Trolls, Haters, or Teammates: Classifying Sentiment in the Language of PvP Game Communications

Kevin Borst
Stanford University
kborst2@stanford.edu

Thomas Yang
Stanford University
thyang23@stanford.edu

Melinda Zhu
Stanford University
melinda7@stanford.edu

June 3, 2021

1 Introduction

Since the first PvP (player vs. player) video games began allowing live communications during matches, toxic players have utilized in-game chats as platforms for unfiltered cyberbullying with little meaningful consequences. Existing filters used in games today only identify specific profanity at best, and players rarely suffer reprisal for racist, sexist, homophobic, or other insulting and derogatory remarks during live gameplay. Simultaneously, a unique lexicon of "gamer slang" (Fig. 1) has arisen, both in general and for specific games. Gamer slang is relatively unknown and unused in regular conversation, and as a result, sentiment analysis algorithms (the identification of emotional connotations from text) adapted to normal English conversations are unable to comprehend and recognize toxic messages in games. Our project aims to extend existing sentiment analysis models to the "language of gaming" through a supervised learning setting. The inputs to our algorithms are in-game chat messages in vectorized forms. We then compare the performance of several classifiers such as Support Vector Machines, Naive Bayes, K-Nearest Neighbors, Random Forests, and neural networks to output one of three sentiments – neutral/positive (class 0), negative attitude (class 1), or derogatory (class 2) – for each message. This text classification is foundational to any effort to regulate live toxic language.

message	label
report for unskilled player is useless	negative attitude
ggwp	neutral/positive
such a feeder...xD	negative attitude
shut up f g	derogatory
totally noob team	negative attitude
go play barbie like a girl	derogatory

Figure 1: Examples of subtle aggression in game chats. ("ggwp" = good game, well played. "feeder" = an insult towards a player who gets killed frequently by enemies.)

2 Related work

Past studies have recognized the existence of a "gaming language" and its relevance to addressing cyberbullying in gaming, but the specific applications of sentiment analysis and machine learning remains open for exploration. The lexicon-based sentiment extractor built by Blair et al. for StarCraft 2 incorporated game slang [1], but relied on an inflexible dictionary of expected words without consideration of context or frequency. Another study by Buchanan et al. utilized automatic classification through SQL database queries, and while this setup could identify profanity and some aspects of racist sentiments, it identified fewer than 45% of manually-labeled toxic messages [2].

As a result, to identify toxic and derogatory language in an online game setting, we aim to apply tools from sentiment analysis and NLP that have been used frequently in the domain of social media [3]. For instance, Arabnia et al. utilized Naive Bayes and Random Forest classifiers (techniques we implement in this study) on annotated Twitter data, correctly detecting 85% of cyberbullying in social media comments [4]. Neural networks are also a powerful tool for text classification and especially relevant to our desired multiclass model, where negative and positive attitudes must be clearly distinguished from explicit, derogatory text. In particular, applications such as text vectorization and feedforward neural networks in classifying Reuters newswires [5] serve as inspiration for our neural network implementations, where we also use similar embeddings that only retain words most often associated with each class.

Even where moderation filters are actively in place, such as the first-person shooter Overwatch [6], game-specific insults and microaggressions continue to pervade the game, creating a persistently toxic environment. In our work, we aim to extend Blair et al.'s and Buchanan et al.'s work to include ML and neural networks, and ultimately be able to detect subtleties that cannot be captured by current game chat filters.

3 Dataset and Features

We compiled 33,000 chat messages from DOTA 2 and League of Legends [7,8], with manual labels of the three classes. We balanced the dataset to reflect the general distribution of messages during gameplay, where neutral/positive messages hold a slight majority, and derogatory messages appear not as frequently (Fig. 2). We randomly divided the dataset into training and test sets with a 80/20 split, and then we split the training set into 80% training and 20% validation data, using scikit-learn's `train_test_split` function [9].

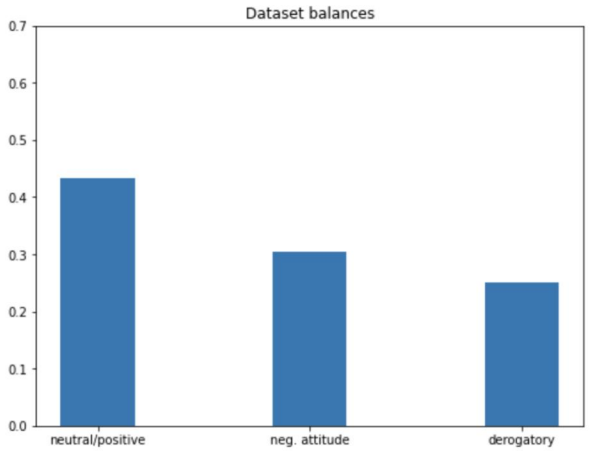


Figure 2: Balance of classes from the dataset.

3.1 TF-IDF Encodings

For our ML classifiers, we vectorized each message using the TF-IDF (Term Frequency Inverse Document Frequency) Vectorizer, which measures words' originality by comparing the number of times a word appears in a message with the number of messages the word appears in [10]. Each column of the matrix is an index in the vocabulary of the dataset. The TF-IDF values for each term t are calculated as follows:

$$\text{tf-idf}(t) = \text{tf}(t,d) \times \text{idf}(t)$$

$$\text{tf}(t,d) = \text{number of times word } t \text{ appears in msg. } d$$

$$\text{idf}(t) = \log\left(\frac{1+|\text{messages}|}{1+\mathbb{1}\{d : t \in d\}}\right)$$

The primary benefit of using TF-IDF, instead of assigning binary appearance values to each word in the vocabulary (which considers only individual messages), is that TF-IDF takes into account the frequency of words in the entire dataset, and does not rely on pre-trained vectors. This is particularly significant for our data because of the many acronyms and slang terms used throughout.

3.2 Word2vec and GloVe Embeddings

For the neural networks, we incorporated word2vec and GloVe in our Embedding layer, instead of using relatively high-dimensional vectors of TF-IDF ($\approx$ 2700 dimensions). Word2vec (from Python's Gensim library) and GloVe group similar words together in a vector space and makes highly accurate predictions about a word's meaning from past appearances. While Word2vec relies on semantic meaning through local context, GloVe uses global context, making it potentially more effective with its comprehensive coverage of an entire corpus and dataset. For GloVe embeddings, we used vectors of 100- or 50-dimensions for each message, where the vectors are pre-trained from Gigaword 5: a newswire text archive [11]. For word2vec, we use 100- or 50-dimensional pre-trained vectors from a Google news dataset [12]. Compared to TF-IDF, we anticipated that the pre-trainings used in word2vec and GloVe would result in lower accuracies due to the "gamer language" rarely being used in news articles.

4 Methods

4.1 Baseline: Binary SVM

For our first baseline, we combined both negative attitude (class 1) and derogatory language (class 2) into one class for a binary classifier (using TF-IDF encodings), to determine whether there exists a clear separation between neutral/positive (label $y = 1$) and negative attitude/derogatory (label $y = -1$) examples. We chose SVM because of its compatibility with separable data and primary goal of separating examples with a clear hyperplane by maximizing geometric margin. The optimization problem for SVM is as follows [13]:

$$\max_{\gamma,w,b} \ \gamma$$

$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq \gamma, i = 1, \ldots, n$$

with the constraint that $||w|| = 1$. Here, γ represents the geometric margin (distance to a decision boundary), and (w, b) are the weights and bias term that the model learns.

4.2 Naive Bayes

We implemented two iterations of multinomial Naive Bayes: the first of which (our baseline) utilizes a manually constructed dictionary of the 165 most frequently encountered toxic phrases in gaming. We chose Naive Bayes because of its conditional independence assumption; namely, that presence(s) of a certain phrase in a message has no effect on how many times a different phrase in the same message is present [14]. We constructed input matrices that had sparse vectors with each column ($t = 1, \ldots, 165$) containing the frequencies of phrase w_t in message i. Our goal with the predetermined vocabulary was to force messages without any phrases in the vocabulary (i.e. class 0 messages) to have an almost-zero probability of belonging to class 1 or class 2. The $3d - 1$ parameters of the model are as follows (with Laplace smoothing):

$$P(w_t \mid C = k) = \frac{1 + \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = k\} x_t^{(i)}}{|V| + \sum_{s=1}^{|V|} \sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = k\} x_s^{(i)}}$$

$$\text{priors: } P(C = k) = \frac{\sum_{i=1}^{n} \mathbb{1}\{y^{(i)} = k\}}{n}$$

where $|V|$ is the dictionary size and each x is a frequency vector corresponding to the dictionary. For our second iteration of NB, we used TF-IDF encodings instead of a predetermined dictionary.

4.3 K-Nearest Neighbors

Similar to our second iteration of NB, we used TF-IDF encodings to create the input matrices for KNN. With each message, the Euclidean distance (similarity) of its feature vector to other feature vectors in the training matrix would be found. Then, we build an ordered collection of distances and indices of the data. A parameter K determines how many of the training messages that are "closest" to the test message are considered. For example, if $K = 2$, the two closest mes-

sages would be considered. The most frequently occurring label (the mode) of the K closest messages to a given test message is then the predicted class label [15].

4.4 Random Forests

Branching off into alternative methods of classifying language in the PvP game lexicon, we utilized random forests, which constructs multiple randomized decision trees and averages their results to avoid the overfitting that can come from the use of a single decision tree.

We tested this with two common loss functions to determine where to split the data. First is minimizing the entropy loss, given by:

$$L_{cross}(R) = -\sum_{c} \hat{p_c} \log_2 \hat{p_c}$$

where c is the number of classes and each $\hat{p}$ is the prevalence of a given class [16]. We then build random forests by minimizing the Gini impurity, given by:

$$I_G(\hat{p}) = \sum_{i=1}^{c} \hat{p_i}(1 - \hat{p_i}).$$

4.5 Feedforward Neural Network

In addition to ML classifiers, we used neural networks to determine whether intermediate learning steps and feedback loops would lead to higher accuracies. We first employed several implementations of feedforward neural networks with a small number of hidden layers. Several "intermediate labels" were the outputs for each layer, and eventually the final predictions were made based the previous intermediate outputs being fed into the final layer. We experimented with different input sizes, embeddings, and various activation functions such as ReLU, softmax, and sigmoid. Fig. 3 is an example architecture we used, inspired by the Reuters newswires classification [5]. We trained the model for 550 epochs, which we found was sufficient to reach convergence.

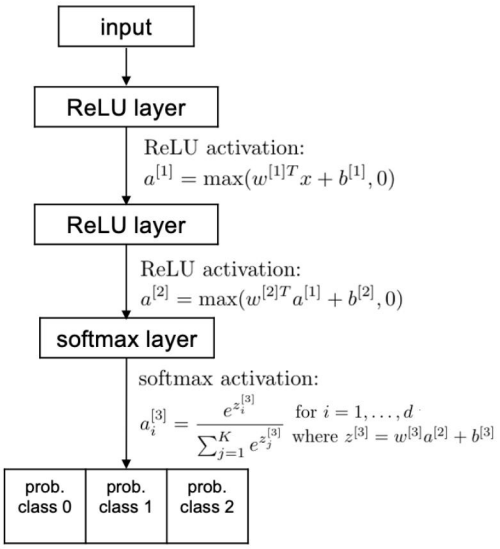
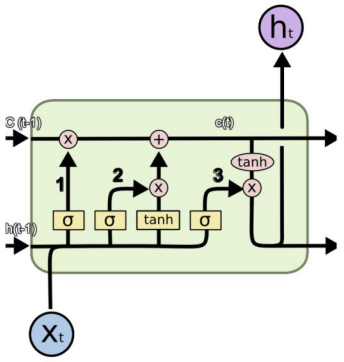


Figure 3: Example feedforward NN architecture, with w and b representing the learned weights and bias terms.

4.6 Long short-term memory (LSTM)

In contrast to feedforward NNs, we applied LSTM: a specified recurrent NN frequently used for sentiment analysis. LSTM units have feedback loops and memory cells that can hold past information and learn from context; for example, cell states can decide what words are important in deriving the sentiment of a certain phrase, and these words are stored in the memory cell of the next LSTM unit. We trained the model for 300 epochs, which was more than sufficient to reach convergence. The following shows one LSTM unit's computations in our context (see Appendix for functions used) [5]:

Figure 4: LSTM unit [17] with labeled steps 1, 2, and 3.



1. Sigmoid layer: decide what words to store in the cell state, based on output from the last LSTM unit h_{t-1} and input x_t.

2. Tanh layer: update cell state using the tanh function, with vectorized words chosen to be kept from the sigmoid layer.

3. Output h_t: compute sigmoid to decide which words in the cell state to output, and put the cell state through tanh to generate all vectorized words to input to the next LSTM unit.

We use the categorical cross-entropy loss function for multi-class classification in the feedforward NN and LSTM, which the models minimize [18]:

$$CE = -\sum_{k=0}^{C-1} t_k \log(f(s)_k)$$

where $f(s)$ is the softmax activation:

$$f(s)_k = \frac{e^{s_k}}{\sum_{j=0}^{C-1} e^{s_j}}.$$

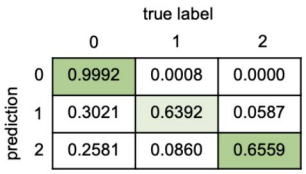
C is the number of classes, t is the target one-hot vector, and s refers to the output scores (probabilities) after 1 epoch.

5 Results and Discussion

5.1 Baselines: Binary SVM & NB

Model	Accuracy
SVM (linear kernel)	0.9565
SVM (rbf kernel)	0.9635
SVM (polynomial kernel)	0.9435
Naive Bayes	0.8500

Table 2: Accuracy scores on SVM & Naive Bayes classifiers.



Class 0	Class 1	Class 2
0.90	0.77	0.61

Figure 5: Confusion matrix and F1 scores for Naive Bayes baseline.

The SVM accuracies indicate that messages can be separated into positive/neutral and negative examples, with the linear and RBF kernels performing particularly well. These results demonstrate substantial distances between vectors across classes, thus identifying a clear decision boundary. In comparison, the polynomial kernel was slightly weaker, most likely due to overfitting.

On the other hand, our Naive Bayes baseline, which utilized a predetermined vocabulary of the most frequently used toxic phrases, was able to correctly identify most positive/neutral examples but performed poorly in distinguishing negative and derogatory examples from positive examples (Fig. 5). Words such as "f**k" and "bad" were used across negative, derogatory, and positive/neutral examples (e.g. "my bad" vs. "you're bad", or "f**k you" vs. "f**k yeah"). Without considering context, many negative and derogatory examples were incorrectly labeled as neutral/positive, as shown by the F1 scores. This exposes a potential issue with the conditional independence assumption; phrases may not be entirely independent of one another in a message, and it is insufficient to condition only on the class label without considering context within messages. We also attributed the flaws to our manually constructed dictionary, which could not generalize well to new data that contain toxic language that is not present in the dictionary.

5.2 ML Classifiers

Model	Accuracy
Naive Bayes	0.9506
K-Nearest Neighbors ($K = 1$)	0.9853
K-Nearest Neighbors ($K = 2$)	0.9711
K-Nearest Neighbors ($K = 3$)	0.9683
K-Nearest Neighbors ($K = 6$)	0.8424
Random Forest (entropy, max depth = 32)	0.8775
Random Forest (Gini)	0.9912
Random Forest (Entropy)	0.9910

Table 3: ML classifier accuracies, with TF-IDF encodings.

Model	Class 0	Class 1	Class 2
Naive Bayes	0.97	0.94	0.87
KNN ($K = 1$)	0.99	0.98	0.95
KNN ($K = 2$)	0.98	0.96	0.91
KNN ($K = 3$)	0.98	0.96	0.91
KNN ($K = 6$)	0.89	0.77	0.67
RF (Entropy, depth=32)	0.91	0.86	0.53
RF (Entropy)	0.99	0.99	0.98
RF (Gini)	0.99	0.99	0.98

Table 4: F1 scores for ML classifiers.

Our second iteration of NB with TF-IDF encodings demonstrated better performance in identifying negative (class 1) and derogatory (class 2) messages compared to the baseline, indicating that the baseline fell short due to the predetermined dictionary rather than the conditional independence assumption. Here, our model was able to capture a more comprehensive scope of words used throughout both the training and test sets. As confirmation, the most indicative words in each class were: "gg" (0), "thanks" (0), "noob" (1), "wtf" (1), "useless" (1), "r*tarded" (2), and "f*gg*t" (2).

KNN with TF-IDF also demonstrated this better performance. We tested multiple K values and saw that smaller K performed well, with $K = 1$ being ideal; instead of having large groups of similar messages, the training data had 1-2 very similar messages to a given test message, which meant that when the classifier attempted to classify a message by referring to several (> 3) nearest neighbors, those neighbors had different labels (likely due to the impactful difference of 1-2 words). As shown by the F1 scores, the classifier had high accuracy in correctly identifying messages from class 1 and class 2 for $K = 1, 2$, and 3, compared to Naive Bayes.

Random forests, without an arbitrarily enforced maximum depth, were able to precisely classify the test data, and performance using Gini impurity as the loss function were comparably good to those using entropy. However, an arbitrarily set depth performed significantly worse, particularly when classifying explicitly derogatory language as opposed to neutral/positive or negative attitude language: this is owed largely to a tendency to misclassify as neutral/positive that is mitigated with higher depths. This overfitting does not seem to be mitigated by averaging over many estimator decision trees. The F1 scores for entropy and Gini once again show the model's ability to correctly distinguish messages, even with very subtle differences between class 1 and 2.

5.3 Neural Networks

Model	Activation	Input	Score
FF	ReLU, softmax	Word2Vec 50-dim	0.8449
FF	ReLU, ReLU, softmax	Word2Vec 100-dim	0.8618
LSTM	softmax output	Word2Vec 100-dim	0.8803
LSTM	sigmoid output	Word2Vec 50-dim	0.9068
LSTM	softmax output	GloVe 100-dim	0.9175
FF	ReLU, sigmoid, softmax	Word2Vec 50-dim	0.9183
FF	ReLU, ReLU, softmax	GloVe 50-dim	0.9486
LSTM	softmax output	Word2Vec 50-dim	0.9499
FF	ReLU, ReLU, softmax	Word2Vec 50-dim	0.9572
LSTM	softmax output	GloVe 50-dim	0.9586

Table 5: NNs accuracies, with various activation functions and input sizes. (FF = feedforward NN)

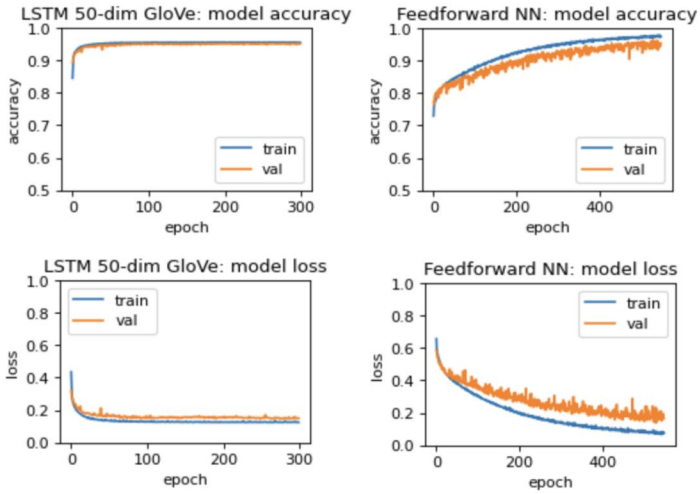


Figure 6: Loss and accuracy curves neural network models.

5.3.1 Comparison of Setups

Among the highest-performing models that we tested (Table 4), we found that lower-dimensional vectors consistently performed best, with high F1 scores (0.96 & 0.89 & 0.95). Both high- and low-dimensional vectors had high training accuracies throughout (94-97%), but higher-dimensional vectors had lower test set accuracy, demonstrating slight overfitting. Consistent with standard multiclass classifiers, we also found that the softmax activation function (and the ReLU function for the feedforward NN) was most suitable for ensuring that messages would have only one associated label [19].

Between word2vec and GloVe, we found that both performed well for both the feedforward and LSTM models, with the key difference that GloVe embeddings achieved 90% validation accuracy significantly sooner than word2vec embeddings. For LSTM, GloVe achieved 95% accuracy within 80-90 epochs while word2vec achieved 95% within 235 epochs. We attributed this to the GloVe embeddings being dependent on global context, with similar vector representations for certain words across the whole dataset leading to ease of classification with the same label, while word2vec's reliance on local context can result in subtle differences in word representations that may be classified differently across the data.

Comparing the feedforward NN and LSTM models, the best LSTM model converged on a 90% validation accuracy under 50 epochs (250 less epochs than feedforward). We attribute this to the recurrent NN structure of LSTM; the model is able to constantly learn from past phrases associated with certain classes, and hold familiar word representations in its memory cells. As a result, such references to words associated with derogatory messages, for example, were learned early on. In contrast, the feedforward model does not have available memory cells; its only reference is the previous iteration's learned weights, rather than specific words that often associate with an output class.

5.3.2 Discussion of DL vs. ML

Overall, the feedforward and LSTM models were slightly weaker than our ML classifiers (5.2). We attributed this to the vector representations of the input data; the pre-trained word2vec and GloVe embeddings for our NN classifiers rely on regular English words, while our ML classifiers utilized word frequencies from the slang-populated data itself, without pre-trained embeddings. In the context of gaming, this focus on word frequencies in the data resulted in more accurate classifications. Pre-trained embeddings may not view words like "bronze" and "feeder" as insults in gaming, or observe that "gg" (good game) frequents positive/neutral examples. This supports our finding that encodings such as TF-IDF (which were used in our ML classifiers and compute frequencies based on the dataset alone) would be more effective in identifying what words in games are more indicative of insults or positivity.

6 Conclusions and Future Work

Our work indicates that ML classifiers (particularly KNN and Random Forest) are able to adapt to "gamer slang" and accurately classify sentiment for League of Legends and DOTA 2 chats, with their own unique lexicons reflecting the competitive PvP genre. We achieved high (85-99%) overall accuracies, though F1 scores indicated that models tend to perform slightly better with classifying positive/neutral and negative attitude messages than derogatory language. We also observed higher accuracies with word frequency-based encodings rather than manually-constructed vocabularies or pre-trained embeddings, which is a potentially promising area of study as gaming lexicons continue to evolve with zeitgeist shifts. There is no guarantee that the games we study here will be applicable in the future, but the method of utilizing word frequencies for encoding can continue to be applied as new data emerges. Moreover, models such as convolutional neural networks, boosted decision trees, other deep learning architectures, and applications of voice chats and newer games could improve the detection of toxicity. Nevertheless, we hope that our analysis of ML algorithms applied to toxicity in PvP games can contribute to fostering more cooperative and less hostile environments in what ought to be a source of relief and entertainment rather than stress and harm.

7 Appendix

7.1 LSTM unit layers

1. Sigmoid layer: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$, where h_{t-1} is the output of the last LSTM unit, x_t is the current input, and W and b refer to the learned weights and bias term.

2. Tanh layer: updating the new cell state C_t (i_t is another sigmoid deciding which values to update):

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \Rightarrow \quad C_t = f_t * C_{t-1} + i_t * \tilde{C}_t.$$

3. Output h_t: run a sigmoid layer that decides which parts of the cell state to output, then put the cell state through tanh to push the values between -1 and 1 [17].

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \Rightarrow \quad h_t = o_t * \tanh(C_t).$$

8 Contributions

All members contributed to the project design, data labelling, and write-up. Melinda was the project lead and focused on the baselines, neural networks, and data pre-processing, while Thomas and Kevin focused on implementing the ML classifier methods and performing error analysis. We would also like to thank the CS 229 course staff for providing feedback throughout the process and running an engaging and informative class, without which this project would not have been possible.

References

[1] Blair, M. R., Leung, B., Taboada, M., & Thompson, J. J. (2017). Sentiment analysis of player chat messaging in the video game StarCraft 2: Extending a lexicon-based model. *Knowledge-Based Systems*, 137:149-162.

[2] Buchanan, W. J., Murnion, S., Russell, G., & Smales, A. (2018). Machine learning and semantic analysis of in-game chat for cyberbullying. *Computers & Security*, 76:197-213.

[3] Chen, W., Li, X., Yin, M., Yue, L., & Zuo, W. (2019). A survey of sentiment analysis in social media. *Knowledge and Information Systems*, 60:617-663.

[4] Arabnia, H. R., Balakrishnan, V., & Khan, S. (2020). Improving cyberbullying detection using Twitter users' psychological features and machine learning. *Computers & Security*, vol. 90.

[5] Chollet, F. (2017). *Deep Learning with Python*, pp. 79-80, 202-203. Manning Publications Co.

[6] Zimbler, A. (2020). Overwatch is Decreasing Toxicity in Chat With Machine Learning. Screen Rant. Retrieved from https://screenrant.com/overwatch-fighting-toxic-chat-machine-learning/.

[7] GOSU.AI Dota 2 (Valve) Game Chats: https://www.kaggle.com/romovpa/gosuai-dota-2-game-chats.

[8] League of Legends (Riot Games) Tribunal Chatlogs: https://www.kaggle.com/simshengxue/league-of-legends-tribunal-chatlogs.

[9] `sklearn.model_selection.train_test_split`: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[10] "What does tf-idf mean?" Retrieved from http://www.tfidf.com/.

[11] Manning, C. D., Pennington, J., & Socher, R. (2014). GloVe: Global Vectors for Word Representation. Retrieved from https://nlp.stanford.edu/projects/glove/.

[12] Word2vec: Tool for computing continuous distributed representations of words. (2013). Google Code Archive. Retrieved from https://code.google.com/archive/p/word2vec/.

[13] Ma, T. & Ng, A. (2020). CS229 Lecture Notes. Part VI: Support Vector Machines, pp.13-16.

[14] Ng, A. CS229 Lecture Notes. Part IV: Generative Learning Algorithms, pp. 9-12.

[15] Harrison, O. (2018). Machine Learning Basics with the K-Nearest Neighbors Algorithm. *Towards Data Science*. Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[16] CS 229 Lecture. May 14, 2021: Decision Trees.

[17] Almudever, C. G., Bertels, K., & Varsamopoulos, S. (2018). Designing neural network based decoders for surface codes. *ResearchGate*.

[18] Categorical crossentropy. *Peltarion*. Retrived from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

[19] Wood, T. Softmax Function. *DeepAI*. Retrieved from https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.