Real-time Hate Speech Detection on Social Media with Spark

Sevak Harutyunyan
Faculty of Applied Statistics and Data Science, YSU
sevak.g.harutyunyan@gmail.com

Introduction

With the massive increase in social interactions on online social networks, there has also been an increase of hateful activities that exploit such infrastructure. On Twitter, and Facebook, hateful posts are those that contain abusive speech targeting individuals (a politician, a celebrity, a product) or particular groups (a country, a religion, gender, an organization, etc.).

The need in software to effectively and reliably detect such hateful speech is important for analyzing public sentiment of a group of users towards another group, and for discouraging associated wrongful activities. Moreover, these kind of technologies help governments and social networks stop the spread of hate speech and consequences that may arise as a result of similar content.

1. Implementation

As a final product, we built an API to easily connect with Twitter and Facebook individual or group accounts to monitor the activity of that social network users (currently available only for posts) and with constant retraining of the **baseline model** to get a **trained model** at the time we want. Since Spark structured streaming doesn’t support pySpark MLlib API, we used regular streaming.

In this paper, we experiment with multiple classifiers such as Logistic Regression, Random Forest, Gradient Boosted Decision Trees (GBDTs) and Deep Neural Networks (DNNs).

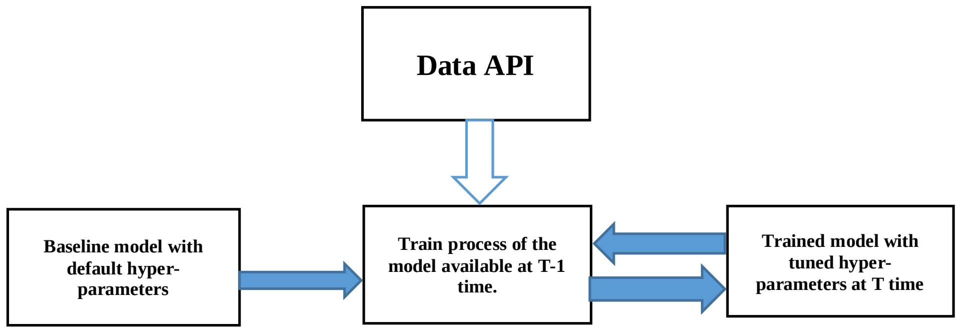
Even though, DNNs prove themselves to be more accurate in inference time, it takes too long to make an inference (stress-test is done by Apache JMeter).

Stress test results 1.1

	1	10	100
Logistic Regression	0.03	0.4	5
Random Forest	0.07	0.9	11
GBDTs	0.02	0.8	7
DNNs	0.1	2	18

Note that this paper doesn’t address the issue of preprocessing the dataset and mainly focuses on deploying the above-mentioned classifiers in stream processing using the Spark engine.

From baseline to trained 1.2


Data API
Baseline model with default hyper-parameters
Train process of the model available at T-1 time.
Trained model with tuned hyper-parameters at T time

The diagram structure above shows that once we defined the baseline model and fed with even one training data point, we get the trained model at time t and no longer we need the baseline model. The continuous process goes on and the stream retraining is happening as the service starts updating the hyper-parameters using the Spark MLlib grid search technique.

2. Results

For the final estimator we choose Logistic regression. One of the reasons is the results shown in 1.1 table.

The second reason is the performance accuracy compared to the other estimators shown in the diagram below.

	Precision	Recall	Accuracy	F1
Logistic Regression	**0.73**	0.78	0.77	0.75
Random Forest	0.7	0.8	0.76	0.74
GBDTs	0.72	**0.81**	0.77	**0.76**