Twitch Plays Pokémon, Machine Learns Twitch

Unsupervised Context-Aware Anomaly Detection for Identifying Trolls in Streaming Data

UTCS
Department of Computer Science
College of Natural Sciences

Albert Haque
Department of Computer Science, University of Texas at Austin

Abstract

With the increasing importance of online communities, discussion forums, and customer reviews, Internet "trolls" have proliferated thereby making it difficult for information seekers to find relevant and correct information. In this paper, we consider the problem of detecting and identifying Internet trolls, almost all of which are human agents. Identifying a human agent among a human population presents significant challenges compared to detecting automated spam or computerized robots.

To learn a troll's behavior, we use contextual anomaly detection to profile each chat user. Using density-based clustering methods, we use contextual data such as the group's current goal, the current time, and the user to classify each point as an anomaly. A user with consistent anomalies will be labeled as a troll. We have successfully trained our algorithm using k-means clustering methods on a dataset consisting of 38 million data points from the viral Internet social fad, Twitch Plays Pokémon. Using MapReduce techniques for preprocessing and user profiling, we are able to classify trolls based on 10 features extracted from a user's lifetime history.

Introduction & Motivation

In Internet slang, a "troll" is a person who sows discord on the Internet by starting arguments or upsetting people, by posting inflammatory, extraneous, or off-topic messages in an online community (such as a forum, chat room, or blog), either accidentally or with the deliberate intent of provoking readers into an emotional response or of otherwise disrupting normal on-topic discussion [1].

Contributions

1. We propose a set of features used for identifying trolls in Twitch Plays Pokémon. We use context-based techniques to understand the scenario a human is faced when entering input into the chat room. We then compare the effects of different distance measures to understand their strengths and weaknesses.
2. Development of an online classification algorithm. It is initially trained using unsupervised methods on offline data. When switched to online mode, this algorithm updates as new data points are received from a live stream. We apply this algorithm to the Twitch Plays Pokémon data stream and visualize the results.

The primary purpose of this project, through these two contributions, is to in real-time, distinguish between trolls and humans on the Internet. Future work can be extended to email classifiers, comment moderation, and anomaly detection for online forums, reviews, and other communities.

[1] Wikipedia: Troll (Internet)

Context Awareness & Representation

We define a context as a snapshot or period of time. It can also be thought of a sliding window of time with a specific duration. The context stores important information such as the button frequencies, number of messages, and percentage of spam messages.

The context is critical to distinguishing trolls from non-trolls. In Pokémon, there are times when a specific set of button inputs must be entered before the game can proceed. If one button is entered out of sequence, the sequence must restart. For those familiar with the Pokémon game, we are referring to the "cut" sequence where the player must cut a bush. Sequence 1 below lists the required button chain.

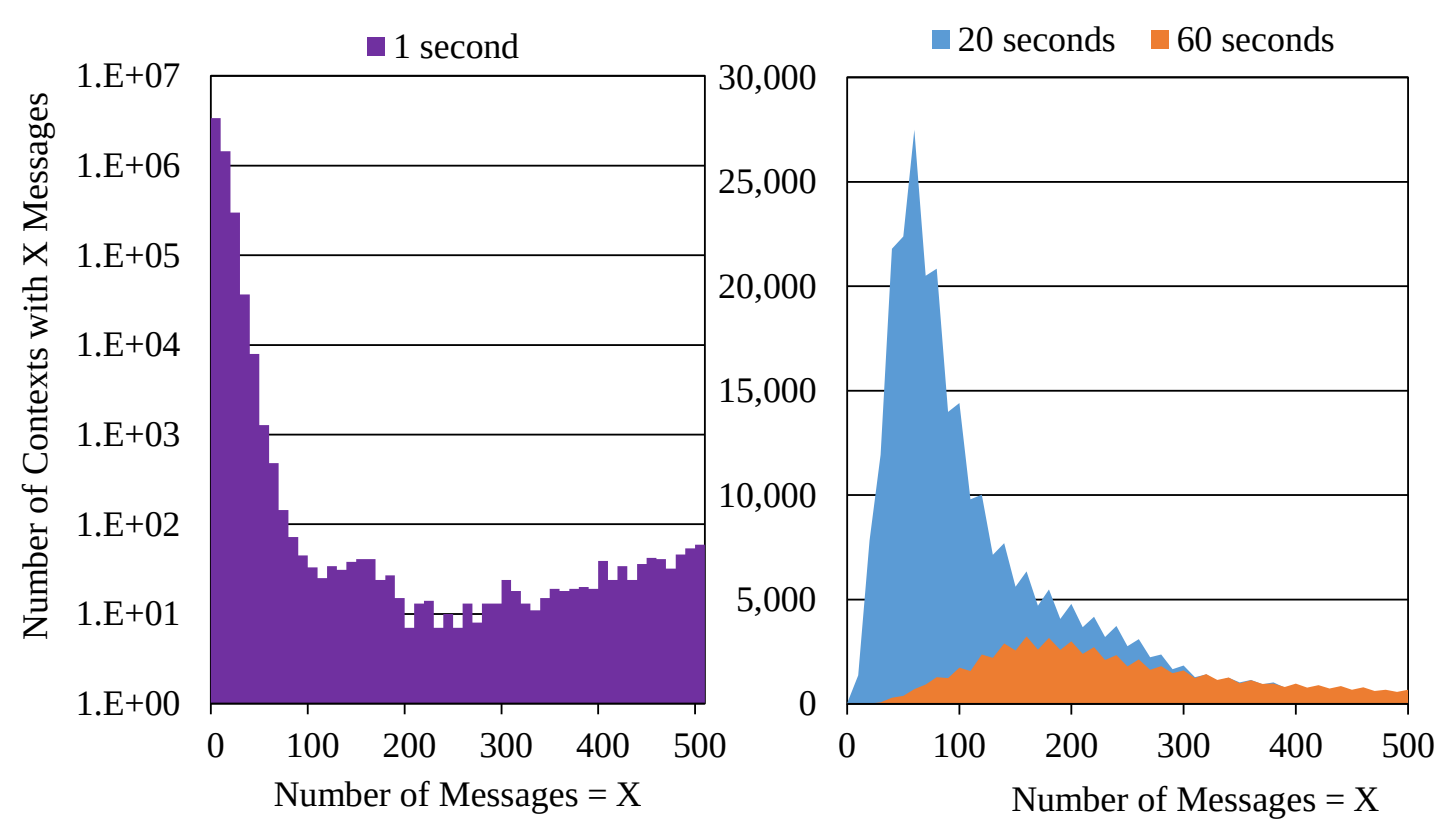
Open up the menu screen, navigate to the Pokémon list, select a specific Pokémon, select the CUT action

Sequence 1: DOWN > START > UP > UP > A > UP > A > DOWN > DOWN > A

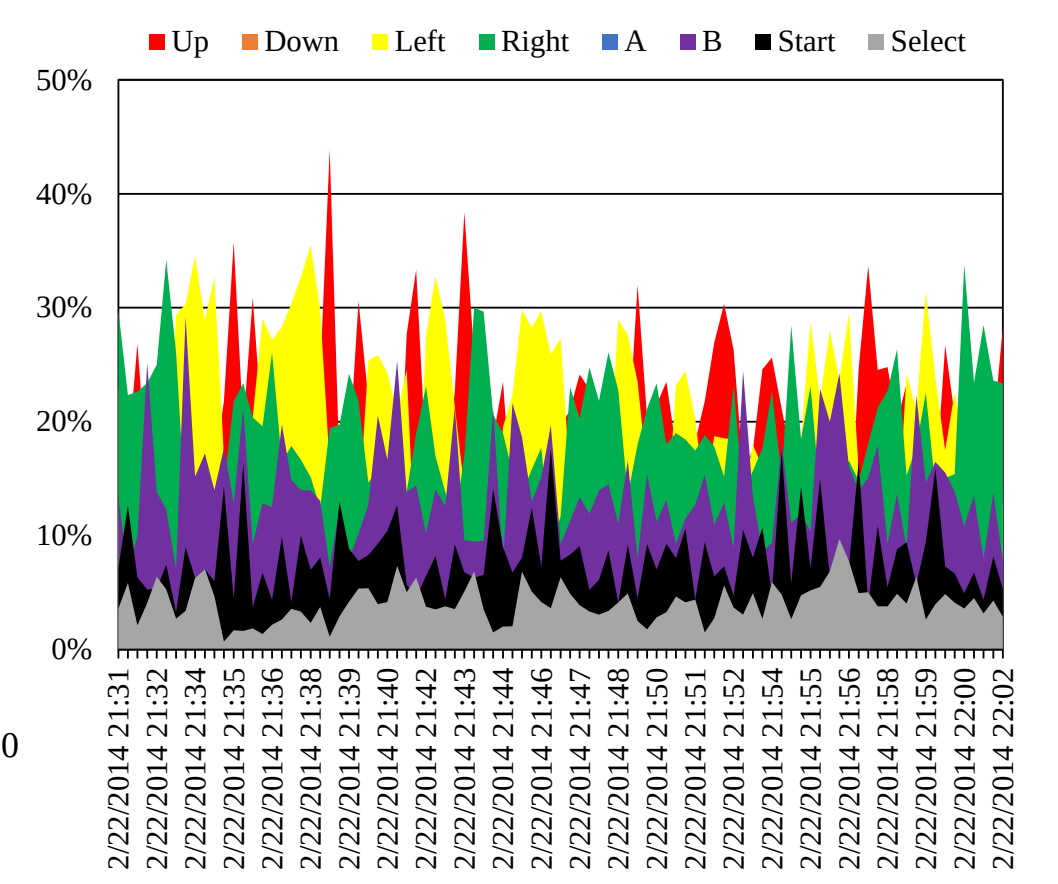
In normal gameplay, the START button brings up the menu and delays progress of the game. Users entering START are not contributing to the group's goal and are then labeled as trolls. However, if the collective goal is to cut a bush or execute Sequence 1, users inputting START may actually be non-troll users. Therefore we need to distinguish between Sequence 1 and normal gameplay (where START is not needed).

This is done using a context. Every context maintains statistics about the user inputs during a period of time (context duration). If many people are pressing START – more than usual – it's possible we are in a Sequence 1 event and we can classify accordingly.

Effect of Context Duration on the Messages Per Context
"Changing the window size has what effect on context size?"



Frequencies of Each Button
Context Duration: 20 seconds



We experimented with context durations of 60 seconds, 20 seconds, and 1 second. Every 60, 20, and 1 second(s), we analyze the chat log create the context. As the context duration increases, the effect of noise is reduced, and the context is smoother. We chose to use a context duration of 20 seconds for this project. On the right graph, you can see how the context goal changes over time. The peaks at each point in time indicate which command was most popular in that context. Sometimes it is left, sometimes it is up. It is most likely that the players are attempting to navigate through the world as the frequencies of A and B are low (i.e. not in a menu).

Background: Twitch Plays Pokémon

Twitch Plays Pokémon [1] is a "social experiment" and channel on the video streaming website Twitch.tv, consisting of a crowd-sourced attempt to play Game Freak and Nintendo's Pokémon video games by parsing commands sent by users through the channel's chat room [2]. Users can input any message into the chat but only the following commands are recognized by the bot (server-side script parsing inputs): up, down, left, right, start, select, a, b, anarchy, democracy. Any other message will have no effect on the game. Anarchy and democracy refer to the "mode" of the game. In anarchy mode, inputs are executed in pseudo-FIFO order (see next paragraph for definition). In democracy mode, the bot collects user input for 20 seconds, after which it executes the most frequently entered command.

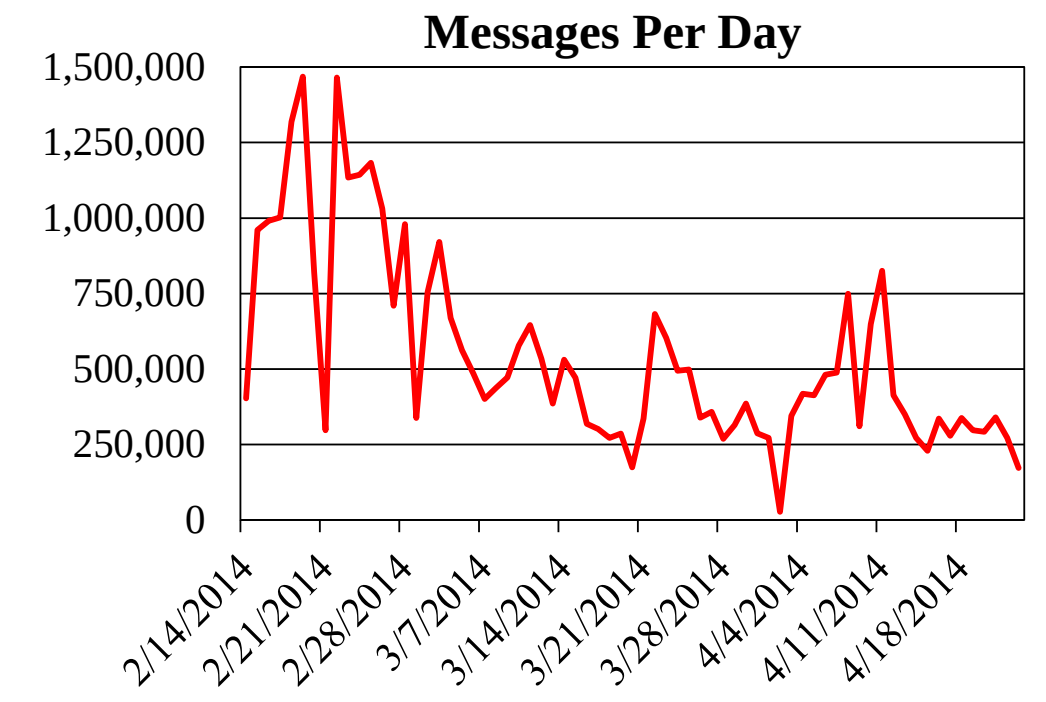
The bot attempts to execute commands sequentially in a pseudo-FIFO order. Many commands are skipped to empty the queue faster and thus an element of randomness is introduced, hence the name pseudo-FIFO. The uncertainty in the queue reduces to selecting an action at random where the frequency of user input serves as the underlying probability distribution.

[1] http://www.twitch.tv/twitchplayspokemon
[2] Wikipedia: Twitch Plays Pokémon

Start and End Dates of Pokémon Games

Game	Start	Completed	Completion Time
Pokémon Red	2/12/2014	3/1/2014	16 days, 7 hours, 45 minutes, 30 seconds
Pokémon Crystal	3/2/2014	3/15/2014	13 days, 2 hours, 2 minutes, 55 seconds
Pokémon Emerald	3/21/2014	4/11/2014	20 days, 22 hours, 1 minutes, 5 seconds
Pokémon FireRed	4/11/2014	*Ongoing*	

Messages Per Day



Most Frequent Words

Rank	Word	Count
1	anarchy	2,623,439
2	democracy	2,506,334
3	the	1,290,375
4	we	991,997
5	to	900,017
13	ヽ༼ຈل͜ຈ༽ﾉ	320,374

Design

Data Collection & Preprocessing

- Data was collected using a Ruby script connected to the twitch.tv IRC channel
- The stream started on 2/12/2014; we started collecting data on 2/14/2014
- The raw XML data totals roughly 3.5 GB and consists of 37.8 million messages submitted from roughly 1 million unique users
- We used Hadoop 2.4 and Python 3.4/2.7 for context and feature extraction

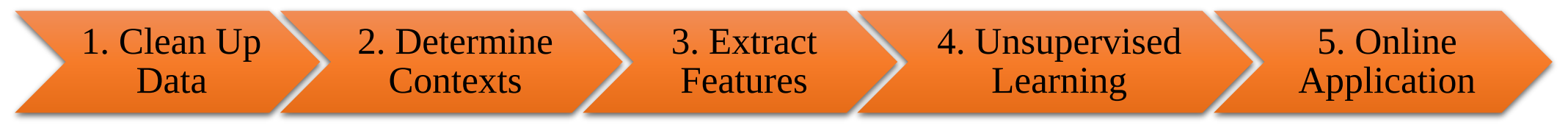
The raw data looks like this:

```
<date>2014-02-14</date><time>08:16:23</time><user>yeniuss</user><msg>A</msg>
```

Software Design

- NumPy 1.8.1 and SciPy 1.8.1 were used for most machine learning tasks
- Since everything is done in-memory, we used memory-optimized Amazon EC2 (r3.xlarge) instances, to perform feature extraction and some training. These machines were equipped with 32 GB of memory and 80 GB SSD disks.

Procedure & Workflow



1. Clean Up Data
2. Determine Contexts
3. Extract Features
4. Unsupervised Learning
5. Online Application

Algorithms & Distance Measures

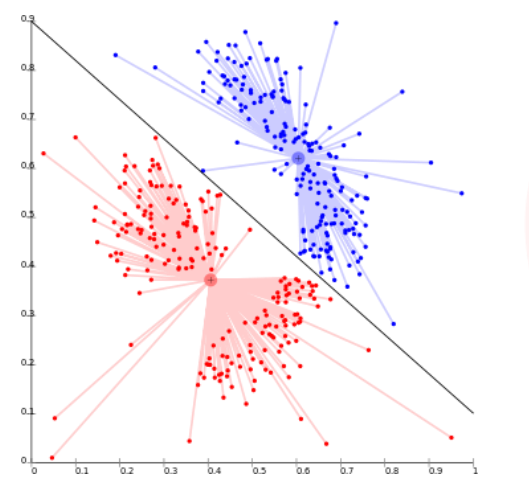
We used *k*-means clustering as our primary algorithm. We use a variety of distance measures to calculate the anomaly score. These methods are outlined below.

Anomaly Scoring

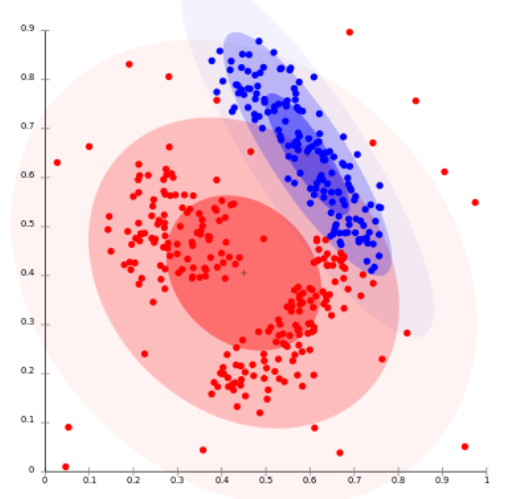
We assign a decimal value between 0 and 100 representing our confidence that the point is an outlier, or a troll. This score can be calculated in many ways. In this project, we used the following scoring techniques:

1. Distance to *k*-nearest neighbor (KNN)
2. Sum of distances to *k*-nearest neighbor (SKNN)
3. Relative density (RD)
4. Local outlier factor (LOF)
5. Multi-granularity deviation factor (MDEF)

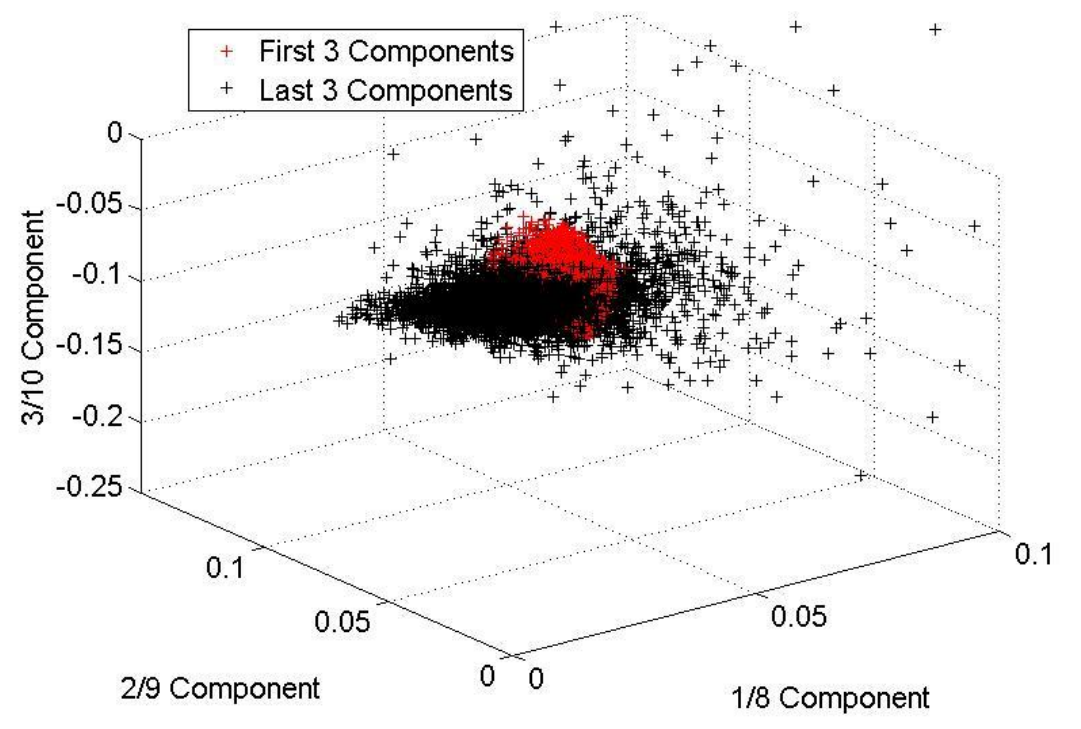
SKNN Illustration



RD Illustration



PCA Illustration



Dimensionality Reduction

Furthermore, we used principal component analysis and selected the first three principal components $w_{(1)}, w_{(2)}, w_{(3)}$. Since our original matrix was 1 million by 10, we used a modified version of SVD that is more computationally compact:

$$A^T A = V S^2 V^T \text{ and then } U = A V S^{-1}$$

A plot of the first three principal components is shown above. A plot of the last three principal components is shown as a comparison.

Feature Selection

We used the following features to describe a user:

Percentage of time inputs are aligned with top *n* goals or least *n* important goal

$$f_1 = \sum_i P(b_i | b_i = g_1) \qquad \begin{array}{c} b_i = user\ entered\ button\ b\ in\ context\ i \\ g_i = i^{th} strongest\ goal\ in\ context\ i \end{array} \qquad f_8 = \sum_i P(b_i | b_i = g_8)$$

$$f_2 = \sum_i P(b_i | b_i = g_1 \wedge b_i = g_2) \qquad f_3 = \sum_i P(b_i | b_i = g_1 \wedge b_i = g_2 \wedge b_i = g_3)$$

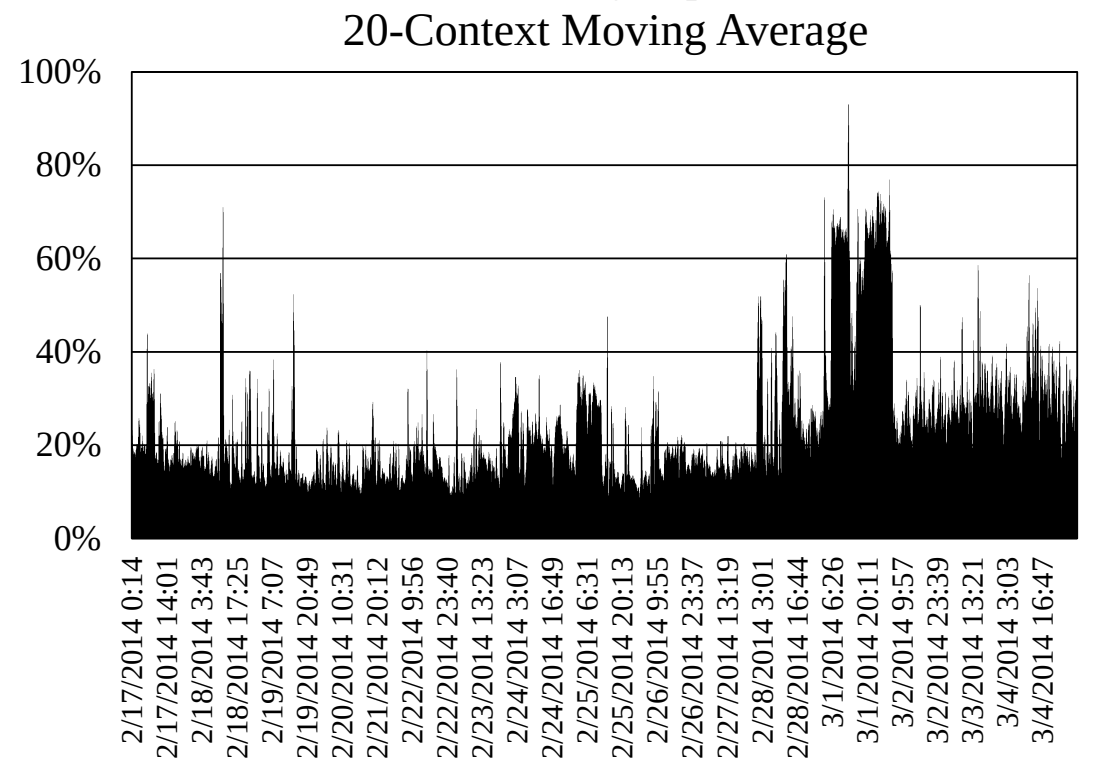
Percent of messages that are non-inputs (spam)

$$f_4 = \sum_i P(m_i = \bar{b})$$

$$m_i = user\ entered\ message\ m\ in\ context\ i$$

Spam levels give us a clue as to what is happening in-game. See the long spike above 60%? That is the period between two games when the stream is offline. Additionally, when the players defeat a gym leader (major accomplishment), spam temporarily spikes.

Percentage Spam
20-Context Moving Average



Percent of button commands sent during anarchy mode

Trolls tend to talk more during anarchy mode. If the mode is democracy, they generally leave the chat room.

$$f_5 = \frac{\sum_i P(m_i | anarchy)}{\sum_i P(m_i)}$$

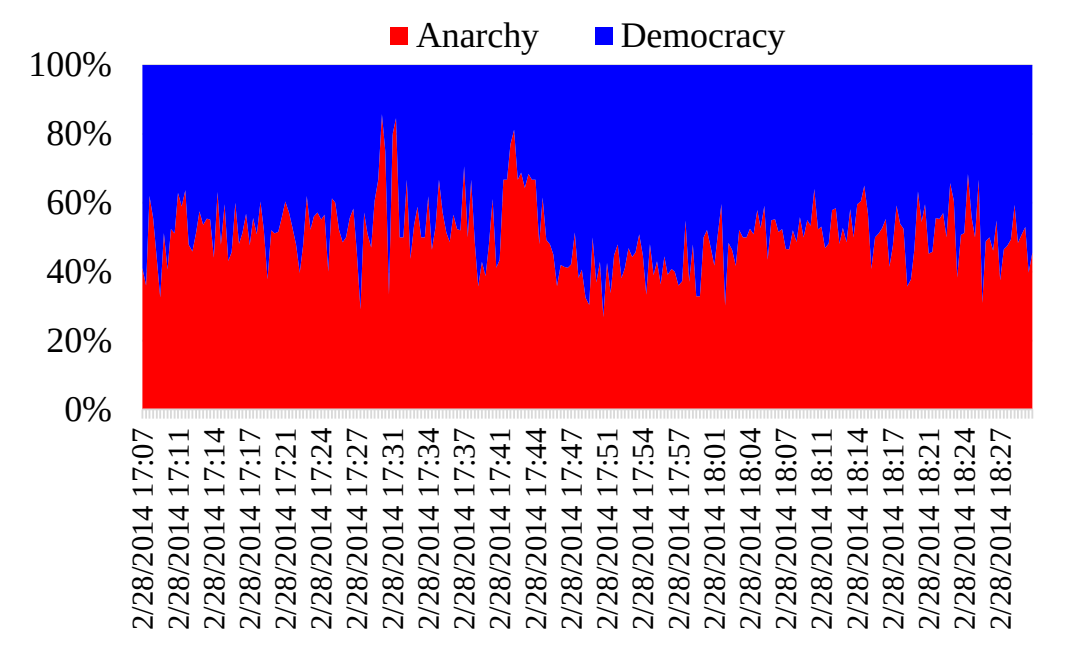
Percent of START commands

$$f_6 = \frac{|b_i = start|}{|b_i|}$$

Percent of anarchy votes

$$f_7 = \frac{|m_i = anarchy|}{|m_i = anarchy \cup m_i = democracy|}$$

Anarchy vs. Democracy
Context Duration: 60 Seconds



Feature Set

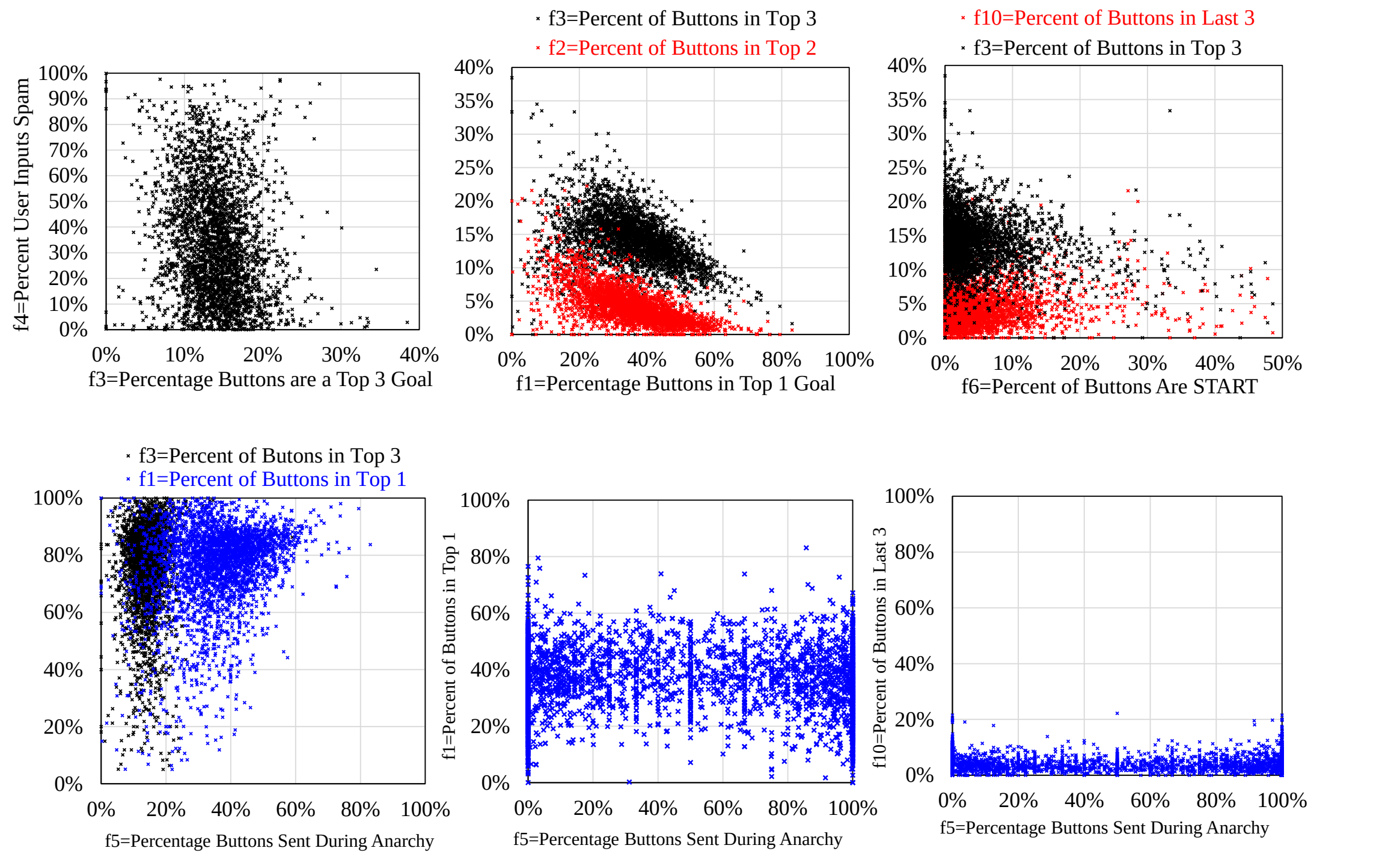
In addition to the features below, we collected information regarding the interval and total messages sent per user. This information was not used as features but instead assisted with throwing out impossible anomalies and/or incomplete data points.

The following features were used for clustering and unsupervised learning:

- f_1 Percent of button inputs that are in the top 1 goal of each context
- f_2 Percent of button inputs that are in the top 2 goals of each context
- f_3 Percent of button inputs that are in the top 3 goals of each context
- f_4 Percent of messages that are spam
- f_5 Percent of button inputs sent during anarchy mode
- f_6 Percent of button inputs that are START
- f_7 Percent of mode inputs that are ANARCHY
- f_8 Percent of button inputs that are in the bottom 1 goal of each context
- f_9 Percent of button inputs that are in the bottom 2 goals of each context
- f_{10} Percent of button inputs that are in the bottom 3 goals of each context

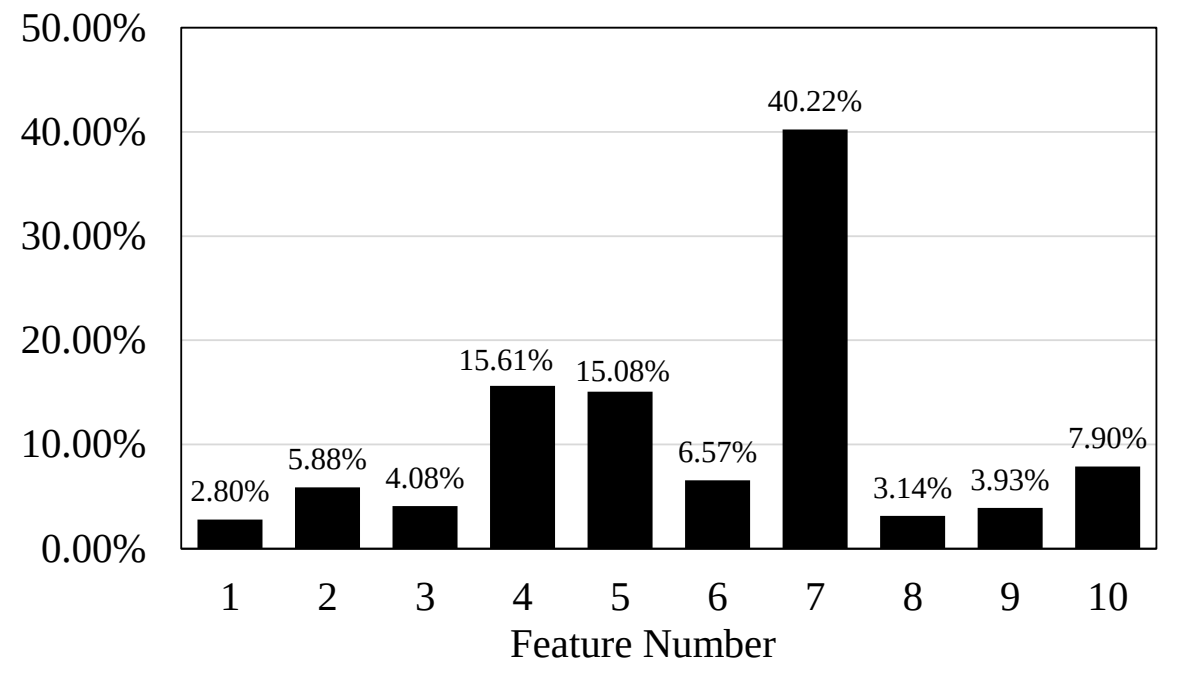
Results

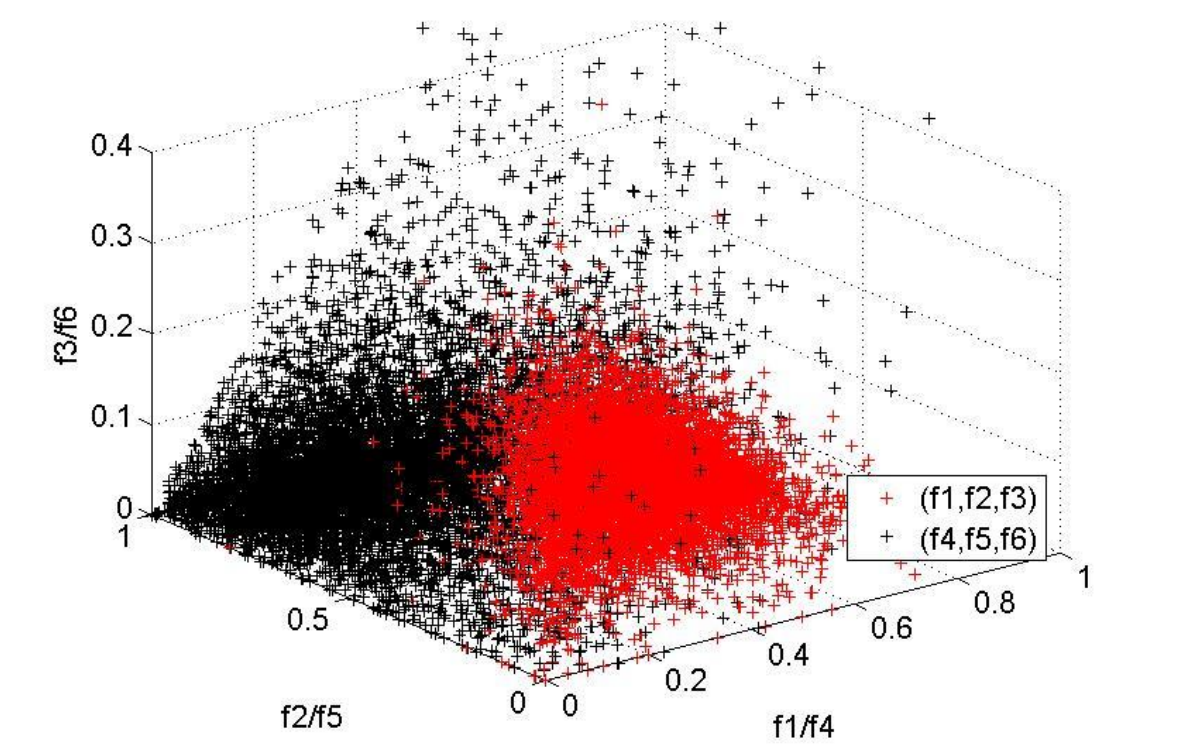
We train our algorithm offline on the 38 million data points collected. We plotted the relationship between several features.



Density based clustering was performed on feature plots similar to those shown above. We experiment by using a single feature to identify a troll and note the percentage of total users classified as trolls:

Percent of Users Labeled As Trolls Using Features





A lot of this work is currently ongoing. Results pertaining to the labeling of users in a streaming environment are currently under way and will be compiled soon.

Related & Future Work

A lot of the work done here can be extend to anomaly detection in the online setting. During popular interview events, moderators often use an internet method of collecting questions to ask the speaker. University courses use twitter so students can ask questions anonymously. Constructing contexts, using methods developed in this project, can be used to generate contexts during classes and interviews. It would then be possible to filter out irrelevant questions. In Jiang et al. [1], context is used for mass surveillance videos. The time of day, place, day of the week, and number of people all have an impact on how people act. Modeling this as a context helps machines learn contextual variables – especially in computer vision [2]. The work done in this project serves as a starting point for future research in online, context-based communities, and gives researchers new insights as to which areas require greater focus.

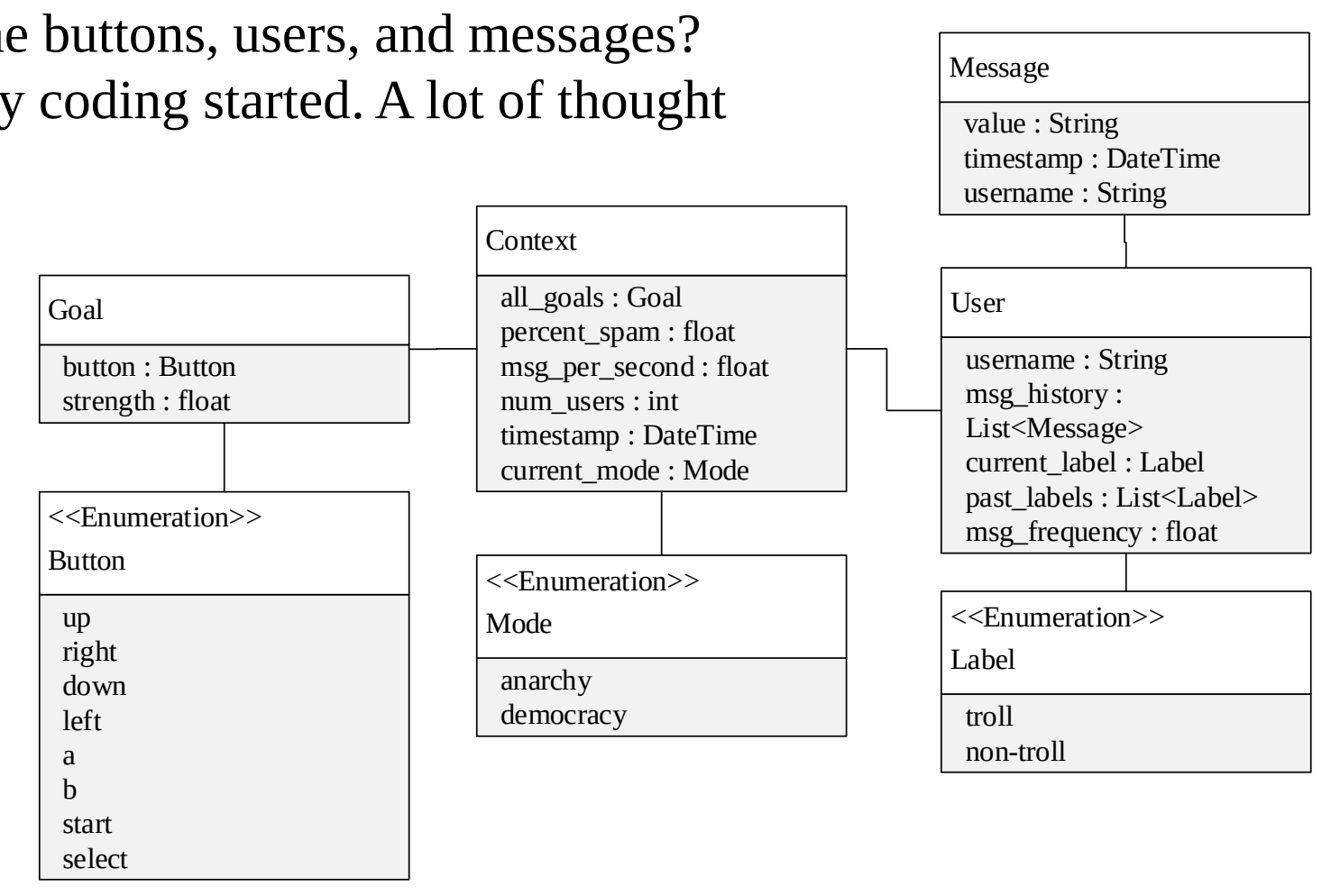
[1] Jiang, Fan, Ying Wu, and Aggelos K. Katsaggelos. "Detecting contextual anomalies of crowd motion in surveillance video." *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 2009.
[2] Jiang, Fan, et al. "Anomalous video event detection using spatiotemporal context." Computer Vision and Image Understanding 115.3 (2011): 323-333. APA

Frequently Asked Questions

Q: There's a time delay of about 20-40 seconds when a user inputs a command and when the bot parses it. How do you deal with this?
A: We ignore it. We are concerned with what happens among the users rather than how it relates to what is actually happening in the game. Both non-trolls and trolls will face this lag. As a result, the chat will be behind the video by about 20-40 seconds. We can generate the context just fine because everyone is acting in unison. They may be 20-40 seconds behind, but they're in unison, and that's enough for us to build the context.

Q: Did you do anything differently in democracy and anarchy mode?
A: No. Democracy allows for complex inputs such as "2up3right" and "2a1start2down." Since we did not want to deal with this complexity, we ignored democracy and anarchy mode when labeling users. We do take note of the current mode at any given time. This is used as one of our features.

Q: How did you integrate the context, the buttons, users, and messages?
A: We created a class diagram before any coding started. A lot of thought went into the design. As a result, we were able to solve any potential issues before the code was written. This also accelerated the development step since most of the requirements were already specified. The design was followed closely, for the most part, with some deviations regarding the goal and labels. Additionally, methods are not included in the class diagram on the right.



Message
value : String
timestamp : DateTime
username : String

Context
all_goals : Goal
percent_spam : float
msg_per_second : float
num_users : int
timestamp : DateTime
current_mode : Mode

User
username : String
msg_history : List<Message>
current_label : Label
past_labels : List<Label>
msg_frequency : float

Goal
button : Button
strength : float

<<Enumeration>>
Button
up
right
down
left
a
b
start
select

<<Enumeration>>
Mode
anarchy
democracy

<<Enumeration>>
Label
troll
non-troll

CS 363D Statistical Learning & Data Mining Term Project
All code is publically available on Github: https://github.com/ahaque/twitch-troll-detection/
Albert Haque, May 2014