Multi Platform-based Hate Speech Detection

Shane Cooke
School of Computer Science
University College Dublin, Ireland
shane.cooke@ucdconnect.ie

Damien Graux
Inria, Université Côte d'Azur
CNRS, I3S
Sophia Antipolis, France
damien.graux@inria.fr

Soumyabrata Dev
ADAPT SFI Research Centre
School of Computer Science
University College Dublin, Ireland
soumyabrata.dev@ucd.ie

Abstract

A major issue faced by social media platforms today is the detection, and handling of hateful speech. The intricacies and imperfections of online communication make this a difficult task, and the rapidly changing use of both non-hateful, and hateful language in the online sphere means that researchers must constantly update and modify their hate speech detection methodologies. In this study, we propose an accurate and versatile multi-platform model for the detection of hate speech, using first-hand data scraped from some of the most popular social media platforms, that we share to the community. We explore and optimise 50 different model approaches, and evaluate their performances using several evaluation metrics. Overall, we successfully build a hate speech detection model, pairing the USE word embeddings with the SVC machine learning classifier, to obtain an average accuracy of 95.65% and achieved a maximum accuracy of 96.89%. We also develop and share an application allowing users to test sentences against a collection of the most accurate hate speech detection models. Our application then returns a aggregated hate speech classification, together with a confidence level, and a breakdown of the methodologies used to produce the final classification for explainability.

1 Introduction

The exact definition of hate speech is a topic of great discussion within society today. Philosophers, researchers and law-makers all have their own variations of the definition, however there are a set of facts upon which almost all parties agree on, the first being that the message is directed at an individual or group, and the second being that based on that message the group is viewed as negative, unwelcome or undesirable which warrants hostility towards them (Rudnicki and Steiger, 2020). In EU law, hate speech is defined as "the public incitement to violence or hatred on the basis of certain characteristics, including race, colour, religion, descent and national or ethnic origin" (Jourová, 2016).

Online hate speech however is a special case of hate speech that occurs in the online environment, making the perpetrators more anonymous, which in turn may make them feel less accountable, and as a result potentially more ruthless. To effectively fight online hate speech, non-government organisations aim to be more flexible than the justice system allow, in particular it is increasingly common to define hate speech much more broadly and include messages that do not explicitly incite violence only, but instead spread prejudice, stereotypes, biases and a general sense of ostracism. Hate speech online has been a major problem since the spread of the Web, however in light of the rapid rise in popularity of social media sites, this problem has since increased in size exponentially. For instance, Hawdon et al. (2015) found that approximately 53% of American, 48% of Finnish, 39% of British and 31% of German survey-respondents had been exposed to online hate material. These figures are worrying when considering the fact that online hate speech has risen almost exponentially in more recent years. Twitter for example, reported a 54% increase in the number of accounts actioned for violations of their hateful conduct policy in Q2 of 2019 (Twitter Inc., 2020). A study conducted by an AI-technology company, found that Twitter hate speech against China and the Chinese had increased 900% in early 2020, and found that a 70% increase in hate between kids and teens in online chatrooms had occurred in the same timeframe (L1GHT, 2020). TikTok removed 380 000 videos from their platform that violated its hate speech policy in August 2020 alone (Han, 2020) and Facebook reported a record 25 million instances of hate speech on its platform in Q1 of 2021 (Facebook, 2021).

In this study, we propose an accurate and versatile multi-platform model for the detection of hate

speech, using first-hand data scraped from some of the most popular social media platforms. Our contributions are threefold: First, we annotated manually a corpus of 3 000 comments from three social media platforms and share it to the community[1]. Second, we explore and optimise 50 different model approaches, and evaluate their performances using several evaluation metrics. Third, in addition, we also develop and share[2] an application allowing users to test sentences against a collection of the most accurate hate speech detection models to give the possibility to have finer results made from a combination of several models working together.

The rest of the article is structure as follows: in Section 2, we briefly remind the state-of-the-art in hate-speech detection. Then, we describe the data acquisition process in Section 3. Section 4 gives the details of the approach followed and Section 5 discusses the obtained results and performances. Section 6 presents the final application we developed. And finally, Sections 7 & 8 respectively mark the limitations of our method and draw our conclusions.

2 Related Work

Detecting hate speech online amongst the millions of posts every day is a difficult task and carries many associated challenges with it. Kovács et al. (2020) outlined some of these challenges. They reviewed over fifty works by other researchers in the field of hate detection online such as Davidson et al. (2017) or Bhattacharya and Weber (2019). Some of the main challenges identified in the form of key-word based search approaches. Another huge challenge in hate speech detection which spans all forms of search is the detection of context and nuance. Röttger et al. (2021), found that many hate speech detection models really struggled with "reclaimed slurs" and often mislabelled them as hateful[3]. In parallel, Sap et al. (2019) aimed to outline the major challenge of bias consideration.

Salminen et al. (2020) presents a multi-platform machine learning approach to online hate detection is proposed. They observed that most studies (*e.g.* (Kansara and Shekokar, 2015; Ramampiaro, 2018; Lee and Lee, 2018)) tend to focus on one platform, which they saw as problematic because "there are no guarantees the models that researchers develop generalize well across platforms".

There are many ways to evaluate the performance of hate speech detection models, however there is some level of disagreement as to which metrics are best suited for this problem. Mozafari (2020) remark that "In general, classifiers with higher precision and recall scores are preferred in classification tasks. However, due to the imbalanced classes in the hate speech detection datasets, we tend to make a trade-off between these two measures". For this reason they decided to use macro averaged F1-measures to summarize the performance of their models. On the other hand, Alshalan and Al-Khalifa (2020) decided to evaluate their models using precision, recall, F1-score, accuracy, hate class recall, and AUROC. Vigna et al. (2017) proposed that the best evaluation metrics to use are accuracy, precision, recall and F-score.

3 Data Acquisition

In order to create a versatile and well-rounded hate speech detection system, we decided to collect comment and post data from three different sources: Reddit, Twitter and 4Chan. The use of language, both hateful and non-hateful, can vary extremely between platforms, for this reason we believe the use of a multi-platform approach should help hate speech detection. Each of the three platforms boast varying levels and methods of moderation: with Reddit having community-based moderation, Twitter having automatic or employee-based moderation, and 4Chan having virtually zero moderation. Due to these highly differing methods of platform moderation, it is easy to pinpoint the subtleties of the hateful language used on each platform. Due to Reddit's community-based moderation, the hateful speech exhibited is often very subtle and very few slurs are used, while the automatic and employee-driven moderation used by Twitter promotes "leetspeak" and disguised slurs. These are both in sharp contrast to the language used in the unmoderated 4Chan forums, where extreme slurs are used regularly, and hateful speech is not only tolerated, but encouraged by some. This choice of three diverse data sources was ultimately made to ensure that the hate speech dataset curated for this project would be heterogenous, and would feature a wide array

[1]Links are hidden for double-blind policy.

[2]Links are hidden for double-blind policy.

[3]Their system is composed of 29 functional tests grouped into 11 classes, with 18 of these tests being for distinct expressions of hate and 11 tests being for contrastive non-hate (content that shares linguistic features with hateful expressions). Röttger et al. (2021) tested four models: (Davidson et al., 2017; Founta et al., 2018; Perspective, 2021; twohat, 2021).

of different forms of both non-hateful and hateful language.

Overall, we decided that a procured dataset of 3 000 posts and comments would be the best solution. The dataset exhibits an equal split of 1 000 posts from each of the three social media platforms, and each post is classified and labelled as either non-hateful ('0') or hateful ('1'). The posts and comments are split into classifications of 2 400 non-hateful posts (80%), and 600 hateful posts (20%).

4 General Approach

4.1 Word Embeddings

Word Embeddings are a class of techniques in which individual strings are mapped to vector or numerical representations. The chosen form of representation varies widely depending on the word embedding method being employed, however every method follows the same core principle of mapping a single string to a single defined value. In order to efficiently and accurately analyse and model the posts contained in our database, we used a variety of **five** different word embedding methods which all employ very different embedding methodologies.

First, we used *TFIDF* or "Term Frequency-Inverse Document Frequency". TFIDF is a machine learning algorithm based on a statistical measure of finding the relevance of words in a text. The "Term Frequency" or "TF", is calculated by dividing the number of occurrences of words by the total number of words in the text base. The "IDF" or "Inverse Document Frequency", is calculated by dividing the total number of comments by the number of comments containing the word. The overall TFIDF embedding is then equal to (TF) $\times$ (IDF).

Second, we consider *Doc2Vec* (Le and Mikolov, 2014) which is an NLP tool for representing documents as a vector, and is a generalisation of the "Word2Vec" model. The Word2Vec model maps words to their representative vector format, while the Doc2Vec model vectorises words to their representative format, and includes a paragraph numerical representation tied to these word vectors.

Third, we used a *Hashing Vectorizer* algorithm which converts a collection of text or strings into a matrix of token occurrences, where each token directly maps to a column position in a matrix where its size is predefined. The mapping happens using a process called "Hashing", and the hash function that is used is "Murmurhash3", which yields a 32-bit or 128-bit hash value.

Fourth, we exploited Google's *Universal Sentence Encoder* (Cer et al., 2018). USE encodes any given sentence into a 512-dimensional sentence embedding which is designed to work on multiple generic tasks. It will capture only the most informative features of any given sentence and discard noise, giving it the ability to transfer universally to a wide variety of NLP tasks such as hate speech detection.

Finally, we included also Google's (Bidirectional Encoder Representations from Transformers), namely *BERT* (Devlin et al., 2018) which uses a Transformer learning the contextual relations between words in a text.

4.2 Classifiers

We trial a diverse collection of **ten** machine learning classifiers. We ensure that within this group of classifiers there are both classical machine learning algorithms such as the Decision Tree classifier, and more modern, task-specific algorithms such as the XGBoost classifier. We also pay attention that there is a diverse array of methodologies used within this group of classifiers, which would ensure that a sufficiently wide array of approaches are tested. The selected classifiers are:

1. Random Forest Classifier (Pal, 2005);
2. Decision Tree(Safavian and Landgrebe, 1991);
3. Naive Bayes (Rish et al., 2001);
4. SVC (Vapnik, 1998);
5. AdaBoost (Freund and Schapire, 1997);
6. Gaussian Process (Gibbs, 1998);
7. K-Neighbours (Guo et al., 2003);
8. Multi-layer Perceptron (Hornik et al., 1989);
9. XGBoost (Chen and Guestrin, 2016);
10. Linear Discrimination (Izenman, 2013).

Once the word embedding methods are chosen and implemented, we begin to test all possible combinations of word embedding, machine learning classifier pairs. First, the classifiers are trained using the training data vectors produced by the word embedding process. Each one of these vectors has a corresponding "Hateful" value of either '0' or '1', which is the 'target' variable, or the variable which the classifier is aiming to predict. In the pursuit of the fairest evaluation possible, we run each classifier twenty times with a new train and test data

```
randomForest_grid = {'n_estimators':[200,400,600,800,1000], 'criterion':['gini', 'entropy']}
decisionTree_grid = {'max_depth':[2,4,6,8], 'splitter':['best', 'random'], 'criterion':['gini', 'entropy']}
svc_grid = {'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
adaboost_grid = {'n_estimators':[50,100,150,200], 'algorithm':['SAMME', 'SAMME.R']}
mlp_grid = {'max_iter':[500,1000,1500], 'activation':['identity', 'logistic', 'tanh', 'relu']}
linearDis_grid = {'solver':['svd', 'lsqr', 'eigen']}
```

Figure 1: Parameter dictionaries for the GridSearchCV algorithm.

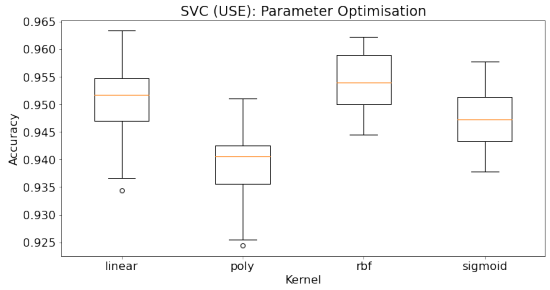

Figure 2: Parameter Optim. of the SVC classifier.

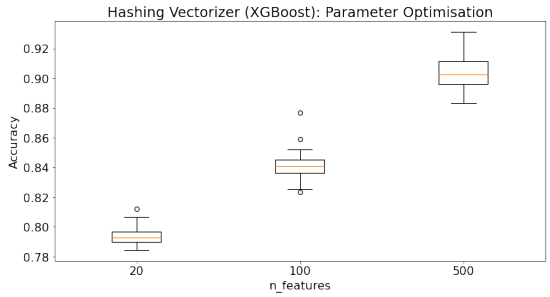

Figure 3: Parameter Optim. of the HV embeddings.

split for each iteration, and then take an average of each of the evaluation metrics across the twenty iterations and achieved a set of final results.

4.3 Optimisation Strategies...

4.3.1 ...for Classifiers

In order to optimise these results, the parameters or configuration variables of each classifier had to be tested and refined in pursuit of the highest possible results. While some classifiers do not take parameters such as the 'GaussianNB', 'GaussianProcess' and 'XGBoost' classifiers, the other classifiers can take upwards of eight parameters[4]. We thus implement exhaustive searches over specified parameter values, and implements fitting and scoring methods to evaluate each combination of parameters, as depicted on Figure 1.

Results The parameter optimisation of the machine learning classifiers had a majorly positive effect on the results produced by the classifiers across all evaluation metrics. While some algorithms do not take parameters such as 'GaussianNB', 'GaussianProcess' and 'XGBoost', the majority of algorithms do take parameter variables, and the overall optimisation process was extremely effective. For instance, Figure 2 shows an example of the results produced by the parameter optimisation for the SVC classifier. The highest performing kernel parameter ('rbf') has more than 1% higher accuracy than the lowest performing one ('poly').

4.3.2 ...for Word Embeddings

Similarly, we optimise the parameters of the word embedding methods. While some of the word embedding methods such as USE and BERT do not take parameters due to the fact that they are pre-trained algorithms, the TFIDF, Doc2Vec and Hashing Vectorizer methods take parameters. In pursuit of the most efficient word embedding parameter optimisation process possible, we created multiple different word embedding instances from the same word embedding method, each initialised with different parameters.

Results. The parameter optimisation of the word embedding methods was also successful, and had a much more pronounced and noticeable effect on the results produced by the models, across all evaluation metrics. While the pre-trained word embedding algorithms such as 'USE' and 'BERT' do not take parameters, the other ones do, and the overall optimisation process was extremely effective. For example, Figure 3 shows some examples of the results achieved by the parameter optimisation for the Hashing Vectorizer. When 500 'n_features' are selected as opposed to 20, there is more than a 10% increase in the accuracy produced by the word embedding method.

Unlike the parameter optimisation of the machine learning classifiers where changes in accuracy were often subtle, the word embedding parameter optimisation exhibited major improvements to the accuracy of the models. Increases in accuracy due to word embedding parameter optimisations were non-uniform and varied widely, however an increase in the range of +0.25% (TFIDF) and

[4]Practically, we used the "GridSearchCV" function from the "sci-kit learn" python library.

+10.5% (Hashing Vectorizer) was exhibited across all embeddings.

5 Experimental Validation

To evaluate and analyse the performance of the hate detection models, we relied on four main evaluation metrics: accuracy, precision, recall and F1-Score. The proportion of positive identifications that where actually correct (Precision) and the proportion of actual positives that where identified correctly (Recall) are major factors in determining the overall performance of a hate speech detection system, and F1-Score (a harmonic mean of both precision and recall) is also an extremely valuable metric when judging overall performance. Accuracy is used to determine the ability of the model to accurately identify patterns or relationships between the data in a dataset based on the training data that it has received.

5.1 Single Platform

The overall goal is to produce an efficient and effective multi-platform hate speech detection model, however it is important to analyse and evaluate the performance of each of the three individual single-platform models. To carry out this evaluation, we first split the full 3 000 comment dataset into three datasets of 1 000 comments, with each dataset only containing data from one specific platform. This resulted in each platform having its own dataset with 800 (80%) of comments being non-hateful, and 200 (20%) of those being hateful. Each dataset was then initialised used the same training to testing data split ratio of 0.3, and tested using the exact same methodology, classifiers and word embeddings. Each platforms data was then used to create and test fifty models (number of word embeddings times the number of classifiers). Once this testing had been completed and all evaluation metrics had been noted, we singled out[5] the top performing machine learning classifiers for each of the five word embedding methods, for each of the three platform datasets. The results of this process are shown in Figures 4, 5 & 6. We notice that there is a wide variety in the highest achieving machine learning classifier and word embedding pairs depending on what data the model had been trained and tested on. The Reddit datasets highest performing model was the USE word embeddings paired with the Naïve Bayes classifier, for the Twitter datasets it was the

[5]See Appendix A for additional details.

Reddit Dataset

Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / AdaBoost	0.912333	0: 0.92 1: 0.88	0: 0.93 1: 0.63	0: 0.95 1: 0.74
Doc2Vec / Linear Discrimination	0.809667	0: 0.84 1: 0.54	0: 0.94 1: 0.27	0: 0.89 1: 0.36
Hashing / Multi-Layer Perceptron	0.878000	0: 0.88 1: 0.87	0: 0.98 1: 0.46	0: 0.97 1: 0.86
USE / Naive Bayes	0.947333	0: 0.96 1: 0.90	0: 0.98 1: 0.83	0: 0.97 1: 0.86
BERT / Multi-Layer Perceptron	0.921667	0: 0.94 1: 0.84	0: 0.96 1: 0.75	0: 0.95 1: 0.79

Figure 4: Best ML classifier for each word embedding using the Reddit data.

Twitter Dataset

Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Decision Tree	0.987833	0: 0.99 1: 0.99	0: 1.00 1: 0.95	0: 0.99 1: 0.97
Doc2Vec / Linear Discrimination	0.865167	0: 0.88 1: 0.75	0: 0.96 1: 0.51	0: 0.92 1: 0.60
Hashing / XGBoost	0.947500	0: 0.96 1: 0.90	0: 0.98 1: 0.84	0: 0.97 1: 0.86
USE / SVC	0.960167	0: 0.95 1: 0.99	0: 1.00 1: 0.81	0: 0.98 1: 0.88
BERT / SVC	0.934333	0: 0.94 1: 0.93	0: 0.99 1: 0.73	0: 0.96 1: 0.81

Figure 5: Best ML classifier for each word embedding using the Twitter data.

4Chan Dataset

Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Random Forest	0.921167	0: 0.91 1: 0.98	0: 1.00 1: 0.63	0: 0.95 1: 0.77
Doc2Vec / Random Forest	0.795833	0: 0.80 1: 0.45	0: 0.99 1: 0.04	0: 0.89 1: 0.06
Hashing / XGBoost	0.894833	0: 0.90 1: 0.84	0: 0.97 1: 0.59	0: 0.94 1: 0.69
USE / Multi-Layer Perceptron	0.956833	0: 0.96 1: 0.97	0: 0.99 1: 0.80	0: 0.97 1: 0.88
BERT / SVC	0.934500	0: 0.93 1: 0.93	0: 0.99 1: 0.72	0: 0.96 1: 0.81

Figure 6: Best ML classifier for each word embedding using the 4Chan data.

TFIDF paired with the Decision Tree classifier, and with 4Chan datasets it was the USE paired with the Multi-Layer Perceptron. There is also diversity in the highest average accuracy achieved by each dataset, with Reddit achieving maximum of 94.73%, Twitter a maximum of 98.78% and 4Chan a maximum of 96.02%.

Due to the fact that the exact same methodologies, embeddings and classifiers where employed on each of the three datasets, this diversity in embedding and classifier pairs, and in the results achieved by these pairs can be explained by the

Word Embedding / ML Model	Accuracy	Precision	Recall	F1 Score
TFIDF / Random Forest	0.949444	0: 0.95 1: 0.95	0: 0.99 1: 0.79	0: 0.97 1: 0.86
Doc2Vec / Linear Discrimination	0.900778	0: 0.90 1: 0.90	0: 0.99 1: 0.56	0: 0.94 1: 0.69
Hashing / XGBoost	0.902378	0: 0.92 1: 0.84	0: 0.97 1: 0.66	0: 0.94 1: 0.73
USE / SVC	0.956500	0: 0.96 1: 0.96	0: 0.99 1: 0.82	0: 0.97 1: 0.88
BERT / SVC	0.933944	0: 0.94 1: 0.91	0: 0.98 1: 0.74	0: 0.96 1: 0.81

Figure 7: Best ML classifier for each of the word embedding methods using the multi-platform dataset.

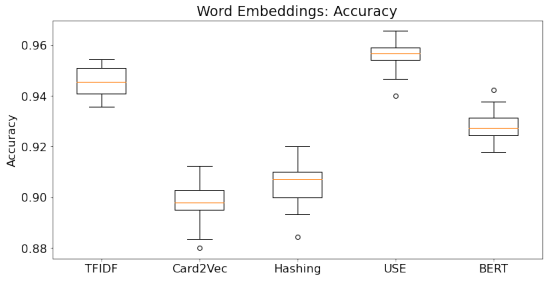

Figure 8: Accuracies achieved by the best machine learning classifier for each of the word embedding methods for the multi-platform dataset.

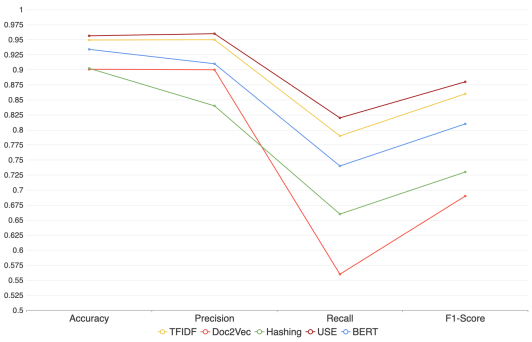

Figure 9: Evaluation metrics achieved by each of the best word embedding and classifier pairs when classifying data as hateful (X-Axis starts at 0.5).

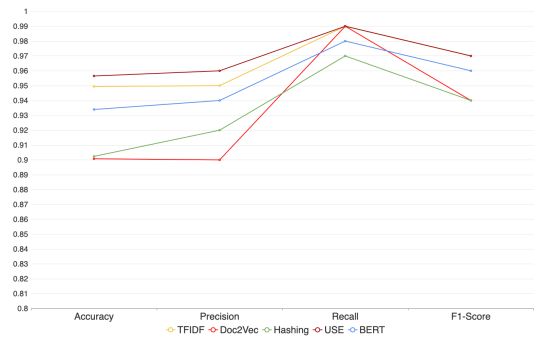

Figure 10: Evaluation metrics achieved by each of the best word embedding and classifier pairs when classifying data as non-hateful (X-Axis starts at 0.8).

differences in the collected data. As stated before, each of the three social media platforms has specific moderation methods. We believe that the diversity in results achieved by each single-platform model is largely due to the relative difficulty to detect the specific forms of hateful speech exhibited on that platform. Reddit's community based and strong moderation leads to "slurless" and subtle hateful language (e.g. "Get them all out of our country") which is difficult to detect and classify, where-as the automated and somewhat inadequate moderation on Twitter promotes the common use of typical hateful slurs (e.g. n*gger, f*ggot) which is much easier to detect and classify. 4Chan's no moderation policy leads to a diverse array of hateful speech, language and slurs (e.g. k*ke, n*gger, f*ggot, towelhead, mudskin), which ultimately makes it easier to detect and classify than the subtle Reddit hate speech, but harder to detect and classify than the repetitive Twitter hate speech.

For this reason it is extremely important in this day and age to produce multi-platform, versatile hate speech detection models that don't rely on the specific language used on a single social media platform.

5.2 Multi Platform

The core goal of our study is to produce a high-achieving hate speech detection model that could span multiple social media platforms and produce reliable and replicable results. To carry out a multi platform analysis, we use the full 3 000 comment dataset of combined platform data. The dataset was then split into training and testing data in a ratio of 0.3, and tested against the fifty word embedding and machine learning classifier pairs. The results of this process are in Figure 7.

The highest performing multi-platform hate speech detection model produced in this project was a combination of the Universal Sentence Encoder word embeddings paired with the Support Vector Machine (SVC) machine learning classifier, which achieved a peak average accuracy of 95.85%. The USE word embeddings achieving the highest accuracy result is relatively unsurprising due to the analysis carried out on the single-platform models, in which USE was identified as an extremely consistent and versatile word embedding method, regardless of the platform data. The box plot shown in Figure 8 shows that the pair (USE,SVC) exhibits the highest upper bound accuracy of all combina-

	TFIDF	Doc2Vec	Hashing	USE	BERT	Average
Random Forest	15.5070	6.9077	5.9888	18.0517	27.1622	14.7235
Decision Tree	0.4148	0.0747	0.0620	0.6194	0.9643	0.4270
Naive Bayes	0.3136	0.0101	0.0574	0.0570	0.0933	0.1063
SVC	9.6417	0.1189	1.1423	0.7407	0.9411	2.5170
AdaBoost	2.0077	0.4864	0.3037	3.8224	5.8252	2.4891
Gaussian Process	10.4660	4.1397	5.1421	5.7909	2.4921	5.6062
K Neighbours	0.5193	0.0549	0.1422	0.1472	0.1317	0.1991
Multi-Layer Perceptron	14.2263	0.9683	2.7452	3.4606	2.8556	4.8512
XGBoost	4.2010	0.4059	0.7215	1.3560	2.3949	1.8159
Linear Discrimination	5.6602	0.1172	0.4563	0.4749	0.4895	1.4396
Average	6.2958	1.3284	1.6762	3.4521	4.3350	

Figure 11: Comparative average run times of each model tested on the multi-platform dataset.

tions at a value of 96.89%, and also exhibits a lower variation in accuracy results when compared to all other embedding methods.

The Universal Sentence Encoder word embeddings combined with the SVC classifier exhibited a maximum average precision of 0.96, recall of 0.82 and F1-Score of 0.88 when classifying a comment as hateful. Each one of these individual four values where the highest evaluation metric results achieved by any model trained using the multi-platform dataset, as can be seen in Figure 9. It also exhibited a maximum average precision of 0.96, recall of 0.99 and F1-Score of 0.97 when classifying comments as non-hateful, which apart from recall where some models equalled the highest result, were also the highest evaluation metrics achieved by any model trained on the multi-platform dataset. These classification results for non-hateful comments can be seen in Figure 10.

5.3 Run Time & Efficiency

In the course of this study we measured both the single run time and the twenty run time average of the word embeddings and machine learning classifiers. The twenty runtime average for all classifiers and embeddings came out to exactly 20x the signle run time, so we decided to only focus on the single runtime metric. In order to fairly evaluate this metric, we calculated both the average runtime of each machine learning classifier across all word embeddings, and the average run time of each word embedding method across all classifiers and noted the results.

The classifier with the highest average run time by quite a large margin was the Random Forest Classifier (14.7235s), and the classifier with the lowest average run time was the Naïve Bayes classifier (0.1063s). The Gaussian Process and Multi-Layer Perceptron classifiers also had notably high average run times (5.6062s and 4.8512s respectively), while the K-Neighbours and Decision Tree classifiers had notably low average run times (0.1991s and 0.4270s respectively). Regarding the word embedding with the highest average run-time, it was TFIDF (6.2958s), with BERT having the second highest average run-time of 4.3350s. Doc2Vec was the fasted performing word embedding method with an average run-time of 1.3284s and the Hashing word embeddings also performed well with a 1.6762s average run time.

With the run-time of each model calculated and noted, we determine which models exhibited the highest levels of efficiency. Efficiency refers to the ability of a machine learning model to produce accurate results, while also exhibiting a very short relative run-time. Figure 11 contains all of the run-times of the machine learning models tested during this project. We then used this run-time table along with the table of accuracies produced by each model in order to come to a conclusion as to the most efficient models.

Ultimately, we determined that the three models circled in red in the above table exhibited the highest levels of efficiency out of all tested models. The Doc2Vec and Naïve Bayes model, the Doc2Vec and K-Neighbours model, and the USE and Naïve Bayes model all exhibited an average run time of less than one second (0.0101s, 0.0549s and 0.0570s respectively), and also all exhibited a notably high degree of accuracy when compared to other models. The Doc2Vec and Naïve Bayes model achieved an overall accuracy of 83.02%, the Doc2Vec and K-Neighbours model achieved an overall accuracy of 88.55%, and the USE and Naïve Bayes model achieved an overall accuracy of 93.61%. All three of these models exhibited fast run-times in comparison to other models, and also achieved an high accuracy in comparison to other models, which makes them the most highly efficient and economical models.

6 HateChecker Application

HateChecker is the application we developed using the “streamlit” python library for the purpose of testing some of the most accurate hate speech detec-

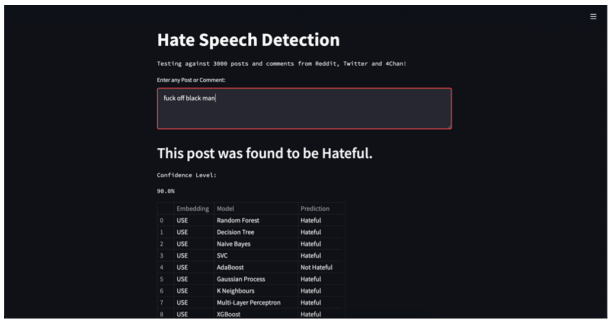

Figure 12: Output screenshot of our HateChecker.

tion models against a wide variety of different user inputted comments and posts. The HateChecker application takes input in the form of a comment or post like sequence of strings. Each of the twenty individual models selected will then use its own methodology to classify the user inputted comment as non-hateful ('0') or hateful ('1'), and an aggregation of the classifications produced by these twenty models is then calculated and returned as an overall classification with a confidence level percentage included. Practically, the models that we selected for use in the HateChecker application employ each of the ten classifiers paired with both the USE and BERT word embeddings. We selected USE and BERT word embeddings for the HateChecker application over the other word embeddings, because the models produced using these word embeddings exhibited an absolute minimum average of 81.00%, while other word embedding methods exhibited accuracies as low as 43.78%. To ensure that the results produced by the HateChecker application where to a sufficiently high standard, we decided to rule out the other word embedding methods.

The HateChecker application was ultimately created and designed so that individual models could be tested by carefully designed user inputted data which may have not occurred in either the training or testing data which the model was built on. An example of this would be using the HateChecker application to analyse the models ability to classify sequences of strings filled with punctuation, numbers and special characters. By calculating an overall classification, confidence level percentage, and displaying which individual models made which classifications, we could begin to test my models on a wide array of different sequences of strings allowing me to evaluate and analyse weaknesses and strengths within my models.

7 Limitations

The presented findings of this study rely on the first (manual-)step of annotating comments, thereby the consideration of additional platforms and more comments could refine our results, indeed, each time we added more comments to the database, the results achieved by the models across the board would increase by 0.5% - 1%. Similarly, considering a larger set of embeddings techniques and classifiers could lead to finer results and different explanations at the end of the process when HateChecker returns its confidence score. Regarding the application, technical strategies could be deployed to improve the performance of HateChecker as many intermediate results are not yet saved, leading to long loading times when opening the software. On the hate detection performances, advance annotations could be explored to detect subtle hate-sentences. Finally, the quality of the presented and shared set of annotations is bound in time as languages and usages evolve across time, with haters often finding new ways to convey their ideas.

8 Conclusion

In this study, we explore a strategy to detect hate-speech. We based our approach on considering messages from several online social-media platforms at once, betting that their different internal moderation policies would provide a larger set of haters' methods. In additions to sharing[6] our annotated set with the community, we also develop an application building up our strategy of combining/comparing multiple pairs of word embeddings and classifiers. Overall, we successfully build a hate speech detection model, pairing USE and SVC, to obtain an average accuracy of 95.65% and achieved a maximum accuracy of 96.89%. Moreover, our application allows to define an aggregating strategy by *e.g.* choosing which pairs should be taken more into account. Therefore, we hope that this two-side strategy of involving several platforms and combining multiple pairs of embeddings and classifiers, will inspire the community to improve our results and refine our performance score.

[6]Links are hidden to respect anonymity.

Ethical Considerations

Sensitive Content Warning – Due to the nature of this study, there are some points in this article where hateful language and terms are used. While we did try and keep the use of these terms and phrases to a minimum, and while we obviously do not approve these message, at some points it was vital to provide the reader with a proper understanding of the context and methodologies used in the process of completing this project.

References

Raghad Alshalan and Hend Al-Khalifa. 2020. A deep learning approach for automatic hate speech detection in the saudi twittersphere. *MDPI*.

Debsamita Bhattacharya and Ingmar Weber. 2019. Racial bias in hate speech and abusive language detection datasets. *ACL Anthology*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Facebook. 2021. Hate speech.

Antigioni-Maria Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, and Jeremy Blackburn. 2018. Large scale crowdsourcing and characterization of twitter abusive behavior.

Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Mark N Gibbs. 1998. *Bayesian Gaussian processes for regression and classification*. Ph.D. thesis, Citeseer.

Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. 2003. Knn model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 986–996. Springer.

Eric Han. 2020. Countering hate on tiktok.

James Hawdon, Atte Oksanen, and Pekka Räsänen. 2015. Online extremism and online hate exposure among adolescents and young adults in four nations.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Alan Julian Izenman. 2013. Linear discriminant analysis. In *Modern multivariate statistical techniques*, pages 237–280. Springer.

Vêra Jourová. 2016. Code of conduct - illegal online hate speech questions and answers.

Krishna Kansara and N Shekokar. 2015. A framework for cyberbullying detection in social network. *Semantic Scholar*.

György Kovács, Pedro Alonso, and Rajkumar Saini. 2020. Challenges of hate speech detection in social media. *Springer Nature*.

L1GHT. 2020. Rising levels of hate speech & online toxicity during this time of crisis.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Ho Suk Lee and Hong Rae Lee. 2018. An abusive text detection system based on enhanced abusive and non-abusive word lists. *Yonsei University*.

Marzieh Mozafari. 2020. Hate speech detection and racial bias mitigation in social media based on bert model. *PLOS ONE*.

Mahesh Pal. 2005. Random forest classifier for remote sensing classification. *International journal of remote sensing*, 26(1):217–222.

Perspective. 2021. Using machine learning to reduce toxicity online.

Heri Ramampiaro. 2018. Detecting offensive language in tweets using deep learning. *Cornell University*.

Irina Rish et al. 2001. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3-22, pages 41–46.

Konrad Rudnicki and Stefan Steiger. 2020. Online hate speech.

Paul Röttger, Bertram Vidgen, Dong Nguyen, Zeerak Waseem, Helen Margetts, and Janet Pierrehumbert. 2021. Hatecheck: Functional tests for hate speech detection models.

S Rasoul Safavian and David Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Joni Salminen, Maximilian Hopf, Shammur Chowdhury, Soon gyo Jung, Hind Almerekhi, and Bernard Jansen. 2020. Developing an online hate classifier for multiple social media platforms. *Human-centric Computing and Information Sciences*.

Maarten Sap, Dallas Card, Saadia Gabriel, Yejin Choi, and Noah Smith. 2019. The risk of racial bias in hate speech detection.

Twitter Inc. 2020. Introducing the new twitter transperency center. *Twitter*.

twohat. 2021. Introducing... sift ninja!

Vladimir Vapnik. 1998. Statistical learning theory new york. *NY: Wiley*, 1(2):3.

Fabio Del Vigna, Andrea Cimino, and Marinella Petrocchi. 2017. Hate me, hate me not: Hate speech detection on facebook. *First Italian Conference on Cybersecurity*.

A Results Full-Tables

A = Accuracy, P = Precision, R = Recall, F1 = F1-Score.
Values coloured red received ill-defined or 0.0 values in all twenty simulations for precision, recall and F1-Score.
Values coloured orange received some ill-defined or 0.0 values in the twenty simulations for precision, recall and F1-Score.
Values coloured green exhibited the best results for the word embedding method.

1000 Comment Reddit Dataset Results (20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.901667 P = (0: 0.89, 1: 0.97) R = (0: 1.00, 1: 0.53) F1 = (0: 0.94, 1: 0.68)	A = 0.793667 P = (0: 0.80, 1: 0.52) R = (0: 0.99, 1: 0.02) F1 = (0: 0.88, 1: 0.04)	A = 0.879000 P = (0: 0.89, 1: 0.82) R = (0: 0.97, 1: 0.51) F1 = (0: 0.93, 1: 0.63)	A = 0.894167 P = (0: 0.88, 1: 0.99) R = (0: 1.00, 1: 0.47) F1 = (0: 0.94, 1: 0.63)	A = 0.887500 P = (0: 0.88, 1: 0.97) R = (0: 1.00, 1: 0.46) F1 = (0: 0.93, 1: 0.62)
Decision Tree	A = 0.874167 P = (0: 0.88, 1: 0.86) R = (0: 0.98, 1: 0.44) F1 = (0: 0.93, 1: 0.58)	A = 0.738000 P = (0: 0.80, 1: 0.26) R = (0: 0.89, 1: 0.15) F1 = (0: 0.84, 1: 0.18)	A = 0.861333 P = (0: 0.88, 1: 0.75) R = (0: 0.96, 1: 0.45) F1 = (0: 0.92, 1: 0.56)	A = 0.850500 P = (0: 0.91, 1: 0.63) R = (0: 0.91, 1: 0.64) F1 = (0: 0.91, 1: 0.63)	A = 0.816500 P = (0: 0.89, 1: 0.54) R = (0: 0.89, 1: 0.53) F1 = (0: 0.89, 1: 0.53)
Naive Bayes	A = 0.756000 P = (0: 0.91, 1: 0.42) R = (0: 0.78, 1: 0.68) F1 = (0: 0.84, 1: 0.52)	A = 0.604333 P = (0: 0.79, 1: 0.23) R = (0: 0.69, 1: 0.28) F1 = (0: 0.72, 1: 0.20)	A = 0.620667 P = (0: 0.84, 1: 0.27) R = (0: 0.65, 1: 0.52) F1 = (0: 0.73, 1: 0.35)	A = 0.947333 P = (0: 0.96, 1: 0.90) R = (0: 0.98, 1: 0.83) F1 = (0: 0.97, 1: 0.86)	A = 0.821333 P = (0: 0.94, 1: 0.54) R = (0: 0.83, 1: 0.78) F1 = (0: 0.88, 1: 0.63)
SVC	A = 0.844167 P = (0: 0.84, 1: 0.98) R = (0: 1.00, 1: 0.25) F1 = (0: 0.91, 1: 0.40)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.845000 P = (0: 0.84, 1: 0.91) R = (0: 0.99, 1: 0.27) F1 = (0: 0.91, 1: 0.41)	A = 0.944167 P = (0: 0.94, 1: 0.98) R = (0: 1.00, 1: 0.74) F1 = (0: 0.97, 1: 0.84)	A = 0.919500 P = (0: 0.91, 1: 0.97) R = (0: 1.00, 1: 0.61) F1 = (0: 0.95, 1: 0.75)
AdaBoost	A = 0.912333 P = (0: 0.92, 1: 0.88) R = (0: 0.98, 1: 0.63) F1 = (0: 0.95, 1: 0.74)	A = 0.774500 P = (0: 0.80, 1: 0.30) R = (0: 0.96, 1: 0.06) F1 = (0: 0.87, 1: 0.10)	A = 0.855833 P = (0: 0.88, 1: 0.73) R = (0: 0.95, 1: 0.51) F1 = (0: 0.91, 1: 0.60)	A = 0.922000 P = (0: 0.94, 1: 0.85) R = (0: 0.97, 1: 0.73) F1 = (0: 0.95, 1: 0.79)	A = 0.886667 P = (0: 0.91, 1: 0.76) R = (0: 0.95, 1: 0.64) F1 = (0: 0.93, 1: 0.69)
Gaussian Process	A = 0.797333 P = (0: 0.80, 1: 0.97) R = (0: 1.00, 1: 0.02) F1 = (0: 0.89, 1: 0.05)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.807167 P = (0: 0.81, 1: 0.97) R = (0: 1.0, 1: 0.05) F1 = (0: 0.89, 1: 0.10)	A = 0.887833 P = (0: 0.88, 1: 1.00) R = (0: 1.00, 1: 0.45) F1 = (0: 0.93, 1: 0.62)	A = 0.895500 P = (0: 0.93, 1: 0.74) R = (0: 0.94, 1: 0.71) F1 = (0: 0.94, 1: 0.73)
K Neighbours	A = 0.816333 P = (0: 0.82, 1: 0.96) R = (0: 1.00, 1: 0.15) F1 = (0: 0.90, 1: 0.24)	A = 0.778833 P = (0: 0.80, 1: 0.26) R = (0: 0.96, 1: 0.05) F1 = (0: 0.87, 1: 0.08)	A = 0.811167 P = (0: 0.81, 1: 0.85) R = (0: 0.99, 1: 0.11) F1 = (0: 0.89, 1: 0.17)	A = 0.934333 P = (0: 0.97, 1: 0.82) R = (0: 0.95, 1: 0.86) F1 = (0: 0.96, 1: 0.84)	A = 0.907500 P = (0: 0.92, 1: 0.83) R = (0: 0.97, 1: 0.65) F1 = (0: 0.94, 1: 0.73)
Multi-Layer Perceptron	A = 0.881833 P = (0: 0.87, 1: 0.97) R = (0: 1.00, 1: 0.43) F1 = (0: 0.93, 1: 0.59)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.878000 P = (0: 0.88, 1: 0.87) R = (0: 0.98, 1: 0.46) F1 = (0: 0.93, 1: 0.60)	A = 0.943500 P = (0: 0.94, 1: 0.95) R = (0: 0.99, 1: 0.76) F1 = (0: 0.97, 1: 0.84)	A = 0.921667 P = (0: 0.94, 1: 0.84) R = (0: 0.96, 1: 0.75) F1 = (0: 0.95, 1: 0.79)
XGBoost	A = 0.877000 P = (0: 0.88, 1: 0.83) R = (0: 0.97, 1: 0.50) F1 = (0: 0.93, 1: 0.62)	A = 0.777500 P = (0: 0.81, 1: 0.32) R = (0: 0.95, 1: 0.10) F1 = (0: 0.87, 1: 0.15)	A = 0.870167 P = (0: 0.89, 1: 0.77) R = (0: 0.96, 1: 0.50) F1 = (0: 0.92, 1: 0.60)	A = 0.926833 P = (0: 0.93, 1: 0.92) R = (0: 0.99, 1: 0.69) F1 = (0: 0.96, 1: 0.79)	A = 0.905333 P = (0: 0.91, 1: 0.85) R = (0: 0.97, 1: 0.62) F1 = (0: 0.94, 1: 0.71)
Linear Discrimination	A = 0.580500 P = (0: 0.89, 1: 0.33) R = (0: 0.54, 1: 0.72) F1 = (0: 0.66, 1: 0.42)	A = 0.809667 P = (0: 0.84, 1: 0.54) R = (0: 0.94, 1: 0.27) F1 = (0: 0.89, 1: 0.36)	A = 0.772500 P = (0: 0.89, 1: 0.45) R = (0: 0.81, 1: 0.61) F1 = (0: 0.85, 1: 0.52)	A = 0.843500 P = (0: 0.94, 1: 0.59) R = (0: 0.86, 1: 0.77) F1 = (0: 0.90, 1: 0.66)	A = 0.709500 P = (0: 0.89, 1: 0.38) R = (0: 0.72, 1: 0.65) F1 = (0: 0.80, 1: 0.48)

1000 Comment Twitter Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.983000 P = (0: 0.98, 1: 1.00) R = (0: 1.00, 1: 0.92) F1 = (0: 0.99, 1: 0.96)	A = 0.814333 P = (0: 0.82, 1: 0.74) R = (0: 0.99, 1: 0.14) F1 = (0: 0.89, 1: 0.23)	A = 0.951833 P = (0: 0.97, 1: 0.90) R = (0: 0.97, 1: 0.87) F1 = (0: 0.97, 1: 0.88)	A = 0.908500 P = (0: 0.90, 1: 0.99) R = (0: 1.00, 1: 0.55) F1 = (0: 0.95, 1: 0.70)	A = 0.885500 P = (0: 0.88, 1: 0.90) R = (0: 0.99, 1: 0.49) F1 = (0: 0.93, 1: 0.63)
Decision Tree	A = 0.987833 P = (0: 0.99, 1: 0.99) R = (0: 1.00, 1: 0.95) F1 = (0: 0.99, 1: 0.97)	A = 0.748000 P = (0: 0.83, 1: 0.35) R = (0: 0.85, 1: 0.32) F1 = (0: 0.84, 1: 0.33)	A = 0.946833 P = (0: 0.97, 1: 0.86) R = (0: 0.97, 1: 0.87) F1 = (0: 0.97, 1: 0.87)	A = 0.868833 P = (0: 0.92, 1: 0.65) R = (0: 0.91, 1: 0.68) F1 = (0: 0.92, 1: 0.66)	A = 0.829500 P = (0: 0.90, 1: 0.57) R = (0: 0.89, 1: 0.58) F1 = (0: 0.89, 1: 0.58)
Naive Bayes	A = 0.765167 P = (0: 0.87, 1: 0.42) R = (0: 0.83, 1: 0.50) F1 = (0: 0.85, 1: 0.45)	A = 0.542000 P = (0: 0.87, 1: 0.25) R = (0: 0.51, 1: 0.68) F1 = (0: 0.64, 1: 0.37)	A = 0.715167 P = (0: 0.85, 1: 0.34) R = (0: 0.78, 1: 0.47) F1 = (0: 0.81, 1: 0.39)	A = 0.947167 P = (0: 0.96, 1: 0.91) R = (0: 0.98, 1: 0.82) F1 = (0: 0.97, 1: 0.86)	A = 0.825667 P = (0: 0.95, 1: 0.55) R = (0: 0.82, 1: 0.84) F1 = (0: 0.88, 1: 0.66)
SVC	A = 0.879500 P = (0: 0.87, 1: 1.00) R = (0: 1.00, 1: 0.40) F1 = (0: 0.93, 1: 0.57)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.909500 P = (0: 0.90, 1: 0.98) R = (0: 1.00, 1: 0.56) F1 = (0: 0.95, 1: 0.71)	A = 0.960167 P = (0: 0.95, 1: 0.99) R = (0: 1.00, 1: 0.81) F1 = (0: 0.98, 1: 0.89)	A = 0.934333 P = (0: 0.94, 1: 0.93) R = (0: 0.99, 1: 0.73) F1 = (0: 0.96, 1: 0.81)
AdaBoost	A = 0.984833 P = (0: 0.99, 1: 0.97) R = (0: 0.99, 1: 0.95) F1 = (0: 0.99, 1: 0.96)	A = 0.792000 P = (0: 0.83, 1: 0.46) R = (0: 0.94, 1: 0.21) F1 = (0: 0.88, 1: 0.29)	A = 0.936333 P = (0: 0.96, 1: 0.86) R = (0: 0.97, 1: 0.82) F1 = (0: 0.96, 1: 0.83)	A = 0.939833 P = (0: 0.95, 1: 0.88) R = (0: 0.97, 1: 0.79) F1 = (0: 0.96, 1: 0.83)	A = 0.897833 P = (0: 0.93, 1: 0.77) R = (0: 0.95, 1: 0.70) F1 = (0: 0.94, 1: 0.73)
Gaussian Process	A = 0.820833 P = (0: 0.82, 1: 1.00) R = (0: 1.00, 1: 0.09) F1 = (0: 0.90, 1: 0.16)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.827000 P = (0: 0.82, 1: 1.00) R = (0: 1.00, 1: 0.16) F1 = (0: 0.90, 1: 0.28)	A = 0.923167 P = (0: 0.91, 1: 1.00) R = (0: 1.00, 1: 0.62) F1 = (0: 0.95, 1: 0.76)	A = 0.854333 P = (0: 0.94, 1: 0.62) R = (0: 0.88, 1: 0.76) F1 = (0: 0.91, 1: 0.68)
K Neighbours	A = 0.923167 P = (0: 0.93, 1: 0.87) R = (0: 0.97, 1: 0.72) F1 = (0: 0.95, 1: 0.79)	A = 0.787333 P = (0: 0.81, 1: 0.43) R = (0: 0.95, 1: 0.14) F1 = (0: 0.88, 1: 0.21)	A = 0.910833 P = (0: 0.92, 1: 0.86) R = (0: 0.97, 1: 0.67) F1 = (0: 0.95, 1: 0.75)	A = 0.921000 P = (0: 0.99, 1: 0.73) R = (0: 0.91, 1: 0.97) F1 = (0: 0.95, 1: 0.83)	A = 0.900000 P = (0: 0.94, 1: 0.73) R = (0: 0.93, 1: 0.76) F1 = (0: 0.94, 1: 0.74)
Multi-Layer Perceptron	A = 0.911167 P = (0: 0.90, 1: 1.00) R = (0: 1.00, 1: 0.56) F1 = (0: 0.95, 1: 0.71)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.922167 P = (0: 0.92, 1: 0.95) R = (0: 0.99, 1: 0.66) F1 = (0: 0.95, 1: 0.78)	A = 0.957833 P = (0: 0.96, 1: 0.97) R = (0: 0.99, 1: 0.82) F1 = (0: 0.97, 1: 0.89)	A = 0.932000 P = (0: 0.95, 1: 0.87) R = (0: 0.97, 1: 0.80) F1 = (0: 0.96, 1: 0.83)
XGBoost	A = 0.979000 P = (0: 0.98, 1: 0.99) R = (0: 1.00, 1: 0.90) F1 = (0: 0.99, 1: 0.94)	A = 0.812500 P = (0: 0.84, 1: 0.60) R = (0: 0.95, 1: 0.28) F1 = (0: 0.89, 1: 0.37)	A = 0.947500 P = (0: 0.96, 1: 0.90) R = (0: 0.98, 1: 0.84) F1 = (0: 0.97, 1: 0.86)	A = 0.943167 P = (0: 0.95, 1: 0.93) R = (0: 0.99, 1: 0.77) F1 = (0: 0.97, 1: 0.84)	A = 0.914833 P = (0: 0.92, 1: 0.86) R = (0: 0.97, 1: 0.68) F1 = (0: 0.95, 1: 0.76)
Linear Discrimination	A = 0.784167 P = (0: 0.90, 1: 0.70) R = (0: 0.82, 1: 0.64) F1 = (0: 0.84, 1: 0.60)	A = 0.865167 P = (0: 0.88, 1: 0.75) R = (0: 0.96, 1: 0.51) F1 = (0: 0.92, 1: 0.60)	A = 0.812167 P = (0: 0.91, 1: 0.52) R = (0: 0.85, 1: 0.67) F1 = (0: 0.88, 1: 0.59)	A = 0.863000 P = (0: 0.95, 1: 0.61) R = (0: 0.87, 1: 0.82) F1 = (0: 0.91, 1: 0.70)	A = 0.730500 P = (0: 0.91, 1: 0.39) R = (0: 0.74, 1: 0.69) F1 = (0: 0.81, 1: 0.50)

1000 Comment 4Chan Dataset Results
(20 simulation average)

	TFIDF	Doc2Vec	Hashing	USE	BERT
Random Forest	A = 0.921167 P = (0: 0.91, 1: 0.98) R = (0: 1.00, 1: 0.63) F1 = (0: 0.95, 1: 0.77)	A = 0.795833 P = (0: 0.80, 1: 0.45) R = (0: 0.99, 1: 0.04) F1 = (0: 0.89, 1: 0.06)	A = 0.896167 P = (0: 0.90, 1: 0.90) R = (0: 0.98, 1: 0.56) F1 = (0: 0.94, 1: 0.69)	A = 0.921833 P = (0: 0.91, 1: 1.00) R = (0: 1.00, 1: 0.61) F1 = (0: 0.95, 1: 0.76)	A = 0.903833 P = (0: 0.90, 1: 0.93) R = (0: 0.99, 1: 0.57) F1 = (0: 0.94, 1: 0.70)
Decision Tree	A = 0.917500 P = (0: 0.91, 1: 0.97) R = (0: 0.99, 1: 0.62) F1 = (0: 0.95, 1: 0.76)	A = 0.721333 P = (0: 0.81, 1: 0.24) R = (0: 0.86, 1: 0.18) F1 = (0: 0.83, 1: 0.20)	A = 0.893167 P = (0: 0.9, 1: 0.85) R = (0: 0.97, 1: 0.57) F1 = (0: 0.94, 1: 0.68)	A = 0.860667 P = (0: 0.91, 1: 0.65) R = (0: 0.91, 1: 0.66) F1 = (0: 0.91, 1: 0.65)	A = 0.840500 P = (0: 0.90, 1: 0.59) R = (0: 0.90, 1: 0.59) F1 = (0: 0.90, 1: 0.59)
Naive Bayes	A = 0.675500 P = (0: 0.88, 1: 0.32) R = (0: 0.69, 1: 0.61) F1 = (0: 0.77, 1: 0.42)	A = 0.675667 P = (0: 0.83, 1: 0.28) R = (0: 0.74, 1: 0.41) F1 = (0: 0.79, 1: 0.33)	A = 0.514833 P = (0: 0.82, 1: 0.20) R = (0: 0.52, 1: 0.51) F1 = (0: 0.63, 1: 0.29)	A = 0.948667 P = (0: 0.96, 1: 0.91) R = (0: 0.98, 1: 0.82) F1 = (0: 0.97, 1: 0.86)	A = 0.851667 P = (0: 0.96, 1: 0.59) R = (0: 0.85, 1: 0.87) F1 = (0: 0.90, 1: 0.70)
SVC	A = 0.838833 P = (0: 0.83, 1: 0.99) R = (0: 1.00, 1: 0.21) F1 = (0: 0.91, 1: 0.35)	A = P = (0: , 1:) R = (0: , 1:) F1 = (0: , 1:)	A = 0.863833 P = (0: 0.86, 1: 0.93) R = (0: 0.99, 1: 0.34) F1 = (0: 0.92, 1: 0.49)	A = 0.943833 P = (0: 0.94, 1: 0.98) R = (0: 1.00, 1: 0.74) F1 = (0: 0.97, 1: 0.84)	A = 0.934500 P = (0: 0.93, 1: 0.93) R = (0: 0.99, 1: 0.72) F1 = (0: 0.96, 1: 0.81)
AdaBoost	A = 0.916167 P = (0: 0.92, 1: 0.90) R = (0: 0.98, 1: 0.67) F1 = (0: 0.95, 1: 0.76)	A = 0.777667 P = (0: 0.80, 1: 0.29) R = (0: 0.96, 1: 0.07) F1 = (0: 0.87, 1: 0.11)	A = 0.891167 P = (0: 0.91, 1: 0.79) R = (0: 0.96, 1: 0.62) F1 = (0: 0.93, 1: 0.69)	A = 0.933333 P = (0: 0.95, 1: 0.88) R = (0: 0.97, 1: 0.77) F1 = (0: 0.96, 1: 0.82)	A = 0.909833 P = (0: 0.93, 1: 0.80) R = (0: 0.96, 1: 0.70) F1 = (0: 0.94, 1: 0.75)
Gaussian Process	A = 0.809500 P = (0: 0.81, 1: 1.00) R = (0: 1.00, 1: 0.02) F1 = (0: 0.90, 1: 0.05)	A = 0.797833 P = (0: 0.81, 1: 0.69) R = (0: 0.99, 1: 0.02) F1 = (0: 0.89, 1: 0.04)	A = 0.809000 P = (0: 0.81, 1: 0.99) R = (0: 1.00, 1: 0.07) F1 = (0: 0.89, 1: 0.13)	A = 0.925000 P = (0: 0.92, 1: 1.00) R = (0: 1.00, 1: 0.60) F1 = (0: 0.96, 1: 0.75)	A = 0.872500 P = (0: 0.94, 1: 0.66) R = (0: 0.90, 1: 0.77) F1 = (0: 0.92, 1: 0.71)
K Neighbours	A = 0.801167 P = (0: 0.83, 1: 0.59) R = (0: 0.94, 1: 0.25) F1 = (0: 0.88, 1: 0.32)	A = 0.772833 P = (0: 0.81, 1: 0.26) R = (0: 0.94, 1: 0.09) F1 = (0: 0.87, 1: 0.13)	A = 0.804000 P = (0: 0.82, 1: 0.79) R = (0: 0.99, 1: 0.11) F1 = (0: 0.90, 1: 0.17)	A = 0.939333 P = (0: 0.98, 1: 0.82) R = (0: 0.95, 1: 0.90) F1 = (0: 0.96, 1: 0.85)	A = 0.897333 P = (0: 0.93, 1: 0.76) R = (0: 0.94, 1: 0.73) F1 = (0: 0.94, 1: 0.74)
Multi-Layer Perceptron	A = 0.879833 P = (0: 0.87, 1: 0.98) R = (0: 1.00, 1: 0.39) F1 = (0: 0.93, 1: 0.55)	A = 0.797167 P = (0: 0.81, 1: 1.00) R = (0: 1.00, 1: 0.02) F1 = (0: 0.89, 1: 0.03)	A = 0.877667 P = (0: 0.88, 1: 0.89) R = (0: 0.98, 1: 0.46) F1 = (0: 0.93, 1: 0.60)	A = 0.956833 P = (0: 0.96, 1: 0.97) R = (0: 0.99, 1: 0.80) F1 = (0: 0.97, 1: 0.88)	A = 0.934500 P = (0: 0.95, 1: 0.87) R = (0: 0.97, 1: 0.79) F1 = (0: 0.96, 1: 0.83)
XGBoost	A = 0.912333 P = (0: 0.91, 1: 0.91) R = (0: 0.98, 1: 0.63) F1 = (0: 0.95, 1: 0.74)	A = 0.775000 P = (0: 0.81, 1: 0.30) R = (0: 0.94, 1: 0.10) F1 = (0: 0.87, 1: 0.15)	A = 0.894833 P = (0: 0.90, 1: 0.84) R = (0: 0.97, 1: 0.59) F1 = (0: 0.94, 1: 0.69)	A = 0.946167 P = (0: 0.94, 1: 0.96) R = (0: 0.99, 1: 0.77) F1 = (0: 0.97, 1: 0.85)	A = 0.925833 P = (0: 0.93, 1: 0.90) R = (0: 0.98, 1: 0.70) F1 = (0: 0.96, 1: 0.79)
Linear Discrimination	A = 0.581500 P = (0: 0.86, 1: 0.29) R = (0: 0.56, 1: 0.66) F1 = (0: 0.67, 1: 0.40)	A = 0.776500 P = (0: 0.82, 1: 0.38) R = (0: 0.93, 1: 0.16) F1 = (0: 0.87, 1: 0.23)	A = 0.778833 P = (0: 0.89, 1: 0.46) R = (0: 0.82, 1: 0.60) F1 = (0: 0.86, 1: 0.52)	A = 0.845333 P = (0: 0.94, 1: 0.59) R = (0: 0.87, 1: 0.77) F1 = (0: 0.90, 1: 0.67)	A = 0.728833 P = (0: 0.90, 1: 0.40) R = (0: 0.75, 1: 0.65) F1 = (0: 0.81, 1: 0.49)