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.

The source code and documentation for the project can be found on the following public Github repository: https://github.com/freesoft/detox_bot2

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.

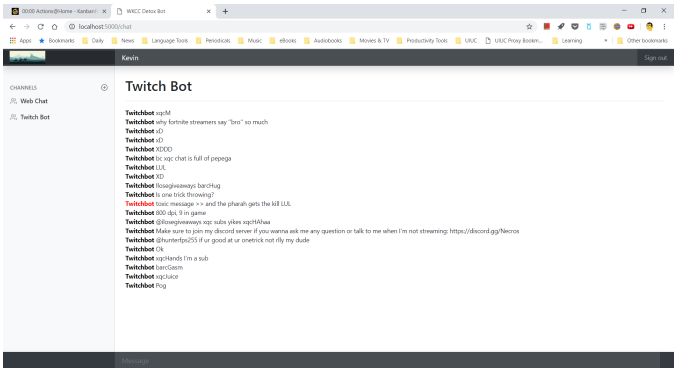
- Flask, WebSockets, JavaScript, Bootstrap - web application framework and client-server sockets
- Docker, Kubernetes - easy deployment and scale out
- Google Cloud Platform (GCP) - to deploy the solution into a mainstream IaaS/PaaS infrastructure
- scikit-learn - machine learning for the detox engine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

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

IV. Method/Design

A. Detox Bot project

The original idea for the toxic chat detoxifier came from the final CS410 project (called "detox_bot" and hosted on Github) of one of the authors. They implemented a simple detoxifier consisting of a Python console application and rudimentary webpage hosted on a Docker image. The motivation for the current project is to re-implement detox_bot as a distributed framework for toxic chat detection, applying the distributed systems technologies covered by the Cloud Computing Applications course at the University of Illinois Urbana-Champaign. The technologies to be applied include Docker containers on Kubernetes pods; IaaS/PaaS deployment on GCP; Spark; HDFS; a proper load-balanced web application hosted in the cloud, etc. We also wanted to do more formal and deeper research into the other approaches taken to solve the toxic chat classification problem and improve the performance of the classifier itself.



The New Detoxifier Web Application

B. Application framework overview

The framework is divided into the following elements:

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

1) 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.

2) 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.

3) 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. The final project uses PySpark with ML library for improved scalability.

4) Container system: The application is deployed to a Kuernetes cluster on Google Cloud Platform (GCP), with load balancing for high availability and scalable performance. The application container is automatically built on GCP when changes are committed to the master branch of the repository. Thus the application team was able to work off of local branches, verify changes via pull requests to a development branch, and then trigger automated builds of the container image in GCP by committing approved changes the master branch.

V. Preliminary Evaluation/Results

A. Toxic chat classification accuracy

We originally used scikit-learn library to prototype a TF-IDF Multinomial Naive Bayes classifier to classify toxic comments. The performance for the scikit-learn based classifier was:

	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

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. We showed the performance for non-toxic comments as well, since erroneously flagging non-toxic comments could lead to user dissatisfaction. However, scikit-learn uses a lot of memory as training data gets bigger. Thus we moved to Apache Spark's ML library in order to achieve better scalability and high volume capacity.

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

	Precision	Recall	F1-Score	Support
0	0.95	0.97	0.95	8165
1	0.69	0.58	0.69	995
Avg/Total	0.92	0.93	0.92	9160

We can see that PySpark's dataframe-based classifier performs slightly better on toxic detection than that of scikit-learn (F1 score of 0.69 instead of 0.64). However since the overall networks are nearly the same, performance is expected to be similar. Our main objective here is to prove that we can achieve similar results using the Apache Spark framework, while providing better scalability.

B. Cloud Services

We have evaluated several IaaS and PaaS cloud services, incldluing 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. We have tested our application and the following is the cluster requirement:

1 node, total of 2 vCPUs with 7.5 GB. idle CPU utilization (current/target value): 19%/80%

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.

It has been challenging to move our classifier from scikit-learn to the PySpark DataFrame-based ML libraries. First, most of the tutorials on PySpark ML libraries are still RDD-based, rather than DataFrame-based. Second, most of the Spark environment is tested in the Linux environment, and there is a lot of special setup needed to make it work on Windows for local development and testing. Third, PySpark has sparse compatibility with other libraries. In order to use common text processing libraries such as NLTK or loading pre-trained word2vec model using gensim, we had to either load data using pandas data frame first then convert it to PySpark data frame, or change the underlying pre-trained word2vec model to make it PySpark compatible

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

VII. Related Work

A. Literature

We conducted a focused search of recent academic literature to identify related work and to understand the current state-of-the-art in toxic chat detection systems. The search was performed on Scopus, Science Direct, IEEE, and Google Scholar. Based on the content of the abstracts, we identified nine research articles that were futher analyzed for content.

The research we found did not deal with the systems themselves, but rather with the performance of the classification algorithm. We thus feel our work fills an important gap in the literature in describing a framework for deploying the toxic chat classification function at scale on the web for real-world use.

B. Key topics

The topic coverage in the papers we reviewed falls into the following general categories:

a) reports on specific implementations of the toxic chat classifier
b) comparison of different toxic chat classification approaches and algorithms
c) methods for dealing with data quality issues and imbalanced class labels
d) countermeasures for adversarial toxic comments (i.e. comments written specifically to circumvent automatic toxic chat detection)

While the research helps to inform our selection of a toxic chat classifier, our goal for this project is primarily to define and prototype a cloud computing application framework for toxic chat detection. Our concern is only tangentially related to the performance of the comment classification algorithm itself. A properly designed system will allow for different algorithms to be plugged into the architecture as the state of the art advances with respect to toxic comment classification.

C. Important conclusions from literature

For our purposes, the research suggests a number of helpful conclusions.

1. Deep learning approaches such as convolutional or recurrent neural networks consistetly out-perform shallow approaches [5]–[7]. However, shallow learning techniques - and especially Linear Regression, NBSVMs, and SVMs - perform quite well and may require substantially less effort to develop and tune that neural networks. There is thus a tradeoff to consider as to whether the relatively small gains in accuracy are worth the time spent building, tuning, and maintaining a complex model [5], [6], [8].
2. Class labels are imbalanced (i.e. there are many more non-toxic comments than toxic comments in the training data) but there are techniques available for augmenting data to rebalance the classes [5].
3. Chat comments, especially those entered on a cell phone, often contain typos and stylized or abbreviated spelling that present challenges to effective tokenization. However, there are techniques for normalizing such data [9]
4. Notwithstanding issues such as imbalanced classes and garbled input data, even without augmentation simple classifiers can still perform well [9]
5. Users may engage in adversarial chat, using intentionally vague or distorted text to defeat toxic chat classifiers [10], [11]. We leave the augmented processing suggested in some of these articles to future work.

D. Google Perspective API

An important intiative for toxic chat classification is the Perspective API (http://www.perspectiveapi.com), developed by Google and its incubation company Jigsaw. The API allows developers to access Google's toxic chat classifier via public interace. Most recent research uses Jigsaw's Wikipedia comment dataset [1] (which is the basis of Kaggle's toxic comment challenge) to train and test classifiers.

Use of the Perspective API can be considered in future work for this system.

VIII. Future Work

A. Classifier upgrade

We replaced the ML and NLP library from scikit-learn with Apache Spark and ML library. The demo that can classify comments at the F1 scores reported above. We continue tuning and testing the model, and verifying it on Google Cloud

Platform prior to committing changes to the repository. For future versions beyond the end of the CCA course, we may experiment with deep classifiers (CNNs or RNNs) or ensemble approaches. Future work will add data augmentation to deal with imbalanced classes, and pre-processing to normalize chat text to recognizable terms. Future versions may also include methods to prevent bias by analyzing more contextual information in the chat stream.

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.

Another future improvement may allow users to classify messages as toxic or non-toxic, thus allowing for crowd-sourcing of the training data for the classifier.

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 are considering solutions that support Infrastructure as Code, such as Puppet, Chef, Ansible, etc, and leaning towards Terraform or Terragrunt at this point.

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

As the solution matures, we are considering to automate the build, integration, and deployment pipeline to allow for more rapid development and validation of changes and then deployment to a working system. The project already uses GCP's Cloud Build feature, that triggers the project's build whenever a change is submitted to the Github repo's develop branch. However, the team is researching if there are better alternatives for the build pipeline through other tools like Jenkins 2 or other commericial CI/CD solutions.

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 have created a load-balanced web application on a Kubernetes cluster. Future work may 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.

IX. 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
- Related works research
- Team leadership
- 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
- Literature review of related works
- Framework for paper generation
- Code reviews
- Paper writing

C. Cindy Tseng

- Multinomial Naive Bayes classifier on Spark with ML library
- scikit-learn and PySpark/ML library classifier performance comparison
- Rewrite Dockerfile to accomodate PySpark enviornment
- Tested initial deployment of PySpark classifier on google cloud platform
- Code reviews
- Paper writing

D. Conrad Harley

- Kubernetes and classifier model storage research

Acknowledgment

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.

[5] M. Rybinski M., "On the design and tuning of machine learning models for language toxicity classification in online platforms," *Studies in Computational Intelligence*, vol. 798, pp. 329–343, 2018.

[6] M. Saif M.A., "Classification of online toxic comments using the logistic regression and neural networks models," *AIP Conference Proceedings*, vol. 2048, 2018.

[7] A. G. V. Spiros V. Georgakopoulos Sotiris K. Tasoulis, "Convolutional neural networks for toxic comment classification," *http://arxiv.org/pdf/1802.09957v1*, 2018.

[8] D. Noever, "Machine learning suites for online toxicity detection," *http://arxiv.org/pdf/1810.01869v1*, 2018.

[9] F. Mohammad, "Is preprocessing of text really worth your time for online comment classification?" *http://arxiv.org/pdf/1806.02908v2*, nos. 11 pages,including Appendix, 2018.

[10] S. R.-G. Nestor Rodriguez, "Shielding google's language toxicity model against adversarial attacks," *http://arxiv.org/pdf/1801.01828v1*, 2018.

[11] B. Z. Hossein Hosseini Sreeram Kannan, "Deceiving google's perspective api built for detecting toxic comments," *http://arxiv.org/pdf/1702.08138v1*, no. 4 pages, 2017.

Wonhee Jung (wonheej2@illinois.edu)

Kevin Mackie (kevindm2@illinois.edu)

Cindy Tseng (cindyst2@illinois.edu)

Conrad Harley (harley3@illinois.edu)