* models. After that, the *Teacher* model reclassifies the filtered results to obtain the final texts category. To avoid increasing the *Student* models unlimitedly as the number of samples increases, the discriminator model is used to filter the *Student* models based on their similarity. The Learn# method has the advantage of a shorter training time than the One-Time model, because it only needs to train a new *Student* model each time, without changing the existing *Student* models. Furthermore, it can also obtain feedback during application and tune the models parameters over time. Experiments on different datasets show that our method for text classification outperforms many traditional One-Time methods, reducing training time by nearly 80%.

1. Introduction

Text classification plays a vital role in natural language processing. Many natural language processing problems can be regarded as text classification tasks, including sentiment analysis (Alaei, Becken, & Stantic, 2019; Wehrmann, Becker, & Barros, 2018), news classification (Duong & Hoang, 2019; Qiu, Jiang, Zhang, Lu, & Pei, 2018), spam filtering (Mallik & Sahoo, 2019; Peng, Huang, Jia, & Ingram, 2018) and so on. With accessible labeled data, a text classification model can be trained under a supervised learning manner and used to classify unseen texts.

Text classification based on traditional machine learning methods often relies on a lot of human-designed features, such as rule-based (Hadi, Al-Radaideh, & Alhawari, 2018) and statistics based (Zhai, Song, Liu, Liu, & Zhao, 2018) features. However, these methods cost a lot to construct artificial features (Jiang et al., 2018). Improper features might influence the classification effect directly. Besides, it is unable to effectively extract semantic information and utilize the contextual information behind the texts (Hu et al., 2019). At last, in the machine learning paradigm, certain feature combination methods can significantly impact the final result, but it is too expensive to get all the feature combination for machine learning (Labani, Moradi, Ahmadizar, & Jalili, 2018).

With the rapid development of deep learning (LeCun, Bengio, & Hinton, 2015), text classification tasks continue to achieve higher performance (Liang, Sun, Sun, & Gao, 2017). However, deep learning for text classification has some limitations. First, deep learning models are trained in a batch learning setting with the entire training data (Ioffe & Szegedy, 2015; Tsoumakas, Katakis, & Vlahavas, 2008). So it is cumbersome when used in a continual learning setting because storing previous text data is a memory expensive task (Papernot et al., 2016). Second, it is difficult to obtain total sufficient labeled samples for text classification with deep learning at the beginning because the labeling cost is prohibitively expensive or they do not occur frequently enough to be collected (Sun, Sun, Zhou, & Lv, 2019). At last, model performances highly depend on the distribution of the data samples. The distribution of the previous datasets might differ from that of the newly collected data, which might lead to overfitting.

As a continual learning method, incremental learning can be adopted to solve these problems (Chen & Liu, 2018). The great significance of incremental learning is that when new samples are added, there is no need to retrain with all samples, reducing memory consumption and training time cost. Incremental learning requires that the algorithm learns from streaming data with limited memory without sacrificing accuracy. An ideal incremental learning system would be able to assimilate new information without

* Corresponding author.
E-mail addresses: guangxush@tongji.edu.cn (G. Shan), xushiyao@tongji.edu.cn (S. Xu), li.yang@tongji.edu.cn (L. Yang), shengbinjia@tongji.edu.cn (S. Jia), tjdxxiangyang@gmail.com (Y. Xiang).

https://doi.org/10.1016/j.eswa.2020.113198
0957-4174/

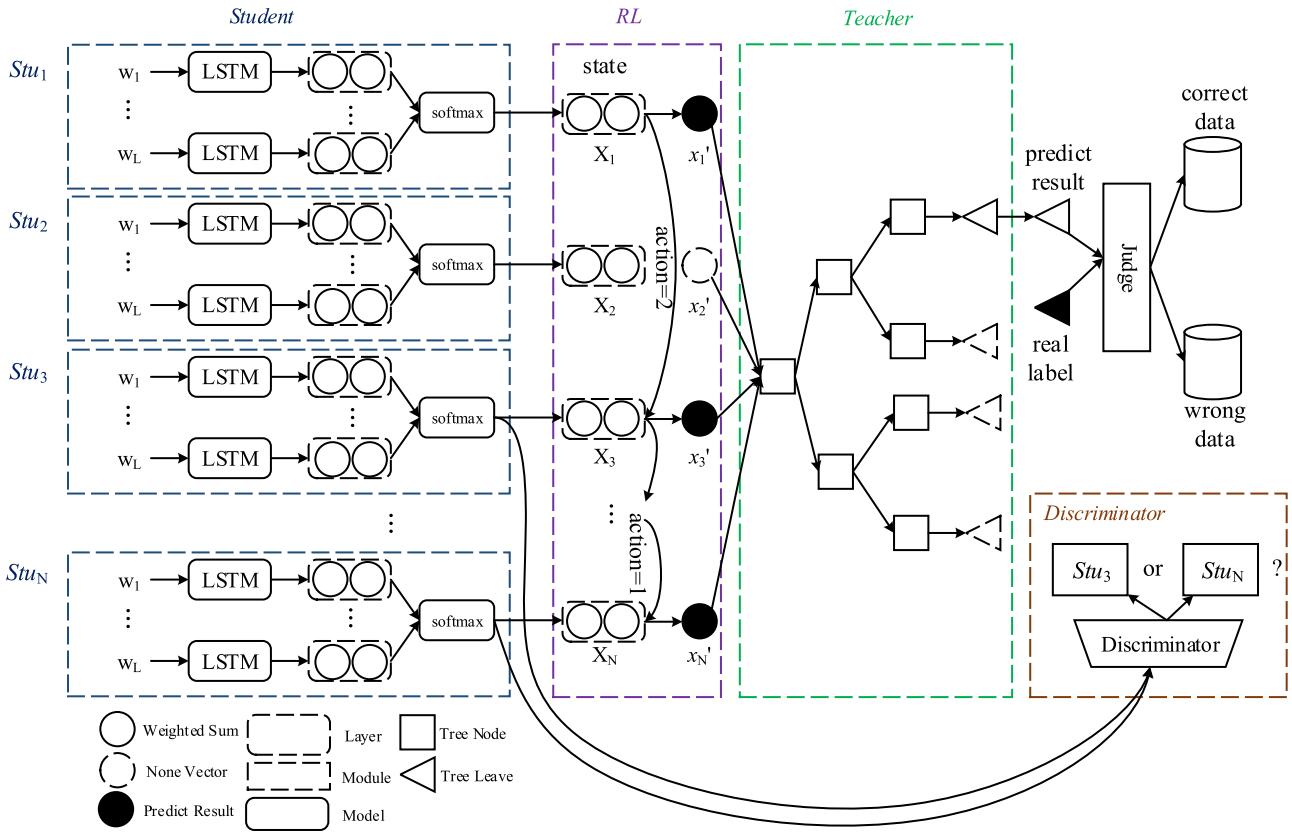


Fig. 1. The structure of the Learn#, from left to right, the *Student* models give preliminary predictions with LSTM, then RL selects the adaptive predictions from the *Student* models and feeds the results to the *Teacher* model. Finally, the *Teacher* model provides the text categories with XGBoost. The discriminator model calculates the similarity of each model, and then eliminates the similar *Student* models, thereby keeping the number of *Student* models steady.

the need to store the entire training dataset. So It can keep the time and space complexity within a controllable range. However, incremental learning suffers from a problem called catastrophic forgetting (Kirkpatrick et al., 2017), which is current model tends to forget the knowledge they obtained from previous training sessions when trained incrementally. To be more specific, fine-tuning a model on new data usually results in a significant performance drop on previous data.

One way to overcome catastrophic forgetting is Elastic Weight Consolidation (Huszár, 2017), which penalizes the modification of parameters deemed to be important to prior tasks. Another way is Learning without Forgetting (Li & Hoiem, 2017), it trains the model without using the previous data, while incrementally learning new labels, which tries to preserve activations of the initial network while training on new data. All of them have tried to preserve knowledge important to prior tasks through the use of proxy losses, but sometimes it fails to filter the existing similar preliminary models (Leung, Leung, & Wong, 2019; Stojanov et al., 2019) and can not assign reasonable weights to the preliminary models (Chen & Liu, 2018).

Distinct from such prior works, our proposed incremental learning method, Learn#, allows the model to retain a fraction of the training data of previous samples while incrementally learning new datasets and retrain the part of networks with minimal loss. Furthermore, we remove the similar results that are obtained from the preliminary models, which avoiding influence the performance of reclassification and save storage space. Fig 1 shows Learn#'s four components: *Student* model, reinforcement learning (RL) module, *Teacher* model, and discriminator model. First, multiple *Student* models provide preliminary classification results with the same dataset, and then the RL module selects a part of the appropriate predictions. The *Teacher* model reclassifies the above predictions to obtain the final result. In service, we record the misclassified samples, then feed them into a new *Student* model to learn each time while keeping the parameters of other existing *Student* models unchanged. After the number of *Student* models reaches a threshold, the discriminator model is used to eliminate similar *Student* models.

Our primary contributions are:

- We adopt an incremental learning method, Learn#, for training Student models and a *Teacher* model. Learn# can be used in an open environment and can continuously learn from new training samples. It is designed to obtain feedback from the environment over time and improve the learning ability. Training time is significantly reduced without the need to retrain the historical data. Section 3.5 presented a detailed description of this algorithm.
- To reduce the over-matching of excessive error results generated by the *Student* model in incremental learning, we use a reinforcement learning (RL) module to select different *Student* models' predictions. The predictions of multiple *Student* models are globally optimal. Section 3.2 describes the framework of the RL module.
- To avoid increasing the *Student* models unlimitedly as the number of datasets increases, we use a discriminator model to calculate the similarity between each pair of *Student* models offline and remove one of the most similar *Student* models to ensure the stability of the model and improve models' training speed. The experimental results described in Section 4.2 prove the model's effectiveness.

2. Related work

Traditional machine learning models achieved great performance in classification, where feed the statistical features to classifiers, for which training time is relatively short. However, the classification of NLP seems more complicated. This is because many machine learning methods classify the texts based on statistical features rather than semantic information, which is unable to learn adaptive text structure sufficiently (Vilar, 2019; Wohlwend, Elenberg, Altschul, Henry, & Lei, 2019). SVM is one of the most widely used algorithms for classification problems. It has been shown to have many advantages in dealing with nonlinear and high-dimension machine learning questions (Zanaty, 2012). However, training SVMs can be computationally expensive in text classification. Moreover, SVMs lose a part of previously learned information in incremental learning (Goudjil, Koudil, Bedda, & Ghoggali, 2018). Xu et al. (2018) proposed the MR-ISVM for classifications in real-world, but it is only used in linear prediction methods. Qiubo, Guozheng, and Keyuan (2019) proposed a detection approach based on relevant features and XGBoost incremental learning, but it needs to filter out some weaker features depend on information gains and only used new samples to update original model incrementally. (Xu, 2018) proposed the text classification with Naive Bayesian classifier that it can use the posterior over the parameters after training on one task as a prior for the next one. But the posterior over the weights of a neural network is typically intractable because models with many features typically resulted in larger models that were slow to train and update. Overall, the main drawback of traditional machine learning models is the lack of semantic representation, which results in poor performance.

Supervised deep learning methods have been successfully applied to text classification as they learn sophisticated mapping rules between the texts and the labels. So text classification with deep learning has achieved breakthroughs (Jiang et al., 2018; 2018; Yao, Mao, & Luo, 2019). However, there are still some limitations in deep learning. First, deep learning is notoriously greedy for large labeled datasets at the beginning, which limits the generalizability of deep models to new classes due to annotation cost (Xu, Cheng, Luo, Liu, & Zhang, 2019). Second, the rules learned by the majority of current deep learning methods used for text classification are largely fixed and do not vary with different conditions. This limits the network's ability to work in more complex and on-line environments in which the mapping rules themselves are not fixed and constantly change according to contexts, such as different distribution and labels (Partalas et al., 2015). Third, deep learning models need to be updated over time as the amount of data increases in online environments. When the amount of data becomes too large, the problems of how to reduce the training time and memory usage arise.

Humans are readily capable of rapidly learning new knowledge of concepts with few examples of stimulation. This notable gap provides a fertile ground for few-shot learning methods (Snell, Swersky, & Zemel, 2017). It has been proposed to reduce the number of necessary labeled samples and succeeded in text classification. The common few-shot learning methods are as follows. 1. Transfer learning is a few-shot learning method, it uses the knowledge acquired from one task to help learn another. BERT learned from a big text dataset is often used to extract the text features for NLP tasks with few datasets (Devlin, Chang, Lee, & Toutanova, 2018). 2. Optimization-based methods, which aim to learn to optimize the model parameters given the gradients computed from the few-shot examples. Wang, Cheng, Yu, Guo, and Zhang (2019a) proposed the Meta-Metric-Learner for few-shot learning, which is a combination of a Meta-SGD meta learner and a base metric models to predict the parameters of the task-specific classifiers. 3. Distance metric learning, the core idea is similar to nearest neighbors and kernel density estimation. The predicted probability over a set of known labels is the distances of labels of support set samples. Yan, Zheng, and Cao (2018) proposed a short text classification framework based on Siamese CNNs and few-shot learning and outweighed some traditional machine learning methods.

Most of the achievements in text classification show that more labeled samples can improve the performance of classifiers, but it performed worse than training on enough labeled data (Yan et al., 2018). Although quite a lot of research has been done in few-shot learning, learning new concepts or classes with only a few samples is still one of the major challenges in deep learning tasks. And few-shot learning is not suitable to continually learn from a stream of data while maintaining knowledge.

So the increment learning as a potential solution for online learning, can start the task in few labels and improve the performance with the training. It is better for online text classifications than few-shot learning in some situations. When we do not know how many labeled samples are enough for text classifier training, we can apply incremental learning to resolve the text classification tasks. When learning data sequentially while incremental learning can keep the minimum amount of necessary data. At the same time, incremental learning can enable the object function to learn incrementally by updating the current model in accordance with the newly arriving data (Grollman & Jenkins, 2010) and it is able to handle newly introduced labels (Muhlbaier, Topalis, & Polikar, 2004). Furthermore, it does not require access to the total original data used to train the existing classifiers in order to limit memory requirements(Masud, Gao, Khan, Han, & Thuraisingham, 2010).

Incremental learning aims at adapting a learned model to new tasks while retaining the knowledge gained earlier. A key challenge for increment learning is how to strike a balance between the preservation of old tasks and the adaptation to a new one within a given model. There are some methods in incremental learning, such as parameter-based. These methods tried to estimate the importance of each parameter in the original model and add more penalty to the changes on significant parameters (Lughofer, 2008; Parikh & Polikar, 2007). Distillation-based is another effective way to transfer knowledge from one network to another (Lu, Fang, Lin, Chen, & Chen, 2007; Shah, Javed, & Shafait, 2018).

In order to overcome the risk of catastrophic forgetting of incremental learning, there are some methods proposed depends on the increment learning, including two main factions, one method retraining with the previous data and other retraining without the previous data. Arguably, the best method for incremental learning with the previous data is iCaRL (Rebuffi, Kolesnikov, Sperl, & Lampert, 2017), but it requires storing training examples for each class, making it challenging to scale. Leung et al. (2019) proposed the two node fault-tolerant incremental algorithms for single-hidden-layer feedforward networks. It determines the output weight of the newly inserted node and updates all of the previously trained output weights. Incremental learning without using previous data does not require any history data. Kirkpatrick et al. (2017) proposed a technique to remember previous tasks by constraining the important weights when optimizing a new task. One limitation of this approach is that the previous and new tasks may conflict with these important weights. Lv, Chen, Li, and Yang (2018) presented an unsupervised incremental learning algorithm, TFusion, which was aided by the transfer learning of the pedestrians spatio-temporal patterns in the target domain without using previous data.

With the exception of resolving the catastrophic forgetting, incremental learning also has other limitations of research in recent years. Polikar, Upda, Upda, and Honavar (2001) introduced Learn++, an algorithm for incremental training of neural network pattern classifiers. The proposed algorithm enables supervised neural network paradigms to accommodate new data, including ex-

amples that correspond to previously unseen classes. However, it cannot filter preliminary models effectively. Kho, Lee, Choi, and Kim (2019) also used the incremental method to propose a new spoof detection framework to learn new types of fakes incrementally without retraining the existing spoof detector. Nevertheless, the model cannot consider an adaptive expert construction method to improve its accuracy and need a lot of space to store all expert models. Bulat, Kossaifi, Tzimiropoulos, and Pantic (2019) presented a method for multi-domain learning without catastrophic forgetting, using a fully tensorized architecture. However, directly repeating the training process with both previous and new data is often infeasible, due to various issues such as computation cost, storage budget, and privacy. Wang, Chen, Li, Dong, and Tarn (2019b) proposed an incremental RL algorithm that learns from dynamic environments without having any prior knowledge or making any assumptions about the ever-changing environments. However, it needs detector-agent to detect the change of environment and it's not fully automated.

In this paper, we focus on research related to text classification in increment learning. Our proposed model, Learn#, uses the novel incremental learning to overcome catastrophic forgetting and improve the training speed and accuracy of text classification. The reinforcement learning can make the optimal decision depends on the experience accumulated in the past and the discriminator model improves the efficiency of incremental learning automatically, in terms of sample complexity, computational and memory cost. This learning method introduces the distribution of new datasets based on the existing models and ensures that the final result is reasonable.

3. Method

In this section, we introduce the Learn# method, which consists of four components: *Student* model, a reinforcement learning (RL) module, a *Teacher* model, and a discriminator model. The Learn#'s structure is shown in Fig 1.

3.1. Student model

In the *Student* model, we solve the text classification task with deep learning and choose a baseline model to train it, which can highlight the effects of incremental learning.

The word vectors of the texts are pre-trained before being input into the *Student* model. We feed the text corpus into the language model with unsupervised training, and obtain the corresponding word vector features with the Word2Vec (Rehurek & Sojka, 2010). The word vector sequence is represented as $\mathbf{W} = < \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \ldots, \mathbf{w}_\mathrm{L} >$, (L is the max length of the texts).

The training process is as follows, where $y_i^{(1)}$ represents the LSTM layers (Gers, Schmidhuber, & Cummins, 1999) for obtaining the semantic information of each texts:

$$y_i^{(1)} = LSTM(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \ldots, \mathbf{w}_\mathrm{L}) \tag{1}$$

Moreover, the sentence's feature vector input into the multilayer neural network (Lewis, Yesildirek, & Liu, 1996). We chose the ReLU as the activation function here (Nair & Hinton, 2010). The $y_j^{(2)}$ is the output of the multilayer neural network:

$$y_j^{(2)} = \sum_{i=1}^{L} ReLU\left(\omega^{(2)} y_i^{(1)} + b^{(2)}\right) \tag{2}$$

where $\omega^{(2)}$ is the neural network's parameter and $b^{(2)}$ is the bias.

Finally, the category of the texts is calculated by the softmax classifier. The p_j is the possibility of classification and n represents the number of categories, so the class j's probability is:

$$p_j = \frac{\exp\left(y_j^{(2)}\right)}{\sum_{k=1}^{n} \exp\left(y_k^{(2)}\right)} \tag{3}$$

After the *Student* model converges, we obtain the final categories of the texts and the vector of the last layer. The last layer's vector is the texts representation's feature vectors.

3.2. Reinforcement learning(RL) module

We find that the output of the *Student* models is a sequence because the models grows chronologically. Hence we design a reinforcement learning strategy to select the correct prediction results of the *Student* models (Sutton, Barto et al., 1998).

Over time, the *Student* model can be considered as a sequence: $< \mathrm{Stu}_1, \mathrm{Stu}_2, \mathrm{Stu}_3, \ldots, \mathrm{Stu}_\mathrm{N} >$. The feature vectors can be represented as: $< \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \ldots, \mathbf{X}_\mathrm{N} >$ and the corresponding classification results are: $< x_1, x_2, x_3, \ldots, x_\mathrm{N} >$. This process can be described as a Markov decision process, which consists of:

States : $\mathrm{S} = < \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \ldots, \mathbf{X}_t, \ldots, \mathbf{X}_\mathrm{N} >$, the $\mathbf{X}_\mathrm{t}$ is the feature vector of one *Student* model at time t;

Action : If the current state is $\mathbf{X}_\mathrm{t}$, the next state is $\mathbf{X}_{\mathrm{t}+1}$, and the action is the a_t, so that transition dynamics $T(\mathbf{X}_{\mathrm{t}+1}|\mathbf{X}_\mathrm{t}, a_\mathrm{t})$ maps a *Student* model's prediction vectors $\mathbf{X}_\mathrm{t}$ at time t onto another prediction vector $\mathbf{X}_{\mathrm{t}+1}$ in next step ($\mathrm{t}+1 <= \mathrm{N}$);

Reward : The reward r_t is obtained from matching between the predicted result of *Student* model and the real label. If the predicted result is the same as the real category, the reward is 1; otherwise, it is -1.

This filter layer uses DQN to find an optimal policy (Mnih et al., 2013). We design a multilayer neural network to fit Q values by the function like this (the Q network parameter is w).

$$Q(\mathbf{X}, a; w) = Q'(\mathbf{X}, a) \tag{4}$$

The z_i is the predicted action and decides which *Student* model will be chosen next. We calculate the $\min (z_i - Q(\mathbf{X}_i, a_i; w))^2$ to make the difference between the q-target value and the q-eval value as small as possible.

The entire objective function can be optimized by the stochastic gradient descent (Zhang, 2004).

$$\nabla_w L_i(w) = E_{a\sim\rho(.),\mathbf{X}_i'\sim\varepsilon} \nabla_w \left[z_i - Q\left(\mathbf{X}_i', a_i'; w\right)\right] \tag{5}$$

We save the prediction results for empirical playback and then update the reinforcement learning model based on historical samples (Ziebart, Maas, Bagnell, & Dey, 2008). The ultimate goal of reinforcement learning is to maximize the prediction's accuracy of the entire model (Osband, Blundell, Pritzel, & Van Roy, 2016). Once the model is trained, the prediction results of the *Student* model can be input into the reinforcement learning model until all the predictions are filtered. So far, the results of the predictions are prepared for *Teacher* model.

The RL module filters part of the *Student* models' predictions, reduces the excessive error results generated by the *Student* model and enhances the learning effect. It ensures that the predictions of multiple *Student* models are optimized globally.

3.3. Teacher model

The *Teacher* model classifies the *Student* model's output that has been selected by RL module and obtains the final classification results of the text. It performs as a secondary classification. Here we use the boosting trees model (Chen & Guestrin, 2016) to solve the classification task and obtain the category of the text.

From the RL model, we obtain the filtered result: $< x_1', x_2', x_3', \ldots, x_\mathrm{N}' >$. We use the result as input data for *Teacher* model, and the *Teacher* model's output is the final prediction.

More specifically, we use the XGBoost model (Chen & Guestrin, 2016) to perform the classification. Once the XGBoost model is trained, we can obtain the final prediction. The object is to find a function to minimize the $Obj(t)$:

$$Obj(t) = \sum_{i=1}^{n} l\left(h_i, \hat{h}_i\right) + \sum_{i=1}^{t} \Omega(f_i) \tag{6}$$

where the h_i is the correct label and the $\hat{h}_i$ is the predicted value. The t represents the XGBoost tree's parameter.

The *Teacher* model serves as a global model to integrate the local results. On the one hand, it eventually overwrites local misclassification results. On the other hand, it is an easy-to-use and non-parametric classifier, does not require any assumptions on the data and the amount of the input data is small, while the speed of calculation is repaid and the model's performance is robust.

3.4. Discriminator model

As the size of datasets increases, the number of *Student* models will also increase accordingly, because we will add a new *Student* model each time as the error samples accumulate to the threshold. For eliminating some similar *Student* models, we utilize the idea of the discriminator in a generative adversarial network (GAN) (Goodfellow et al., 2014). The discriminator model calculates the similarity of the existing model offline to keep the number of *Student* models from growing unlimitedly.

Assume that one of the *Student* models is Stu_i and the other model is Stu_j, The $S(x)$ is the Stu_i's predicted probability and the $S'(x)$ is the model Stu_j's predicted probability. We expect the discriminator to recognize which *Student* model each prediction comes from, which is a binary classification problem. The loss function is:

$$L = \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log S\left(x^{(i)}\right) + \log\left(1 - S'\left(x^{(i)}\right)\right)\right] \tag{7}$$

We randomly select a portion of the data from the existing samples, of which 70% train the discriminator model (Zhang, Lertvittayakumjorn, & Guo, 2019). After the model converges, the remaining 30% of the data calculate the similarity. The lower the classification efficiency, the more similar the *Student* models.

The results of excessive similar *Student* models can influence the effect of the *Teacher* model's effect. So it is necessary to remove the models by calculating the similarity. We use the discriminator model to control the number of *Student* models, avoid excessive *Student* models, and increase time complexity.

3.5. Learn# method

From this, we introduce all of the models that we designed in the Learn# method. Next we will discuss how to combine these models. The training process of the Learn# method is shown in Fig 2.

Step 1: Segment texts and train word vectors, then use *Student* model to extract the feature of texts, obtain the feature vectors and the preliminary classification results of all the texts.

Step 2: Divide the training dataset D into D_a and D_b. According to the process in Step 1, train Stu_k with D_a, and then input D_b into Stu_1 ~ Stu_k. The results of Stu_1 ~ Stu_k are then obtained.

Step 3: The results sequence Stu_1 ~ Stu_k and the text category of D_b are used to train the RL module. After the RL module is trained, the *Student* models' result sequence is input into RL for filtering. The prediction sequences with high reward are used to train the *Teacher* model.

Step 4: Input the *Student* model's results were chosen in Step 3 into the *Teacher* model, then train the XGBoost model by the gradient descent optimization algorithm.

Step 5: Use these models that have been trained from Step1 ~ Step4. Once the number of misclassified samples reached the threshold, we used 70% misclassified samples and 30% correct classified samples to train a new *Student* model Stu_{k+1}, while the earlier *Student* models are unchanged. At the same time, we updated the RL module and the *Teacher* model by Step 3 and Step 4.

Step 6: As the number of samples increases, if the number of *Student* models becomes too large, the discriminator model will be used to calculate the similarity of each *Student* model pair, and remove one of the pair models with the most similarity. Then repeat these processes from Step 1 ~ Step 6.

The detail training process of Learn# is described in Algorithm 1.

Algorithm 1: The Training Process of Learn# Method.

Input: Divide the dataset D into D_a and D_b, D_c is the online data,x is upper limit of error data's count,y is upper limit of *Student* model's count, z represents the number of iterations

Output: *Student* model S, *Teacher* model T, RL and Discriminator model D

```
foreach i = 1; i < z; i++ do
    R_1 = φ;
    S_i = new Student(D_a[i]); // train a new Student model;
    S.append(S_i);
    foreach j = 1; j < S.length; j++ do
        R_1.append(S_j.classify(D_b[i]));
    k = R_1.length;
    RL = new RL(R_1[0 : 0.7k]); // update the RL model;
    R_2 = RL.filter(R_1[0.7k : k]);
    T = new Teacher(R_2); // update Teacher model;
    E = φ; // init the error data;
    while true do
        if T(RL(S(D_c[t]))) ≠ D_c[t].label then
            E.append(D_c[t]);
            count(E)++;
        if count(E) ≥ x then
            break;
    D_a[i + 1].append(E); // add the error dataset to next iterator;
    if count(S) ≥ y then
        Discriminator = new Discriminator(S); // update Discriminator;
        S = Discriminator.select(S);
return S, RL, T, D
```

4. Experiments

4.1. Settings

In this section, we introduce datasets, evaluation metrics and the parameters of the experiments.

4.1.1. Dataset

To verify the classification effect of Learn#, we used different text classification in the experiments, including ontology classification (DBPedia), topic classification (the AG's news), and senti-

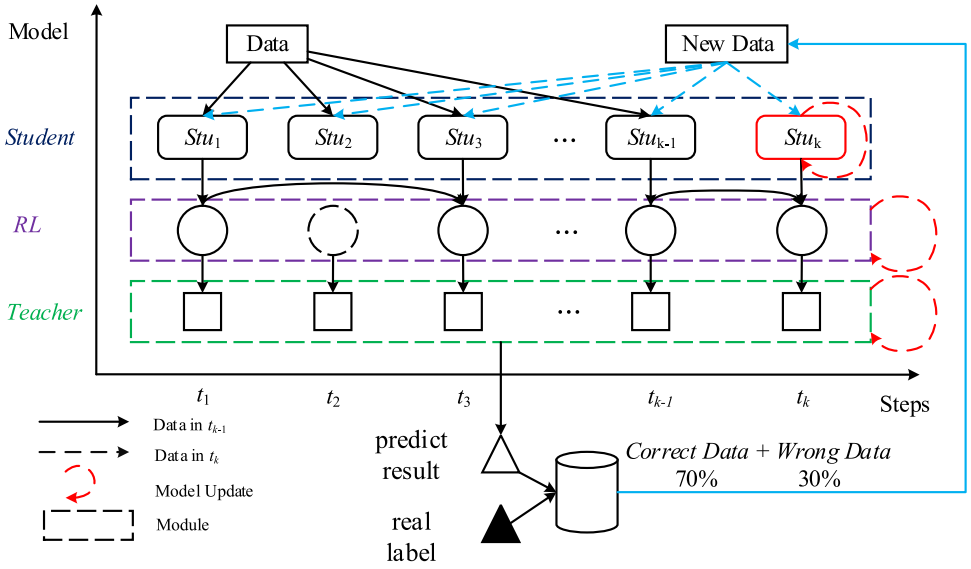


Fig. 2. The Learn# method. From top to bottom, the data is input into the $Stu_1 \sim Stu_{k-1}$ to generate preliminary prediction results and then input them to the RL module at t_{k-1}. The RL module filters out the predictions from the *Student* models and passes them to the Teacher model to generate the final category. We put these models into the usage, recorded the misclassified dataset and correctly predicted the data samples. When the number of all data samples reached a threshold, we selected 30% of the correct samples and 70% of the incorrect samples, and refactored a new dataset: New Data. At t_k, we divided the New Data into D_a and D_b, then retrained a new *Student* model Stu_k on dataset D_a, and kept the other *Student* models unchanged. The new Data D_b was added to the $Stu_1 \sim Stu_k$ to generate preliminary predictions, and then the RL module and *Teacher* model were retrained using the preliminary predictions and D_b's labels with the supervised method. With this, the iterative training process of the Learn# is complete.

Table 1
The description of the dataset which we used in the experiments. The # represents the quantity of the dataset. The Avg_L is the average length of the sentences.

Dataset	Task	#Classes	(#D)	#D′	$\#D_a$ / $\#D_b$/ $\#D_c$	Avg_L
AG's news	News categorization	4	120k	7.6k	3600	44
Amazon Review Full	Sentiment analysis	5	100k	10k	3300	93
Amazon Review Polarity	Sentiment analysis	2	100k	10k	3300	91
Yelp Review Full	Sentiment analysis	5	120k	10k	3600	158
Yelp Review Polarity	Sentiment analysis	2	100k	10k	3300	156
DBPedia	Ontology classification	14	120k	7.6k	3600	55
Twitter	Sentiment analysis	2	35k	3.5k	1000	11

ment analysis (Twitter[1], Yelp and Amazon reviews). These classification tasks included binary classification and multiple classifications. The AG's news contained 496,835 categorized news articles from more than 2000 news sources, from which we chose the four largest classes from this corpus to construct our dataset. The Yelp review dataset was obtained from the Yelp Dataset Challenge in 2015. This dataset contained 1,569,264 samples that have review texts. The Amazon review dataset is from the Stanford Network Analysis Project (SNAP), which spans 18 years with 34,686,770 reviews from 6,643,669 users on 2,441,053 products. DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia (Zhang, Zhao, & LeCun, 2015). The Twitter is from the Kaggle's sentiment analysis competition. Table 1 lists the statistics of the dataset used in the experiments. It is worth mentioning that the experiments contained both long text datasets and short text datasets.

Since our focus was on text classification in online environment, the data was collected incrementally. In order to simulate the online environment, we selected the part of data from the original dataset and divide those into the different parts for Learn# (Khodorchenko, 2019; Singh, Chaudhari, & Singh, 2017; Xiao, Zhang, Yang, Peng, & Zhang, 2014). Assume that the total training dataset is D, the common test dataset is D′, and the number of iterations is k. Then we divide D/*k* into three equal portions: D_a, D_b and D_c, where D_a is used to train the *Student* model. We input D_b into the trained *Student* models and obtained the *Student* model's output results. Next, we used these results and true labels to train the RL module and *Teacher* model. Next, D_c was used to evaluate the above model. In these experiments, we found that after 10 iterations, the accuracy rate in the validation dataset was balanced as shown in Fig 4, so we set iterator k to 10.

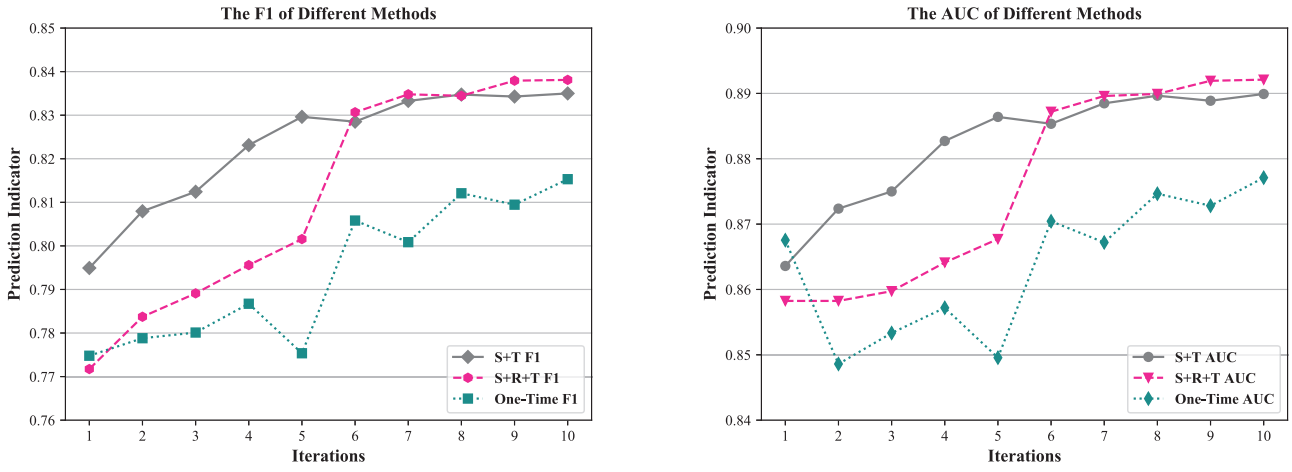
Finally, we used all of the dataset (D_a, D_b and D_c) that had been used in Learn# to train One-Time *Student* model (One-Time). The common test dataset D′ was then used to test the effects of the Learn# model and One-Time model for each iteration. The size of the dataset that we used is shown in Table 1.

4.1.2. Experimental configurations

In our experiments, we adopted the same text classification model (LSTM) with different training strategies, for the One-Time model and Learn# model, to compare their AUC, F1, training time and storage (Jefferts et al., 2002). We used the LSTM as baseline in order to highlight the effect of the Learn# method rather than the classifiers.

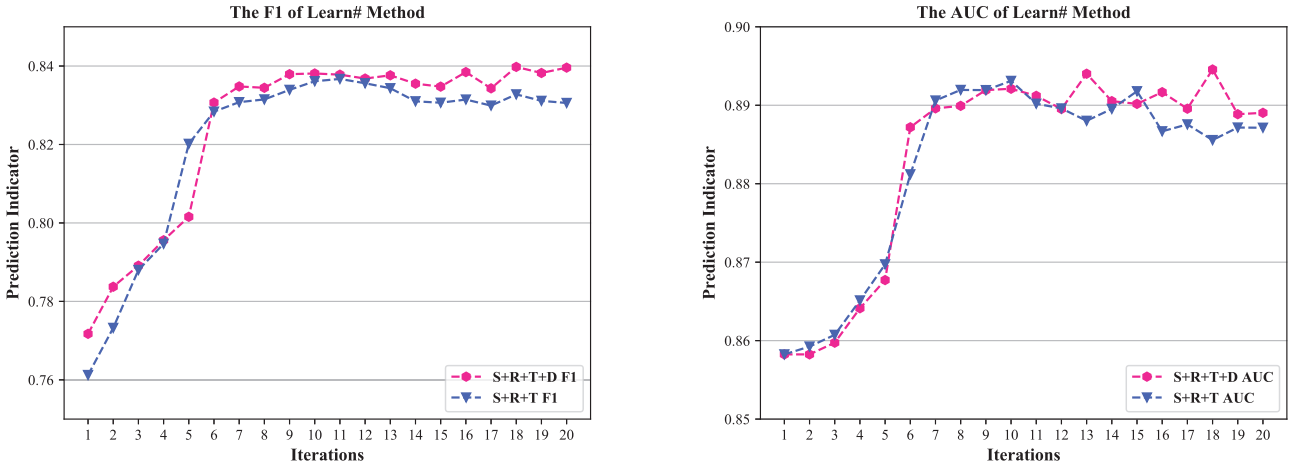
We mainly focused on the performance of Learn# and compare it with that of the One-Time model. We calculate the AUC, F1, training time and storage of Learn#: *Student* model + *Teacher*

[1] https://www.kaggle.com/c/twitter-sentiment-analysis2/data.



(a) The F1 of different methods. (b) The AUC of different methods.

Fig. 3. The F1 and AUC of different methods in 10 iterations. The performance of Learn#(S+R+T) and Learn#(S+T)both exceed that of the One-Time model in F1 and AUC. Learn#(S+R+T) can improve the F1 and AUC compared to the Learn#(S+T) in later steps of the iterations.



(a) The F1 of different methods. (b) The AUC of different methods.

Fig. 4. The F1 and AUC of Learn#(S+R+T+D) in 20 iterations. With the addition of the new dataset, the effect of the Learn#(S+R+T+D) tends to be stable.

model (S+T), and *Student* model + RL module + *Teacher* model (S+R+T). Then we applied the discriminator model for the comparative experiment (S+R+T+D). Through the experiments, we demonstrated the effectiveness of the proposed solution in the AG's news, and learned some interesting facts based on the other datasets.

We use the Keras library written in Python to implement the neural network (Gulli & Pal, 2017). The word embeddings are pre-trained using the Gensim Python library. In these experiments, we set the dimension of the word vector as [100, 300, 500], and we found the 100 dimension is best on a validation set, so we used 100-dimensional word2vec vectors to initialize word representations. For other parameters, we used 30% of the dataset as a validation set, and chose the best hyperparameters based on the validation performance. The dimensionality of the LSTM hidden states was 100. We used ReLU as the layer's activation function and Adam (Kingma & Ba, 2014) algorithm as the models optimizer for the models. We used four fully-connected layers for the *Student* model and two dropout layers (50% dropout rate) to avoid overfitting. We conducted classification experiments by changing the learning rate from {0.0, 0.1, 0.001, ..., 0.00001}, while fixing the optimal learning rate at 0.001.

The Q network in *Teacher* model consisted of five fully-connected layers and one dropout layer (35% dropout rate). The ϵ of e-greedy was selected from {0.85, 0.86, ..., 1}. All of the experiments were repeated three times and the average performance are reported that ϵ was suitable for setting to 0.995. We employed the RMSprop (Tieleman & Hinton, 2012) optimizer to train our RL model, with an initial learning rate of 0.003 and the weight of L2 regularization term at 0.0001. The memory buffer's max length for experience replay was 2000, and the discount factor for the Q value was 0.95.

The *Teacher* model was trained by the XGBoost classifier, the maximum depth of the tree was six, and we applied a grid search for hyper-parameters: the learning ratio was tuned amongst {0.0, 0.1, ..., 0.8} and we selected the best parameters 0.3 depending on the performance of validation data.

The discriminator model used the three fully-connected layers and one dropout layer (75% dropout rate) to regularize models. The optimizer used was also RMSprop, and the learning rate was set to 0.002, a value determined by optimizing F1 on a validation set.

Table 2
Text classification's performances on different datasets with different methods. Learn#(S+R+T)'s performance exceeds Learn#(S+T) and they both far exceed that of the One-Time model. In addition, the Learn#(S+R+T)'s training time is shorter and the storage is larger than those of the One-Time model. We use the space complexity of algorithm for time complexity.

Dataset	Method	Training Time	F1	AUC	Storage
AG's news	S+T	**4min**	**0.8349**	**0.8899**	500M
	S+R+T	**5min38s**	**0.8381**	**0.8921**	500M
	One-Time	35min34s	0.8168	0.8763	50M
Amazon Review Full	S+T	**2min44s**	**0.3598**	**0.6076**	620M
	S+R+T	**3min37s**	**0.3798**	**0.6128**	620M
	One-Time	36min33s	0.3421	0.5948	62M
Amazon Review Polarity	S+T	4min7s	0.7890	0.7891	610M
	S+R+T	**5min17s**	**0.8084**	**0.8088**	610M
	One-Time	32min23s	0.7896	0.7894	61M
Yelp Review Full	S+T	**3min38s**	**0.4670**	**0.6726**	370M
	S+R+T	**5min33s**	**0.4929**	**0.6816**	370M
	One-Time	35min16s	0.4638	0.6648	37M
Yelp Review Polarity	S+T	**3min12s**	**0.8850**	**0.8851**	340M
	S+R+T	**5min3s**	**0.8859**	**0.8861**	340M
	One-Time	36min5s	0.8758	0.8759	34M
DBPedia	S+T	**4min**	**0.9489**	**0.9725**	920M
	S+R+T	**6min33s**	**0.9578**	**0.9772**	920M
	One-Time	40min16s	0.9482	0.9721	92M
Twitter	S+T	**1min20s**	**0.7282**	**0.7002**	18M
	S+R+T	**2min34s**	**0.7324**	**0.7230**	18M
	One-Time	10min09s	0.7084	0.6849	180M

4.2. Accuracy of text classification

We trained Learn#(S+T) incrementally and obtained the training performance for each iteration. At the same time, we trained the One-Time model using AG's news. Fig 3 shows the performance of incremental model's F1 and AUC. Learn#(S+T)'s performance exceeds the One-Time model in F1 and AUC. The *Teacher* models reclassified the *Student* models' results and improved the effectiveness of the single *Student* models. Our results showed that the Learn#(S+T) can balance the preliminary classification errors and integrate the classification results with the distribution of different samples.

Next, we added the RL module to filter the prediction results of the *Student* model. The Fig 3 shows that the Learn#(S+R+T) can improve the F1 and AUC in later steps of the iterations. This proves that the RL module effectively prevents the occurrence of excessive error results by the *Student* models. They obtain the computational power and ensures the stability of the whole models.

Here we used the discriminator model to constrain the number of *Student* models. Fig 4 shows the performance of the Learn# with discriminator model (S+R+T+D) between 1 and 20 iterations. We can see that the results of the models with discriminator are relatively stable. From the results, we see that the discriminator model can filter the similar preliminary models effectively and maintain a better classification effect.

At the same time, we used another dataset for comparison and obtained the effects of these methods on a different dataset. All of the experiments were repeated three times and the average performance was reported. These models' performance on a different dataset in the 10th iteration is shown in Table 2. The performance of Learn#(S+R+T) also exceeded that of the Learn#(S+T), and it far exceeded that of the One-Time model.

4.3. Training time and storage for text classification

Training time represents the time spent in training the models in this iteration. Here, we reported results for the training time (forward pass, backward pass and cross-entropy) of different training methods in the AG's news dataset. We compared results on the Nvidia 1 Titan GPU implemented in Keras. The Learn#'s training time and One-Time's training time are also calculated when new data coming. The Learn# calculate the last iterator's training time and the One-Time calculate the whole model's training time when the new data input. Fig 5 summarizes the training time for the Learn#(S+T), Learn#(S+R+T), and One-Time model. It is obvious that the Learn#'s training time is much less than the One-Time model. The models' performance on different datasets in the 10th iteration is also shown in Table 2. As the number of iterations increases, more *Student* models are generated, and the storage space required for the model is also increases accordingly. However, we used the discriminator model to filter and keep the number of *Student* models at around 10, making the model more stable.

We used the new dataset to train the Learn# while using all of the other datasets retrain the One-Time model. This meant that the One-Time model's time complexity was enormous, and parameters updated slowly. What is more, the input vector dimension of the *Teacher* model was small compared with the *Student* model's word vectors, and because the *Teacher* model's network structure was more straightforward than *Student* model, so the whole update speed was faster. The Learn# method for text classification outperformers server One-Time models, nearly reducing the training time nearly 80%. Thus, it is designed for feedback from the environment in time and adjust the parameters to give a better result. It is suitable for learning in online environments.

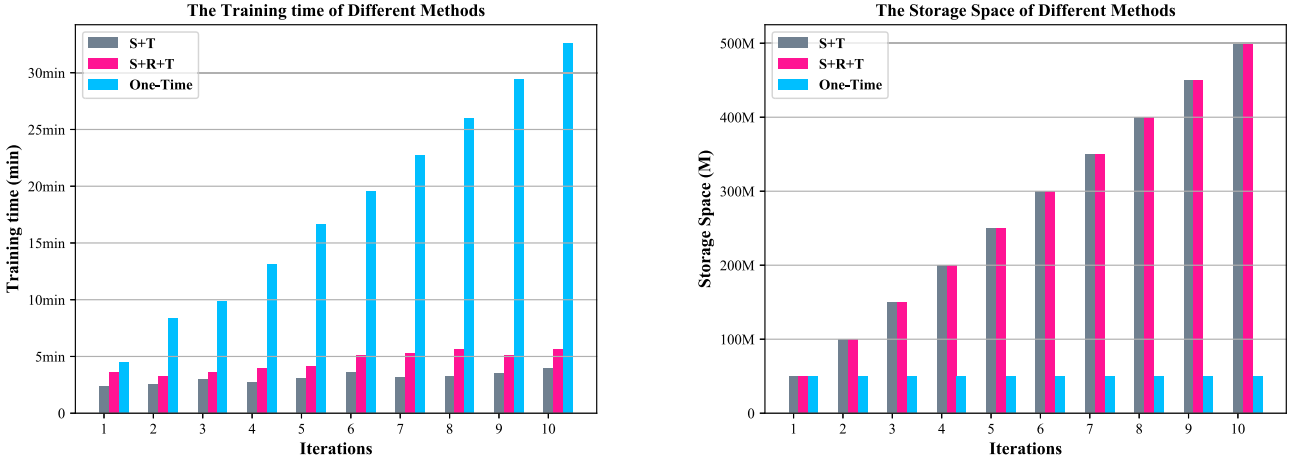
When we added the RL module to the Learn#(S+T), the experiment's results showed that the training time of the Learn#(S+R+T) was relatively longer because of the RL model's training. Usage also consumes time, so that you can trade off between the accuracy and the training speed, depending on the situation.

4.4. Comparisons

To validate the effectiveness of our method, we conducted classification experiments by comparing Learn# in the following four different models:

S(TF-IDF+SVM)+R+T: This model used SVM based on the TF-IDF feature as the *Student* model for text classification, then combined the RL model and *Teacher* model to obtain final result.

S(TF-IDF+XGBoost)+R+T: We also used the XGBoost classifier based on TF-IDF features as the *Student* model for text classification and then used the RL model to filter the results. We then used the *Teacher* model to generate the final result.



(a) Training time in 10 iterators. (b) Storage Space in 10 iterators.

Fig. 5. Training time and Storage Space of different methods in 10 iterators. The One-Time's training time is nearly 70% ~ 80% longer than Learn#(S+R+T) and Learn#(S+T)'s. The training time of the Learn#(S+R+T) is relatively long because of the RL model's training and usage also need time. The storage space grows linearly as the iteration increases, because the count of models increases, when the model is stable, we maintain a constant storage size through the Discriminator model.

Table 3
The text classification's performances on AG news with different methods. We use different Student models, such as TF-IDF+SVM, TF-IDF+XGBoost, Word2Vec+LSTM and BERT, and compare them with the One-Time model with the same models. We see the Learn#(S(BERT)+R+T)'s performance exceeds that of the other models, and Learn# is better than the One-Time model in F1 and AUC.

Method	Variant	Training Time	F1	AUC	Storage
Learn#(S+R+T)	S(TF-IDF+SVM)	4min43s	0.7252	0.7068	35M
	S(TF-IDF+XGBoost)	3min12s	0.7741	0.7328	30M
	S(Word2Vec+LSTM)	5min38s	0.8381	0.8921	500M
	S(BERT)	**9min53s**	**0.9436**	**0.9722**	**4150M**
One-Time	TF-IDF+SVM	38min19s	0.7081	0.6953	3.5M
	TF-IDF+XGBoost	29min3s	0.7405	0.7022	3M
	Word2Vec+LSTM	35min34s	0.8168	0.8763	50M
	BERT	**57min2s**	**0.9317**	**0.9617**	**415M**

S(Word2Vec+LSTM)+R+T: This model is as same as the S+R+T model in 4.2.

S(BERT)+R+T: We directly use the [CLS] representation of BERT as a classification feature to fine-tune the BERT model for paired sentence classification. Then the output of S(BERT) was fed to the RL model and *Teacher* model to obtain the final predictions.

As shown in Table 3, the performance of the traditional machine learning methods is poor compared with that of the deep learning method, but Learn# can improve performance of traditional machine learning methods. The S(TF-IDF+XGBoost)+R+T contributed 3% points and the S(TF-IDF+SVM)+R+T improved by 2% points. This proves that the traditional machine learning methods ignore the semantic information and only on the statistics features, but the Learn# method can be used to improve the text classification effect and reduce the training time.

In addition, explicitly capturing sentence information consistently improves the performance of the S(BERT)+R+T over the S(Word2Vec+LSTM)+R+T model. On average, the F1 of S(BERT)+R+T increased by 11% points. This shows that the Learn# method for text classification is also useful for the BERT-based model as well, because the sentence information has been embedded in the BERT representation. S(BERT)+R+T outperforms the One-Time(BERT) model. On average, the Learn# contributes 1% points to the final performance of the only BERT. It also shows that the Learn# method contributes to the accuracy and training speed of the One-Time model based on BERT.

In the summary in Section 4, according to the experimental results of different datasets, the model has the capability of incremental learning. The performance of the classifier tends to improve with incremental steps, and the performance can exceed or be very close to that of the One-Time learning model. This indicates that the Learn# with short training time is effective. Using BERT representations further boosts the performance of our model. Even though the original BERT model already provides strong predictive power, our model consistently improves over BERT-CLS, which indicates that Learn# can better utilize these semantic representations for text classification.

5. Conclusion and future work

In this paper, we propose a novel incremental learning method, called Learn#. The *Student* models in this method use a neural network model to extract text features and predict the category of the text. The predictions are filtered by the reinforcement learning module. The *Teacher* model is used to reclassify the classification results of the *Student* models to further improve the classification effect. Once the number of *Student* models reaches a threshold, the discriminator model will eliminate the similar *Student* models for keeping the number of models in steady. The experimental results prove that Learn# is meaningful for text classification on different datasets.

We only used the LSTM as the *Student* models. In our future work, we will combine other classification methods, including logistic regression, support vector machine, decision tree, random forests, and other deep learning methods depending on the dataset's distribution to form an ensemble classifier as different

Student models. We expect to see further performance gains in a better design of *Student* models. We adopted the widely used Q-learning algorithm as the basic implementation for the *Teacher* Model, which can be extended to other specific RL algorithms (e.g., A3C (Mnih et al., 2016))) in principle. We believe that our methodology is applicable to other supervised natural language processing tasks with Learn#, such as language inference and relation classification and we plan to apply the proposed approach to those tasks in the future.

Declaration of Competing Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from tjdxxiangyang@gmail.com.

Credit authorship contribution statement

Guangxu Shan: Conceptualization, Formal analysis, Investigation, Methodology, Writing - original draft, Writing - review & editing. **Shiyao Xu:** Formal analysis, Supervision. **Li Yang:** Formal analysis, Supervision. **Shengbin Jia:** Formal analysis, Supervision. **Yang Xiang:** Funding acquisition, Supervision.

Acknowledgements

We would like to thank all the reviewers for their insightful and valuable comments, which significantly improve the quality of this paper. This work is supported by the National Natural Science Foundation of China under Grant no. 71571136 and the Project of Science and Technology Commission of Shanghai Municipality under Grant no. 16JC1403000.

References

Alaei, A. R., Becken, S., & Stantic, B. (2019). Sentiment analysis in tourism: Capitalizing on big data. *Journal of Travel Research, 58*(2), 175–191.

Bulat, A., Kossaifi, J., Tzimiropoulos, G., & Pantic, M. (2019). Incremental multi-domain learning with network latent tensor factorization. arXiv:1904.06345.

Chen, C. P., & Liu, Z. (2018). Broad learning system: An effective and efficient incremental learning system without the need for deep architecture. *IEEE transactions on neural networks and learning systems, 29*(1), 10–24.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). ACM.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805.

Duong, H.-T., & Hoang, V. T. (2019). A survey on the multiple classifier for new benchmark dataset of vietnamese news classification. In *2019 11th international conference on knowledge and smart technology (kst)* (pp. 23–28). IEEE.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Goudjil, M., Koudil, M., Bedda, M., & Ghoggali, N. (2018). A novel active learning method using svm for text classification. *International Journal of Automation and Computing, 15*(3), 290–298.

Grollman, D. H., & Jenkins, O. C. (2010). Incremental learning of subtasks from unsegmented demonstration. In *2010 ieee/rsj international conference on intelligent robots and systems* (pp. 261–266). IEEE.

Gulli, A., & Pal, S. (2017). *Deep learning with Keras*. Packt Publishing Ltd.

Hadi, W., Al-Radaideh, Q. A., & Alhawari, S. (2018). Integrating associative rule-based classification with naïve bayes for text classification. *Applied Soft Computing, 69*, 344–356.

Hu, H., Phan, N., Geller, J., Iezzi, S., Vo, H., Dou, D., & Chun, S. (2019). An ensemble deep learning model for drug abuse detection in sparse twitter-sphere. arXiv:1904.02062.

Huszár, F. (2017). On quadratic penalties in elastic weight consolidation. arXiv:1712.03847.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Jefferts, S. R., Shirley, J., Parker, T., Heavner, T., Meekhof, D., Nelson, C., … Drullinger, R., et al. (2002). Accuracy evaluation of nist-f1. *Metrologia, 39*(4), 321.

Jiang, M., Liang, Y., Feng, X., Fan, X., Pei, Z., Xue, Y., & Guan, R. (2018). Text classification based on deep belief network and softmax regression. *Neural Computing and Applications, 29*(1), 61–70.

Kho, J. B., Lee, W., Choi, H., & Kim, J. (2019). An incremental learning method for spoof fingerprint detection. *Expert Systems with Applications, 116*, 52–64.

Khodorchenko, M. (2019). Distant supervision and knowledge transfer for domain-oriented text classification in online social networks. *Procedia Computer Science, 156*, 166–175.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., … Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences, 114*(13), 3521–3526.

Labani, M., Moradi, P., Ahmadizar, F., & Jalili, M. (2018). A novel multivariate filter method for feature selection in text classification problems. *Engineering Applications of Artificial Intelligence, 70*, 25–37.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature, 521*(7553), 436.

Leung, H., Leung, C., & Wong, E. (2019). Fault and noise tolerance in the incremental extreme learning machine. *IEEE Access, PP*. doi:10.1109/ACCESS.2019.2948059. 1–1

Lewis, F. L., Yesildirek, A., & Liu, K. (1996). Multilayer neural-net robot controller with guaranteed tracking performance. *IEEE Transactions on Neural Networks, 7*(2), 388–399.

Li, Z., & Hoiem, D. (2017). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence, 40*(12), 2935–2947.

Liang, H., Sun, X., Sun, Y., & Gao, Y. (2017). Text feature extraction based on deep learning: A review. *EURASIP journal on wireless communications and networking, 2017*(1), 1–12.

Lu, J., Fang, N., Lin, J., Chen, F., & Chen, G. (2007). Constructing the model of propylene distillation based on neural networks. In *2007 ieee 22nd international symposium on intelligent control* (pp. 430–434). IEEE.

Lughofer, E. D. (2008). Flexfis: A robust incremental learning approach for evolving takagi–sugeno fuzzy models. *IEEE Transactions on fuzzy systems, 16*(6), 1393–1410.

Lv, J., Chen, W., Li, Q., & Yang, C. (2018). Unsupervised cross-dataset person re-identification by transfer learning of spatial-temporal patterns. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7948–7956).

Mallik, R., & Sahoo, A. K. (2019). A novel approach to spam filtering using semantic based naive bayesian classifier in text analytics. In *Emerging technologies in data mining and information security* (pp. 301–309). Springer.

Masud, M., Gao, J., Khan, L., Han, J., & Thuraisingham, B. M. (2010). Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering, 23*(6), 859–874.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., … Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv:1312.5602.

Muhlbaier, M., Topalis, A., & Polikar, R. (2004). Learn++. mt: A new approach to incremental learning. In *International workshop on multiple classifier systems* (pp. 52–61). Springer.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 807–814).

Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems* (pp. 4026–4034).

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. (2016). The limitations of deep learning in adversarial settings. In *2016 ieee european symposium on security and privacy (euros&p)* (pp. 372–387). IEEE.

Parikh, D., & Polikar, R. (2007). An ensemble-based incremental learning approach to data fusion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37*(2), 437–450.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R., & Galinari, P. (2015). Lshtc: A benchmark for large-scale text classification. arXiv:1503.08581.

Peng, W., Huang, L., Jia, J., & Ingram, E. (2018). Enhancing the naive bayes spam filter through intelligent text modification detection. In *2018 17th ieee international conference on trust, security and privacy in computing and communications/12th ieee international conference on big data science and engineering (trustcom/bigdatase)* (pp. 849–854). IEEE.

Polikar, R., Upda, L., Upda, S. S., & Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews), 31*(4), 497–508.

Qiu, S., Jiang, M., Zhang, Z., Lu, Y., & Pei, Z. (2018). Chinese news text classification of the stacked denoising auto encoder based on adaptive learning rate and additional momentum item. In *International symposium on neural networks* (pp. 578–584). Springer.

Qiubo, H., Guozheng, F., & Keyuan, J. (2019). An approach of suspected code plagiarism detection based on xgboost incremental learning. *2019 international conference on computer, network, communication and information systems (cnci 2019)*. Atlantis Press.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2001–2010).

Rehurek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. *In proceedings of the lrec 2010 workshop on new challenges for nlp frameworks*. Citeseer.

Shah, H., Javed, K., & Shafait, F. (2018). Distillation techniques for pseudo-rehearsal based incremental learning. arXiv:1807.02799.

Singh, N., Chaudhari, N. S., & Singh, N. (2017). Online url classification for large-scale streaming environments. *IEEE Intelligent Systems, 32*(2), 31–36. doi:10.1109/MIS.2017.39.

Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in neural information processing systems* (pp. 4077–4087).

Stojanov, S., Mishra, S., Thai, N. A., Dhanda, N., Humayun, A., Yu, C., … Rehg, J. M. (2019). Incremental object learning from contiguous views. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 8777–8786).

Sun, S., Sun, Q., Zhou, K., & Lv, T. (2019). Hierarchical attention prototypical networks for few-shot text classification. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 476–485).

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*: 135. MIT press Cambridge.

Tieleman, T., & Hinton, G. (2012). Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *Tech. Rep., Technical report*, 31.

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ecml/pkdd 2008 workshop on mining multidimensional data (mmd??08): 21* (pp. 53–59). sn.

Vilar, E. (2019). Word embedding, neural networks and text classification: What is the state-of-the-art? *Junior Management Science, 4*(1), 35–62.

Wang, D., Cheng, Y., Yu, M., Guo, X., & Zhang, T. (2019a). A hybrid approach with optimization-based and metric-based meta-learner for few-shot learning. *Neurocomputing, 349*, 202–211.

Wang, Z., Chen, C., Li, H.-X., Dong, D., & Tarn, T.-J. (2019b). Incremental reinforcement learning with prioritized sweeping for dynamic environments. *IEEE/ASME Transactions on Mechatronics, 24*(2), 621–632.

Wehrmann, J., Becker, W. E., & Barros, R. C. (2018). A multi-task neural network for multilingual sentiment classification and language detection on twitter. In *Proceedings of the 33rd annual acm symposium on applied computing* (pp. 1805–1812). ACM.

Wohlwend, J., Elenberg, E., Altschul, S., Henry, S., & Lei, T. (2019). Metric learning for dynamic text classification,.

Xiao, T., Zhang, J., Yang, K., Peng, Y., & Zhang, Z. (2014). Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd acm international conference on multimedia* (pp. 177–186). ACM.

Xu, D., Cheng, W., Luo, D., Liu, X., & Zhang, X. (2019). Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In *Proceedings of the 28th international joint conference on artificial intelligence* (pp. 3947–3953). AAAI Press.

Xu, J., Xu, C., Zou, B., Tang, Y. Y., Peng, J., & You, X. (2018). New incremental learning algorithm with support vector machines. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Xu, S. (2018). Bayesian naïve bayes classifiers to text classification. *Journal of Information Science, 44*(1), 48–59.

Yan, L., Zheng, Y., & Cao, J. (2018). Few-shot learning for short text classification. *Multimedia Tools and Applications, 77*(22), 29799–29810.

Yao, L., Mao, C., & Luo, Y. (2019). Graph convolutional networks for text classification. In *Proceedings of the aaai conference on artificial intelligence: 33* (pp. 7370–7377).

Zanaty, E. (2012). Support vector machines (svms) versus multilayer perception (mlp) in data classification. *Egyptian Informatics Journal, 13*(3), 177–183.

Zhai, Y., Song, W., Liu, X., Liu, L., & Zhao, X. (2018). A chi-square statistics based feature selection method in text classification. In *2018 ieee 9th international conference on software engineering and service science (icsess)* (pp. 160–163). IEEE.

Zhang, J., Lertvittayakumjorn, P., & Guo, Y. (2019). Integrating semantic knowledge to tackle zero-shot text classification. arXiv:1903.12626.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on machine learning* (p. 116). ACM.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems* (pp. 649–657).

Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning.. In *Aaai: 8* (pp. 1433–1438). Chicago, IL, USA.