A Distributed Architecture for Toxic Chat Detection

Wonhee Jung (wonheej2@illinois.edu) *University of Illinois Urbana-Champaign*, Kevin Mackie (kevindm2@illinois.edu) *University of Illinois Urbana-Champaign*, Cindy Tseng (cindyst2@illinois.edu) *University of Illinois Urbana-Champaign* and Conrad Harley (harley3@illinois.edu) *University of Illinois Urbana-Champaign*

***Abstract*—This paper presents the architecture, development, and use of a distributed system for toxic chat filtering. This system differentiates itself in that a) its filtering is based on machine learning and deeper contextual analysis, and b) it is deployed as a scalable and easily integrated web framework that can be adapted to any source of text for online interaction of any size. The platform is based on Docker and Kubernetes for easy deployment and dependency management, and uses state-of-the-art distributed systems technology to allow for fast scale-out to large systems. The system is presented in the context of a web chat application and Twitch chat bot as motivating examples.**

***Index Terms*—toxic comment, web, chat, detoxifier, detection, classifier, distributed, architecture**

I. Introduction

ONLINE platforms allow people to express their opinions freely, and stimulate collaboration across the globe. Unfortunately, online interaction may often come with loosened inhibitions in making profane, bigoted, or offensive remarks. We refer to such unwelcome remarks as “toxic chat”. Online systems may or may not have their own embedded profanity filtering, and those that do typically use pre-registered terms and simple pattern matching. This approach lacks the deeper contextual understanding needed to identify sentences that are toxic but that may not contain banned terms.

Thus we propose a new toxic chat filtering system that differentiates itself in that a) its filtering is based on machine learning and deeper contextual analysis, and b) it is deployed as a scalable and easily integrated web framework that can be adapted to any source of text for online interaction of any size. The platform is based on Docker and Kubernetes for easy deployment and dependency management and to allow for fast scale-out to large systems. It uses state-of-the-art distributed systems technology for processing and storage, to allow for rapid scaling to any size while maintaining a shared file space (HDFS) between each Kubernetes Zone. This paper presents the architecture, development, and use of this system in the context of a web chat application and Twitch chatBot as motivating examples.

II. Datasets

There is a dearth of labelled datasets for training classifiers to detect toxic comments. An online search and literature review were conducted on IEEE Xplore, Scopus, and Science Direct. We concluded that the best dataset available is Toxic Comment Classification Challenge dataset released by Jigsaw and Google on Kaggle in 2018, see [1]. Note that three additional datasets were identified - from Reddit [2], [3], Wikipedia [2], and Twitter [3], [4]. However, the datasets were not appropriate for use with our classifier.

III. Technologies and Tools

The following technologies and solutions have been integrated to date.

- Flask, WebSockets, JavaScript, Bootstrap - web application framework and client-server sockets
- Docker, Kubernetes - easy deployment and scale out
- Google Cloud Platform (GCP) - to deploy the solution into a mainstream IaaS/PaaS infrastructure
- scikit-learn - machine learning for the detox engine
- Github - source code management

See “Future Work” for other technologies to be assessed for integration over the coming weeks.

IV. Method/Design

The framework is divided into the following elements:

- Web server
- Chat stream
- Classifier
  - Training data set
  - Model storage
  - Classifier
- Container system

A. Web server

The user interface is implemented as a web application reachable via public URL. Python flask was chosen for the web application development framework, due to its simplicity, modularity, and compatibility with deployment to Google Cloud Platform. Client-side programming is done in JavaScript with Bootstrap for the GUI framework and WebSockets for client-server socket connectivity for real-time chat.

B. Chat stream

The web application connects the user to both a webchat and Twitch TV stream. For webchat, the server acts as a hub to relay messages arriving from each client to all other connected clients. The web server attaches to a hard-coded Twitch TV channel and receives and processes a continuous stream of comments. Text typed into the web chat or received from Twitch is passed to the classifier. Toxic messages are marked in red with a prefix indicating that the message is toxic.

C. Classifier

We use a Multinomial Naive Bayes classifier with TF-IDF for toxic chat classification. The classifier is trained on the Jigsaw dataset on Kaggle and the resulting model is persisted in serialized form and used for future invocations of the framework. The scikit-learn library was chosen for prototyping the Machine Learning and NLP components because it was easy to use and is supported with many online examples.

V. Preliminary Evaluation/Results

A. Toxic chat classification accuracy

We used TF-IDF with Multinomial Naive Bayes for toxic chat classification. scikit-learn uses a lot of memory as training data gets bigger, and we are assessing Apache Spark's MLlib as an alternative to run as a cloud service for better scalability and high volume capacity.

With PySpark's dataframe-based ML library, the performance metrics are:

Accuracy	0.871943231441048
Precision	0.4434561626429479
Recall	0.7015075376884422
F1	0.5434021019852082

Work is still in progress to improve the scores so the F1 score is on-par, if not better than, the F1 score for scikit-learn implementation (which was 0.64).

Precision/Recall/F1-Score : With scikit-learn, we shuffled the training set and took 20% as the test set for evaluation. Note the performance numbers vary slightly with each run due to the random shuffling of data.

	Precision	Recall	F1-Score	Support
0	0.94	1.00	0.97	28629
1	0.95	0.48	0.64	3286
Avg/Total	0.95	0.94	0.94	31915

In the result, the non-toxic comments (labeled as 0) have a 0.97 F1-score and the toxic comments have a 0.64 F1-score. On average, F1 is 0.94 for the both cases. The result is acceptable, at least for non-toxic comment classficiation, because the classifier rarely classfied non-toxic chat as toxic (0.03 error rate). We hope to improve the F1 score for the toxic comments class in the final report.

B. Cloud Services

We have evaluated several IaaS and PaaS cloud services, inclduing AWS, GCP, and Heroku. We have decided to use GCP because of its easy configuration of Kubernetes Clusters, which the team decided to use for docker container scale-out and deployment. K8s gets direct support from Google, and GCP already has GKE services. We were able to create a docker build pipeline using GCP's Cloud Build with a few mouse clicks.

AWS was problematic for our usage, and its Kubernetes support was not as mature or functional as GCP's. Heroku was also considered, but it does not support any Kubernetes services or Docker deployments using CI/CD build pipelines. Users can deploy the dockerized application, but it has to be done locally from the user's console using the Heroku CLI.

VI. Discussion

Our final objective is to develop a flexible and scalable distributed architecture for toxic chat classification. At this intermediate stage, we have most of the technical components prototyped at varying levels of completeness. There is a web application that allows users to exchange chat messages and monitor a fixed online chat source. A Multinomial Naive Bayes classifier is invoked by the web framework to classify the web chat messages as either toxic or non-toxic. The application is built as a docker image and deployed onto a Kubernetes pod on Google Cloud Platform.

See the next section "Future Work" for details about the work to be undertaken in the coming weeks to develop the final system.

VII. Future Work

A. Classifier upgrade

As mentioned above, we are assessing the possibility of replacing the ML and NLP library from scikit-learn with Apache Spark and MLlib. We have already created a running demo that can classify comments at a slightly lower F1 score. We continue tuning and testing the model, and verifying it on Google Cloud Platform prior to committing changes to the repository.

B. Better sharing of trained model for classifier

Training the model is expensive, so we do not want to train the classifier everytime the Docker container runs. The application currently saves the trained model to a file (in a serialized format called "pickle" in Python), and then restores the model from the file (if it exists) when it starts up and needs to classify a comment. However, we are expecting to have multiple docker containers running on K8s clusters, so a better way is needed to share the trained model with other K8s pods. We will expore options for this, such as K8s volume pods and Docker volume mounting on HDFS paths, etc.

C. Infrastructure as Code

GCP is easy to use, but it requires a lot of manual effort when shutting down and later recreating all of the K8s clusters, load balancers, etc. This is frequently done during development to avoid GCP's daily charges. We will consider solutions that support Infrastructure as Code, such as Puppet, Chef, Ansible, etc. Also, we will possibly use Terraform to automate our infrastrcture build and maintain it as code.

D. Continuous Integration and Continuous Deployment (CI/CD) pipeline

As the solution matures we will automate the build, integration, and deployment pipeline to allow for more rapid development and validation of changes and then deployment to a working system.

E. RESTful APIs

We may consider RESTful interfaces to allow easier integration of the facilities provided by our framework (detox engine, chat stream acquisition, etc.) with other applications.

F. Load-balanced web application

We will investigate using a load balancer to distribute traffic among several instances of the web application. Also, a complete application will allow plugs ins for different chat sources (Twitter, web forums, Reddit, Wikipedia), with the user able to specify the chat source and channel (currently this is hard-coded). We may also consider user authentication and configuration settings for the web application.

VIII. Division of Work (May overlap)

This section summarizes the areas of responsibility of the respective members of the team.

A. Wonhee Jung

- Docker design and build
- Google Cloud Platform and Kubernetes deployment
- Repository management
- Code reviews
- Paper writing

B. Kevin Mackie

- Unified web application prototype for web chat and Twitch TV bot
- Literature review of classification datasets
- Framework for paper generation
- Code reviews
- Paper writing

C. Cindy Tseng

- Multinomial Naive Bayes classifier on Spark and MLlib
- scikit-learn and MLlib classifier performance comparison
- Code reviews
- Paper writing

D. Conrad Harley

- Kubernetes and classifier model storage research

Acknowledgment

The authors would like to thank ...

References

[1] Jigsaw, "Toxic comment classification challenge," *https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data.*.

[2] R. K. Éloi Brassard-Gourdeau, "Impact of sentiment detection to recognize toxic and subversive online comments," *http://arxiv.org/pdf/1812.01704v1*, 2018.

[3] R. G. Chandra Khatri Behnam Hedayatnia, "Detecting offensive content in open-domain conversations using two stage semi-supervision," *http://arxiv.org/pdf/1811.12900v1*, no. NIPS CONVAI Workshop 2018, 2018.

[4] R. K. Betty van Aken Julian Risch, "Challenges for toxic comment classification: An in-depth error analysis," *http://arxiv.org/pdf/1809.07572v1*, nos. ALW2: 2nd Workshop on Abusive Language Online to be held at EMNLP 2018 (Brussels, Belgium), October 31st, 2018, 2018.

Wonhee Jung (wonheej2@illinois.edu)

Kevin Mackie (kevindm2@illinois.edu)

Cindy Tseng (cindyst2@illinois.edu)

Conrad Harley (harley3@illinois.edu)