Poster: Reproducible and Reliable Distributed Classification of Text Streams

Artem Trofimov,[1,2] Mikhail Shavkunov,[3] Sergey Reznick,[4] Nikita Sokolov,[5] Mikhail Yutman,[3]
Igor E. Kuralenok,[1] and Boris Novikov[3]
[1]JetBrains Research, [4] Kofax, [5] ITMO University
[2]Saint Petersburg State University, [3]National Research University Higher School of Economics
St. Petersburg, Russia
{trofimov9artem,mv.shavkunov,sergey.reznick,faucct,myutman,ikuralenok}@gmail.com,borisnov@acm.org

ABSTRACT

Large-scale classification of text streams is an essential problem that is hard to solve. Batch processing systems are scalable and proved their effectiveness for machine learning but do not provide low latency. On the other hand, state-of-the-art distributed stream processing systems are able to achieve low latency but do not support the same level of fault tolerance and determinism. In this work, we discuss how the distributed streaming computational model and fault tolerance mechanisms can affect the correctness of text classification data flow. We also propose solutions that can mitigate the revealed pitfalls.

CCS CONCEPTS

• **Information systems → Data streams**; **Clustering and classification**; *Data stream mining*;

KEYWORDS

Data streams, text classification, reproducibility, exactly once

ACM Reference Format:
Artem Trofimov,[1,2] Mikhail Shavkunov,[3] Sergey Reznick,[4] Nikita Sokolov,[5] Mikhail Yutman,[3] Igor E. Kuralenok,[1] and Boris Novikov[3] . 2019. Poster: Reproducible and Reliable Distributed Classification of Text Streams. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3328905.3332514

1 INTRODUCTION

Classification of large text streams is hard, but important task for researchers and practitioners. It has a wide range of applications including detection of emerging news and current user interests, suspicious traffic analysis, spam detection, etc. Popular open-source libraries like sklearn [5] provide a rich set of tools, but they mostly aim at handling static datasets. The lack of scalability across multiple computational units is another limitation of these solutions. There are plenty of works which adapt batch processing systems for text classification [6]. Their advantages are fault tolerance, high throughput, and scalability. On the other hand, these systems do not provide low latency that is a strong requirement for most streaming applications.

An immediate idea is to employ a distributed stream processing engine such as Flink [2] or Storm [1]. However, unlike batch engines, stream processing systems have several peculiarities:

- In a general case, failure and recovery are not transparent for a user. The guarantees on data in case of failures are defined in terms of delivery guarantees: *at least once* and *exactly once*. The choice of a guarantee may affect the correctness of text classification.
- Most of streaming systems are inherently non-deterministic. It means that different runs on the same data may produce different results. This feature can influence the classification process as well.

In this work, we investigate the applicability of state-of-the-art stream processing systems to the text classification and demonstrate the challenges that a developer can experience. In particular, we discuss how the delivery guarantees and non-deterministic pipelines may affect the results. Possible solutions to the mentioned issues are proposed.

2 PROBLEM STATEMENT

Let us consider the news topic classification as an illustration of a text stream multi-classification task. There are two input streams: pre-labeled and raw. The latter stream elements must be labeled by a classifier and delivered to end-user. The pre-labeled stream is used for updating a machine learning model with new data in order to adapt it to current events. The ultimate purpose is to achieve the distribution of news topics that is changing over time. This task is a typical representative of the text stream classification problem.

As the main requirement to a stream processing engine we claim the following: each output element must depend only on the input. In other words, node failures and execution environment parameters must not affect the results, similar to batch systems. Otherwise, it is hard to design a solution that provides reproducible and interpretable classification results.

3 DATA FLOW

Typical classification pipeline based on bag-of-words text representation consists of three steps. The first one is computing TF-IDF features. The second one is training a classifier on these features or making a prediction. To adapt this pipeline for a stream processing engine, one needs to represent it in the form of a *logical graph*. It serves as a language for defining streaming computations.

DEBS '19, June 24–28, 2019, Darmstadt, Germany

ACM ISBN 978-1-4503-6794-3/19/06…$15.00
https://doi.org/10.1145/3328905.3332514

Vertices of a logical graph denote operations, while edges indicate data subscriptions between them.

The initial point in our data flow is an *Source* vertex. It receives input texts from data producers and computes term frequencies. Computing of inverse document frequencies is a separate operation because it maintains a state and requires different data partitioning in a physical execution. *TF-IDF* vertex joins features corresponding to the same text and passes them to the *Text Classifier*. *Text Classifier* is the very last vertex that predicts a label and delivers it to a data consumer. The scheme of the proposed logical graph is shown in Figure 1.

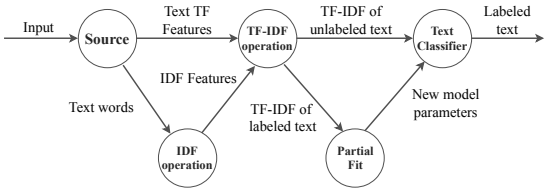

Figure 1: Text classification data flow

Training pipeline is a separate branch within the logical graph introduced above. For already labeled text its features are sent to a *Partial fit* vertex instead of the *Classifier*. *Partial fit* vertex buffers all input elements until training is triggered. After training, the buffer flushes. Updated parameters of a machine learning model are saved for further training and broadcasted to all *Text Classifier* vertices.

4 CHALLENGES

Online training. An issue regarding the pipeline is that the training process may be time-consuming. If training and prediction processes run consecutively, there will be significant latency spikes, e.g. if a training process lasts for several minutes, then spikes may be 10 000 times greater than latency for prediction. However, without synchronization, there will be no reproducible correspondence between texts and applied model. It is almost impossible to achieve the same results within a new run on the same data because the training time becomes a hidden parameter that influences output. For instance, assume that we make two runs. On the first run model update consumes 70 seconds, but on the second run 75 seconds due to extra CPU load. If training and predicting are not synchronized, more unlabeled input elements are processed by an outdated model in the second case, so the distribution of news topics may be different between these two runs. We propose two solutions for the issues in question:

- Use online learning algorithms. In this case, model updating is smooth and its synchronization with training does not cause latency spikes.
- Consider model parameters as special input elements that are stored with other input elements in a persistent queue, e.g. using Kafka [3]. To reproduce results, there is just a need to replay elements from this queue.

Delivery guarantees. Requirements on reproducibility and predictability of classification results affect the choice of a delivery guarantee. If a stream processing system provides *at least once*, some input texts can be processed more than one time in case of failures. This behavior may lead to biased prediction results. For example, if a single sports article is processed many times due to multiple failures, the resulted topics distribution will show that sport is a hot news topic right now. Hence, the only suitable delivery guarantee is *exactly once*. The problem here is that it is hard to achieve both low latency (less than 500 ms) and exactly once. Table 1 shows if a state-of-the-art system supports both features. To the best of our knowledge, among open systems only FlameStream provides for both low latency and exactly once. This property is achieved using optimistic order enforcement that implies system-wide idempotence. The details of this approach are discussed in [4].

Table 1: Support of exactly and low latency (less than 500ms) by stream processing systems

System	Exactly-once	Latency
Storm,Heron,Samza	–	low
Spark Streaming	+	high
Flink	+	high*
MillWheel	+	NA
FlameStream	+	low

* with enabled exactly-once [4]

5 CONCLUSION

In this work, we investigated the suitability of distributed stream processing engines to the problem of text streams classification with the following requirements:

- Unbiased by distributed environment: node failures or races do not affect the ultimate result distribution.
- Reproducible: if input elements are stored in persistent storage, the same predictions are obtained on each new run.

We discussed several pitfalls with a straightforward approach and highlight several limitations regarding the choice of a processing engine and the structure of data flow:

- Exactly once is a strong requirement.
- Embedding of time-consuming train process in the prediction pipeline leads to significant latency spikes.

As future work, we plan to implement a text classification framework that satisfies the proposed requirements and provides for low latency.

REFERENCES

[1] Apache Storm 2017. Apache Storm. (Oct. 2017). http://storm.apache.org/

[2] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State Management in Apache Flink&Reg;: Consistent Stateful Distributed Stream Processing. *Proc. VLDB* 10, 12 (Aug. 2017), 1718–1729.

[3] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*. 1–7.

[4] Igor E. Kuralenok, Artem Trofimov, Nikita Marshalkin, and Boris Novikov. 2018. FlameStream: Model and Runtime for Distributed Stream Processing. In *Proceedings of the 5th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR'18)*. ACM, New York, NY, USA, Article 8, 2 pages. https://doi.org/10.1145/3206333.3209273

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[6] Alexey Svyatkovskiy, Kosuke Imai, Mary Kroeger, and Yuki Shiraito. 2016. Large-scale text processing pipeline with Apache Spark. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 3928–3935.