sensors

MDPI

Article

Effective Model Update for Adaptive Classification of Text Streams in a Distributed Learning Environment

Min-Seon Kim [1], Bo-Young Lim [1], Kisung Lee [2] and Hyuk-Yoon Kwon [3,*]

[1] Department of Industrial Engineering, Seoul National University of Science and Technology, 232 Gongneung-ro, Nowon-gu, Seoul 01811, Republic of Korea
[2] Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA
[3] Department of Industrial Engineering, The Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, 232 Gongneung-ro, Nowon-gu, Seoul 01811, Republic of Korea
* Correspondence: hyukyoon.kwon@seoultech.ac.kr

Abstract: In this study, we propose dynamic model update methods for the adaptive classification model of text streams in a distributed learning environment. In particular, we present two model update strategies: (1) the entire model update and (2) the partial model update. The former aims to maximize the model accuracy by periodically rebuilding the model based on the accumulated datasets including recent datasets. Its learning time incrementally increases as the datasets increase, but we alleviate the learning overhead by the distributed learning of the model. The latter fine-tunes the model only with a limited number of recent datasets, noting that the data streams are dependent on a recent event. Therefore, it accelerates the learning speed while maintaining a certain level of accuracy. To verify the proposed update strategies, we extensively apply them to not only fully trainable language models based on CNN, RNN, and Bi-LSTM, but also a pre-trained embedding model based on BERT. Through extensive experiments using two real tweet streaming datasets, we show that the entire model update improves the classification accuracy of the pre-trained offline model; the partial model update also improves it, which shows comparable accuracy with the entire model update, while significantly increasing the learning speed. We also validate the scalability of the proposed distributed learning architecture by showing that the model learning and inference time decrease as the number of worker nodes increases.

Keywords: event classification; text streams; distributed learning; continual learning; dynamic model update

Citation: Kim, M.-S.; Lim, B.-Y.; Lee, K.; Kwon, H.-Y. Effective Model Update for Adaptive Classification of Text Streams in a Distributed Learning Environment. *Sensors* **2022**, 22, 9298. https://doi.org/10.3390/s22239298

Academic Editor: Meng Han

Received: 2 November 2022
Accepted: 25 November 2022
Published: 29 November 2022

1. Introduction

Twitter is one of the popular social networking services dealing with text streams, which provides a fast and interactive channel where the users write tweets and obtain access to the written tweets related to the latest events [1]. There have been lots of research efforts to classify the tweets and detect certain types of events using the collected tweets, focusing on the offline classification model [2,3]. Because tweets have short texts, it causes great difficulties in classification. Batool et al. [2] extracted knowledge from tweets and classified tweets based on the semantics of tweets. Shin et al. [3] proposed a text classification model for detecting cyber-security-related tweets by introducing two contrastive word embedding models that are positive and negative to the target events.

Considering that new information and responses to events are rapidly generating, it is important to quickly reflect data generated in real time into the model. However, applying the large-scale data streams to the classification model dynamically causes many limitations because it requires high costs in re-training the model to reflect the new data generated in real time. To confirm the overhead of the learning process, we conduct a preliminary experiment that compares the data ingestion, learning, and inference time by scaling the number of tweets. For this experiment, we used one of the datasets used in the experiments, i.e., the CSI dataset, and the bidirectional LSTM (Bi-LSTM)-based classification model. Table 1 shows the results that clearly confirm the learning process is a bottleneck in the entire classification pipeline considering its portion and the increased ratio according to the data scale compared to the ingestion or inference time. In the previous study [4], the relatively high portion of learning time over the data ingestion time has been presented to show the necessity of effective data ingestion to retain the ingested datasets for further learning. In this study, we focus on how the portion of the learning time becomes longer compared to the ingestion time as the data increases, showing the necessity of the efficient model update for large-scale data streams. Specifically, the learning time over the ingestion time is 5.43 times in the case of 200,000 tweets, whereas it is 4.42 times in the case of 50,000 tweets. This indicates that we need an effective model update method in an environment where the streaming data continuously flows in by efficiently learning the classification model while maintaining the model accuracy.

Table 1. The elapsed time for ingesting, learning, and inferencing tweets.

Number of Tweets	Ingestion Time	Learning Time	Inference Time
50,000	266 s	1120 s	0.0671 s
100,000	523 s	2467 s	0.0784 s
150,000	751 s	4076 s	0.0854 s
200,000	1094 s	5604 s	0.1345 s

In this study, we deal with the problem of updating the text classification models dynamically in a distributed environment to respond to the streaming data flowing in at a fast speed. In this regard, we claim two research objectives of this study. First, according to the continuously changing event trends in the streaming data, we need to efficiently reflect the changes into the existing model to respond to those changes in a real-time manner. Second, considering that large-scale streaming data flows at a high speed, we need to design a scalable architecture based on the distributed environment, accelerating the learning and inference speed of the model. We note that the existing classification models for the streaming data were not effective in terms of their efficiency and scalability in reflecting the dynamic changes into the model in a distributed environment.

There have been lots of research efforts to increase the performance of event classification in data streams [3,5–12]. Because the performance of the classifier is significantly affected by the underlying embedding models, effective word-embedding methods have been proposed [3,6,11]. On the other hand, a few studies have investigated building a set of classifiers and selectively utilizing them to respond to non-stationary data streams [9,10,12]. However, a scalable architecture for the streaming event classification in a distributed environment, which is required to efficiently respond to detect and track the events from data sources with large volumes and high velocity, has not been considered before.

Learning the model in a centralized server has an inherent limitation in providing scalability for dealing with massive-scale data. Therefore, distributed learning has evolved as the solution by distributing the overhead of maintaining a global model in a centralized server into multiple nodes. Lots of recent research efforts have focused on the cooperation of multiple nodes to update a global model [13–17]. On the other hand, the efficient reflecting of local changes in each local model in a distributed environment has also been addressed in several studies [18–21]. However, there have been no research efforts to dynamically update the distributed classification model to respond to the changing non-stationary event streams.

To respond to non-stationary data streams, distributed online learning has been explored in various fields [22–26], and they target common research goals of dynamically updating the model in a distributed environment. All previous studies focused on improving the model performance targeting a specific model, but it is essential to immediately update the model by reflecting newly generated data streams to the model in order to respond to a new event trend in real-time data streams. In this regard, our distinguishing research goal that is different from the existing distributed online learning methods is proposing an efficient model update method while minimizing the accuracy loss, not improving the model performance. Furthermore, we do not focus on a specific classification model and, instead, identify common trainable modules for the typical classification models and apply the update strategies to the various classification models.

In this study, we propose the dynamic model update methods for the adaptive classification model of data streams in a distributed learning environment. Based on the scalable architecture in a distributed environment, we present two model update strategies: (1) the entire model update and (2) the partial model update. The former aims to maximize the model accuracy by accumulating all the datasets including recent datasets and rebuilding the model periodically as the trends in data streams change. Accordingly, its accuracy increases over time as the data increases, but the learning time significantly increases at the same time. The latter fine-tunes the model only with a limited number of recent datasets, noting that the data streams are dependent on a recent event. Therefore, it accelerates the learning speed while maintaining a certain level of accuracy. To verify the effectiveness of the proposed update strategies, we extensively evaluate not only fully trainable language models based on CNN, RNN, and Bi-LSTM but also a pre-trained word-embedding model based on BERT. In particular, we identify the partial modules that can be effectively updated with a marginal overhead for all the models in common.

The contributions of the paper can be summarized as follows:

- We design a scalable classification model based on a distributed learning environment that enhances the parallelism of the model learning. Therefore, it can resolve the bottleneck that occurred during the learning process in the entire event stream classification pipeline (Section 3.1).
- Based on a distributed learning architecture, we propose two kinds of model update strategies: (1) the entire model update and (2) the partial model update. Because they have their distinguishing properties in terms of learning efficiency and model accuracy, they can be selectively chosen according to the needs of the target applications (Sections 3.3 and 3.4).
- As the target classification models, we consider not only fully trainable language models based on CNN, RNN, and Bi-LSTM but also a pre-trained word-embedding model based on BERT. In particular, we identify the trainable partial modules that are commonly applied in the deep learning-based classification models (Section 3.2).
- We conduct extensive experiments using two real tweet datasets and show the effectiveness of the proposed update strategies. Specifically, the entire model update gradually improves the classification accuracy in the range of 28.96∼58.63% compared to the pre-trained offline model; the partial model update improves it in the range of 12.34∼50.92%, while significantly reducing the learning time by 69.35∼93.95% compared to entire model update strategy. We also confirm the scalability of the proposed distributed learning architecture by showing that compared to using a single worker node, the learning time decreases by 34.03% in the entire model update and by 45.21% in the partial model update, respectively, when using three worker nodes (Section 4).

The remainder of this paper is organized as follows. In Section 2, we describe the related work. In Section 3, we present the proposed dynamic model update methods. In Section 4, we present the experimental results. In Section 5, we conclude the paper and discuss future work.

2. Related Work

2.1. Data Stream Classification

Lots of research efforts for detecting the events based on the classification model from the streaming data have been conducted. Mittal et al. [27] described the necessity of incremental algorithms to achieve the consistent accuracy of the classifier in streaming environments to respond to the concept drift. They evaluated five data stream mining algorithms in various drifting settings and found that none of each algorithm outperforms the others in all settings due to a trade-off between the model accuracy and training speed of the algorithms. Nishida et al. [5] proposed a classification model for streaming tweets by examining the changes in class distributions and probabilities of word occurrences. Weiler et al. [6] monitored shifts in the inverse document frequency (IDF) of terms to identify events from large-scale SNS streams. Zyblewski et al. [9] proposed a dynamic classifier based on ensemble selection methods to classify the non-stationary data streams. Shin et al. [3] proposed a text classification model that detects cyber-security-related tweets by introducing a contrastive word-embedding model that defines positive and negative embedding models to the target event. Malialis et al. [12] proposed a new density-based active learning strategy based on the similarity in the latent space for non-stationary and imbalanced data streams. Nguyen et al. [7] extracted and tracked social events on real-time data streams by aggregating discrete signals representing relevant keywords from the tweets collected by the event categorization. Eddaoudy et al. [28] proposed a distributed machine learning model on streaming data based on Apache Spark to learn the event streams and to predict events in real time.

The previous studies have not considered the classification model that can be dynamically updated as the input stream changes. In this problem, because event trends fluctuate over time, we need to periodically re-train the classification model to maintain the model accuracy. However, re-training of the entire model iteratively requires considerable costs and time. To resolve this challenge, we propose a partial model update strategy that effectively updates the model with a marginal update overhead while maintaining the model accuracy.

2.2. Distributed Learning

Distributed learning has become popular due to the explosion in the size and complexity of datasets to be learned. Gupta et al. [29] proposed a model partitioning strategy over multiple agents and further incorporated it with semi-supervised learning using a few labeled samples. Huang et al. [30] combined approximate augmented Lagrangian function with time-varying gaussian noise addition in a distributed learning framework with differential privacy, providing performance improvement. Chen et al. [15] updated a central model in an asynchronous manner to cope with the heterogeneity of distributed edge devices. Wang et al. [16] proposed a distributed modulation classification model based on the cooperation of multiple edge devices and a model averaging algorithm. They achieved lower computing overhead than centralized modulation classification to achieve a similar convergence speed. Park et al. [20] presented a communication-efficient distributed learning framework that enables edge nodes to proactively and independently react to local changes. Gao et al. [31] divided the learning process of graph neural networks into two stages to resolve the mismatch between the graphs. They first learned nonlinear representations from raw data at the training stage and retrained the linear representations at the testing stage.

Recently, Apache Spark [32] became a popular choice to convey big data analytics or machine learning tasks for large-scale datasets based in a distributed environment. Several research efforts for distributed learning of neural networks have been conducted based on Apache Spark. Dunner et al. [33] proposed practical techniques to achieve the best performance in Apache Spark, targeting any distributed algorithms and infrastructures. Zhao et al. [34] proposed a scalable stochastic optimization method on Apache Spark that achieves both computation and communication efficiency. Alkhoury et al. [35] proposed the communication-efficient distributed learning model on Apache Spark and applied it to image segmentation on large-scale datasets. In this study, we deal with the dynamic model update problems on deep learning-based classification models on Apache Spark, which has not yet been studied before.

2.3. Continual Learning

Continual learning deals with the problem of learning from potentially infinite data streams with the goal of preserving and extending acquired knowledge. This is also referred to as lifelong learning or incremental learning in the literature. The main challenge with continual learning is catastrophic forgetting, which is a tendency of neural networks that forget previously learned knowledge in the learning process for new data. Various methodologies [36–50] for continual learning have been proposed to effectively handle sequential new tasks while maintaining the classification accuracy of previous tasks. The methodologies can be largely categorized into four groups: (1) regularization-based methods, (2) knowledge distillation methods, (3) rehearsal-based methods, and (4) dynamic architecture methods.

Regularization-based methods modify the gradient of parameters for optimization by assigning constraints to the weights to be updated. Kirkpatrick et al. [36] proposed a regularization to model parameters by selectively learning important weights for old tasks. In a similar way, Zenke et al. [37] extended the loss function to penalize changes to parameters that are unimportant for old tasks. Mirzadeh et al. [38] assessed the impact of different training regimes on catastrophic forgetting and widened the curvature of each task to prevent catastrophic forgetting. Yoon et al. [51] decomposed the entire model parameters into task-generic parameters and task-specific parameters to maintain inference accuracy between different tasks.

Knowledge distillation methods attempt to alleviate the catastrophic forgetting issue by distilling the knowledge learned from the previous data when learning new tasks. Li et al. [39] preserved the knowledge learned from past tasks by using a distillation loss. Rebuffi et al. [40] re-defined the loss function using both classification loss and distillation loss to distill the knowledge learned from the existing tasks when learning new tasks. Castro et al. [41] trained the deep learning-based models by minimizing both cross-entropy loss to learn new classes and distillation loss to retain the previous knowledge.

Rehearsal-based methods build and store a memory of the knowledge learned from old tasks and periodically replay the model to strengthen connections with previous knowledge. Rebuffi et al. [40] maintained the exemplar set of previous representative data samples in memory and updated them when new data are observed. Chaudhry et al. [42] built a dynamic episodic memory of parameter gradients during the learning process for leading to a faster learning process while not forgetting each individual task. Wang et al. [43] performed meta-learning of the model to learn a better initialization for local adaptation. Some studies generated synthetic data by learning the data distribution of previous tasks and used them in learning new tasks. Shin et al. [44] proposed a novel framework with a deep generative model to enable previous training data to be sampled and interleaved with those for a new task. Wang et al. [45] replayed data sampled from the conditional generative adversarial network. They selectively stabilized the parameters of the discriminator for discriminating the pairs of old unlabeled data and their predicted pseudo-labels to overcome the catastrophic forgetting of unlabeled data.

Dynamic architecture methods are typically used in task-incremental learning to learn the task-specific parameters or networks. Rebuffi et al. [49] introduced universal parametric families of neural networks that contain both domain-shared parameters among multiple domains and domain-specific modular adapters and attached them to the network for new tasks. Rusu et al. [46] proposed a training method that grows a network hierarchically to handle new coming data. Mallya et al. [47] pruned and retrained the network by obtaining the sparsity masks for the tasks and utilizing them to freeze the corresponding network weights. Mallya et al. [48] masked unimportant parameters for previous tasks to train the parameters for new tasks. Ashfahani et al. [50] proposed an autonomous deep learning algorithm based on the self-constructing structure generating different depths and widths. Cano et al. [52] proposed an ensemble architecture with the concept drift to deal with imbalanced data streams by reflecting them to the model in an online manner.

Previous studies for continual learning focused on preserving and extending the acquired knowledge in the process of learning new data to prevent catastrophic forgetting. In contrast, in this study, we focus on the efficient model update that can respond to the continuously changing data streams while maintaining a certain level of classification accuracy.

2.4. Distributed Online Learning

Time-dependent data learning has drawn a lot of attention to an online learning framework in the last few years. Specifically, introducing a distributed environment aims to minimize the computing overhead and to provide scalability to the increased data scales while maintaining the model performance. Tekin et al. [22] proposed distributed online learning algorithms that lead each processor to learn itself for maximizing the total expected rewards from its own actions without the interaction between processors. Zhang et al. [23] explored an online conditional gradient algorithm with simple linear optimization steps in the distributed online learning setting. Li et al. [24] developed a privacy-preserving distributed online learning framework by building independent local models based on local datasets and exchanging intermediate parameters with neighboring nodes to achieve convergence. They showed that the Euclidean distance of all learnable parameters became shorter over iterations. Wang et al. [14] managed the dynamic resource constraints by adaptively and periodically choosing the optimal global aggregation frequency, considering the network cost and model performance simultaneously. Wu et al. [26] proposed a distributed hierarchical online learning approach to enhance the robustness by reducing the long-term cost when converging to a new local optimal.

In this study, we target the problem where the trends in the data streams are continuously changing. In this problem, we need to update the model periodically to reflect new data streams according to the changing events, requiring intensive learning overhead. To address the problem effectively, we propose a distributed architecture on Apache Spark that can reflect real-time data streams by updating the model with an appropriate time overhead. The architecture provides methods for dynamically updating the model in a distributed learning environment by considering both learning efficiency and classification accuracy.

2.5. Summary

In this study, we aim to provide an efficient model update method in a distributed environment while maintaining a certain level of accuracy. Table 2 summarizes the coverage of related studies in terms of (1) streaming classification, (2) distributed learning, (3) dynamic model update, and (4) model learning efficiency, which are the four main focuses of this study. As shown in the table, none of the previous studies have considered all four focuses of this study. Distributed online learning, which was described in Section 2.4, has dealt with the most similar issues to our study but has not considered an environment where large amounts of data flow in, requiring the immediate reflection of them to the model [14,23]. As a result, they are not appropriate to be used for updating the classification model efficiently to respond to the streaming data flowing in at a fast speed. To the best of our knowledge, our study is the first research effort to update the model in an online manner to classify the streaming data in a distributed environment.

A preliminary version of this study was presented as a conference short paper [4]. In this paper, we fully rewrite and extensively extend it. The major extensions include (1) proposing a completely new dynamic model update strategy, partial model update, which is a more effective and practical method by providing immediate model updates while maintaining a certain level of accuracy, compared to the entire model update strategy in the preliminary version, (2) extensive experiments on two tweet datasets using two model update strategies, (3) extensive applications of the idea to not only the fully trained language models based on CNN, RNN, and Bi-LSTM but also the pre-trained word-embedding model based on BERT, and (4) the detailed and extensive literature reviews for related work.

Table 2. Comparison between the previous studies and our model.

Papers	Streaming Classification	Distributed Learning	Dynamic Model Update	Model Learning Efficiency
[5]	O		O	
[7]	O			
[12]	O		O	
[14]	O	O		O
[20]		O		O
[23]	O	O	O	
[29]		O		
[30]	O	O		
[40]	O		O	
[48]			O	
[51]		O	O	
[52]	O		O	O
Our model	O	O	O	O

3. Proposed Method

3.1. Overall Framework

Figure 1 shows the overall architecture of our proposed framework. For distributed learning and classification, our framework is designed to run on an Apache Spark cluster with one master node and a set of worker nodes. We run Spark ML pipelines on the cluster to classify streaming data in real time and update classification models for each time window in a distributed manner. In our experimental settings, we prepare a total dataset related to the target event in advance and feed them by the time window to control the data streams according to our intention. Here, we use a set of seed keywords defined for an event. Our classification model on each worker node can determine if each streaming data (e.g., tweet selected using the seed keywords) is actually related to the event in real time. To efficiently handle large-scale streaming data in a scalable manner, we distribute the collected data to the worker nodes, which take data using the data stream consumer. We develop our distributed training and classification pipelines using the transformers (e.g., feature vector generation, classification using a learned model) and estimators (i.e., learning algorithms) provided by Spark's machine learning library (Apache Spark MLlib). The word representation update module processes each data to use it as an input for model training or classification and has two main processing steps: (1) pre-processing step, including stemming and lemmatization, and (2) word-embedding step, including text vectorization. To dynamically update our classification model using recently collected data, our framework has the batch jobs scheduler that updates the frequent word list and coordinates model updates. For the model updates, our framework assumes that all or a part of the newly collected data in the current window are labeled by using any available labeling techniques, such as crowdsourcing-based labeling and emerging approaches (e.g., active learning, semi-supervised learning, and self-supervised learning) [53–58]. It is worth noting that our study focuses on reflecting non-stationary features presented in real-time data streams more efficiently on the model, not improving model performance using more accurate labeling. For distributed model updates on the cluster, we initialize a deep learning-based model on the driver of the master node and ship its serialized version to the worker nodes with model parameters. Each worker node deserializes the model, trains it using its chunk of data, and sends its gradients back to the master node, which aggregates the gradients and updates the master model. We distribute the updated master model to worker nodes and replace the classification model with the updated one to classify streaming data in real time.

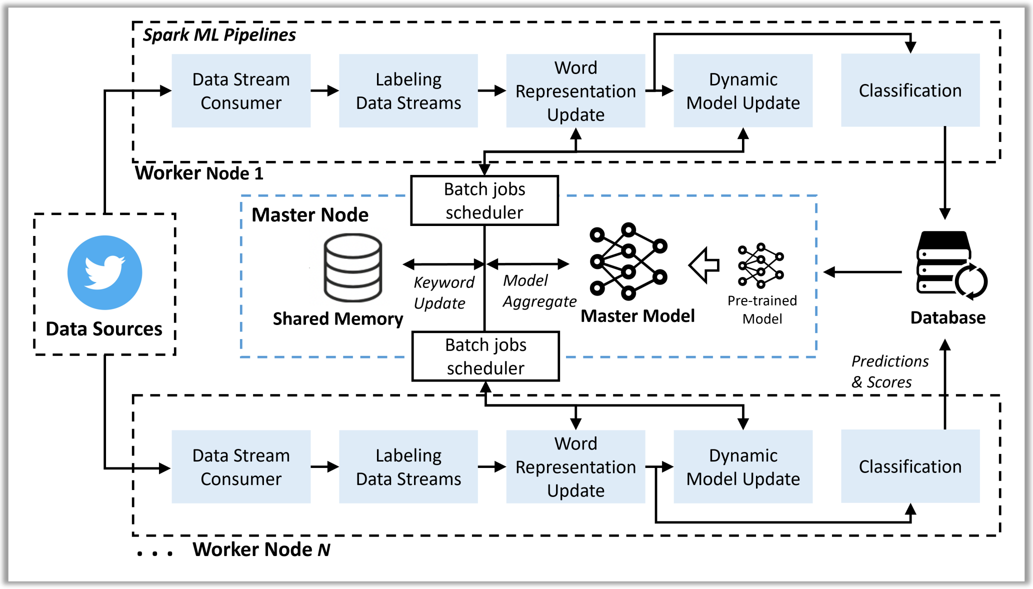


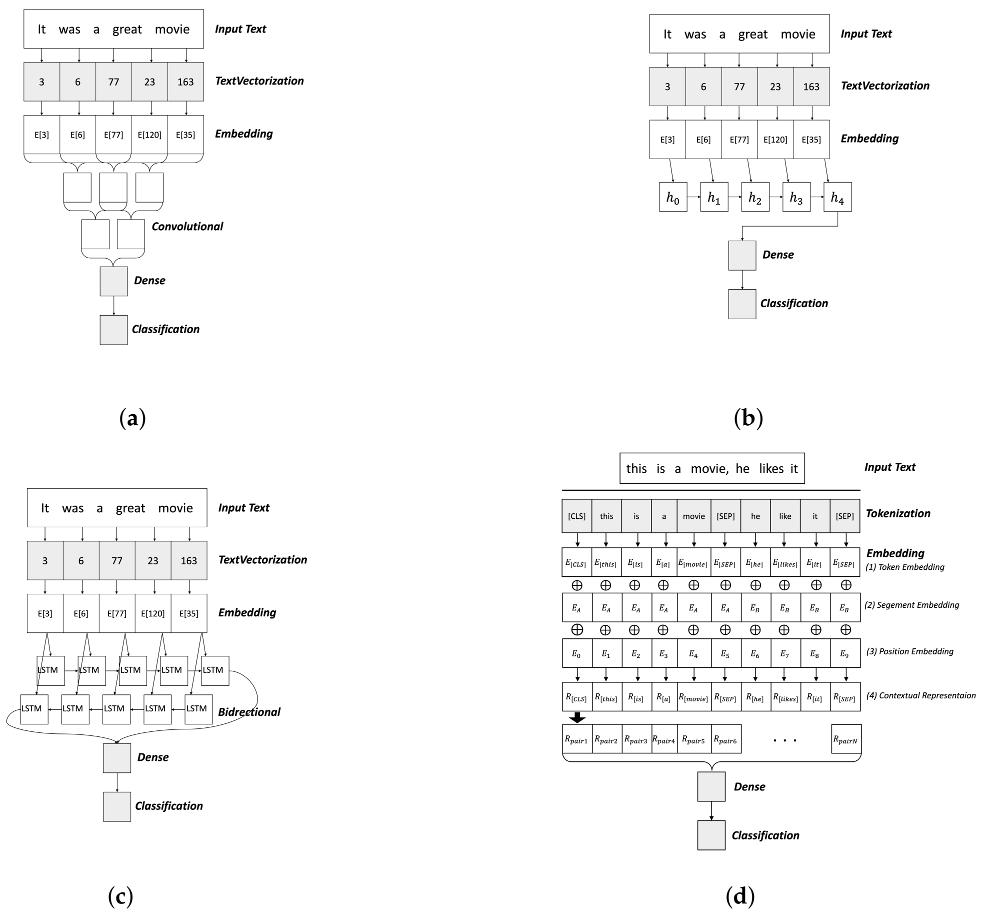
Figure 1. Overall framework architecture.

3.2. Classification Model

In the data streams, the trends of the data can be dynamically changed over time. In particular, the streaming data are significantly sensitive to recently occurred events. Therefore, we need to respond to them by incrementally updating the model using the newly collected data, in particular, focusing on recent data. However, as streaming data are continuously reached, reflecting them to the existing models is a challenging issue because the learning overhead of the models significantly increases as described in Table 1. In this study, we focus on the efficient model update reflecting the evolving recent trends in the streaming data while maintaining the accuracy of the classification model. In particular, we consider the learning model in a distributed environment to deal with massive-scale datasets with continuously increasing volumes. We present two kinds of model update strategies: (1) the entire model update (described in Section 3.3) and (2) the partial model update (described in Section 3.4).

For dynamic model updates in this study, we employ learning architectures based on three typical neural networks (CNN, RNN, and Bi-LSTM) and a transformer-based pre-trained language model (BERT) as shown in Figure 2 in which trainable layers are highlighted in gray. In the CNN-, RNN-, and Bi-LSTM-based architectures, an embedding layer consists of a 100-word sequence of each sentence. A sequence of integers with text data is fed as input, which corresponds to word indices in a sentence to the layer. The layer generates up to 100-word embeddings for each text using the top-5000 frequently occurred keywords in the text corpus and learns the vector representation of each word. We denote this embedding model as *event-specific word embedding*, fully training the embedding layer with the datasets for the target event, and we also apply it to the BERT-based architecture to compare it with the original BERT pre-trained word embedding. By continuously updating the top 5000 keywords, the embedding layer can reflect trend changes in streaming data.

In the CNN-based architecture (Figure 2a), the word embeddings of input data are connected to a 1D convolutional layer that extracts features by sliding along the word embeddings in sequence to look at embeddings of multiple consecutive words at the same time.In the RNN-based architecture (Figure 2b), the ordered word embeddings of data (i.e., an embedding sequence) are used as an input to a hidden layer that processes the sequence in the forward direction. In the Bi-LSTM-based architecture (Figure 2c), by processing the embedding sequence in the forward and backward directions, we can keep track of information in the sequence from both directions. Each architecture has a dense layer with the sigmoid activation function as its last layer to perform binary classification for each data. The BERT-based architecture is described in Section 3.5.



It was a great movie
Input Text
3 6 77 23 163
TextVectorization
E[3] E[6] E[77] E[120] E[35]
Embedding
Convolutional
Dense
Classification
(a)
h_0 h_1 h_2 h_3 h_4
(b)
LSTM
Bidrectional
(c)
this is a movie, he likes it
[CLS] this is a movie [SEP] he like it [SEP]
Tokenization
(1) Token Embedding
(2) Segement Embedding
(3) Position Embedding
(4) Contextual Representaion
R_{pair1} R_{pair2} R_{pair3} R_{pair4} R_{pair5} R_{pair6} . . . R_{pairN}
(d)

Figure 2. Deep learning-based architectures with trainable layers (**a**) CNN, (**b**) RNN, (**c**) Bi-LSTM, (**d**) BERT.

3.3. Entire Model Update

As the first strategy for dynamic model updates, we propose the *entire model update*, which trains a new classification model from scratch using all accumulated data in each time window, as shown in Figure 3. It is worth noting that the word frequency list is being continuously updated as our framework takes new data, and we utilize the updated top-5000 frequent keywords in each time window for the entire model update to reflect the recent trends of streaming data. After the model is trained with new input vectors, it is serialized and shipped to the distributed worker nodes and used to classify the newly generated data streams during the upcoming time window. Even though the entire model update can generate a more accurate model by digesting all accumulated data, one main limitation of this strategy is limited scalability, because we need to keep all collected data on our Spark cluster, and consequently, the model learning time will continuously increase as we use more data for training in each window, as evidenced in Table 1.

3.4. Partial Model Update

To address the limited scalability of the entire model update strategy, we propose a lightweight strategy, called the *partial model update*, that updates only a portion of the classification model based on the pre-trained offline model, as shown in Figure 3. In this strategy, we pre-train a classification model using previously collected data in an offline manner and fine-tune only a part of the model using newly streamed data in the current time window. Specifically, only the dense layer is fine-tuned with the newly streamed data in each time window. Like the entire model update strategy, the word frequency list is continuously updated as our framework takes new data, and we utilize the updated top-5000 frequent keywords in each time window to reflect the recent trends of streaming data. Unlike the entire model update strategy, the partial model update strategy does not keep all accumulated data on a cluster because it needs only newly streamed data in the current window for fine-tuning. Assuming a similar number of streaming data in each window, we expect that the partial model update (i.e., fine-tuning) would require a consistent learning time for all windows, tackling the scalability issue of the entire model update strategy.

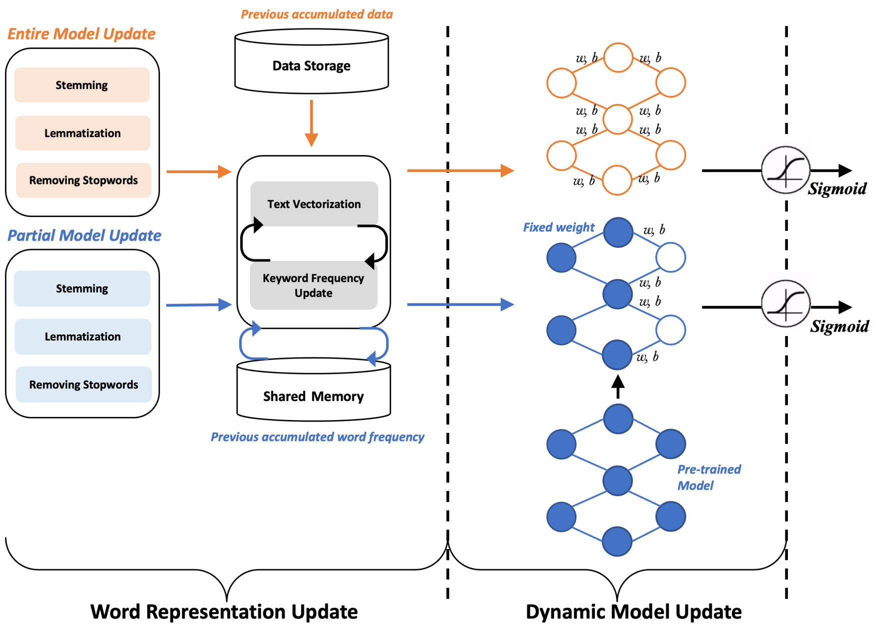


Figure 3. Continual machine learning pipeline for classification supporting dynamic model update.

3.5. Application to Pre-Trained Embedding Model

In this study, we also consider pre-trained embedding models, which have been known to show better performance than the typical fully trained deep learning models in most natural language processing (NLP) tasks. Devlin et al. [59] proposed Bidirectional Encoder Representations from Transformers (BERT) that is trained using unlabeled data extracted from BooksCorpus [60] and English Wikipedia. It can be fine-tuned by adding the output layer tailored to target NLP tasks. There is a major computational advantage of pre-computing representations of input data and then using lightweight models on top of these representations to apply them to downstream tasks. As shown in Figure 2d, BERT reads a complete sequence of words in parallel, enabling the model to understand each word’s context as a result of the relationship with neighboring words. We update the top-5000 frequent keyword sets in the tokenization process similar to the other models before embedding processes (e.g., token embedding, segment embedding, and position embedding). Here, we set the maximum word sequences for each sentence as 100. By embedding updated keywords that are considered important at each time window, this model can also adaptively respond to the changes in input data streams. BERT has a total of 110 million trainable parameters, and such high model complexity calls for expensive computational resources and extremely excessive training costs. Thus, the iterative re-training of BERT to respond to the streaming events is not a feasible approach [61,62]. Therefore, we consider only the partial model update strategy for the BERT-based architecture in our proposed framework.

4. Performance Evaluation

In this study, we perform extensive experiments to verify the effectiveness of our proposed framework using real-world datasets. The experiments aim to show that our strategies for the dynamic model updates are feasible to respond to the trend changes in the data streams in terms of both model accuracy and learning efficiency. Here, we apply two model update strategies (entire and partial) described in Section 3. We collected two kinds of datasets related to different target events from Twitter: (1) the cybersecurity intelligence (CSI) dataset and (2) the disaster dataset. To control the input data events as the time varies as we want, we prepare two kinds of datasets for each event: (1) the relevant dataset to the event and (2) the irrelevant dataset to the event. Then, we divide them into sub-datasets to feed each sub-dataset to each time window. The datasets used for training the model in each time window depend on each update strategy. We extensively apply our proposed dynamic model update strategies into three typical neural networks (CNN, RNN, and Bi-LSTM) and a transformer-based pre-trained model (BERT). We measure the classification accuracy and the learning time as the evaluation metrics.

4.1. Datasets

Table 3 shows the dataset name, the number of tweets, the number of time windows, and the time period for our datasets. By leveraging the accounts and keyword sets that are relevant to the target event, we collect the actual tweets by crawling them in time order. We use 80 percent of the data in each time window to train and validate the model and the remaining 20 percent to test the model performance. We explain the details of CSI and disaster datasets in Sections 4.1.1 and 4.1.2, respectively.

Table 3. The used twitter datasets.

Dataset Name	Total Data Size	Number of Time Windows	Target Written Duration (Years)
CSI Dataset	1,000,000 tweets	6	2007 $\sim$ 2015
Disaster Dataset	1,400,000 tweets	5	2007 $\sim$ 2015

4.1.1. CSI Dataset

For collecting the CSI-related dataset, we focus on the tweets containing the keyword ‘exploit’. Through the analysis engine on the target tweets by Recorded Future (https://www.recordedfuture.com (accessed on 31 October 2022)), we obtain 100 accounts and 639 keywords relevant to the cyber-security, e.g., ‘Internet-security’, ‘flaw’, ‘PoC’, and ‘CVE’. We collect all the tweets from 2007 to 2015 posted by the selected accounts. Then, we filter only the tweets containing at least one keyword in the relevant keyword set and define them as the CSI-related dataset. For the CSI-unrelated dataset, we collect random tweets containing general keywords, which are in the top-10 most commonly used English words, such as ‘the’, ‘to’, and ‘a’, based on an analysis of the Oxford English Corpus (http://oxforddictionaries.com/us/words/the-oxford-english-corpus (accessed on 31 October 2022)). This dataset has one million tweets including 500,000 CSI-related tweets and 500,000 CSI-unrelated tweets. Examples of CSI-related tweets are as follows: “A very deep dive into iOS Exploit chains found in the wild” and “Binary Exploitation—Buffer Overflow Explained in Detail”. We use 400,000 tweets to pre-train the model and an additional 100,000 tweets to re-train the model for each time window, with the same portion between CSI-related and CSI-unrelated datasets.

4.1.2. Disaster Dataset

For the disaster-related dataset, we use a disaster-related keyword set consisting of 24 keywords (e.g., Flood, Epidemics, Windstorm) identified by Apronti et al. [63]. Since some of them could not be related to actual disasters, we collect the tweets satisfying the following conditions: (1) containing at least two keywords in the disaster keyword set, (2) having more than 10 characters, and (3) having more than five distinct words. We collect tweets satisfying the conditions posted from 2007 to 2015. For the disaster-unrelated data, we collect them in the same way as the CSI-unrelated dataset. This dataset has 1.4 million tweets including 700,000 disaster-related tweets and 700,000 disaster-unrelated tweets.

Examples of disaster-related tweets are as follows: "Ian Downgraded to Tropical Storm, Flooding Threats Remain." and "typhoon linpha causes flooding in northern philippines storm natural disaster". We use 400,000 tweets to pre-train the model and an additional 200,000 tweets for each time window, with the same portion between disaster-related and disaster-unrelated datasets.

4.2. Experimental Methods and Environments

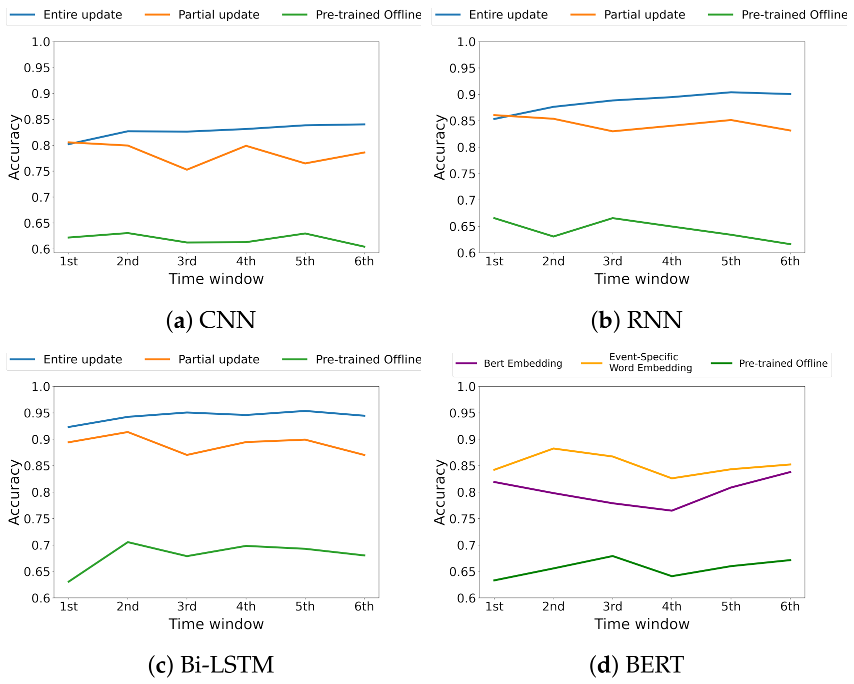
For the experiments, we use one master node and three worker nodes with Apache Spark 2.4.7 managed by Hadoop Yarn. Each node is equipped with Intel Xeon Silver 4210R 2.40 GHz CPU and 32 GB RAM and runs Ubuntu 18.04. To focus on the distributed environments, we only utilize the CPU-based environments without GPU devices. To apply the proposed method to various deep learning models, we evaluate four kinds of deep learning-based classification models: (1) CNN, (2) RNN, (3) Bi-LSTM, and (4) BERT. For training the event-specific word embedding of the first three models, we feed a sequence of integers with 400,000 data samples as input and learn the vector representation of each word.

For the model based on CNN, we employed a one-dimensional convolutional neural network layer with 256 units for feature extraction and one hidden layer for the classifier. For the model based on RNN, we adapted three SimpleRNN layers with 256 units for feature extraction and one hidden layer for the classifier. For the model based on Bi-LSTM, we adapted the Bi-LSTM layer with 128 units for feature extraction and one hidden layer after the embedding layer for the classifier. For all models, we adapted the dense layer with the sigmoid activation function after the hidden layer and performed binary classification for input tweets using the Adam optimizer. We commonly set the number of epochs to 10, the batch size to 128, and binary cross-entropy as a loss function in all settings. Using Elephas Estimator (https://github.com/maxpumperla/elephas (accessed on 31 October 2022)) supported by Spark MLlib, we run distributed classification models at scale on Apache Spark. For the fine-tuning of BERT-based architecture, we use another BERT model supported by SparkNLP (https://github.com/JohnSnowLabs/spark-nlp (accessed on 31 October 2022)) because we actually observe that the fine-tuning of BERT-based architecture only for the final dense layer using Elephas Estimator takes more than an hour in processing one epoch with 100,000 data samples, which is infeasible to apply to the dynamic model update in the streaming classification. SparkNLP utilizes the transformer itself of Spark MLlib without the process of abstracting the learning algorithm for customizing the transformer tailored to the datasets, efficiently fine-tuning the model. Although there might occur accuracy loss in the trained model, we use the SparkNLP-based approach to focus on the efficient model update according to our research goal.

4.3. Experimental Results

4.3.1. Model Accuracy

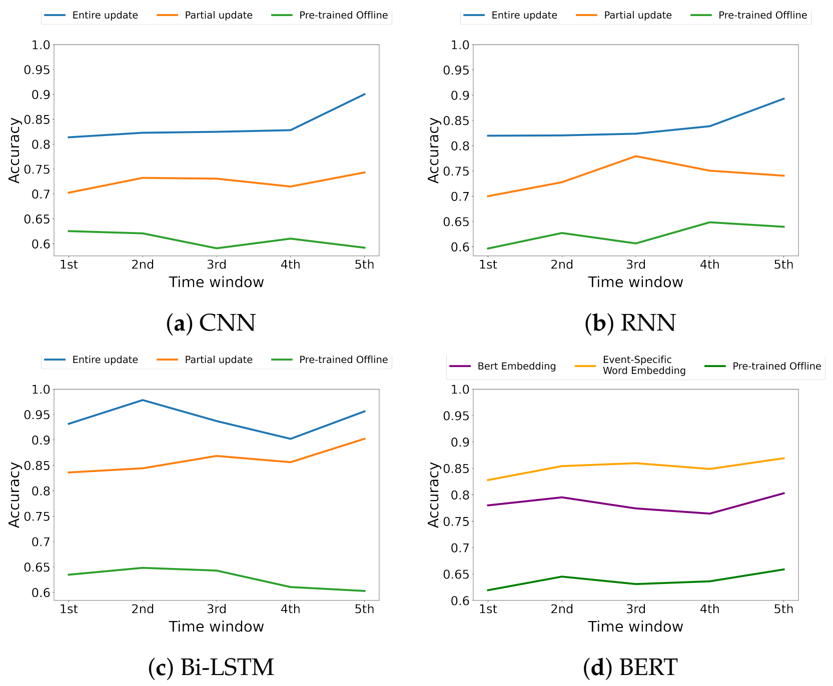
Figures 4 and 5 represent the accuracy over time when each strategy (i.e., (1) Pre-trained offline, (2) Entire model update, (3) Partial model update) has been adopted on the CSI and disaster datasets, respectively. We note that both entire and partial update strategies show a distinct accuracy improvement compared to the strategy where the initial pre-trained offline model has been utilized for the inference. The results show that the entire model update improves the classification accuracy by 28.96$\sim$58.63% compared to pre-trained offline; the partial model update improves it by 12.34$\sim$50.92%, which indicates a competitive accuracy compared to the entire model update. The results indicate that both update strategies show consistent trends for all the employed models over time.



(**a**) CNN (**b**) RNN

(**c**) Bi-LSTM (**d**) BERT

Figure 4. Accuracy comparison between (1) Pre-trained offline, (2) Entire model update, and (3) Partial model update on CSI dataset.



(**a**) CNN (**b**) RNN

(**c**) Bi-LSTM (**d**) BERT

Figure 5. Accuracy comparison between (1) Pre-trained offline, (2) Entire model update, and (3) Partial model update on disaster dataset.

Figures 4d and 5d represent the accuracy of BERT-based architecture over time when each word embedding strategy (i.e., (1) Pre-trained offline, (2) Partial model update based on keyword-level word embedding, (3) Partial model update based on BERT Embedding) has been adopted on the CSI and disaster datasets, respectively. We note that both partial update models show a distinct performance improvement compared to the pre-trained offline model. The partial model update based on keyword-level word embedding, which is fully trained using the datasets defined for a target event, improves the classification accuracy by 21.21∼26.60% compared to the pre-trained offline model; that based on BERT embedding improves it by 12.08∼22.70%.

From the experimental results on both datasets, we point out the following three major observations. (1) The entire update strategy gradually increases the model accuracy for all the underlying deep learning-based models as time increases. This indicates that the accumulated data contribute to the improvement of the model performance. (2) The partial update strategy significantly improves the model accuracy of the pre-trained offline model and shows a competitive accuracy to the entire model update with a marginal model update overhead while maintaining the accuracy over time. This indicates that the partial update strategy can deal with the changes in the event effectively. (3) BERT is generally known to perform well by learning a large amount of data. However, despite fine-tuning, it does not perform well on the event-specific datasets targeted in this study compared to the other fully trained models using event-related datasets. To verify this, we compare BERT-based architecture using the original BERT embedding model with it using the event-specific word embedding for the other models. The results confirm that the model based on the original BERT embedding shows a classification accuracy of 0.8014 for the CSI dataset and 0.7833 for the disaster dataset, which are lower than 5.98% and 8.07% compared to the model based on the event-specific word embedding, respectively. This indicates that specific events have to be represented by a distinct set of keywords specifically related to those events, and so we need to build our own language models to represent each event.

4.3.2. Model Learning Time

Tables 4 and 5 represent the model learning time on the CSI and disaster datasets, respectively, for each time window of the following different update strategies: (1) no online update, (2) entire model update, and (3) partial model update. Here, we note that as more data are accumulated over time, the model learning time of the entire model update proportionally increases. In contrast, the partial update strategy consistently maintains the learning time as time varies on both datasets. We partially update the model by fine-tuning only the final dense layer to all the models in common. However, we observe that the model learning time significantly differs depending on the model. This stems from the inference complexity of each model because the inference from the fixed model is required for the input of the dense layer. In the case of BERT, we fine-tune only the dense layer through SparkNLP by using the learned transformer without fitting it. Accordingly, its learning time is comparatively short considering the massive-scale pre-trained model of BERT.

Table 4. The learning time of the proposed strategies on the CSI dataset (seconds).

		1st Time Window	**2nd Time Window**	**3rd Time Window**	**4th Time Window**	**5th Time Window**	**6th Time Window**
CNN	Entire	489.35	530.83	790.23	1040.21	1250.54	1560.38
	Partial	130.13	143.62	131.99	156.23	139.37	147.01
RNN	Entire	300.1	440.63	532.44	640.34	784.78	838.81
	Partial	91.4	97.17	86.66	78.28	75.91	73.26
Bi-LSTM	Entire	1550.45	1750.27	2250.17	3240.83	4010.45	4630.2
	Partial	1430.59	1382.2	1357.06	1436.02	1402.12	1419.21
BERT	Partial	60.36	62.27	70.23	51.29	54.33	58.15

Table 5. The learning time of the proposed strategies on the disaster dataset (seconds).

		1st Time Window	2nd Time Window	3rd Time Window	4th Time Window	5th Time Window
CNN	Entire	210.16	745.88	1140.34	1754.71	2165.9
	Partial	138.45	131.62	127.17	134.8	131.01
RNN	Entire	173.3	517.62	820.29	1209.09	1876.43
	Partial	116.17	112.53	109.8	128.41	115.69
Bi-LSTM	Entire	1840.23	4073.32	7580.61	8073.73	12587.11
	Partial	1770.3	1520.13	1646.1	1602.36	1457.14
BERT	Partial	159.66	127.52	133.68	149.84	130.62

4.3.3. Scalability on a Cluster

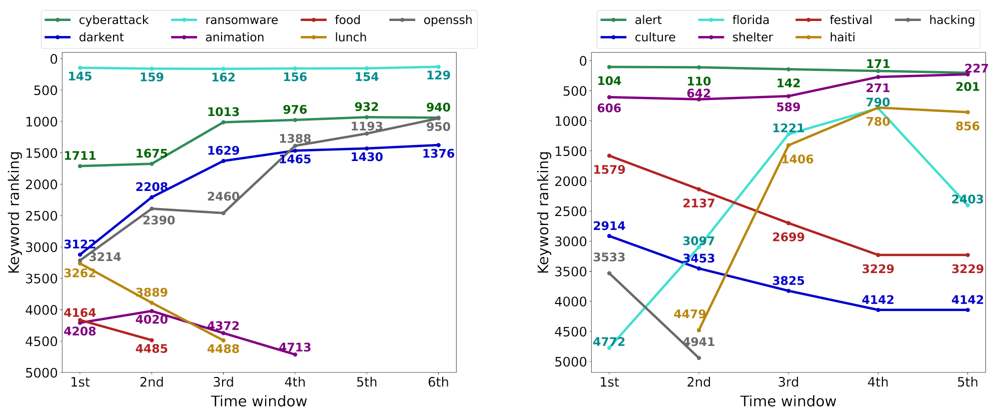
Table 6 shows the elapsed time for learning the classification model using 50,000 tweets as the number of worker nodes in the Spark cluster increases. Here, we use 10 epochs to train the Bi-LSTM model in Section 4.2 and assign one executor to each node consisting of five cores. The results indicate that the learning time of both entire and partial model updates effectively decreases as the number of worker nodes increases. Specifically, it decreases the learning time by 34.03% in the entire model update and by 45.21% in the partial model update, respectively, verifying the scalability of the proposed distributed learning pipeline. In particular, in the entire update model, this scalable architecture alleviates its learning overhead by scaling the distributed environments as we want. In the case of the model inference time as well, we can observe an explicit tendency where the inference time significantly decreases as the number of worker nodes increases. Compared to using a single worker node, a distributed configuration of using three worker nodes decreases the inference time by 57.41% in the entire model update and by 50.37% in the partial model update, respectively. Although it only requires a relatively very short time compared to the learning time, decreasing the inference time is quite important in real-time classification.

Table 6. The elapsed time for training model by the number of worker nodes using proposed strategies (seconds).

Update Strategy	Number of Worker Nodes	Learning Time	Inference Time
Entire Model Update	1	1834.23	0.1599
	2	1587.05	0.0967
	3	1210.04	0.0681
Partial Model Update	1	1668.87	0.1362
	2	1424.88	0.0857
	3	914.30	0.0676

4.3.4. Case Study

Figure 6 shows the ranking changes of selected keywords based on the frequency over time. As explained in Section 3, we update the list of keyword tokens and their ranking in each time window. The result indicates that most of the relevant keywords extracted from the pre-trained offline model have been maintained or become risen in the top keyword sets, whereas general keywords that are less important to the target events are removed from the top keyword sets.



(**a**) CSI-related keywords. (**b**) Disaster-related keywords.

Figure 6. The change of keyword ranking over time.

As shown in Figure 6a on the CSI dataset, the ranking of highly relevant keywords, such as 'cyberattack' and 'ransomware', has maintained over time. On the other hand, the ranking of keywords associated with a particular cyber-security event, such as 'darknet' and 'openssh', is rapidly increasing as the actual event occurs. In contrast, the ranking of the keywords that were included in the pre-trained offline model, but are not actually relevant to the target event continues to decrease over time, such as 'animation', 'food', and 'lunch'.

A similar tendency was observed in Figure 6b on the disaster dataset. That is, the ranking of highly relevant keywords, such as 'shelter' and 'alert', has maintained in the ranking of keywords over time; the ranking of keywords associated with a particular disaster, such as 'florida' and 'haiti', rapidly increases. In contrast, the ranking of the keywords that are not actually relevant to the target event, such as 'hacking', 'festival', and 'culture', decreases over time. Here, we note that the degree of changes in the keyword ranking in the disaster dataset is relatively dynamic compared to the CSI dataset. This stems from the nature of the disaster events where completely different kinds of new events occur, in contrast to the cyber-security domain where the used terms are limited across the domain and domain-specific terms have been continuously used. This indicates that we need to effectively update the word embedding to reflect newly important keywords and eliminate less important keywords so that we can effectively track the event changes.

5. Conclusions and Future Work

In this study, we investigated the problem of dynamically updating the classification model that can adaptively respond to changes in real-time data streams in a distributed learning environment. We proposed two dynamic model update methods: (1) entire model update and (2) partial model update. The former updated the entire model using the accumulated datasets, maximizing the model accuracy with excessive learning overheads; the latter updated a part of the model using only a limited number of recent datasets, maximizing the learning efficiency while maintaining reasonable accuracy. A scalable architecture based on Apache Spark can effectively resolve the bottleneck that occurred during the learning process in the entire event stream classification pipeline. Through the extensive experiments using two real-world tweet datasets, we showed that the entire model update improved the classification accuracy by 28.96∼58.63% compared to the pre-trained offline model, while its learning overhead incrementally increases. On the other hand, the partial model update improves the accuracy by 12.34∼50.92%, while significantly reducing the learning time by 69.35% up to 93.95% compared to the entire model update strategy.

In this study, we focused on the dynamic model update in a distributed environment with the efficient learning methods to reflect new tweet streams to the model. However, as the orders of the event trends are not predictable in practice, the classification model not only requires learning new tasks but also needs to effectively maintain the learned representations of the previous tasks. While the existing model can be adapted to the current task by incrementally learning the current task based on the previously learned representations, they are prone to catastrophic forgetting, i.e., forgetting the previously learned representations [40]. In this study, we did not focus on maintaining the model performance for previous tasks, but they are required when the previous tasks are performed repeatedly. Therefore, as a further study, we plan to investigate continual learning algorithms based on a distributed environment to resolve those challenges. To achieve this, we will manage separate models for specific tasks, and each model would be updated differently according to the degree of event changes in data streams.

Author Contributions: Conceptualization, M.-S.K. and H.-Y.K.; methodology, M.-S.K., B.-Y.L. and H.-Y.K.; software, M.-S.K. and B.-Y.L.; validation, M.-S.K., B.-Y.L., K.L. and H.-Y.K.; formal analysis, M.-S.K., B.-Y.L., K.L. and H.-Y.K.; investigation, M.-S.K., B.-Y.L., K.L. and H.-Y.K.; resources, M.-S.K. and B.-Y.L.; data curation, M.-S.K. and B.-Y.L.; writing—original draft preparation, M.-S.K., B.-Y.L., K.L. and H.-Y.K.; writing—review and editing, M.-S.K., B.-Y.L., K.L. and H.-Y.K.; visualization, M.-S.K. and B.-Y.L.; supervision, K.L. and H.-Y.K.; project administration, K.L. and H.-Y.K.; funding acquisition, H.-Y.K. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the Research Program funded by the SeoulTech (Seoul National University of Science and Technology).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
BERT	Bidirectional Encoder Representations from Transformers
CSI	Cyber-Security Intelligence
Bi-LSTM	Bidirectional Long Short Term Memory

References

1. Weng, J.; Lee, B.S. Event detection in twitter. *Proc. Int. Aaai Conf. Web Soc. Media* **2011**, *5*, 401–408. [CrossRef]
2. Batool, R.; Khattak, A.M.; Maqbool, J.; Lee, S. Precise tweet classification and sentiment analysis. In Proceedings of the 2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS), Niigata, Japan, 16–20 June 2013; Volume 5, pp. 461–466.
3. Shin, H.S.; H.-Y.K.; Ryu, S.J. A new text classification model based on contrastive word embedding for detecting cybersecurity intelligence in twitter. *Electronics* **2020**, *9*, 1527. [CrossRef]
4. Kim, M.S.; Kwon, H.Y. Distributed Classification Model of Streaming Tweets based on Dynamic Model Update. In Proceedings of the 2022 IEEE International Conference on Big Data and Smart Computing (BigComp), Daegu, Republic of Korea, 17–20 January 2022; pp. 47–51.
5. Nishida, K.; Hoshide, T.; Fujimura, K. Improving tweet stream classification by detecting changes in word probability. In Proceedings of the 35th international ACM SIGIR conference on Research and Development in Information Retrieval, Portland OR USA, 12–16 August 2020; pp. 971–980.
6. Weiler, A.; Grossniklaus, M.; Scholl, M.H. Event identification and tracking in social media streaming data. In Proceedings of the EDBT/ICDT, Athens, Greece, 28 March 2014; pp. 282–287.
7. Nguyen, D.T.; Jung, J.J. Real-time event detection on social data stream. *Mob. Net. Appl.* **2015**, *20*, 475–486. [CrossRef]
8. Hasan, R.A.; Alhayali, R.A.I.; Zaki, N.D.; Ali, A.H. An adaptive clustering and classification algorithm for Twitter data streaming in Apache Spark. *TELKOMNIKA (Telecommun. Comput. Electron. Control.)* **2019**, *17*, 3086–3099. [CrossRef]

9. Zyblewski, P.; Sabourin, R.; Woźniak, M. Data preprocessing and dynamic ensemble selection for imbalanced data stream classification. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*; Springer: Cham, Switzerland, 2019; pp. 367–379.
10. Krawczyk, B.; Cano, A. Adaptive Ensemble Active Learning for Drifting Data Stream Mining. In Proceedings of the IJCAI, Macao, China, 10–16 August 2019; pp. 2763–2771.
11. Bermejo, U.; Almeida, A.; Bilbao-Jayo, A.; Azkune, G. Embedding-based real-time change point detection with application to activity segmentation in smart home time series data. *Expert Syst. Appl.* **2021**, *185*, 115641. [CrossRef]
12. Malialis, K.; Panayiotou, C.G.; Polycarpou, M.M. Nonstationary data stream classification with online active learning and siamese neural networks. *Neurocomputing* **2022**, *512*, 235–252. [CrossRef]
13. Wang, J.; Kolar, M.; Srebro, N.; Zhang, T. Efficient distributed learning with sparsity. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 3636–3645.
14. Wang, S.; Tuor, T.; Salonidis, T.; Leung, K.K.; Makaya, C.; He, T.; Chan, K. Adaptive federated learning in resource constrained edge computing systems. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1205–1221. [CrossRef]
15. Chen, Y.; Ning, Y.; Slawski, M.; Rangwala, H. Asynchronous online federated learning for edge devices with non-iid data. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 15–24.
16. Wang, Y.; Guo, L.; Zhao, Y.; Yang, J.; Adebisi, B.; Gacanin, H.; Gui, G. Distributed learning for automatic modulation classification in edge devices. *IEEE Wirel. Commun. Lett.* **2020**, *9*, 2177–2181. [CrossRef]
17. Hsieh, K.; Phanishayee, A.; Mutlu, O.; Gibbons, P. The non-iid data quagmire of decentralized machine learning. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual Event, 13–18 July 2020; pp. 4387–4398.
18. Abad, M.S.H.; Ozfatura, E.; Gunduz, D.; Ercetin, O. Hierarchical federated learning across heterogeneous cellular networks. In Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 8866–8870.
19. Cha, H.; Park, J.; Kim, H.; Bennis, M.; Kim, S.L. Proxy experience replay: Federated distillation for distributed reinforcement learning. *IEEE Intell. Syst.* **2020**, *35*, 94–101. [CrossRef]
20. Park, J.; Samarakoon, S.; Elgabli, A.; Kim, J.; Bennis, M.; Kim, S.L.; Debbah, M. Communication-efficient and distributed learning over wireless networks: Principles and applications. *Proc. IEEE* **2021**, *109*, 796–819. [CrossRef]
21. Jiang, Y.; Wang, S.; Valls, V.; Ko, B.J.; Lee, W.H.; Leung, K.K.; Tassiulas, L. Model pruning enables efficient federated learning on edge devices. *IEEE Trans. Neural Net. Learn. Syst.* **2022**, 1–13. [CrossRef]
22. Tekin, C.; Van Der Schaar, M. Distributed online learning via cooperative contextual bandits. *IEEE Trans. Signal Process.* **2015**, *63*, 3700–3714. [CrossRef]
23. Zhang, W.; Zhao, P.; Zhu, W.; Hoi, S.C.; Zhang, T. Projection-free distributed online learning in networks. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 4054–4062.
24. Li, C.; Zhou, P.; Xiong, L.; Wang, Q.; Wang, T. Differentially private distributed online learning. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1440–1453. [CrossRef] [PubMed]
25. Paternain, S.; Lee, S.; Zavlanos, M.M.; Ribeiro, A. Distributed constrained online learning. *IEEE Trans. Signal Process.* **2020**, *68*, 3486–3499. [CrossRef]
26. Wu, Y.C.; Lin, C.; Quek, T.Q. A Robust Distributed Hierarchical Online Learning Approach for Dynamic MEC Networks. *IEEE J. Sel. Areas Commun.* **2021**, *40*, 641–656. [CrossRef]
27. Mittal, V.; Kashyap, I. Empirical study of impact of various concept drifts in data stream mining methods. *Int. J. Intell. Syst. Appl.* **2016**, *8*, 65. [CrossRef]
28. Ed-daoudy, A.; Maalmi, K. Application of machine learning model on streaming health data event in real-time to predict health status using spark. In Proceedings of the 2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT), Rabat, Morocco, 21–23 November 2018; pp. 1–4.
29. Gupta, O.; Raskar, R. Distributed learning of deep neural network over multiple agents. *J. Netw. Comput. Appl.* **2018**, *116*, 1–8. [CrossRef]
30. Huang, Z.; Hu, R.; Guo, Y.; Chan-Tin, E.; Gong, Y. DP-ADMM: ADMM-based distributed learning with differential privacy. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 1002–1012. [CrossRef]
31. Gao, Z.; Gama, F.; Ribeiro, A. Wide and deep graph neural network with distributed online learning. *IEEE Trans. Signal Process.* **2022**, *70*, 3862–3877. [CrossRef]
32. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), Boston, MA, USA, 22–25 June 2010.
33. Dünner, C.; Parnell, T.; Atasu, K.; Sifalakis, M.; Pozidis, H. Understanding and optimizing the performance of distributed machine learning applications on apache spark. In Proceedings of the 2017 IEEE International Conference on Big Data (big data), Boston, MA, USA, 11–14 December 2017; pp. 331–338.
34. Zhao, S.Y.; Xiang, R.; Shi, Y.H.; Gao, P.; Li, W.J. Scope: Scalable composite optimization for learning on spark. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco CA, USA, 4–9 February 2017.

35. Alkhoury, F.; Wegener, D.; Sylla, K.H.; Mock, M. Communication efficient distributed learning of neural networks in Big Data environments using Spark. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 3871–3877.
36. Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A.A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. USA* **2017**, *114*, 3521–3526. [CrossRef]
37. Zenke, F.; Poole, B.; Ganguli, S. Continual learning through synaptic intelligence. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 3987–3995.
38. Mirzadeh, S.I.; Farajtabar, M.; Pascanu, R.; Ghasemzadeh, H. Understanding the role of training regimes in continual learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 7308–7320.
39. Li, Z.; Hoiem, D. Learning without forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 2935–2947. [CrossRef] [PubMed]
40. Rebuffi, S.A.; Kolesnikov, A.; Sperl, G.; Lampert, C.H. icarl: Incremental classifier and representation learning. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2001–2010.
41. Castro, F.M.; Marín-Jiménez, M.J.; Guil, N.; Schmid, C.; Alahari, K. End-to-end incremental learning. In Proceedings of European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 233–248.
42. Chaudhry, A.; Ranzato, M.; Rohrbach, M.; Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv* **2018**, arXiv:1812.00420.
43. Wang, Z.; Mehta, S.V.; Póczos, B.; Carbonell, J. Efficient meta lifelong-learning with limited memory. *arXiv* **2020**, arXiv:2010.02500.
44. Shin, H.; Lee, J.K.; Kim, J.; Kim, J. Continual learning with deep generative replay. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 2990–2999.
45. Wang, L.; Yang, K.; Li, C.; Hong, L.; Li, Z.; Zhu, J. Ordisco: Effective and efficient usage of incremental unlabeled data for semi-supervised continual learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual Conference, 9–25 June 2021; pp. 5383–5392.
46. Rusu, A.A.; Rabinowitz, N.C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; Hadsell, R. Progressive neural networks. *arXiv* **2016**, arXiv:1606.04671.
47. Mallya, A.; Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7765–7773.
48. Mallya, A.; Davis, D.; Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 67–82.
49. Rebuffi, S.A.; Bilen, H.; Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8119–8127.
50. Ashfahani, A.; Pratama, M. Autonomous deep learning: Continual learning approach for dynamic environments. In Proceedings of the 2019 SIAM International Conference on Data Mining, Calgary, AB, Canada, 2–4 May 2019; pp. 666–674.
51. Yoon, J.; Jeong, W.; Lee, G.; Yang, E.; Hwang, S.J. Federated continual learning with weighted inter-client transfer. In Proceedings of the International Conference on Machine Learning. PMLR, Virtual Event, 13–14 August 2021; pp. 12073–12086.
52. Cano, A.; Krawczyk, B. ROSE: Robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Mach. Learn.* **2022**, *111*, 2561–2599. [CrossRef]
53. Ruder, S.; Plank, B. Strong baselines for neural semi-supervised learning under domain shift. *arXiv* **2018**, arXiv:1804.09530.
54. Yoo, D.; Kweon, I.S. Learning loss for active learning. In Proceedings the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–17 June 2019; pp. 93–102.
55. Smith, J.; Taylor, C.; Baer, S.; Dovrolis, C. Unsupervised progressive learning and the STAM architecture. *arXiv* **2019**, arXiv:1904.02021.
56. Aghdam, H.H.; Gonzalez-Garcia, A.; Weijer, J.v.d.; López, A.M. Active learning for deep detection neural networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 27 October–2 November 2019; pp. 3672–3680.
57. Tiwari, P.; Uprety, S.; Dehdashti, S.; Hossain, M.S. TermInformer: Unsupervised term mining and analysis in biomedical literature. *Neural Comput. Appl.* **2020**, 1–14. [CrossRef]
58. Ashfahani, A.; Pratama, M. Unsupervised Continual Learning in Streaming Environments. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–12. [CrossRef]
59. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805.
60. Zhu, Y.; Kiros, R.; Zemel, R.; Salakhutdinov, R.; Urtasun, R.; Torralba, A.; Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 19–27.
61. You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; Hsieh, C.J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv* **2019**, arXiv:1904.00962.
62. Chen, X.; Cheng, Y.; Wang, S.; Gan, Z.; Wang, Z.; Liu, J. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv* **2020**, arXiv:2101.00063.
63. Apronti, P.T.; Osamu, S.; Otsuki, K.; Kranjac-Berisavljevic, G. Education for disaster risk reduction (DRR): Linking theory with practice in Ghana's basic schools. *Sustainability* **2015**, *7*, 9160–9186. [CrossRef]