
Twitch Plays Pokemon, Machine Learns Twitch: Unsupervised Context-Aware Anomaly Detection for Identifying Trolls in Streaming Data

Albert Haque

Department of Computer Science
University of Texas at Austin
May 2, 2014

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 clustering and distance-based methods, we use contextual data such as the group’s current goal, the current time, and the username to classify each point as an anomaly. A user whose features significantly differ from the norm will be classified as a troll. We collected 38 million data points from the viral Internet fad, Twitch Plays Pokemon. Using clustering and distance-based methods, we develop heuristics for identifying trolls. 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.

1 Introduction

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].

Application of the term troll is subjective. Some readers may characterize a post as trolling, while others may regard the same post as a legitimate contribution to the discussion, even if controversial [1]. Regardless of the circumstances, controversial posts may attract a particularly strong response from those unfamiliar with the dialogue found in some online, rather than physical, communities [1]. Experienced participants in online forums know that the most effective way to discourage a troll is usually to ignore it, because responding tends to encourage trolls to continue disruptive posts hence the often-seen warning: “Please don’t feed the trolls” [5].

Given the novelty of the Twitch Plays Pokemon event, we wanted to leverage known clustering heuristics to identify trolls in real-time. This requires the application of several preprocessing steps which we performed in MapReduce [4], a distributed processing system.

The contributions of this project are as follows. First, we propose a set of features used for identifying trolls in the viral, crowdsourced Internet game, Twitch Plays Pokemon [2]. We use context-based techniques to understand the scenario a human is faced when entering input into the chatroom. We then compare the effects of different distance measures to understand their strengths and weaknesses.

The second major contribution of this paper is 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. The primary purpose of this project, through these two contributions, is to in realtime, 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. The full dataset and code are made available online¹.

The remainder of this paper is organized as follows: In Section 2 we give an overview of Twitch Plays Pokemon and discuss the dataset we collected. Section 3 outlines how we extract features from our dataset. These features are then used by algorithms outlined in Section 4. Results are shown in Section 4 followed by a discussion in Section 5. We then conclude in Section 6 by discussing related and future work.

2 Background

2.1 Twitch Plays Pokemon

Twitch Plays Pokemon 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 Pokemon video games by parsing commands sent by users through the channel’s chat room. 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. This is shown in Figure 1.

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 mode of the game changes when a certain percentage of users vote for democracy or anarchy. The mode will remain in that state until users vote the other direction.



Figure 1: Screenshot of Twitch Plays Pokemon. User input messages are shown on the right side. The top of the list is the front of the queue.

In anarchy mode, 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<https://github.com/ahaque/twitch-troll-detection>

2.2 Dataset

We wrote an Internet Relay Chat robot to collect user messages, commands, time, and user information. The bot collected data for 73 days. It is 3.4 GB in size and contains approximately 38 million data points. A data point is defined as a user’s chat message and metadata such as their username and timestamp at millisecond resolution. Figure 2 shows a data point in an XML-like format.

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

Figure 2: XML-Like Data Point

Users can enter non-commands into the chat as well. We call these types of messages “spam.” It is possible for a spam message to contribute positively to a conversation. However, these messages have no impact on the game. When a major milestone is accomplished in the game, it is common to see an increase in the frequency of spam messages. Certain spikes can be seen in Figure 3. It is possible to gain further insight about each spam message through natural language processing techniques but that is outside the scope of this project.

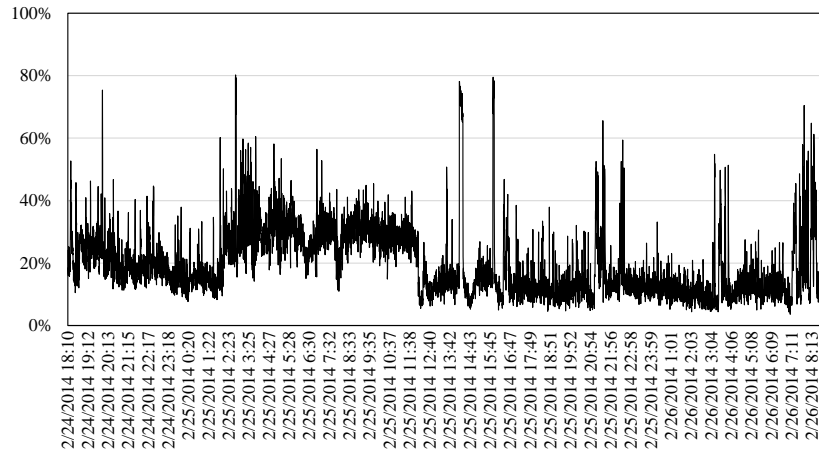


Figure 3: Percent of Messages Considered Spam. Each point corresponds to a 20-period moving average with periods of one second each.

Since a human logs into the system with the same username, we are able to build profiles of each user and learn from their past messages and commands. We do this by constructing a “context” every twenty seconds using all messages submitted during the twenty seconds. The notion of contexts is explained in the next section.

3 Feature Selection

3.1 Context

We define a context as a summary of a period of time. It can also be thought of as 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 Pokemon, 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 Pokemon game, we are referring to the “cut” sequence where the player must cut a bush. Sequence 1 (Figure 4) below lists the required button chain.

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

DOWN > START > UP > UP > A > UP > A > DOWN > DOWN > A

Figure 4: Sequence 1. An ordered list of button commands.

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 all user inputs during a period of time (context duration). If many people are pressing START – more than usual – it is possible we are in a Sequence 1 event and we can classify accordingly.

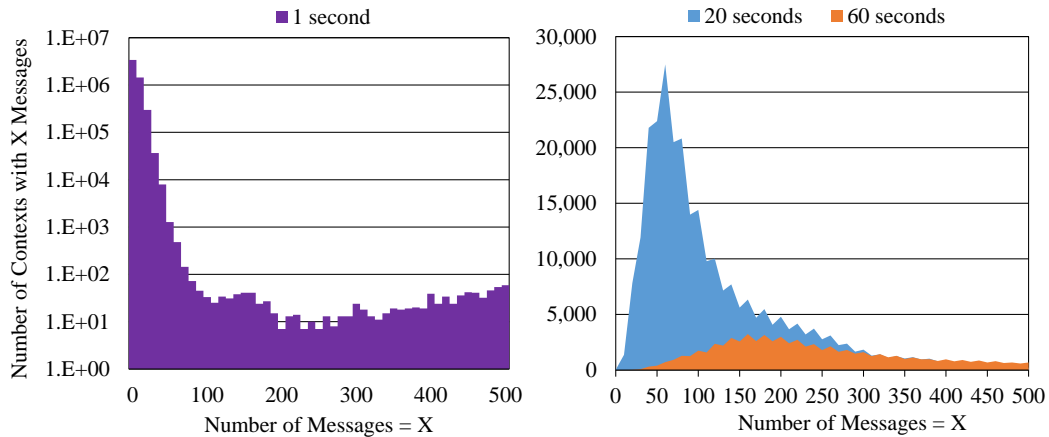


Figure 5: Effect of Varying the Context Duration on Messages per Context

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 a single context. As the context duration increases, the effect of noise is reduced (due to less granular samples as shown in Figure 5), and the context is smoother.

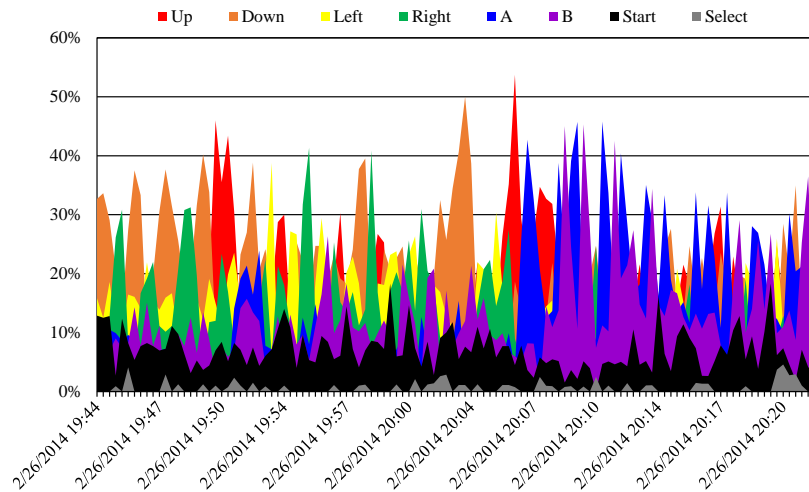


Figure 6: Button Frequencies with Context Duration of 20 Seconds

We chose to use a context duration of 20 seconds for this project. In Figure 6, 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. For the left half of the graph, we can infer that users are attempting to navigate through the world as the frequencies of A and B are low

(i.e. not in a menu). On the right half, the A and B frequencies dramatically increase – indicating a menu or message dialog has appeared.

3.2 Feature Set

We collected information regarding the intervals and total messages sent by each 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

Features f_1, \dots, f_{10} were selected after using domain knowledge about the game and the behavior of Internet trolls. Additionally, we used principal component analysis to compare the first three principal components with the last three components. This is shown in Figure 7. The components were not used as features.

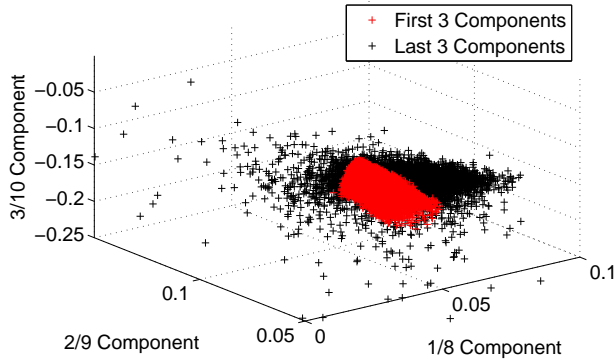


Figure 7: First and Last Three Principal Components

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

Because our original dataset consists of 1 million data points, using the standard SVD formula would require significant computational resources. To solve this issue, we used an alternate formation of SVD shown in Equation (1), where A is a matrix containing all 1 million feature vectors. The result of this compression is a reduction from a matrix A (1 million by 10) to matrix $A^T A$ (10 by 10). The latter resulted in a significantly faster running time.

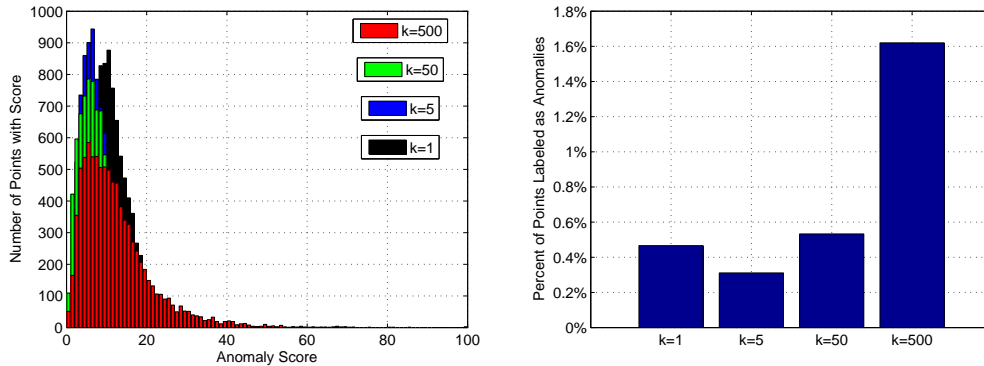
4 Anomaly Scoring & Results

We used k -means clustering and two distance-based measures to calculate an anomaly score. In particular, we looked at the distance to k -nearest neighbor (DKNN) and the sum of the distances to

the k -nearest neighbors (SKNN). To reduce the algorithm running time, we used a subset of 10,000 feature vectors randomly selected from the original 1 million users. This dataset is provided with the submitted code. All distances were normalized such that the minimum distance had an anomaly score of 0 and the maximum distance had an anomaly score of 100.

4.1 Distance to k -Nearest Neighbor

We used Euclidean distance as the distance metric with $k = 1, 5, 50, 500$. Our objective is to minimize the anomaly score of points central to the cluster and maximize the score of those far from the cluster. The results are shown in Figure 8a.



(a) Number of Points with Anomaly Score. Histogram bucket size is 1.

(b) Percent of Dataset Labeled as Anomalies Using Anomaly Score Threshold of 40.

Figure 8: Results for Distance to k -Nearest Neighbor (DKNN)

When $k = 1$, it causes the distribution to have the largest median anomaly score. This tells us that $k = 1$ is not a good choice. If two anomalies are near each other but far from the main cluster, they will have a low anomaly score. Because of this, we must look at other values of k .

With $k = 500$, we have a long right tailed distribution (see Figure 8a). This makes $k = 50$ and $k = 5$ preferable over $k = 500$. After analyzing the effect of k , we opted to use $k = 5$ as our metric. Given the distribution in Figure 8a, we used an anomaly score threshold of 40. Any data point with an anomaly score above this value would be labeled as a troll. After labeling the points, the results are shown in Figure 8b.

4.2 Sum of Distances to k -Nearest Neighbor

The sum of distances to the k -nearest neighbors (SKNN) is similar to DKNN but with one modification: we take the sum of the first to k^{th} nearest neighbor and use this as our distance metric. Results are shown in Figure 9a.

The SKNN algorithm produces results similar to DKNN. However, when $k = 500$, SKNN labels half as many anomalies as DKNN (see Figure 9b), whereas $k = 1, 5, 50$ produce almost identical results as DKNN.

4.3 k -Means Clustering

In addition to k -nearest neighbors, we evaluated the k -means clustering algorithm. The value of k was set to 1. Once the prototype was defined, we used Euclidean distance to estimate each point's distance to the prototype. These distances were normalized and an anomaly score between 0 and 100 was calculated. Higher numbers indicate a higher degree of peculiarity. The algorithm was run using MATLAB and results are shown in Figure 10.

Also using an anomaly score threshold of 40, k -means clustering labeled 107 data points as anomalies which is equivalent to 1.19% of the dataset.

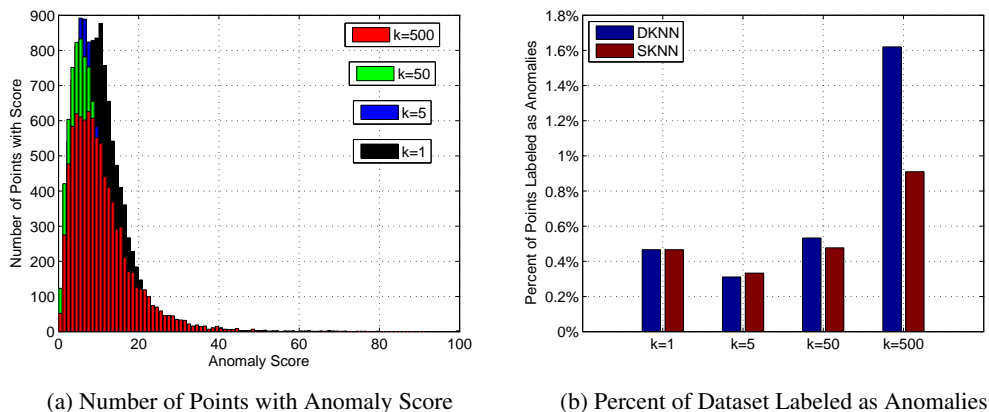


Figure 9: Results for Sum of Distance to k -Nearest Neighbor (SKNN)

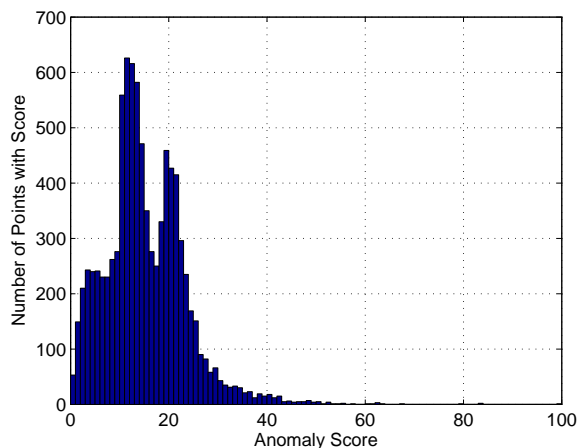


Figure 10: k -Means Clustering, $k = 1$, Number of Points with Anomaly Score

4.4 Online Performance

We apply our anomaly detector to the real-time data stream. Because we cannot obtain features until the context has been generated, there is a maximum delay of 20 seconds until we update our feature vector. We re-cluster the data points once an hour with the updated feature vectors. After re-clustering is complete, we relabel any users who have changed from non-troll to troll and vice versa.

Because this data is unlabeled, we are unable to test the accuracy of this algorithm. However, we have created several test data points (user accounts we own) and have acted as a troll or non-troll. Our online anomaly detector successfully labeled our test data points in the streaming data. Furthermore, we estimate about 1% of all users are actual trolls.

5 Discussion

Since we do not know the true labels of each data point, the results shown in Section 4 tell us that our definition of an anomaly plays an important role in how many anomalies, or trolls, are labeled. Different trolls will exhibit anomalies in different features and as a result, we cannot look solely at a single feature as a basis for identifying a troll. As shown in Figure 11, if we look at f_7 (the number of times a user votes for anarchy) we cannot use this as the sole basis for categorizing users. Simply

too many people vote for anarchy, both trolls and non-trolls, and it is very doubtful that 40% of all users are trolls.

All ten features had a decimal value between 0.0 and 1.0 inclusive. This was generated by analyzing the dataset and corresponds to frequencies. Most features were taken by counting the number of user X’s messages that meet the feature’s condition divided by the total number of messages sent by user X.

Since a single feature does not work, we must look at more sophisticated techniques such as different distance measures and looked at all ten features simultaneously. When this was done, significantly less users were labeled as anomalies. We estimated about 1% of all users are true anomalies and Figure 9a shows how our distance measures were able to classify approximately the same percentage of users as trolls.

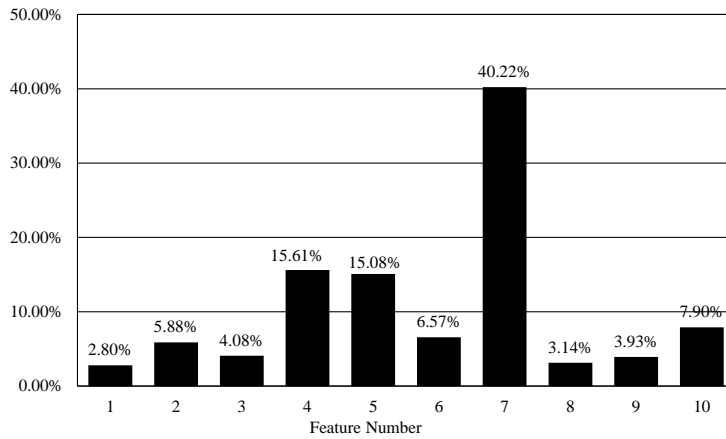


Figure 11: Percent of Users Labeled as a Troll by Using Single Feature. This was done by using an arbitrary threshold using domain knowledge.

We wanted to further understand what differentiates trolls from normal users. To do this, we analyzed the features of trolls labeled by k -means clustering. We took these features and calculated how far, on average, did they lie from the k -means centroid. The results are shown in Figure 12.

The results tell us that trolls tend not to contribute to the group’s goal. Features f_1, f_2, f_3 represent the percentage of a user’s messages that are in agreement with the group’s goal. As shown in Figure 12, trolls are generally not in agreement. Unsurprisingly, trolls tend to submit buttons that are the least frequently submitted in a given context. This is demonstrated by f_8, f_9, f_{10} . This tells us that trolls enter buttons that have the least priority and most of the time have negative effects.

Feature 6 counts how often a user pushes START, which is the “worst” possible move for most contexts. For f_6 , trolls push START almost 20% more than the average user. For feature 7, it is expected that trolls vote for anarchy. However, look at Feature 5 – the number of button inputs that are submitted during anarchy mode. This is surprisingly low and could be explained by the fact that trolls enter the game during democracy, vote for anarchy, then leave once the anarchy has commenced. This is consistent with the definition of a troll who enters a discussion or community with the intention of wreaking havoc and leaves once the job is done.

6 Conclusion

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 by using methods developed in this project can be used to generate contexts during courses and

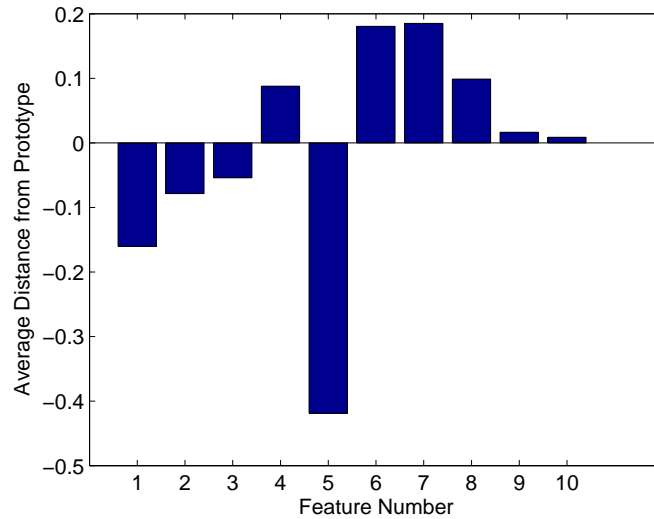


Figure 12: Average Distance from Prototype for Troll Features. For all trolls labeled by k-means clustering, the plot shows the average distance from the prototype for each feature.

interviews. It would then be possible to filter out irrelevant questions depending on the current lecture slide or segment of the interview.

In Jiang et al.[6], 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 [7]. 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.

References

- [1] Wikipedia. <https://www.wikipedia.org/>.
- [2] Twitch plays pokemon. <http://www.twitch.tv/twitchplayspokemon>, April 2014.
- [3] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM Sigmod Record*, volume 29, pages 93–104. ACM, 2000.
- [4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [5] C. Heilmann. De-trolling the web: dont post in anger. <http://christianheilmann.com/2012/06/04/de-trolling-the-web-dont-post-in-anger/>, June 2012.
- [6] F. Jiang, Y. Wu, and A. K. Katsaggelos. Detecting contextual anomalies of crowd motion in surveillance video. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 1117–1120. IEEE, 2009.
- [7] F. Jiang, J. Yuan, S. A. Tsafaris, and A. K. Katsaggelos. Anomalous video event detection using spatiotemporal context. *Computer Vision and Image Understanding*, 115(3):323–333, 2011.