



**DUBLIN INSTITUTE
of TECHNOLOGY**
Institiúid Teicneolaíochta Bhaile Átha Cliath

Anti-Bullying with Machine Learning

Final Year Project Report

DT228

BSc in Computer Science

Shane O'Neill

Dr. Susan McKeever

School of Computing

Dublin Institute of Technology

05/04/2017

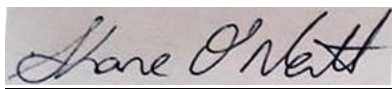
Abstract

In this project, various combinations of machine learning and text mining processes were used to build predictive models to detect cyberbullying or abusive content. The resulting class accuracies of said combinations were recorded and analysed. These methods were defined based on previous research and tested on 4 different datasets labelled for bullying or abusive content. The results show that data quality is key in this area of research. From the given datasets, there are no combinations of methods and processes that encapsulate all social media sources. They also indicate that term frequency normalisation should not be applied to this type of content sensitive data. Future research into classification of cyberbullying and abusive content should bear these findings in mind.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A handwritten signature in black ink, reading "Shane O'Neill", is displayed on a light-colored rectangular background.

Shane O'Neill

05/04/2017

Acknowledgements

I would like to acknowledge and thank my project supervisor Dr. Susan McKeever for her guidance, encouragement and helpful chats throughout the year.

I would like to thank my parents and brother for their continued, unwavering support. I would also like to acknowledge my friends, particularly Paul, Aidan, Caitríona and the college gang. Very special thanks go to Lauren, without whom I could not have made it to this stage.

I would like to sincerely thank the staff of Antionette's Bakery and the Blackbird for their constant, friendly provision of necessary supplies. I would also like to thank the DIT School of Computing staff for the last four years that I have enjoyed as part of DT228 Computer Science.

Table of Contents

Abstract	2
Declaration	3
Acknowledgements	4
1. Introduction	7
1.1. Overview	7
1.2 Objectives.....	7
1.3 Challenges	8
1.4. Structure of this document	9
2. Research	10
2.1. Existing Research.....	10
2.1.1. Bullying	10
2.1.2. Cyberbullying and abusive content detection	10
2.2. Technologies Research.....	11
2.2.1. Machine Learning	11
2.2.2. Coding Languages for Machine Learning.....	18
2.3. Dataset research	19
2.3.1. Dataset 1	20
2.3.2. Dataset 2	21
2.3.3. Dataset 3	23
2.3.4. Dataset 4	25
2.3.5. Discussion	27
3. Methodology	29
3.1. Dataset cleaning	29
3.2. Feature Extraction and Construction	31
3.2.1 Syntactic Features.....	31
3.2.2. Semantic Features	32
3.3. Dataset Rebalancing.....	35
3.4. Dataset Pre-processing	36
3.5. Text representation.....	37
3.7. Feature Reduction.....	38
3.6. Classification	39
3.8. Scoring.....	41

3.9. Cross Validation.....	42
3.10. Final evaluation	44
4. Experimental Process.....	45
4.1. Experiments	45
4.1.1. Experiment 1 – Classifiers.....	45
4.1.2. Experiment 2 – Text Representation	47
4.1.3. Experiment 3 – Dataset Pre-Processing.....	49
4.1.4. Experiment 4 – Feature Addition.....	50
4.1.5. Experiment 5 – Feature Reduction	52
4.1.6. Experiment 6 – Text Representation Normalisation	53
4.2. Results and Findings.....	53
4.2.1. Experiment 1.....	53
4.2.2. Experiment 2.....	54
4.2.3. Experiment 3.....	54
4.2.4. Experiment 4.....	54
4.2.5. Experiment 5.....	55
4.2.6. Experiment 6.....	56
4.2.7. Discussion	56
4.2.8. Demonstration Interface	56
5. Project Management	58
5.1. Plan.....	58
5.2. Outcome.....	59
5.3. Future Work	59
6. Conclusion.....	61
7. Bibliography	62
8. Appendices	65
8.1 Table of Figures	65

1. Introduction

1.1. Overview

The focus of this project is the classification of bullying content on social media through supervised machine learning. The issues of bullying and cyberbullying are highly prevalent in today's social media driven society. From 2010 to 2013 there was an 87% increase in the number of ChildLine's counselling sessions about online bullying [1]. 70% of young people aged between 13 and 22 have reported being a victim of cyberbullying. An estimated 5.43 million young people in the United Kingdom have experienced cyberbullying, with an estimated 1.26 million subjected to extreme cyberbullying on a daily basis [2].

Social media giant Twitter, a highly popular social networking service where users post and interact with messages, recently delved into the prevention side of online abuse and bullying. Twitter have implemented a 12 hour "lock out" of accounts reported for abuse that has been flagged automatically by their machine learning algorithms. "Tweets" containing terms, phrases or images that have been deemed abusive or inappropriate will be automatically hidden from conversation threads. This recent development was announced by Twitter on 1st March 2017 and aims to build on their previous attempts to tackle abuse on the platform. Facebook, the largest social media site in the world, rely heavily on user reporting in order to filter out abusive or inappropriate content and have yet to implement site wide automatic detection. Facebook have currently implement a feature for automatic spam and pornographic content reporting, although this is not site wide and only relates to its "Groups".

Multiple teams of researchers have attempted to determine and classify abusive or bullying text content by using differing, labelled corpuses and varying supervised learning techniques and features, which is discussed in Section 2.1.2. The current project draws from that body of work, in order to determine the best possible combination of machine learning algorithm, features implemented, data representation and pre-processing for a given dataset. This can then be applied either pre-publication in a preventative manner or post-publication in a removal and reporting implementation. A secondary aim of this project is to then build a generalised classifier that can be applied to most modern social media and implement it in a simple, demonstrative manner.

1.2 Objectives

The primary object of this project is to research the combination of machine learning practices that provide the best accuracy in a classifier, for a given dataset. Datasets implemented are deliberately selected from multiple online sources that are naturally structured differently due to their respective differing sources. Determining a method of the most successful classification will be a by-product of this main aim. A number of smaller objectives must be completed to achieve this, and are listed below:

- Acquiring labelled datasets, preferably labelled for bullying content but labelling for abusive content will be acceptable.
- Gaining a detailed understanding of the datasets through analysis and review.

- Investigating the existing applications and recent research into cyberbullying prevention.
- Researching cyberbullying and bullying to understand the aim of successful classification.
- Investigating different machine learning algorithms and testing their effectiveness.
- Determining a coding language or languages in which to perform classification.
- Examining and selecting a number of different features that can be extracted or determined and created manually from the text in order to test their effectiveness.
- Defining an experimental process that can be used to document the results of using different datasets with different features. This will allow for the identification of the most accurate grouping of features with each dataset where accurate classification is possible.

1.3 Challenges

The main challenges of this project are outlined below:

- The multiple concepts and technologies that needed to be learned in order to complete the project including the Python language, machine learning, text and data mining and data analysis.
- Determining which features applied in previous research to implement, and any possible features that could be created or derived.
- Training a classifier or classifiers for general analysis of potential bullying content from an unspecified social media source.
- What is considered bullying or abusive content is subjective, which may be the greatest challenge in this project. What one person interprets as abusive or bullying may not seem as such to another person. The downstream of this subjectivity is that if there are subjective labellers and subsequently subjective evaluations of the classification, then the models are trained and evaluated on this data. Whether the correct prediction has been made is also skewed by subjectivity.
- Textual data, particularly the conversational language and grammar being analysed, is highly unstructured compared to categorical or numerical data typically associated with machine learning. It is therefore quite difficult to apply structure to this data in a manner that a classifier or computer can understand.
- The datasets obtained are labelled subjectively, many text instances contain the original misspellings, abbreviations, and colloquial terms that would be impossible to structure for a classifier to interpret correctly without an extensive dictionary system.
- Time handling and project management.
- Academic research and writing skills that may require development.

1.4. Structure of this document

Chapter 2 of this document will focus on the research performed throughout the project. Sections include: research in cyberbullying and bullying, abusive content classification, machine learning and specific machine learning concepts for text analysis and the coding languages and libraries investigated.

Chapter 3 outlines the scientific method applied to each dataset instance and will detail the dataset cleaning, pre-processing, feature extraction and construction, text representation, classification, feature reduction and evaluation of the final process. Variations across differing datasets will be emphasised clearly in each given subsection.

Chapter 4 explains and displays the experiments performed. It will also provide an analysis and reasoning of the results and findings from the experiments. Again, differences in results across datasets will be made clear.

Chapter 5 will focus on planning and organisation details of this project including suggestions for improvements and possible future work.

Chapter 6 concludes this document.

Chapter 7 contains the Bibliography of references used throughout this document.

Chapter 8 contains the appendices.

2. Research

2.1. Existing Research

Before this project could be executed, the previous work performed in this area of classification and prediction had to be studied. There are many papers and articles written on similar research conducted, each outlining very different methods in terms of how text is represented, how features are derived and what machine learning algorithms are applied to these sets. To further define what features are most applicable, this also required delving into research on bullying and cyberbullying for definitions and insights into the technicalities of this classification. This classification and the effectiveness of the previous work performed is necessarily subjective, due to the subjective nature of what is considered bullying.

2.1.1. Bullying

To detect and classify cyberbullying, initial research included obtaining a definition of bullying and a definition of cyberbullying. Traditional bullying definitions do not focus on this exact area of research, i.e. specifically text-based bullying on social media. However, there are broader aspects of all traditional bullying definitions that do relate, such as a victim always being present in the act and the aggressive nature of the act. This is encapsulated by the current popular definition of cyberbullying being *“an aggressive, intentional act carried out by a group or individual, using electronic forms of contact, repeatedly and over time against a victim who cannot defend against him or herself”* [3].

This definition of cyberbullying contains discrepancies concerning the nature of online social interactions. The first and most obvious of which regards the definition of electronic forms of contact. The second surrounds the repeatable and “over time” nature of the content. The former can be categorised as primarily text based communication due to the ubiquity of communication technologies available across electronic devices such as text messages, e-mail, and social media rather than the ubiquity of the differing electronic devices themselves, such as laptops and smartphones [4]. The second discrepancy is that *“repetitious messages are not exclusive to cyber bullying”* [4] because *“a single act by a perpetrator might reasonably be expected to be repeated by others, and the perpetrator might reasonably be expected to know and even intend this”* [4]. Therefore, cyberbullying can be contained, single instances through text that are aggressive and directed towards an individual victim. This will not be accurate for all cyber bullying instances. No definition of bullying or cyberbullying perfectly encapsulate each individual respective instance, bullying and cyberbullying events are generally unique. What must also be considered is the heavily subjective nature of the content being analysed, which will be further argued in Section 2.3 Dataset Research.

2.1.2. Cyberbullying and abusive content detection

Text mining is a common technique utilised in research studies relating to text based cyberbullying and similar topics. Many of the approaches that have been applied and will be implemented in future in this project have been applied across previous research in this field. Bag of words [5–7], N-gram [5–7] and occasionally term frequency, inverse-document frequency normalisation (TF-IDF, like a weighted bag of words)[8, 9] term representation are applied regularly to the datasets in use. This type of initial feature representation is usually used in conjunction with supervised learning algorithms such as Naïve Bayes [5–7, 10]

,Support Vector Machines in both linear [5–7, 9, 10] and non-linear [8] modes of operation, Logistic Regression [7] or Decision Trees [10, 11].

The vast majority of research involving machine learning and text mining applied in machine learning implement a form of N-Fold cross validation, almost always 10-Fold cross validation [6, 9–12]. Scoring on models is performed in a variety of methods, ranging from overall / average accuracy or average recall to precision to F-1, however, there is a greater emphasis on average recall evaluation [6, 11, 12]. This is due to the nature of this area of text mining, where unbalanced datasets provide high precision and F1 score in only the overrepresented class which skews this scoring. The use of average recall provides a more accurate scoring method by removing the possibility of using precision scores from the overbalanced class, while still accounting for false negatives which are of great importance to this type of classification.

Similar feature additions also appear throughout other research. Inclusions such as punctuation analysis [6], where types and counts of punctuation are included in the feature set are used in conjunction with a ratio of uppercase only words [6]. Other additions to feature sets see counts of offensive words or expletives [6, 8–10] or even labelling an instance when such a word is in proximity to a pronoun. Part of speech labels were also added in some instances [5, 8], where a term or N-gram would also be given the part of speech term used in place of the words. The term “large document” would be labelled as “adjective-noun” in a bigram part of speech analysis. Often in conjunction with abusive terms or expletives, second person pronouns [6, 9, 10] were explicitly labelled in some cases as the nature of bullying revolves around a targeted victim. This can also be accounted for in the parts of speech analysis performed by other research. Less likely but just as relevant to cyberbullying and text analysis were contextual features such as similarity to neighbouring posts [9]. This is a much more difficult process due to the strict requirements of threaded data. Another relevant, if unlikely, aspect to the research is sentiment analysis [5]. The ability to label, even on a relatively simple scale, the positive or negative sentiment ratio on a given piece of text will be very effective in improving classification.

Data rebalancing [11, 12] is another necessary component of machine learning in this field of research. Although not recorded throughout each individual paper, the often overwhelming level of unbalanced datasets must require extensive rebalancing. As explained below in Section 2.3, previous research performed [12] has determined an optimal rebalanced ratio for the datasets that will be manipulated in this project.

2.2. Technologies Research

2.2.1. Machine Learning

At the highest level, there are two types of machine learning; unsupervised and supervised. Unsupervised learning is more along the lines of learning patterns or distributions about a dataset rather than attempting to obtain a definite output from a machine learning algorithm. It’s mostly used for exploratory modelling on a dataset to find the more important features, as there is no given correct answer for each instance in a dataset, only a set of features [13].

Supervised learning is the type of machine learning that will be used for this project. Each given set of feature values is supplied with a given label or class, to which these features are effectively “mapped” to be used for comparison in order to predict labels for future sets of features. Datasets are separated into training and test data, normally a 90% to 10% split. Training data is the set that is used in the learning, or training, of the model and after this has been performed the labels of the test set are predicted by the trained classifier, the model.

2.2.1.1. Terminology and Basic Concepts

Listed below are some of the terms and concepts that are key to machine learning. The explanations are based on learning and understanding gained throughout the implementation and research performed during this project and are high level descriptions of the terms.

- Dataset – the data used in the model, stored in rows and columns of data. In supervised learning, each row has a label which defines that row into a class.
- Label - effectively an answer that the row of data describes. For example, the label might be "banana" if the data were the words "yellow", "long" and "fruit".
- Class – classes are defined by their label. Including the previous example, another row could be labelled “apple” and another labelled “potato”. These three labels are the three classes in that dataset.
- Feature – a feature is a column in the dataset, other than the label. Therefore, implementing the current example the features would be “colour”, “long or short”, “fruit or vegetable”.
- Classifier – it is a machine learning algorithm that maps the features of a row to its label. There are multiple classifiers. They are all relatively unique in how they classify data or how a model “learns” or is trained.
- Model – the result of the classifier being trained. Effectively the machine that has “learned”.
- Cleaning data – normalising each of the pieces of text input so that they are in the same format. This could be as simple as changing it all to lowercase or more complex, for example, translating text.
- Training set – part of the dataset used to train the model with a classifier. Usually 80 or 90 percent of the overall dataset.
- Test set – part of dataset held back from training. This data is then given to a model without the labels to classify. As the model has not seen any of this data before, it can be taken as a validation set. The predicted labels assigned by the model are evaluated against this set’s actual labels.
- Instance - one row of data in the dataset.

- Cross Validation – separating the dataset into smaller sets, and then using one set as the unseen test set and the rest as the training set. Then, change the test set to a different one, and the new remaining sets as training data. Used to ensure all of the data will be used as test and training data at some point in the validation.
- N-Fold Cross Validation - this involves splitting the dataset into N number of smaller subsets or “folds”.
- Average recall - also known as average class accuracy. This is the average accuracy of the model, obtained by comparing the label predicted by the model for each instance in the test set to its actual label, for each instance. The accuracy of each class is then averaged, giving the average accuracy or average recall of the model.
- Vectorised – a dataset is vectorised after bag of words or N-gram representation has been applied to the data.
- Vectorizer – the algorithm that applies bag of words or N-gram term representation to a dataset.
- Term representation – numerically representing text so that a computer may apply meaning and structure to the data.
- Bag of words - a way of representing a text dataset. This method counts each occurrence of a word in a given piece of text but does not keep the order of the text. The “bag” contains all words in a dataset, although only once.
- N-gram - very similar to the bag of words model, but instead of individual words it counts a grouping of words
- Rebalancing - used to balance a dataset with respect to the number of instances in a given class. Unbalanced datasets can lead to false conclusion and misclassification.
- Under sampling - using less of a larger class in a dataset in order to balance the set.
- Over sampling - reusing instances in the smaller class to balance the dataset.
- Overfitting - overfitting occurs when tailoring a model very specifically to a dataset, in order to produce higher results when using the test set. This can cause much greater fluctuation in the accuracy for completely new data
- Underfitting - underfitting might also be an issue, although it is far less common within machine learning compared to overfitting and stems from an underdeveloped feature set used in classification, i.e. not enough features are used to effectively represent the data.

2.2.1.2. Classifiers

2.2.1.2.1. NAÏVE BAYES

Naïve Bayes classifiers use the probability of each feature occurring in each class to determine which prediction to make. The probability of a feature value appearing in each class is calculated, separate to all other features. These probabilities are then combined to find the overall probability of the instance belonging to each class. The class with the highest probability is then chosen based on the combined probabilities calculated from the training data.

Naïve Bayes classifiers are relatively simple probabilistic algorithms. They implement Bayesian probability theory while assuming that each individual feature is independent of others. Therefore, the classifier is considered naïve. Bayesian probability theory *“really involves nothing more than the manipulation of conditional probabilities.”* [14] The probability of a hypothesis, H, given the data D can be defined as:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \quad [14]$$

Naïve Bayes classifiers apply Bayes' Rule using the possible class as the hypothesis value, H, and the current feature value as the data, D. After finding the individual values of probability for each feature in each class, the probabilities are multiplied across each group of features in each class to provide the overall probability of this class being correct. The class with the largest estimated probability is returned.

The assumption of independence in the feature values that gives this particular algorithm its name is *“clearly almost always wrong”* [15]. This is often thought to lead to decreased classification accuracy and performance. However, comparisons made between *“state-of-the-art algorithms for decision tree induction, instance-based learning, and rule induction on standard benchmark datasets”* [15] resulted in the Naïve Bayes algorithm becoming *“sometimes superior to the other learning schemes, even on datasets with substantial feature dependencies”* [15].

2.2.1.2.2. SUPPORT VECTOR MACHINES

Support Vector Machines (SVMs) function by plotting each instance as a point in a two- or three-dimensional space. Linear SVMs attempt to separate the classes within this plotted space via straight-line that supports the ‘optimum separating hyperplane’, an area around and parallel to this separating line. The ‘optimum separating hyperplane’ is the plane that has the largest gap between the outer bounds of the plane and therefore the largest area. This allows for new, unseen data to be plotted in the same way and based on where it is plotted in relation to the optimal hyperplane, the label is predicted. In a linear SVM, where there is less variance within the class instances, the hyperplane can be defined well and there is clear separation between classes [15].

When more variance is recorded between classes i.e. there are many non-binary labels or a less defined feature space, a non-linear SVM can be incorporated. Non-linear SVMs implement a curved hyperplane or multiple curved hyperplanes, which aids in further

classification but can cause increased overfitting for the training set used. SVMs require much more time than other classifiers tweaking the algorithm parameters including kernels and their co-efficient factors and feature selectors [16].

2.2.1.2.3. LOGISTIC REGRESSION

Logistic Regression classification is similar to Naïve Bayes classification in the sense that both implement a variation of probability in their algorithms and apply this probability to the test data to predict labels. Logistic Regression builds upon the logistic function:

$$\frac{1}{1+e^{-x}} \quad [17]$$

A Logistic Regression classification model can have many input variables; a vector of variables, in contrast to the logistic function above which has a single x value as input. Additionally, it consists of weights or coefficients for each input variable. The resulting Logistic Regression is defined as:

$$\frac{1}{1+e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}} \quad [17]$$

This value, β , is the regression coefficient which “*depends, in general, upon the other independent variables. Specifically, the value of the partial coefficient for one independent variable will vary, in general, depending upon the other independent variables included in the regression equation*” [17]. Additionally, similar to Naïve Bayes, the logistic function would not account for the individual features depending on each other without this regression coefficient. Stochastic Gradient Descent (SGD) logistic regression classifiers have been implemented. SGD, or Online Gradient Descent, descends along the cost function towards its minimum for each training instance in the classifier. At every step of this cost function the regression coefficient values are updated.

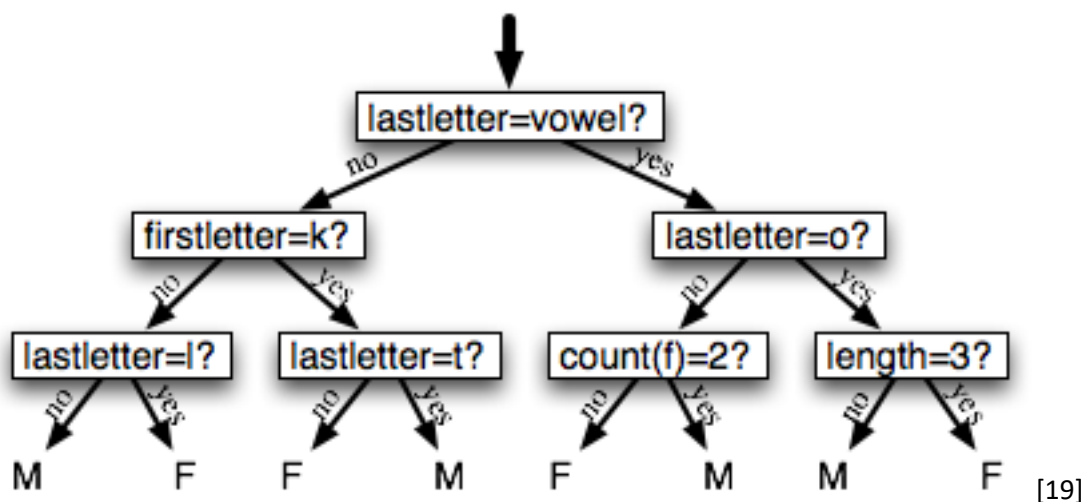
2.2.1.2.4. DECISION TREES

A very high level explanation of a Decision Tree Classifier is to break up the difficult overall decision of classification into a set of lesser and simpler decisions, in the hope that these will result in the correct classification [18]. Decision tree classifiers are tree structures where each node is a feature in an instance of the dataset. The branches represent which values that feature may take, although usually these values are clamped in a certain range that aligns with the training data. This type of classification is very simple as each node poses a rule based question to the value of that particular feature. The root node of the tree, and often the criteria used to determine the quality of each split at each node, is decided by a bias determining function.

The two most frequently used bias measures are the Gini impurity and information gain or entropy. The training set, in conjunction with the bias heuristic selected, effectively decide how a test set will be “sorted” down the tree structure into its eventual class. Due to the possibility of differing heuristics providing alternative root nodes and splitting scores, it is

advised that multiple bias functions are tested. Pruning, the removal of nodes that do not offer any further aid in classification is also heavily advised. Other methods such as limiting the size of the decision tree before fitting, “pre-pruning” or choosing the tree with the fewest leaf nodes can also increase accuracy. There are also no definite best methods of pruning and pre-pruning [15]. An example of a simple decision tree structure used for determining the sex of a person based on the letters used and composition of letter in their name can be seen below in Figure 2.1 [19].

FIGURE 2.1. AN EXAMPLE DECISION TREE



2.2.1.3. Text Mining

Text mining applies the data mining and machine learning algorithms used on numerical data to text. There are several difficulties that arise when implementing text mining. As computers do not understand language and its intricacies, there is a need to find a way of representing words and terms in a way computers can interpret them. This essentially requires transforming a text input into numerical form. The nature of how machines learn and how other data mining and machine learning techniques are applied to regular data must also be considered.

The vast majority of data is categorical or numerical, from a machine learning and data analysis standpoint. This data is often highly structured and easy for a computer to interpret. Categorical data can be as simple as “yes” or “no” and numerical data is a simple, value representation. Therefore, this data can be seen as well structured, bounded data. Text, and language in general, is comparatively highly unstructured. Many terms have the same meanings, many are colloquial and unique. Some terms can be in the same vein as others without sounding or looking remotely similar. Text mining techniques attempt to structure text in such a way that computers can understand the significance of terms, rather than the terms themselves.

Text representation focuses on the same basic concept, in the few popular forms it takes. By counting individual words, groups of words or groups of characters, terms can be quantified in a given data set so that a machine may understand their significance. The simplest form of text representation is “Bag of Words”. This is a simple count of words in each given instance in a dataset. A dictionary of the terms across the data set is constructed, where

each term is one feature in the new representation. Each original text instance is then transformed to a row of these features, every word in the bag. The features in the row are then populated with the corresponding count of the words in this original instance. This often leaves many zeroes in an individual row, as it is highly unlikely that a single instance would contain every word in the dictionary.

Following from this is N-gram representation, the most common of which are bigrams and trigrams. Groups of words are utilised, bigram uses two words while trigram uses three words in sequence as individual terms, and these are counted. Bag of words text representation is essentially unigram or single term N-gram text representation. Another common method is to use groups of sequential characters rather than full words, limiting to four characters in a group being the most popular. Bag of words removes the structure of the text, simply focusing on what words are present, whereas N-gram representation often retains a lot of the structure particularly for shorter text bodies which is incredibly useful for preserving grammatical structure and frequently used word order.

Minimum and maximum document frequency is also a common practice applied to text mining machine learning. This restricts the terms that appear in documents, setting a minimum or maximum number of instances in a dataset that a term must appear in for it to be selected as a representable feature. This is performed in order to remove terms that appear so often or so sparingly that they have little effect on classification and only serve to increase the amount of features constructed from term representations such as Bag of Words or N-grams. Removing these unnecessary extra terms from the feature matrix greatly reduces the amount of input features into the classification algorithms and makes the machine learning process less convoluted.

Post vectorisation feature reduction is also applied through algorithms such as Principle Component Analysis (PSA) and the Chi-Square measure. In this project, the Chi-Square measure, written as χ^2 , is used. Chi-square is a statistical test used for comparing variance, and calculates corresponding χ^2 values between features and the classes in given dataset. The scores from this calculation then allow for feature reduction on a highest to lowest scoring basis.

Normalization is also a common practice in most machine learning and text mining. It is used to smooth input into a classifier, in particular normalised term frequency, which implements Euclidian distance normalization, intending to focus on the direction of the input vector rather than magnitude.

$$tf_{ij} = \frac{Num_{ij}}{\sqrt{\sum_{i=1}^n Num_{ij}^2}} \quad [20]$$

In summary, if a term appears in an instance 10 times, but that instance only has 20 terms, it aims to normalize this figure to avoid the large spike in that frequency or representation. This is useful as it removes the burden of such invariant features from the classifier and helps to aid in better classification. However, normalization itself, much like machine learning, is considered a “black art” in that many different forms exist and they require significant testing and tweaking to find the correct application.

2.2.1.3.1. SENTIMENT ANALYSIS

A subsection of text mining, sentiment analysis attempts to determine the meaning behind the language used in text from a positive and negative sentiment point of view. In relation to this project, this is potentially vital to classifying instances based on more than just the text present and additional numerical features present in the text. Sentiment analysis, also referred to as opinion mining is “an active area of study in the field of natural language processing that analyses people's opinions, sentiments, evaluations, attitudes, and emotions via the computational treatment of subjectivity in text” [21]. In order to apply this, an extensive dictionary of terms is required, and their corresponding sentiment labelled accordingly. This is, of course, a long and intensive process.

The NLTK library provides a sentiment analysis module, known as the VADER (Valence Aware Dictionary for sEntiment Reasoning) sentiment analysis tools [22]. This type of sentiment analysis not only provides a large corpus of labelled terms that represent positive and negative emotion but also during analysis returns an intensity score of positive, negative and a compound sentiment. The scoring ranges from zero to one for the individual polarities. A zero indicates none of the sentiment present, a one represents the instance is entirely positive or negative, depending on the single polarity being tested. For compound scoring, a zero indicates neutrality, minus one represents full negativity and one indicates complete positive sentiment. The terms in the dictionary were labelled on a scale, ranging from the extremely negative (-4) to extremely positive (4), with 0 being neutral. Terms in the sentiment dictionary are then matched to corresponding terms in the text being analysed, if any are present. Other considerations during processing include punctuation, for example exclamation points (!) after a term, capitalization of a term, and the use of what are known as “degree adverbs” [21] such as “very”, “extremely” and “marginally” before sentiment terms increasing the intensity of sentiment. The use of “but” in a sentence also suggests a change in the sentiment of a sentence, and that is accounted for, often resulting in both positive and negative sentiment being split during analysis. Finally, a trigram analysis of the terms before a sentiment term is implement to catch the negation of positive terms in the given example “The food here isn’t really all that great” [21], resulting in a polarity change from the positive term “great” being negated by “isn’t”. This module provides an incredibly accurate and valuable sentiment analysis aspect of the project, however there are weaknesses and issues. It requires that the input be correctly spelled, and have a proper grammatical structure. This is near impossible to enforce on data unless done so at the same time as collection by defining strict parameters for acceptable instances. It is also highly unlikely that any dictionary of positive and negative terms is all inclusive, particularly regarding local and internet slang.

2.2.2. Coding Languages for Machine Learning

This project requires many custom, developer defined functions and as such it makes sense to find a language that performs and defines them with ease. The initial languages considered were R and Python.

R, also known as GNU S, is a functional language that is primarily used in statistical analysis and visualisation of datasets. It is an open source language with lots of community support and specific libraries available for machine learning and predictive analysis. This is obviously a viable language in which to perform this project.

Python is a very common and widely used high level language that is open source and has numerous libraries, including those that allow for machine learning, data analysis and data visualisation. Python has many online resources and offers much more than R in terms of simple data manipulation, specific feature construction and syntax simplicity. Python also offers many more available functionalities without the need for further integration of other programming languages and resources, such as web socket connectivity and the ability to create user interfaces natively.

For the reasons outlined, Python, specifically Python 2.7.12 [23], will be the only coding language used in this project. Python libraries Sci-kit Learn [24], Pandas [25] and the Natural Language Toolkit (NLTK) [26] are all utilised. The Sci-kit Learn library is the most extensively used as it provides multiple, quick and modifiable classifiers necessary for this project within one package. Pandas has been applied specifically for reading and parsing csv files in which the datasets are stored into “DataFrames”. Pandas DataFrames are easy to manipulate in terms of randomising or shuffling the initial dataset, splitting into the necessary subsets or classes of data and resampling to balance the dataset. Pandas and Sci-Kit Learn are also very compatible. The NLTK library is included for determining specific words, removing stop words and for future further implementations of the NLTK functionality. Also imported are SciPy [27] and NumPy [28] which are both powerful scientific python libraries. They are used very sparingly and mainly for mathematical calculations; however, they are also necessary for Sci-kit Learn installation. Standard Python libraries are also used, such as the string and re (regex, regular expression) classes. Also imported are the Tkinter [29], ttk [30], ImageTk and Image [31] classes and modules for use in developing a very simple demonstration front end for the results of the research.

2.3. Dataset research

This project required datasets that had been gathered from social media, preferably from several sources and preferably labelled for bullying or traces of bullying. Rather than perform the time-consuming data collection and labour intensive process of labelling the data, a search for available datasets already composed on the subject was performed. This focused on finding the corpuses used in the previous research in this area due to their specific relation to this project. The data collection would have required numerous scraping of social media sites and the labelling would require an impartial and willing large group of people which was not viable in the given time frame. Previous papers did provide datasets that had been labelled appropriately as described below.

The datasets are of various sizes, both in terms of number of instances and size of each individual instance. Each dataset originates from a single social media or web forum source. Each dataset has been labelled for traces of cyberbullying or harassment, and some include other fields that were implemented in the original research performed and easily available at time of data gathering. While all are labelled for very similar end goals, they will

have been done so under differing rules and circumstances and by differing groups of people. Therefore, each must be treated as an entirely separate dataset, even when from the same source.

2.3.1. Dataset 1

The first dataset is a Twitter dataset collected for similar cyberbullying research [7] and will be referred to as Dataset 1. It contains 1340 instances and was created by scraping Twitter feeds that had been reported for bullying. For non-bullying samples, a general non-specific scrape of Twitter was conducted. The data was then manually labelled for “bullying” or “non-bullying” and led to a distribution of 13% in the positive class and 87% in the negative class. The working definition of cyberbullying was “use of Information Technology to harm or harass other people in a deliberate, repeated, and hostile manner” [7]. Each instance of this dataset contains only the text used in the post and the label for bullying. The average length, in characters, of an instance is 78. Figure 2.2 and Figure 2.3 display the total number of instances and the total number of instances found containing an individual and swearing. Figure 2.2 represents the positive class, an instance of bullying, while Figure 2.3 represents the negative class. Figure 2.4 shows the average number of characters, average number of capital letters, average number of punctuation marks, average number of words in an instance and average number of swears in an instance across each class.

FIGURE 2.2. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 1 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

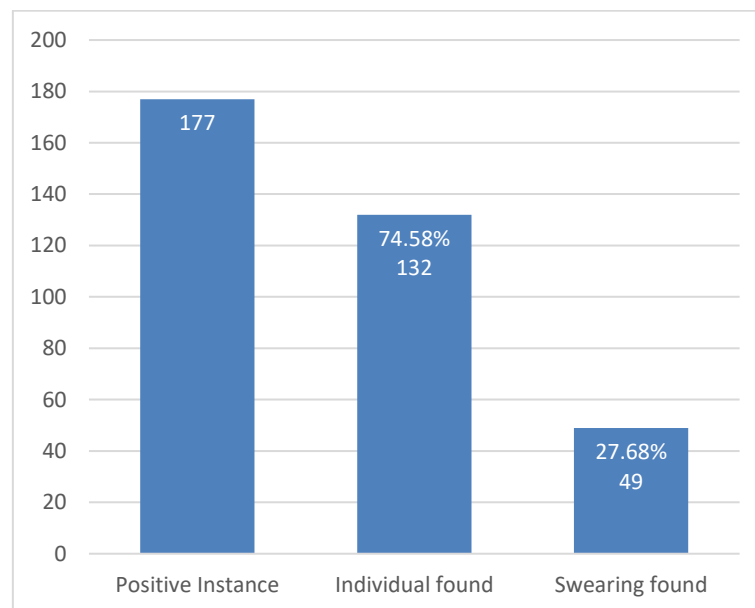


FIGURE 2.3. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 1 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

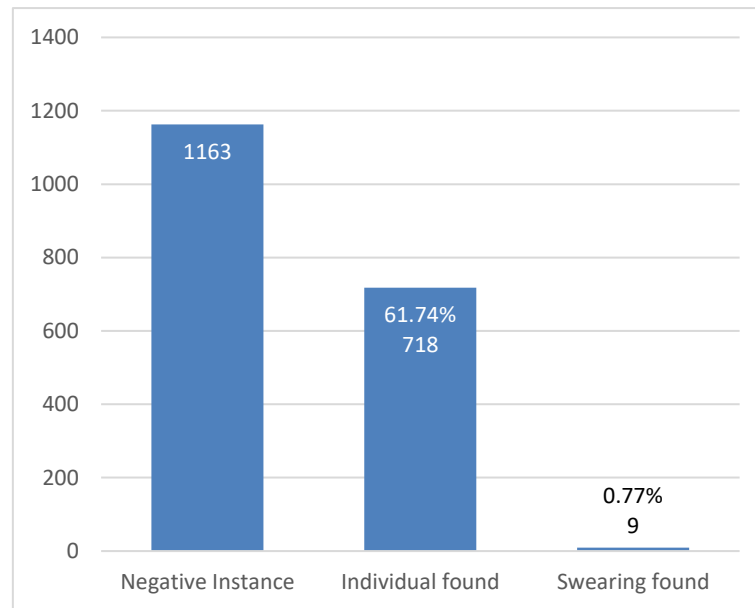
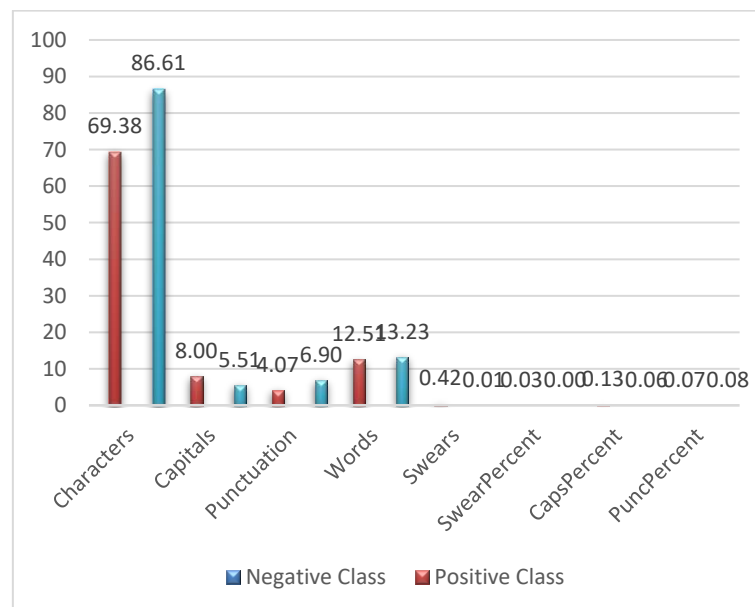


FIGURE 2.4. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 1 FOR BOTH CLASSES



2.3.2. Dataset 2

The second dataset, Dataset 2, was collected for cyberbullying research [10] and contains instances from YouTube comments collected from content posted that was considered “*sensitive to cyberbullying*” [10]. Searches were performed for the videos with this criterion, and the users who commented on the three highest videos in each search were logged. From this list of users, their comments from April to June of 2012 were scraped. The data was then manually labelled for “bullying” or “non-bullying” by two graduate students

using the definition of cyberbullying as, “an aggressive, intentional act carried out by a group or individual, using electronic forms of contact repeatedly and over time against a victim who cannot easily defend him or herself” [10]. The corpus contains 3462 instances and was labelled in a distribution of 12% in the positive class and 88% in the negative class. Each instance contains the original text, the number of comments in the instance, number of subscribers the user has, membership duration, number of uploads the user has, if there is profanity in the User ID, user age and the bullying label. The average length of each text instance is 1293 characters. Figure 2.5 and Figure 2.6 display the total number of instances and the total number of instances found containing an individual and swearing. Figure 2.5 represents the positive class, an instance of bullying, while Figure 2.6 represents the negative class. Figure 2.7 shows the average number of characters, average number of capital letters, average number of punctuation marks, average number of words in an instance and average number of swears in an instance across each class.

FIGURE 2.5. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 2 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

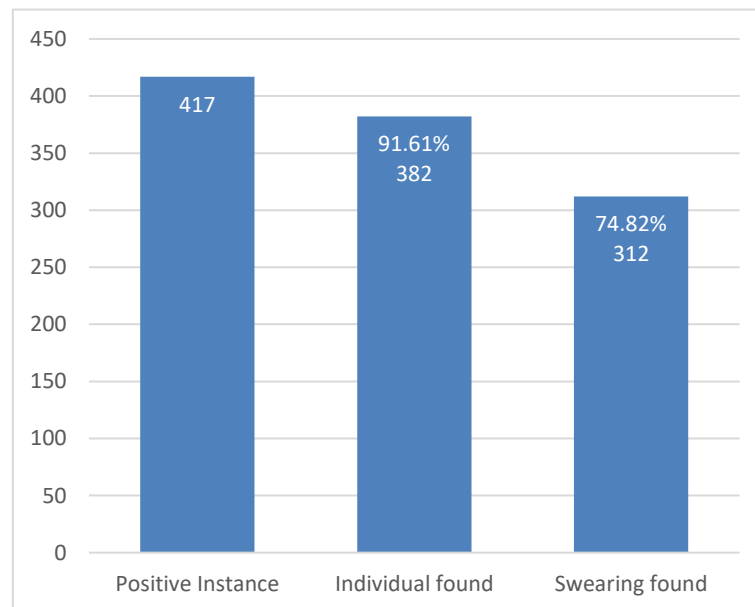


FIGURE 2.6. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 2 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

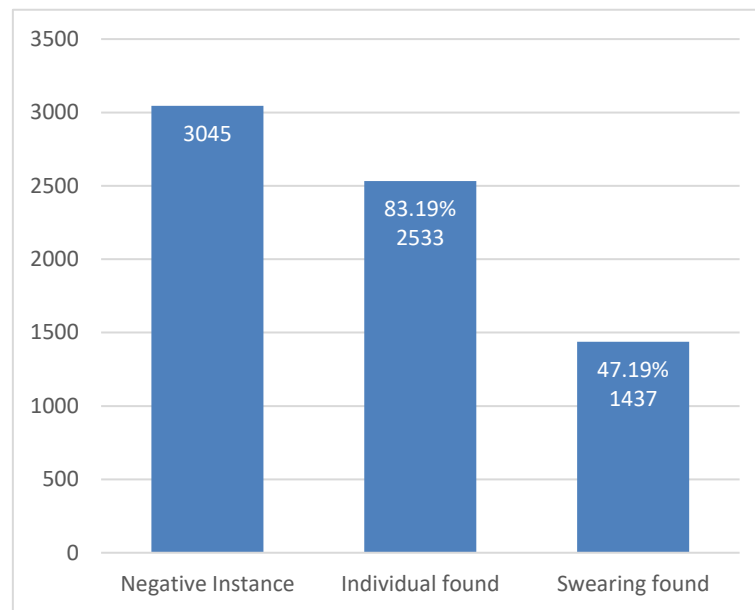
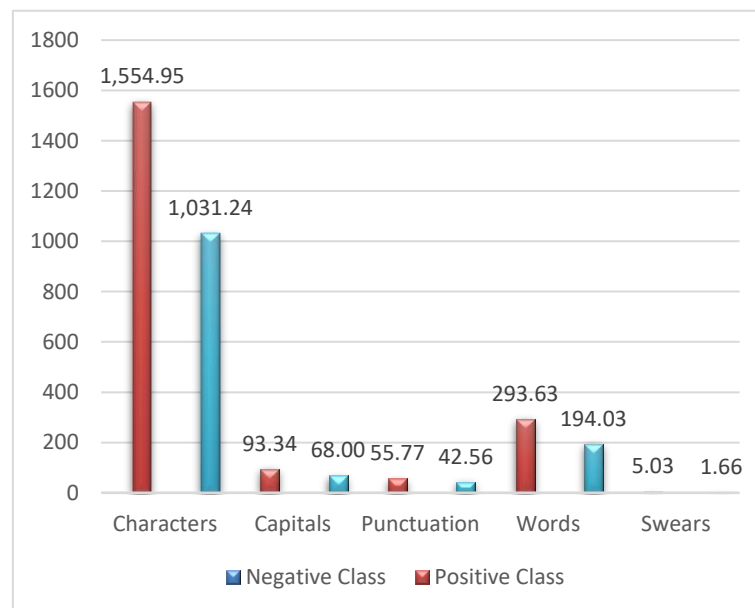


FIGURE 2.7. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 2 FOR BOTH CLASSES



2.3.3. Dataset 3

Dataset 3 was collected for another cyberbullying research paper [32]. Data was retrieved from MySpace, which is a threaded forum style conversation on a given topic for the thread. This dataset differs in that it was collected using a moving window style, meaning that 10 posts in a thread are included in the instance in a dataset. This “sub-thread” is then labelled manually for the presence of cyberbullying by three individuals, requiring at least 2 of these

individuals to label an instance as containing cyberbullying for it to be accepted as such. The definition given for cyberbullying in this paper is “wilful and repeated harm inflicted through the medium of electronic text” [32]. The dataset contains 1658 instances split into 23% in the positive class and 77% in the negative class. Each instance contains text and a label. The average length of an instance is 1503 characters, almost twice the average length of Dataset 2 and 22 times the average length of Dataset 1. However, as this is a moving window style dataset, each window contains repeat posts multiple times. After separating the windows, the average length of an instance is 377 characters. The figures below (Figure 2.8 - Figure 2.10) are based on the separated window. Figure 2.8 and Figure 2.9 display the total number of instances and the total number of instances found containing an individual and swearing. Figure 2.8 represents the positive class, an instance of bullying, while Figure 2.9 represents the negative class. Figure 2.10 shows the average number of characters, average number of capital letters, average number of punctuation marks, average number of words in an instance and average number of swears in an instance across each class.

FIGURE 2.8. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 3 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

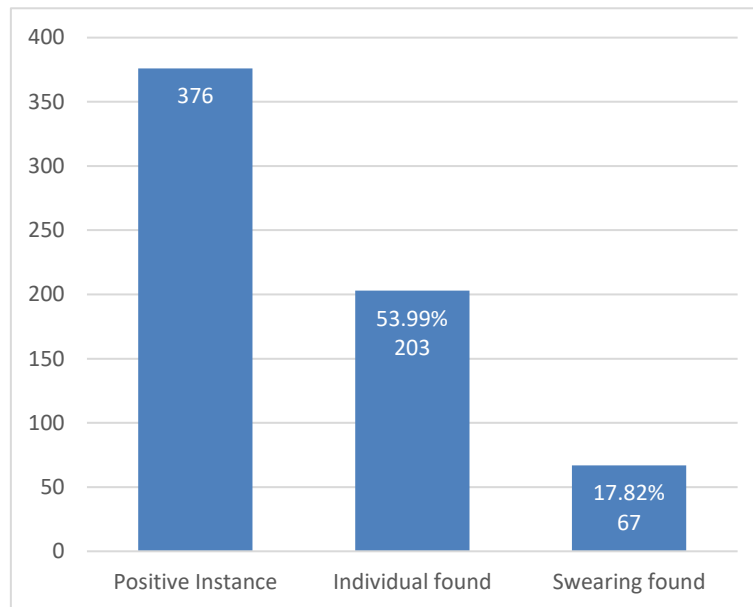


FIGURE 2.9. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 3 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

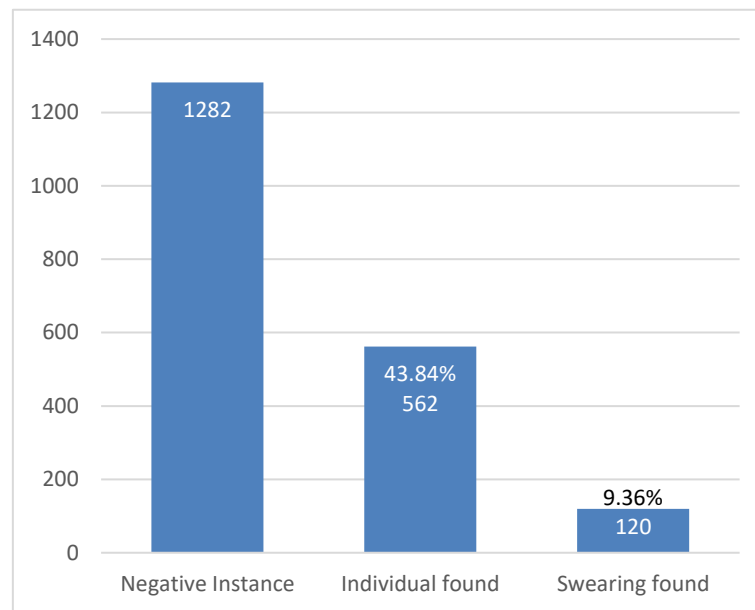
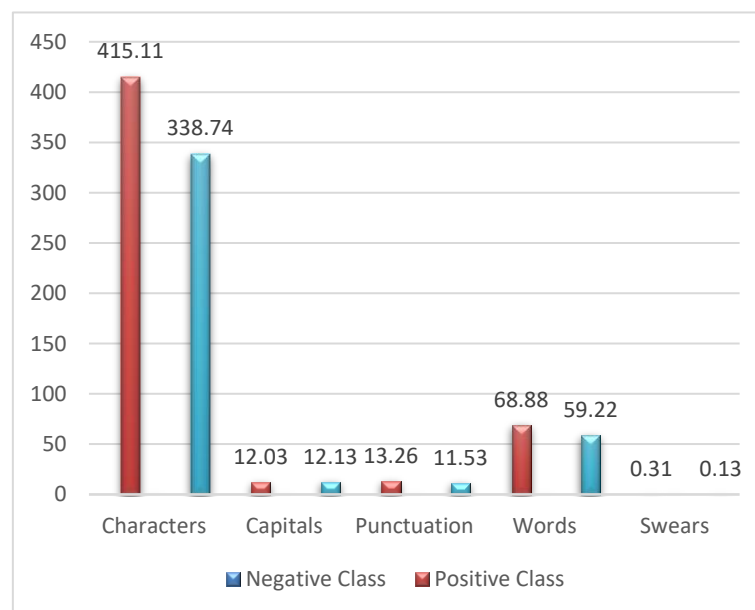


FIGURE 2.10. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 3 FOR BOTH CLASSES



2.3.4. Dataset 4

Finally, Dataset 4, gathered from Kongregate and collected by Fundación Barcelona Media for a research paper on online harassment [9] in which the researchers self-define harassment “restrictively as a kind of action in which a user intentionally annoys one or more

other users in a web community” [9], contains 3991 comments from a single thread. Kongregate is a gaming website that features games playable in a web browser but also has a chat-room aspect to the site where users can communicate directly in threads. This dataset is labelled manually for harassment in a split of less than 1% percent in the positive class and 99% in the negative class. Each instance contains the post body (the text), the post ID, the user ID and the harassment label. The average length of the text across this dataset is 29 characters. Figure 2.11 and Figure 2.12 display the total number of instances and the total number of instances found containing an individual and swearing. Figure 2.11 represents the positive class, an instance of bullying, while Figure 2.12 represents the negative class. Figure 2.13 shows the average number of characters, average number of capital letters, average number of punctuation marks, average number of words in an instance and average number of swears in an instance across each class.

FIGURE 2.11. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 4 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

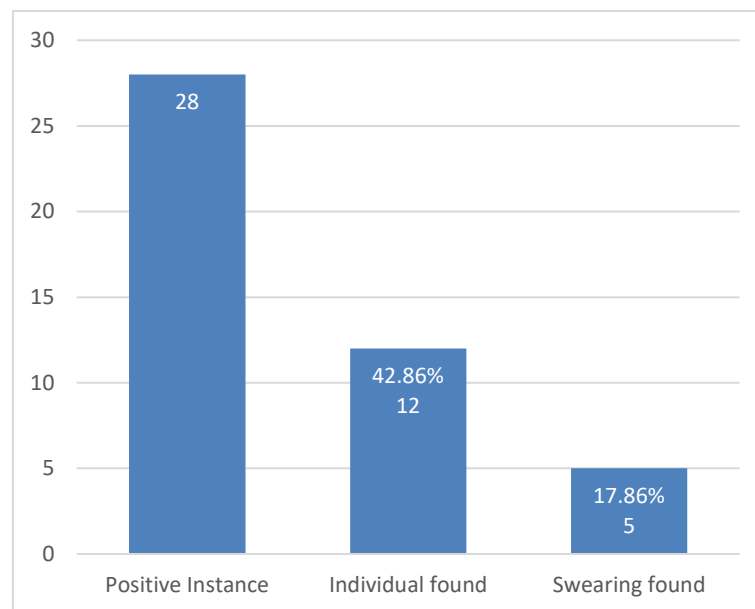


FIGURE 2.12. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 4 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"

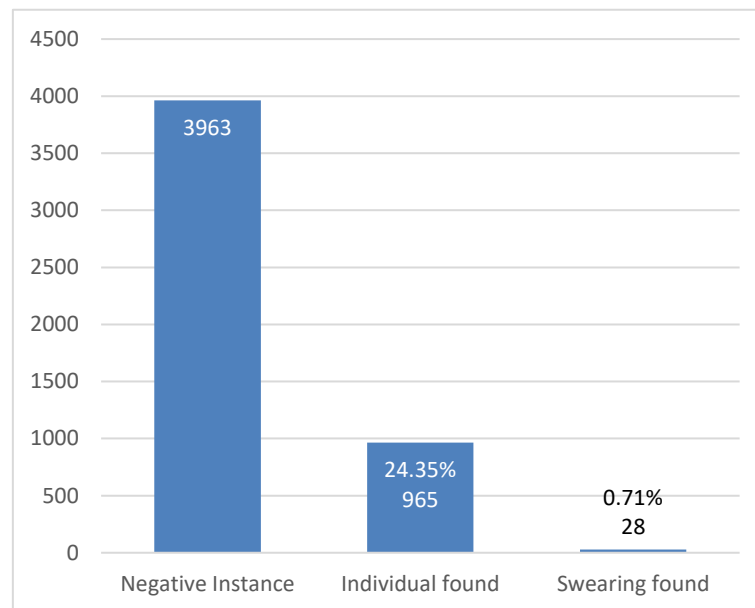
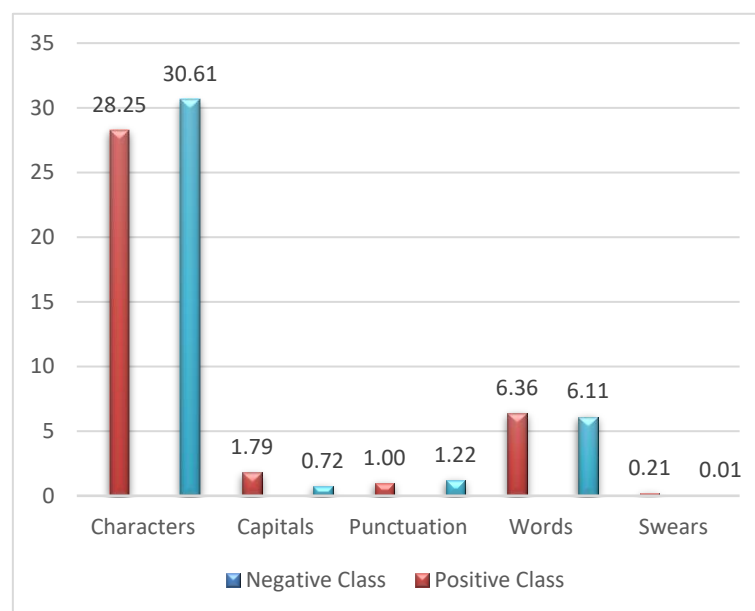


FIGURE 2.13. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 4 FOR BOTH CLASSES



2.3.5. Discussion

It is very important to recognise the manner in which each dataset has been labelled. As previously mentioned, bullying and harassment are very subjective and personal opinions vary depending on numerous factors. What may be considered bullying, abuse or harassment by one may not be considered that way by another. Despite the attempts to vary how the input is labelled, often using a group of people to independently label the instances, it is very difficult to universally define a class to assign to each sample. This is evident even in the

different definitions given in each research paper on the datasets used. Although they are incredibly similar, and imply the same basic concept, the variations in the language used could cause confusion and bias on the part of labeller.

For example, consider the instance from Dataset 1 in Figure 2.14 (the original username has been removed for anonymity). This instance has been labelled as 1, i.e. is in the positive class and contains bullying. The given label is debateable. This is just one example across all the datasets used, although very few such instances appeared, it nonetheless showcases the subjective nature of bullying or abusive content and therefore the data itself.

FIGURE 2.14. A SAMPLE INSTANCE FROM DATASET 1

Text	Label
@username is 30 years old and tweets about belly button rings.'	1

Of particular interest is the notable increase, across all datasets, in the percentage of instances that contain swearing. This is valuable insight into the future feature extraction and construction outlined in Section 3.2. The difference in this percentage between the class labels may be a baseline indicator when detecting bullying and abusive content. Although not a dataset that will be classified, it is necessary to mention the swear word dictionary currently used. It is a list of swears obtained from noswearing.com [33] that contains 349 curse words. Although noswearing.com is “a swear word translator, a list of bad words, and an API to help programmers remove unwanted language from their applications” [33], only the list of bad words has been used and manual parsing of swears has been performed rather than implementation of their API. Also incorporated were lists of popular first names obtained from the US Census Bureau [34]. The list contains only the names that occurred 100 or more times from a Census taken in the 1990s. This is used to find possible victims in each instance.

Another obvious observation is the large imbalance across each dataset. Positive cases are massively underrepresented, and resampling must occur across each dataset to balance the input into each classifier. Chen et al. [12] have determined an optimal resample rate across the datasets in use that retains the prediction accuracy across the negative class while also balancing the dataset and increasing the prediction accuracy of the positive class. The process will be applied and explained in further detail in Section 3.3.

Finally, there are quality issues across most datasets in terms of the language used which should not be overlooked. There are many misspellings in each dataset, some instances are barely recognisable English. This makes text vectorisation incredibly difficult bar a complete and detailed spellcheck of the individual samples. There are also grammatical errors, massive overuse of punctuation and some instances that contain multiple languages which again make adding structure so that a computer can understand the text extremely complicated.

3. Methodology

The methodology discussed below outlines the implementation and process applied for the experiments in Section 4.1. The high-level process begins with reading in a dataset and cleaning it. The new features are then created and assigned from analysis of the sample. A dataset is then rebalanced and the text is vectorised with pre-processing applied. Feature reduction is implemented before the classification algorithm in use produces a trained model. This is evaluated using average recall scoring, cross-validated 10 times. The process from rebalancing onwards, shuffling the dataset before separating into new train and test sets, is repeated 10 times and the final score is averaged and output.

3.1. Dataset cleaning

Although each dataset originates from different sources the initial cleaning is very similar. Duplicate instances are removed from the dataset and the instance itself is cleaned. Additional features, discussed in section 3.3, are also constructed before further cleaning. The cleaning of individual instances primarily applies to text representation such as bag of words and N-gram vectorizers. By cleaning the instances in this manner it applies some extra structure to the text, allowing a vectorizer to represent an upper-case word in the same way as a lower-case word given that they are the same word. When given the same word where even one character is uppercase as opposed to lowercase, text representation vectorizers will view this as a new instance which will skew both text representation and reduce classification accuracy. Hyperlinks are removed, all characters are forced into lower case and any username with the prefix '@' is changed into a generic placeholder, '@username', to retain anonymity and to allow the text representation vectorizers to use each individual username as one all-inclusive feature. This cleaning is performed using the “text_instance_to_words” function, which also implements the “replace_all_usernames_and_RTs” function and “strip_links” function. The “strip_links” function, Figure 3.1, applies a regex or regular expression that replaces links in text with a comma. Figure 3.2 displays the “replace_all_usernames_and_RTs” function that removes the preceding “RRRTTT” from retweets and also replaces any username after the “@” symbol. Figure 3.3 implements both of the above functions in the “text_instance_to_words” function that can also remove stop words. Initial testing across all datasets indicated keeping stop words was the optimal route.

FIGURE 3.1. THE STRIP_LINKS FUNCTION

```
def strip_links(text):
    link_regex =
re.compile('((https?):(//)|(\\\\|\\\/))+([\\w\\d:#@%/;$()~_?\\+-
=\\\\.\\.&](#!)?))', re.DOTALL)
    links = re.findall(link_regex, text)
    for link in links:
        text = text.replace(link[0], ', ')
    return text
```

FIGURE 3.2. THE REPLACE_ALL_USERNAMES_AND_RTS FUNCTION

```
def replace_all_usernames_and_RTs(text):
    entity_prefixes = ['@']
    words = []
    for word in text.split():
        findReTweet = word[:6]
        if (findReTweet == "RRRTTT"):
            word = word[6:]
        word = word.strip()
        if word:
            if word[0] not in entity_prefixes:
                words.append(word)
            else:
                words.append('@username')
    return ' '.join(words)
```

FIGURE 3.3. THE TEXT_INSTANCE_TO_WORDS FUNCTION

```
def text_instance_to_words( raw_text ):
    cleaned_text =
    replace_all_usernames_and_RTs(strip_links(raw_text))
    letters_only = re.sub("[^a-zA-Z@*]", " ", cleaned_text)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    # remove stop words
    meaningful_words = [w for w in words if not w in stops]
    # return meaningful_words to remove stopwords from instance
    return " ".join(words)
```

During this step the window style instances in Dataset 3 were also separated and cleaned. Each instance in this dataset has an ID, in the format XXXXXXXX.YYYY.xml. The XXXXXXXX represents the comment thread number, while the YYYY represents each update as a new comment is added to the window. Each window contains 10 comments in the thread where the first comment is the new addition to the wind. This makes it relatively simple to parse the individual comments based on the difference between the contents of the current window and the next window, as shown in Figure 3.4. The labels change per the content of the window, also making it simple to determine the individual bullying content. This step also requires additional cleaning for the dataset as some individual comments contain only spaces.

FIGURE 3.4. A FOR LOOP TO SEPARATE THE MOVING WINDOW INSTANCES IN DATASET 3 INTO INDIVIDUAL COMMENTS

```
# separate individual sentences from windows
for i, row in dataset.iterrows():
    if i < dataset.shape[0]-1:
        vals = dataset["Text"][i].split(dataset["Text"][i+1][0:10])
        dataset.set_value(i, "Text", vals[0])
```

3.2. Feature Extraction and Construction

The features constructed from the original, raw instance text are explained in the following sections. They can be split into two general categories, the Syntactic Features and the Semantic Features. The Syntactic Features consist of basic syntactic information such as number of words, number of characters for each instance et cetera. The Semantic Features refers to the meaning of the language in the sample such as second and third person pronouns usages, swearing and uppercase letter usage et cetera.

FIGURE 3.5. ADDITIONAL FEATURE COLUMNS APPENDED TO THE PANDAS DATAFRAME FOR THE DATASET IN USE

```
# add the additional feature columns to the dataset DataFrame
dataset["Individual"] = pd.Series(0, index=dataset.index)
dataset["Swearing"] = pd.Series(0, index=dataset.index)
dataset["NumberOfCharacters"] = pd.Series(0, index=dataset.index)
dataset["NumberOfCapitals"] = pd.Series(0, index=dataset.index)
dataset["NumberOfPunctuation"] = pd.Series(0, index=dataset.index)
dataset["NumberOfWords"] = pd.Series(0, index=dataset.index)
dataset["NumberOfSwears"] = pd.Series(0, index=dataset.index)
dataset["SwearPercent"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["CapsPercent"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["PuncPercent"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["NegSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["PosSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["NeuSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["CompoundSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["SwearProximityFlag"] = pd.Series(0, index=dataset.index)
dataset["CountVictims"] = pd.Series(0, index=dataset.index)
dataset["CountSwearNearVictim"] = pd.Series(0, index=dataset.index)
dataset["VictimSwearRatio"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["NegVictimSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["PosVictimSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["NeuVictimSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
dataset["CompoundVictimSentiment"] = pd.Series(0,
index=dataset.index).astype('float64')
```

3.2.1 Syntactic Features

“NumberOfCharacters”, “NumberOfPunctuation” and “NumberOfWords” are the counts of the total character, punctuation marks and words respectively in an instance. These features stem from the initial dataset analysis in Section 2.3, where there can be differences in these averages between the positive and negative classes. These numerical features are

also used in the calculation of some of the following features, such as the percentage of swears or percentage of capital letters, which are both syntactic and semantic. Figure 3.6 and Figure 3.8 demonstrate the implementation of these features in Python.

“PuncPercent” is the percentage of punctuation marks in the instance. This feature was added for use instead of the Number of Punctuation feature, similarly to the “SwearPercent” and “CapsPercent” features in Section 3.2.2. A percentage of the punctuation marks present in a body of text is more accurate across a representation dataset when considering large differences in the size of individual text instances. It is calculated as the number of punctuation found divided by the total number of characters.

FIGURE 3.6. THE COUNTINTEXT FUNCTION

```
def countInText(text):  
    numChars = len(text);  
    numCaps = sum(1 for c in text if c.isupper());  
    numPunc = sum(1 for c in text if c in string.punctuation);  
    return numChars, numCaps, numPunc
```

3.2.2. Semantic Features

“NumberOfCapitals” and “NumberOfSwears” are counts of the characters that are in the upper case and swear words respectively. These features are based on the initial dataset analysis in Section 2.3, where there can be notable differences in these averages between the positive and negative classes. This is used in other additional features to calculate the percentage of capitals and swears in each instance, “CapsPercent” and “SwearPercent”. They are implemented in lieu of the outright counts considering potential differences in sample length skewing results.

“Individual” is a feature constructed using a dictionary of common first names derived from a list by the US Census Bureau [34] and an array of second and third person pronouns and their oft used misspellings. When one of either the common names, one of the pronouns list or ‘@username’ is found in the text, a binary flag is set from 0 to 1 to indicate that an individual has been singled out in the instance. This feature stems from the definitions of bullying and cyberbullying and represents the possible victim found in each instance. Figure 3.7 implements this feature, by setting the “victimFlagValue” to 1. This feature is implemented using the “flagForVictim” function, Figure 3.7.

“Swearing” is a feature constructed in a similar fashion to the “Individual” feature. By searching an instance against a list of swears and profanities obtained from noswearing.com [33], a binary flag is set from 0 to 1 when a swear is found as shown in Figure 3.8. Swearing is a common feature across previous research, discussed in Section 2.1.2. In the current study, it was the baseline feature as swearing was significantly higher in positive cases over negative cases in the majority of the datasets (see Figures 1,2,4,5,7,8,10 and 11 in Section 2.3). During initial research and development of code, the “Swearing” and “Individual” features were the first to be implemented, “Swearing” for the obvious distance across classes in the datasets

and Individual because of the necessity stemming from the definition of cyberbullying. Figure 3.8 implements this feature.

“NegSentiment”, “PosSentiment”, “NeuSentiment” and “CompoundSentiment” are the negative, positive, neutral and compound sentiment scores for the entire instance of text as returned by the VADER sentiment intensity analyser in the NLTK library [22]. Each is a scored metric from zero, no sentiment in the instance, to one, the instance is entirely composed of this given sentiment. For example, text scored as 0.0 positive, 0.0 neutral and 1.0 negative is extremely negative. Only the compound sentiment score differs. It is a scored metric from minus one to one. A minus one compound sentiment score indicates this is a completely negative instance. A one indicates the instance is completely positive. A zero indicates complete neutrality.

“SwearProximityFlag” is a feature that was implemented in line with the definition of cyberbullying, i.e. that cyberbullying requires a victim or to be directed abuse. This feature is very similar to the “Swearing” feature, but requires a swear word to be in proximity of a found second or third person pronoun or name. When a victim is found in the text, the terms surrounding it are analysed for swearing. If present, this binary flag is then set from zero to one. This feature was also included to prevent large bodies of text that contain no bullying or abuse being misclassified due to the presence of a profanity despite no victim being present. See Figure 3.7 for implementation.

“CountVictims”, “CountSwearNearVictim” and “VictimSwearRatio” were implemented to determine whether the number of victims or named persons, number of swear words near a found victim or the ratio of swears-to-victim in an instance had a definitive impact on the classification of an instance. It is a count of the number of victims found within the instance, as defined by the Individual feature previously. Figure 3.7 outlines their implementation.

“NegVictimSentiment”, “PosVictimSentiment”, “NeuVictimSentiment” and “CompoundVictimSentiment” are features that stem from the overall sentiment analysis features and the distance features like “SwearProximityFlag”. Unlike the overall sentiment features such as a “NegSentiment”, which analyses the entire instance, these features only analyse the text surrounding an individual as found per the “Individual” feature. This is to prevent instances that contain a large number of words and are generally positive despite a single, short instance that does contain bullying or abuse, being misclassified. The scoring and output is the same as the overall sentiment features mentioned above in this section, but the analysis is only being performed on the language surrounding the found second or third person pronoun or name. This feature is calculated within the “flagForVictim” function, Figure 3.7.

FIGURE 3.7. THE FLAGFORVICTIM FUNCTION

```
def flagForVictim(inputText):
    #initial cleaning of usernames
    text = replace_all_usernames_and_RTs(inputText)
    # the set of victim pronouns
    victimSet =
['@username','you','yourself','yours','her','hers','they','them','h
im','their','his','he','she','u','he\'s','she\'s','ur','your',
'you\'re','ya']
    # control variables used in the construction of multiple
features
    victimFlagValue=0;
    countVictims=0;
    countSwearNearVictim=0;
    swearFlagValue = 0;
    swearProximityFlag=0;
    sentimentPos = 0
    sentimentNeg = 0
    sentimentNeu = 0
    sentimentCompound = 0
    posTotal = 0
    negTotal = 0
    neuTotal = 0
    compoundTotal = 0

    words = text.split()
    # if a word is in the set of victim pronouns of common names,
    set the victim flag. Find the sentiment
    # intensity and polarity surrounding this potential victim. If
    there is a non-neutral sentiment, find
    # any swearing near the victim and set the relevant flag, swear
    count and totals for sentiment scores.
    for i in range(0, len(words)):
        word = words[i]
        if (word in victimSet) or (word in commonNames):
            victimFlagValue = 1;
            sentimentScores =
sentimentIntensityAnalyzer.polarity_scores(' '.join(words[i-
2:i+7]))

            if (sentimentScores.get("compound") != 0.0):
                countVictims += 1;
                swearFlagValue = flagForSwearing(' '.join(words[i-
2:i+7]))

                if (swearFlagValue[0] == 1):
                    countSwearNearVictim += swearFlagValue[1]

                posTotal += sentimentScores.get("pos");
                negTotal += sentimentScores.get("neg");
                neuTotal += sentimentScores.get("neu");
                compoundTotal += sentimentScores.get("compound");

    # set the swear proximity flag, whether or not there is swear
    near a found victim
    if (countSwearNearVictim > 0):
```

```

        swearProximityFlag = 1;

    # the average sentiment found around victims mentioned.
    if(countVictims > 0):
        sentimentPos = posTotal/float(countVictims)
        sentimentNeg = (negTotal/float(countVictims))
        sentimentNeu = (neuTotal/float(countVictims))
        sentimentCompound = (compoundTotal/float(countVictims))

    return victimFlagValue, swearProximityFlag, countVictims,
countSwearNearVictim, sentimentPos, sentimentNeg, sentimentNeu,
sentimentCompound

```

FIGURE 3.8. THE FLAGFORSWEARING FUNCTION

```

def flagForSwearing(text):
    # return variables
    flagValue=0;
    swearCount = 0;
    wordCount = 0;

    # if a swear is found, assign flags and counts
    for word in text.split():
        word = word.strip()
        word = word.lower()
        wordCount +=1;
        if word:
            if word in swears:
                flagValue = 1;
                swearCount += 1;

    return flagValue, swearCount, wordCount

```

3.3. Dataset Rebalancing

Resampling is performed across the positive class in the training set to balance each dataset. However, while resampling increases the average recall or average accuracy of classification of the positive class by a large amount, it begins to reduce the average recall of the negative class. It is necessary to maintain a balance between improving positive average recall and reducing negative average recall. In their paper [12] utilising the same datasets Chen, McKeever and Delany found an optimal positive to negative ratio for each dataset where resampling is concerned. This project uses the same ratio across each implemented dataset. As per Chen, McKeever and Delaney [12] the ratios of positive and negative classes before and after rebalancing are shown in Figure 3.9 below:

FIGURE 3.9. THE RATIO OF POSITIVE (+) AND NEGATIVE (-) CLASS INSTANCES IN DATASETS 1, 2 AND 4 BEFORE AND AFTER RESAMPLING

	Dataset 1	Dataset 2	Dataset 4
	(+/-)	(+/-)	(+/-)
Before Resample	13% / 87%	12% / 88%	1% / 99%
After Resample	36% / 62%	35% / 65%	9% / 91%

Dataset 3 was deemed to be acceptably balanced and no resampling is performed on this dataset. The rebalancing, as discussed above is only performed on the training set being used during classification. This is vital to note. Resampling across the entire data set before separation into test and training sets, which will be explained in the text representation step, will cause multiple instances to appear in both training and test set. This causes greatly skewed results as the test set is now no longer “unseen”, the model has effectively been trained using the exact instance it is being asked to predict and can do so incredibly accurately. The resampling is performed across the training folds of the dataset. The training set is the sorted by its label or class. The negative samples are returned normally to the training set. The positive samples are oversampled based on the optimum rebalancing ratios outlined in Figure 3.9. This rebalancing is implemented as shown in Figure 3.10, with differing amounts of oversampling via multiplication of the positive class.

FIGURE 3.10. PYTHON CODE IMPLEMENTED TO APPLY OVERSAMPLING TO REBALANCE A DATASET

```
#resample training dataset
neg_Class = train_data.loc[train_data.Label == 0]
pos_Class = train_data.loc[train_data.Label == 1]
resampleResult = pd.DataFrame()
train_data =
resampleResult.append([neg_Class.sample(n=neg_Class.shape[0],
replace=False).reset_index(drop=True),
                        pos_Class.sample(n=pos_Class.shape[0]*4,
replace=True).reset_index(drop=True)])
```

3.4. Dataset Pre-processing

A minimum document frequency value is applied to the dataset, this is set to 0.005 or 0.5%. This applies a threshold on terms that do not appear in at last 0.5% of the instances in the dataset. A maximum document frequency value of 0.995 or 99.5% is also applied and removes terms that appear in 99.5% of the instances in the dataset. These thresholds remove the terms that appear in almost all of the instances in the dataset and those terms that appear so few times that they would have no contribution towards classification from the text alone. This causes a significant reduction in the size of the feature set constructed from text representation alone. This can be seen in Figure 3.15 in Section 3.5, as the document frequency minimum and maximum is applied during text vectorisation.

3.5. Text representation

There are four different ways the text is represented. The first is in the simplest form, Bag of Words. This is implemented with Sci-Kit Learn's CountVectorizer module. As the name implies, it is a count of all individual terms in an instance and returns a large, sparse feature array. The second is an N-gram representation using bigrams. This is again applied using Sci-Kit Learn's CountVectorizer, albeit with different input values declared in the call to the function specifying bigrams. The trigram representation is the third format the text was interpreted in, with the call to the CountVectorizer displaying the change in input variables. Finally, a character N-gram representation was implemented. This was limited to four characters representing one term to be vectorised. As explained in Section 2.2.1.3, the various text representations are all, essentially, counts of the terms in the text. The manner in how a term is defined is what changes between the different representations. Within the vectorizer, there is also the application of the minimum and maximum document frequency thresholds as outlined in Section 3.4.

Also applied when representing text is the normalisation. This applies the term frequency normalisation, discussed in Section 2.2.1.3, to the vectorised text using the TfidfVectorizer module from Sci Kit Learn. This vectorizer has been implemented so that it acts exactly as the CountVectorizer in Bag of Words operation that was previously used, but is implemented to apply the normalisation.

As the feature set has changed dramatically in this step of the classification process, some housekeeping was necessary. After the text instances have been vectorised by whatever current vectorizer is being implemented, the additional features that were constructed based on the original text instance need to be joined to this set. This is a simple mapping operation applied using the SciPy library's hstack operation from the scipy.sparse module. This allows the horizontal stacking of an array to another, basically a row-wise append of the arrays given.

FIGURE 3.11. A BAG OF WORDS COUNT VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON

```
vectorizer = CountVectorizer(min_df=0.005, max_df=0.995)
```

FIGURE 3.12. A UNIGRAM TO BIGRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON

```
vectorizer = CountVectorizer(ngram_range=(1, 2),  
min_df=0.005, max_df=0.995)
```

FIGURE 3.13. A UNIGRAM TO TRIGRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON

```
vectorizer = CountVectorizer(ngram_range=(1, 3),
min_df=0.005, max_df=0.995)
```

FIGURE 3.14. A 4 CHARACTER N-GRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON

```
vectorizer = CountVectorizer(analyzer='char',
ngram_range=(4, 4), min_df=0.005, max_df=0.995)
```

FIGURE 3.15. A TERM FREQUENCY VECTORIZER ACTING AS A BAG OF WORDS VECTORIZER APPLYING MINIMUM AND MAXIMUM DOCUMENT FREQUENCY AND EUCLIDIAN DISTANCE NORMALISATION

```
vectorizer = TfidfVectorizer(min_df=0.005, max_df=0.995,
norm='l2', use_idf=True, smooth_idf=True)
```

FIGURE 3.16. USING THE VALUES FUNCTION OF PANDAS DATAFRAMES AND SCI-PY'S HSTACK FUNCTION TO RE-JOIN THE CONSTRUCTED FEATURES WITH THE VECTORISED TEXT

```
#get additional features
train_features = train_data[["Individual", "Swearing",
"SwearPercent", "CapsPercent", "PuncPercent", "PosSentiment",
"NegSentiment", "NeuSentiment", "VictimSwearRatio",
"NeuVictimSentiment", "PosVictimSentiment", "NegVictimSentiment",
"CompoundSentiment", "CompoundVictimSentiment"]].values
test_features = test_data[["Individual", "Swearing",
"SwearPercent", "CapsPercent", "PuncPercent", "PosSentiment",
"NegSentiment", "NeuSentiment", "VictimSwearRatio",
"NeuVictimSentiment", "PosVictimSentiment", "NegVictimSentiment",
"CompoundSentiment", "CompoundVictimSentiment"]].values

trainMapped = sparse.hstack((trainMapped, train_features))
testMapped = sparse.hstack((testMapped, test_features))
```

3.7. Feature Reduction

Chi Squared feature selection, detailed in Section 2.2.1.3, was implemented as a feature reduction mechanism post text vectorisation and feature addition. The highest scoring k number of features are selected by the Chi Squared algorithm, where k was determined

during testing. This was implemented using Sci-Kit Learn's SelectKBest function and chi2 class imported from the sklearn.feature_selection module. The value of k is a percentage based on the size of the feature set.

FIGURE 3.17. AN IMPLEMENTATION OF CHI SQUARE FEATURE SELECTION USING THE SELECTKBEST FUNCTION

```
size = (trainMapped.shape[1]*9)/10
ch2 = SelectKBest(chi2, k=size)

trainMapped = ch2.fit_transform(trainMapped,
train_data["Label"])
testMapped = ch2.transform(testMapped)
```

3.6. Classification

Five different classification algorithms were used throughout the project. They include a Naïve Bayes classifier, a Decision Tree classifier, a Logistic Regression classifier, a Linear Support Vector Machine and a non-Linear Support Vector Machine. All classifiers were implemented using Sci-Kit Learn modules.

The Naïve Bayes classifier is implemented using the MultinomialNB module from sklearn.naive_bayes. This is the simplest of the classifiers to implement, as the default options in the class are correct for use. After initialisation, the classifier is trained using the “fit” method, which takes 2 arguments, the training dataset and the label for this dataset. With the trained classifier, the predict method is called, passing the test set to the trained model. This returns a label prediction for each instance in the test set in an array.

FIGURE 3.18. INITIALISING AND TRAINING A NAÏVE BAYES CLASSIFIER IN PYTHON

```
# declare and train the classifier
classifier = MultinomialNB()
classifier.fit(trainMapped, train_data["Label"])
# make predictions on test set
prediction = classifier.predict(testMapped)
```

The Decision Tree Classifier is imported from the sklearn.tree module and is trained in the same way as the MultinomialNB classifier, applying the fit method to the training set. There are optional variables when initialising the DecisionTreeClassifier class that need to be addressed. As explained in Section 2.2.1.2.4., there are multiple scoring methods that can be applied to classifier to determine the root node and nodes at each step in the tree making process. This is determined in this class as the “criterion” variable which can be either “gini” or “entropy”, both of which are also discussed in Section 2.2.1.2.4. Another input variable is the “max_features” features variable. This decides the maximum number of features to be tested when looking for the best split on a node. Predict is also used to return an array of class labels as determined by the trained model.

FIGURE 3.19. INITIALISING AND TRAINING A DECISION TREE CLASSIFIER IN PYTHON

```
# declare and train the classifier
classifier = DecisionTreeClassifier(criterion="gini",
max_features=None)
classifier.fit(trainMapped, train_data["Label"])
# make predictions on test set
prediction = classifier.predict(testMapped)
```

The Logistic Regression classifier is instantiated using the Sci-Lit Learn class `SGDClassifier` imported from the `skleran.linear_model` module. There are many available optional variables that can be implemented with this class, as it is multifunctional. This is primarily implemented to apply stochastic gradient descent learning, as explained in Section 2.2.1.2.3. However, setting the “loss” variable to “log” returns a probabilistic logistic regression classifier. The “penalty” argument is set to “elasticnet” to allow for sparsity in the model, which is vital when the text representation applied returns large sparse feature arrays.

FIGURE 3.20. INITIALISING AND TRAINING A LOGISTIC REGRESSION CLASSIFIER IN PYTHON

```
# declare and train the classifier
classifier = SGDClassifier(loss='log', penalty='elasticnet')
classifier.fit(trainMapped, train_data["Label"])
# make predictions on test set
prediction = classifier.predict(testMapped)
```

The Linear SVM classifier is imported from the `sklearn.svm` module as `LinearSVC`. The class has optional arguments such as “C”, which determines the margin of allowed error in the classification, the size of the hyperplane that splits the classes. See Section 2.2.1.2.2 for more on Linear SVMs. A C value of 0.01 was found to be optimum for classification accuracy. “Dual” is another optional argument, that tells the classifier to solve either the dual or primal optimisation problem, which should be set to false given a dataset that contains more samples than features. It is for this reason that dual is set to false. The dual-primal optimisation problem is, as the name suggests, concerned with optimisation. As the datasets in use a very small comparatively the optimisation of their vector calculation is not a primary concern.

FIGURE 3.21. INITIALISING AND TRAINING A LINEAR SUPPORT VECTOR MACHINE CLASSIFIER IN PYTHON

```
# declare and train the classifier
classifier = LinearSVC(C=0.01, dual=False)
classifier.fit(trainMapped, train_data["Label"])
# make predictions on test set
prediction = classifier.predict(testMapped)
```

The non-Linear SVM classifier is imported from the `sklearn.svm` module simply as `SVC`. Again, this class has many optional arguments that change the functionality of the returned

instance. Similarly to the Linear SVM classifier, “C” is present, but a much larger value of optimum C has been found. The chosen kernel is among the most popular non-linear SVM kernels used in machine learning, rbf or radial basis function. The “gamma” value is a specific optional argument that is necessary when implementing an rbf kernel non-linear SVM. Gamma is the standard deviation of the radial basic function, the rbf kernel, and determines how far the influence of a single training instances reaches, i.e. how the grouping of similar instances is affected. The “auto” setting sets this to a value of 1 divided by the number of features, which has been the optimal value.

FIGURE 3.22. INITIALISING AND TRAINING A NON-LINEAR SUPPORT VECTOR MACHINE CLASSIFIER IN PYTHON

```
# declare and train the classifier
classifier = SVC(kernel="rbf", C=100.0, gamma="auto")
classifier.fit(trainMapped, train_data["Label"])
# make predictions on test set
prediction = classifier.predict(testMapped)
```

3.8. Scoring

Classifiers are evaluated using average recall scoring, also known as average accuracy scoring. This is implemented using Sci-Kit Learn’s `recall_score` method imported from the `sklearn.metrics` module. This returns an accuracy for each class using the formula:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad [35]$$

Where the true positives are correctly predicted positive instances and the false negatives are incorrectly labelled as negative but are in fact positive. The `recall_score` method returns an array of the recall scores for the respective classes in the form [X, Y] where X is the negative class recall and Y is positive class recall. Averaging these two scores provides the average recall score. However, the main focus of this type of classification is on the positive class. Relatively speaking, the accuracy of the positive class is far more important than the accuracy of the negative class. A false positive from the negative class, i.e. classifying non-bullying content as bullying is far less detrimental to users than a false negative, i.e. the classification of bullying content as non-bullying.

FIGURE 3.23. RETURNING THE RECALL AND STORING

```
average_recall = recall_score(test_data["Label"], prediction,
                             average=None)

positive_recall_scores += [average_recall[1]]
negative_recall_scores += [average_recall[0]]
```

3.9. Cross Validation

10-Fold cross validation is implemented manually. This process is then performed 10 times. The dataset is randomised, or shuffled, by using the “sample” method available to Pandas DataFrames. Using the “reset_index” argument “drop=True” ensures full randomisation across the dataset by removing previous indexing. With this shuffled dataset, the folds are separated into 10 equally sized subsets using the DataFrame slicing functionality offered by Pandas. The folds, now ranging from Fold1 to Fold10, are used interchangeably as the test and training sets for finding recall scores as demonstrated in Section 3.8. When one of the folds is used as test data, the remaining nine folds are used as training data. This training data is the data resampled in Section 3.3. The test fold is the set classified by the predict method from the classifier defined in Section 3.9. Each fold is used as test data once and there are 10 recall score arrays returned in total. The positive and negative class recall scores are averaged using the recall score arrays returned, providing overall positive and negative class average recall. This can then be averaged again to provide the overall average recall for this model. This score can be then taken as accurate as it has been cross validated using 10-Fold cross validation.

FIGURE 3.24. MANUAL CROSS-VALIDATION IMPLEMENTATION

```
# initialise the final scoring array
output = np.zeros((2, 11))

for i in range(1,11):
    # shuffle the dataset
    dataset = dataset.sample(frac=1).reset_index(drop=True)
    # initialise the scoring arrays
    positive_recall_scores = []
    negative_recall_scores = []

    # separate into folds
    foldAmount = (len(dataset.index))/10
    fold1 = dataset[:foldAmount]
    fold2 = dataset[foldAmount:foldAmount*2]
    fold3 = dataset[(foldAmount*2):foldAmount*3]
    fold4 = dataset[(foldAmount*3):foldAmount*4]
    fold5 = dataset[(foldAmount*4):foldAmount*5]
    fold6 = dataset[(foldAmount*5):foldAmount*6]
    fold7 = dataset[(foldAmount*6):foldAmount*7]
    fold8 = dataset[(foldAmount*7):foldAmount*8]
    fold9 = dataset[(foldAmount*8):foldAmount*9]
    fold10 = dataset[(foldAmount*9):]

    # get the recall scores across each fold.
    recallFold1 = [(getRecallScores(fold1, fold2, fold3, fold4,
    fold5, fold6, fold7, fold8, fold9, fold10))]
    recallFold2 = [(getRecallScores(fold2, fold1, fold3, fold4,
    fold5, fold6, fold7, fold8, fold9, fold10))]
    recallFold3 = [(getRecallScores(fold3, fold2, fold1, fold4,
    fold5, fold6, fold7, fold8, fold9, fold10))]
    recallFold4 = [(getRecallScores(fold4, fold2, fold3, fold1,
    fold5, fold6, fold7, fold8, fold9, fold10))]
    recallFold5 = [(getRecallScores(fold5, fold2, fold3, fold4,
    fold1, fold6, fold7, fold8, fold9, fold10))]
    recallFold6 = [(getRecallScores(fold6, fold2, fold3, fold4,
    fold5, fold1, fold7, fold8, fold9, fold10))]
    recallFold7 = [(getRecallScores(fold7, fold2, fold3, fold4,
    fold5, fold6, fold1, fold8, fold9, fold10))]
    recallFold8 = [(getRecallScores(fold8, fold2, fold3, fold4,
    fold5, fold6, fold7, fold1, fold9, fold10))]
    recallFold9 = [(getRecallScores(fold9, fold2, fold3, fold4,
    fold5, fold6, fold7, fold8, fold1, fold10))]
    recallFold10 = [(getRecallScores(fold10, fold2, fold3, fold4,
    fold5, fold6, fold7, fold8, fold9, fold1))]

    # average recall
    overall_positive_average_recall =
np.mean(positive_recall_scores)
    overall_negative_average_recall =
np.mean(negative_recall_scores)

    # set scores in the final scoring array
    output[0][i] = overall_positive_average_recall
    output[1][i] = overall_negative_average_recall
```

3.10. Final evaluation

While the 10-Fold cross validation and scoring can be taken as accurate, for the sake of consistency, it is applied 10 times when scoring the differing combinations of models, features, text representations, pre-processing and feature reductions methods. Results should be repeatable and the shuffling of each dataset could have a very significant effect on the split of positive to negative samples in each fold. This randomisation and splitting happens before the resample which is incredibly important. Resampling before randomising the dataset and splitting it into folds would cause repeat instances across the train and test data, effectively removing the point of test data. The cross-validation process from Section 3.9 is repeated 10 times. The average recall scores determined across each shuffling of the dataset are themselves averaged across this 10-time repetition. This is the final score used when determining the accuracy of the processes used when creating this model in this dataset.

FIGURE 3.25. FINAL OVERALL SCORING OUTPUT

```
outputDatasetName =  
str(os.path.splitext(input_file)[0]).rsplit('/', 1)[1]  
output[0][0] = np.mean(output[0][1:10])  
output[1][0] = np.mean(output[1][1:10])  
  
outputDf = pd.DataFrame(output)  
timeStamp = str(datetime.datetime.now()).replace(".", "-")  
".replace(":", "-")  
outputDf.columns = ["Overall Average", "Round 1", "Round 2", "Round  
3", "Round 4", "Round 5", "Round 6", "Round 7", "Round 8", "Round 9",  
"Round 10"]  
outputDf = outputDf.rename(index={0: 'Positive Avg'})  
outputDf = outputDf.rename(index={1: 'Negative Avg'})  
outputDf.to_csv("../Results/"+ outputDatasetName + "AccuracyResults  
" + timeStamp + ".csv", sep='\t')
```

4. Experimental Process

4.1. Experiments

Across each experiment, unless explicitly stated otherwise, the individual text instances were cleaned as described in Section 3.1 and each outcome was evaluated as also described in Section 3.9 and Section 3.10, implementing 10-Fold cross-validation repeated 10 times and averaged. As discussed in Section 3.3, dataset rebalancing has been implemented where applicable. The focus of the Experiments rests primarily upon the Positive Class accuracy measurements. This is due in part to the dataset class imbalance but also because of the importance of correct classification in the positive class versus misclassification of the negative class instances. Considering the impact and the nature of the material being classified, a false positive prediction for a negative instance is far less detrimental than a false negative prediction for a positive instance. For these reasons, the primary aim is to increase the accuracy of the positive class predictions, while a secondary aim is to retain the high accuracy in the negative class.

4.1.1. Experiment 1 – Classifiers

The first experiment performed was to determine the most accurate classifier. This was performed on all datasets, in the simplest modes of operation in terms of pre-processing, term representation and feature selection. The initial experiment applied no pre-processing via maximum and minimum document frequency term reduction, no additional features were added and the terms were represented using bag of words. Five classifiers were tested: Naïve Bayes, Decision Tree, Linear SVM, Logistic Regression and a non-Linear SVM. The process applied in Experiment 1 is outlined in Section 3.6 in more detail. The outputs of this experiment are displayed below in Figure 4.1 to Figure 4.4 which represent Dataset 1 to Dataset 4 respectively.

FIGURE 4.1. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 1

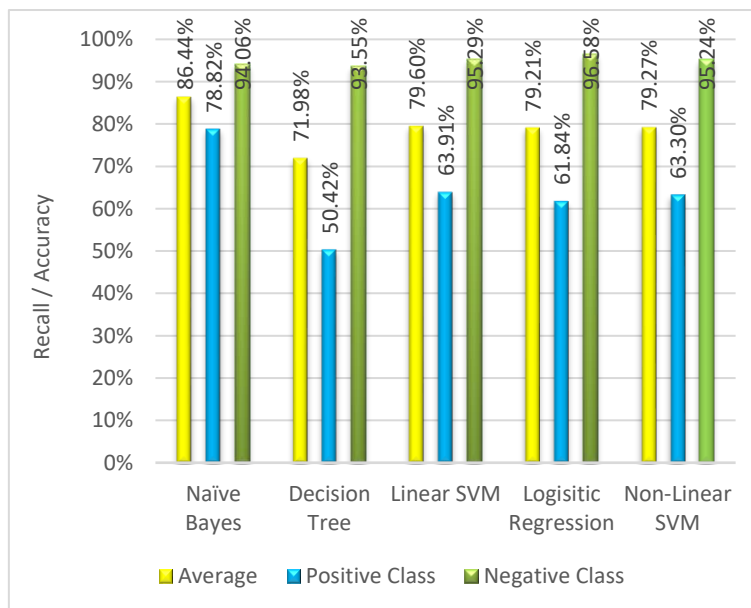


FIGURE 4.2. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 2

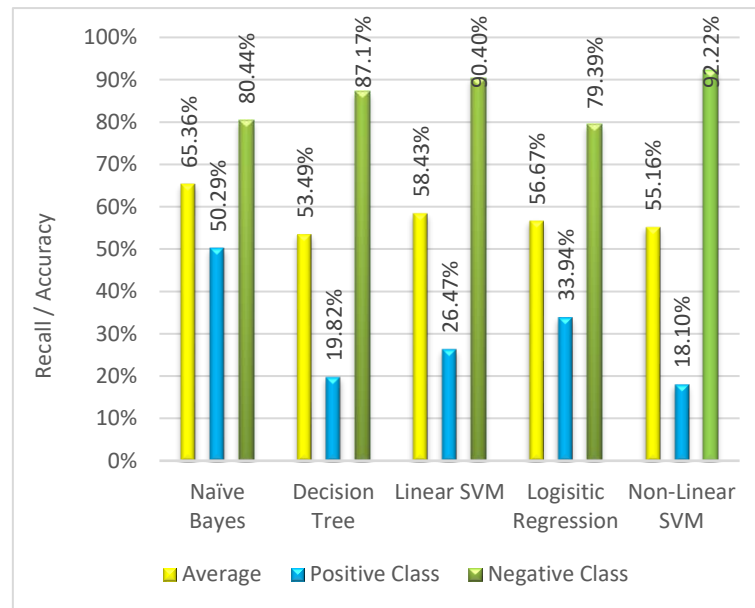


FIGURE 4.3. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 3

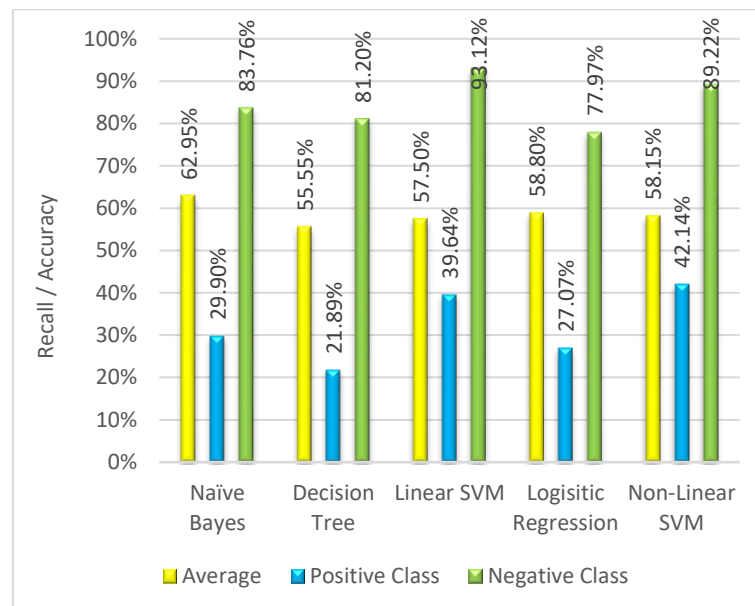
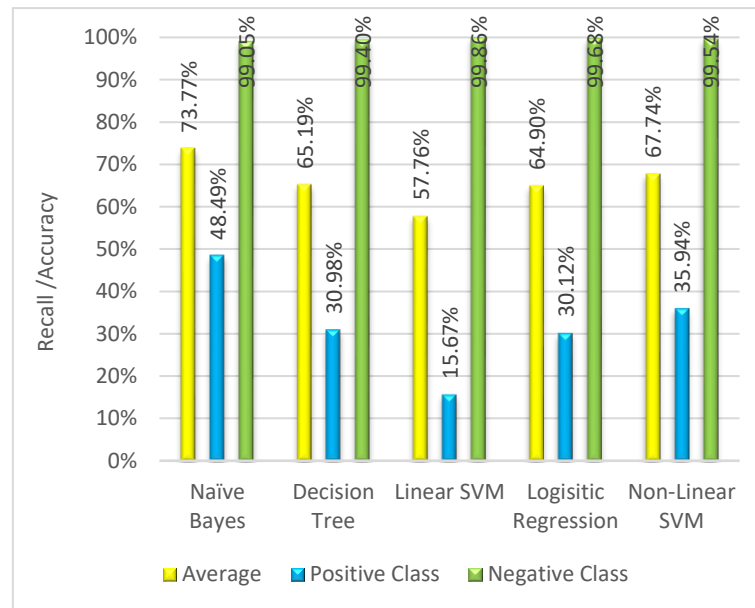


FIGURE 4.4. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 4



4.1.2. Experiment 2 – Text Representation

The second experiment was to determine the most accurate term representation. Based on the outcome of the previous experiment, this was implemented exclusively with the best performing classification algorithm in each dataset, Naïve Bayes. There were 4 types of text representation tested, based on text representation applied in previous research as discussed in Section 2. Bag of words, unigram to bigram, unigram to trigram and four-character N-gram were applied to each dataset. No pre-processing was applied, no additional features were added and the instances in each dataset were cleaned as before. The implementation of this experiment is discussed in Section 3.5 in particular. In the cases of unigram and bigram and unigram to trigram, this means that the vectorizer applies not just bigram or trigram representation. For example, Unigram to Trigram is bag of words (unigram), bigram and trigram representation of each instance. Figure 4.5 to Figure 4.8 show the outcomes of Experiment 2 and represent Dataset 1 to Dataset 4 respectively.

FIGURE 4.5. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 1

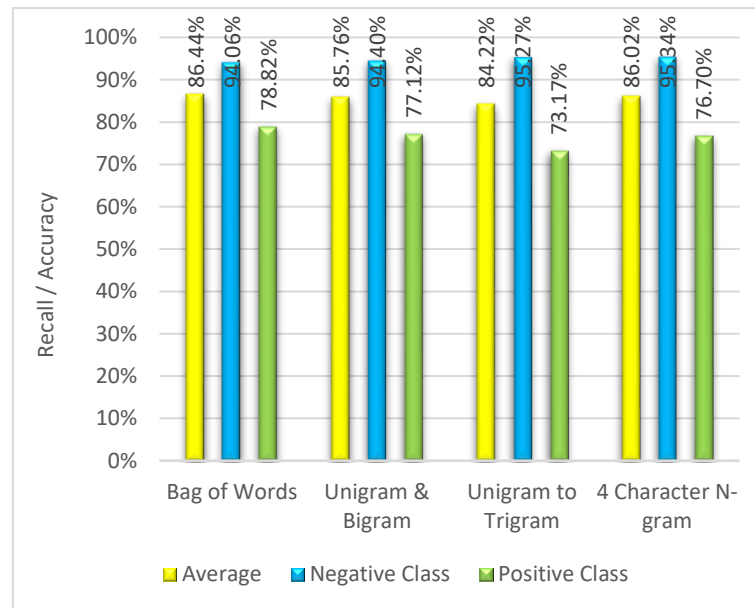


FIGURE 4.6. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 2

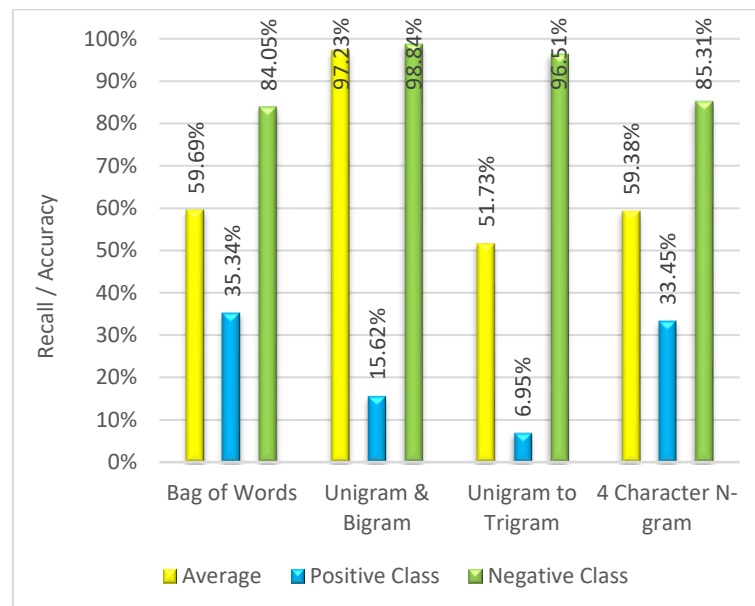


FIGURE 4.7. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 3

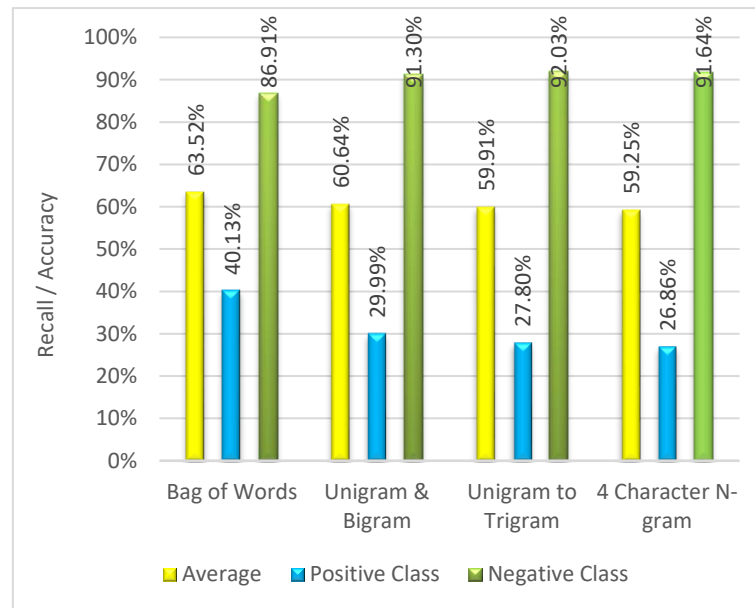
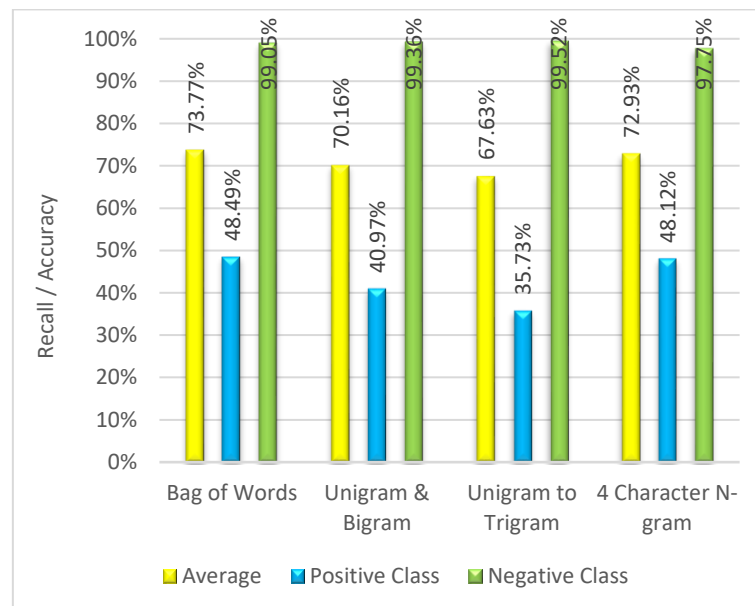


FIGURE 4.8. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 4



4.1.3. Experiment 3 – Dataset Pre-Processing

Pre-processing was performed by applying a minimum and maximum document frequency threshold to each dataset, see Section 3.4 for a detailed description of the implementation. Using the highest accuracy classifier and highest accuracy text representation, along with no additional features, different values of minimum and maximum

document frequency were tested. This was scored with the previously defined evaluation method for accuracy scoring. Figure 4.9 contains the output from Experiment 3.

FIGURE 4.9. NUMBER OF FEATURES, POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING VARYING AMOUNTS OF DOCUMENT FREQUENCY THRESHOLDS FOR DATASETS 1 - 4

		No reduction	Minimum document frequency = 1%, Maximum document frequency = 99%	Minimum document frequency = 0.5%, Maximum document frequency = 99.5%	Maximum document frequency = 95%
Dataset 1	Average Accuracy	86.44%	82.14%	84.31%	84.99%
	Positive Accuracy	78.82%	73.07%	75.67%	74.95%
	Negative Accuracy	94.06%	91.21%	92.96%	95.03%
	Number of features	4325	175	350	4325
Dataset 2	Average Accuracy	59.69%	65.36%	63.75%	59.19%
	Positive Accuracy	35.34%	50.29%	45.28%	34.19%
	Negative Accuracy	84.05%	80.44%	82.21%	84.20%
	Number of features	42350	1600	2820	42350
Dataset 3	Average Accuracy	63.52%	60.22%	62.95%	63.00%
	Positive Accuracy	40.13%	37.40%	42.14%	40.03%
	Negative Accuracy	86.91%	83.03%	83.76%	85.97%
	Number of features	10250	580	1100	10250
Dataset 4	Average Accuracy	73.77%	65.01%	38.93%	73.11%
	Positive Accuracy	48.49%	30.91%	38.93%	47.15%
	Negative Accuracy	99.05%	99.11%	98.89%	99.08%
	Number of features	3500	90	190	3500

4.1.4. Experiment 4 – Feature Addition

Features were tested using the optimal combination of classifier, text representation and pre-processing applied to the dataset as found from the previous experiments. The features were added incrementally one by one, and where an increase in the positive class accuracy was observed, the feature was retained in the set and another new feature was added for testing. When a decrease in the accuracy occurred, the feature was omitted from this contiguous set. Section 3.2 and Section 3.5 explain in further detail some of the processes and implementations in Experiment 4. Figure 4.10 to Figure 4.13 display the results of Experiment 4 for Datasets 1 to 4 respectively.

FIGURE 4.10. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 1

Feature Added	Positive Class Accuracy	Negative Class Accuracy
None	78.82%	94.06%
Swearing Flag	80.44%	94.27%
Individual Found	81.41%	93.83%
Caps Percent	81.79%	93.90%
Punctuation Percent	82.41%	93.89%
Swear Percent	82.37%	94.02%
Swear Proximity Flag	81.23%	94.18%
Victim Swear Ratio	81.94%	94.00%
Number of Swears	81.10%	94.21%
Overall Positive Sentiment	81.84%	94.27%
Overall Negative Sentiment	82.76%	93.57%
Overall Neutral Sentiment	82.10%	96.97%
Overall Compound Sentiment	58.26%	96.97%
Positive Victim Sentiment	81.97%	94.03%
Negative Victim Sentiment	83.24%	93.40%
Neutral Victim Sentiment	83.73%	93.50%
Compound Victim Sentiment	63.04%	97.11%
Number Of Words	81.76%	94.42%

FIGURE 4.11. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 2

Feature Added	Positive Class Accuracy	Negative Class Accuracy
None	50.29%	80.44%
Swearing Flag	50.99%	80.21%
Individual Found	48.70%	81.63%
Caps Percent	50.12%	80.18%
Punctuation Percent	49.56%	80.85%
Swear Percent	49.31%	80.09%
Swear Proximity Flag	49.26%	80.56%
Victim Swear Ratio	50.00%	79.85%
Number of Swears	51.11%	79.65%
Overall Positive Sentiment	51.16%	80.29%
Overall Negative Sentiment	51.47%	79.75%
Overall Neutral Sentiment	49.96%	80.64%
Overall Compound Sentiment	50.39%	77.03%
Positive Victim Sentiment	51.21%	80.34%
Negative Victim Sentiment	50.37%	79.90%
Neutral Victim Sentiment	50.88%	80.59%
Compound Victim Sentiment	48.49%	77.84%
Number Of Words	51.51%	79.89%

FIGURE 4.12. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 3

Feature Added	Positive Class Accuracy	Negative Class Accuracy
None	42.14%	83.76%
Swearing Flag	42.91%	85.36%
Individual Found	38.29%	86.08%
Caps Percent	43.01%	85.73%
Punctuation Percent	42.49%	85.30%
Swear Percent	39.05%	85.34%
Swear Proximity Flag	39.59%	85.18%
Victim Swear Ratio	43.60%	85.31%
Number of Swears	44.42%	85.49%
Overall Positive Sentiment	40.01%	85.31%
Overall Negative Sentiment	41.12%	85.00%
Overall Neutral Sentiment	38.57%	86.18%
Overall Compound Sentiment	27.65%	87.51%
Positive Victim Sentiment	39.98%	85.68%
Negative Victim Sentiment	44.53%	85.09%
Neutral Victim Sentiment	42.40%	85.41%
Compound Victim Sentiment	25.09%	89.69%
Number Of Words	44.58%	85.52%

FIGURE 4.13. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 4

Feature Added	None	Swearing Flag	Individual Found	Caps Percent	Punctuation Percent	Swear Percent	Swear Proximity Flag	Victim Swear Ratio	Number of Swears	Overall Positive Sentiment	Overall Negative Sentiment	Overall Neutral Sentiment	Overall Compound Sentiment	Positive Victim Sentiment	Negative Victim Sentiment	Neutral Victim Sentiment	Compound Victim Sentiment	Number Of Words
Positive Class Accuracy	48.49%	50.65%	45.11%	48.20%	45.22%	47.26%	44.67%	50.83%	51.76%	48.84%	48.06%	46.78%	23.26%	51.24%	48.03%	54.63%	18.27%	48.80%
Negative Class Accuracy	99.05%	98.95%	99.08%	98.98%	98.99%	98.92%	98.94%	98.95%	98.89%	98.87%	98.83%	99.04%	99.33%	98.91%	98.79%	98.85%	99.42%	99.22%

4.1.5. Experiment 5 – Feature Reduction

The Chi Squared feature selection algorithm was implemented in this experiment. It allows for specifying the size of the feature set to be selected. This is applied after the optimal classifier, text representation, pre-processing and additional features have been applied to each dataset respectively. The feature set size is reduced in increments of 10% of the overall dataset sample size. See Section 3.7 for a detailed description of the implementation of Experiment 5. Figure 4.14 displays the outputs from Experiment 5.

FIGURE 4.14. NUMBER OF FEATURES, POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING VARYING VALUES OF FEATURE REDUCTION FOR DATASETS 1 - 4

		No reduction	10% Reduction	20% Reduction	30% Reduction	40% Reduction	50% Reduction	60% Reduction	70% Reduction	80% Reduction	90% Reduction
Dataset 1	Average Accuracy	88.61%	88.01%	88.09%	88.05%	87.61%	87.21%	87.67%	86.73%	86.27%	85.58%
	Positive Accuracy	83.73%	82.76%	82.47%	82.13%	81.18%	80.05%	80.72%	78.63%	77.56%	76.74%
	Negative Accuracy	93.50%	93.26%	93.71%	93.97%	94.05%	94.37%	94.62%	94.82%	94.98%	94.43%
	Number of features	4350	3950	3470	3030	2600	2165	1740	1305	874	435
Dataset 2	Average Accuracy	65.70%	65.46%	65.60%	65.50%	65.51%	65.24%	65.42%	64.46%	63.92%	64.88%
	Positive Accuracy	51.51%	50.99%	51.56%	51.27%	51.27%	50.79%	50.77%	48.75%	45.84%	40.13%
	Negative Accuracy	79.89%	79.92%	79.65%	79.73%	79.74%	79.69%	80.08%	80.17%	82.00%	89.62%
	Number of features	1600	1450	1310	1110	960	795	640	490	290	105
Dataset 3	Average Accuracy	65.05%	62.50%	63.44%	62.72%	63.01%	63.03%	62.05%	62.30%	60.97%	61.19%
	Positive Accuracy	44.58%	39.82%	41.65%	40.49%	41.01%	40.82%	38.82%	36.45%	32.32%	32.39%
	Negative Accuracy	85.52%	85.18%	85.24%	84.96%	85.01%	85.24%	85.27%	88.15%	89.61%	89.99%
	Number of features	1100	990	880	785	670	560	445	320	210	106
Dataset 4	Average Accuracy	76.74%	75.85%	75.97%	74.94%	74.90%	74.90%	74.38%	74.12%	74.46%	73.83%
	Positive Accuracy	54.63%	52.84%	53.07%	51.09%	51.09%	51.10%	50.11%	49.69%	50.26%	48.98%
	Negative Accuracy	98.85%	98.85%	98.86%	98.79%	98.71%	98.70%	98.65%	98.54%	98.67%	98.67%
	Number of features	3500	3160	2790	2440	2110	1750	1380	1070	690	350

4.1.6. Experiment 6 – Text Representation Normalisation

In this experiment, normalisation was applied to the vectorised feature set of terms. All previous experiment outcomes were taken into account, with best possible pre-processing, text vectorizer, classification algorithm, feature set and level of feature reduction applied in each dataset. Section 3.5 contains further information on the process applied for Experiment 6 and Figure 4.15 contains the results of Experiment 6 for each dataset.

FIGURE 4.15. POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING DIFFERENT NORMALISATION TO VECTORISED TEXT FOR DATASETS 1 - 4

		No Normalisation	Euclidian distance normalisation	Vector norm normalisation
Dataset 1	Average Accuracy	88.61%	85.00%	73.36%
	Positive Accuracy	83.73%	73.25%	48.31%
	Negative Accuracy	93.50%	96.75%	98.41%
Dataset 2	Average Accuracy	65.05%	54.48%	51.67%
	Positive Accuracy	51.51%	42.83%	41.01%
	Negative Accuracy	79.89%	86.50%	84.58%
Dataset 3	Average Accuracy	65.05%	54.48%	51.67%
	Positive Accuracy	44.58%	11.99%	4.81%
	Negative Accuracy	85.52%	96.97%	98.54%
Dataset 4	Average Accuracy	76.74%	63.25%	58.57%
	Positive Accuracy	54.63%	27.20%	17.78%
	Negative Accuracy	98.85%	99.30%	99.36%

4.2. Results and Findings

4.2.1. Experiment 1

As mentioned in Section 4.1, the accuracy of the positive class is the main measurement being made due to its importance in comparison with the negative class and the previously established excellent accuracy of the negative class. The results from Experiment 1 indicate that across all datasets the models are not classifier neutral. Each dataset has a definitive best performing classifier, Naïve Bayes. Other studies have indicated classifier neutrality, meaning that despite the algorithm used in classification the results were similar. Taking that into account makes the current results rather surprising, particularly given the high differences in accuracies recorded. This is most likely due to the variation in language used across individual samples in each dataset. The naivety of Naïve Bayes stems from the algorithm treating each feature, or in this case term in an instance, as independent of the others. Given the individual probability of the terms used in positive cases, Naïve Bayes would outperform other algorithms as there simply aren't enough overall samples in any of the available databases.

4.2.2. Experiment 2

The highest accuracy form of text representation across all datasets is bag of words, the simplest form. This is again most likely due to the small sample size of the datasets. Terms vectorised as a single word have a much higher likelihood of appearing in multiple samples rather than groups of two or three terms when vectorised by the bigram and trigram vectorizers. Another factor concerning the bag of words representation is that the abusive or bullying content or language is specific and relatively unvaried in each dataset. The words surrounding those particular terms are not, decreasing the accuracy for higher and lower range N-gram representation.

4.2.3. Experiment 3

Not all datasets benefit from document frequency threshold vectorizer feature reduction. Dataset 2 and Dataset 3 do improve with the inclusion of the document frequency threshold due to the number of features or vectorised terms dropping dramatically, from 42350 to 2820 and from 10250 to 1100 respectively as illustrated in Figure 4.9. For Datasets 2 and 3, the minimum and maximum thresholds applied retain the terms that allow accurate classification while removing those that are detrimental. This most likely relates to the collection of the datasets or their source where the words or language used is relatively consistent across sample instances of bullying or abuse. Datasets 1 and 4 have much smaller initial feature sets of vectorised terms and are again drastically shrunk by the document frequency thresholds. However, as their term feature sets are so small initially, this reduces the accuracy of classification in the positive class. Sample source also has a part to play. The terms on the lower fringes of the thresholds seem to contain those that are pertinent in predicting the label for that instance for Datasets 1 and 4. It is probable that inconsistency or variation in the language used in bullying and non-bullying cases in Dataset 1 and Dataset 4 have a much greater effect on the likelihood of exclusion during document frequency thresholding. There are not enough individual instances of the words used across these datasets.

4.2.4. Experiment 4

The optimal feature set for Dataset 1 is displayed in Figure 4.10 and shows that swearing, an individual found and the percentage of capital letters and punctuation cause an increase in the positive class accuracy. The overall negative sentiment score and the negative and neutral sentiment scores near an individual found also cause an accuracy increase. This is due to the source of the data. The small number of average characters in Dataset 1 causes users to add punctuation and capitals for emphasis, profanities are not monitored and many of the samples are directed at other users in both positive and negative cases. The short length also allows the overall negative sentiment to represent the whole instance correctly as most are single sentences. The amalgamated negative and neutral sentiment near a found victim also reflects the directed nature of the content. The remaining features cause a decrease in the positive class accuracy. This implies there is no significant difference in the class scores or measurements for those features. Of particular interest is the ineffectiveness of the compound sentiment scores. As mentioned previously, when an instance is poorly structured or contains misspellings initially, the compound sentiment scoring algorithm is essentially moot, often returning a default value. This convolutes the feature and causes

misclassification. The positive sentiment scoring is, understandably, not helpful in classification of bullying instances which are by nature negative. The neutral scoring, both overall and near a potential victim, are interesting results. The measure of neutral sentiment overall reduces accuracy, but near a victim aids in determining the positive class accuracy. This most likely indicates the feature aids in correctly determining a negative instance, reducing the likelihood of a false positive.

Dataset 2 gains accuracy increases in the positive class from much fewer features. The samples across this dataset contain many spelling and grammatical errors, making both syntactic and semantic feature addition incredibly difficult. This is reflected in the results in Figure 4.11. However, the overall number of words and swear words are good features, including the baseline swearing flag. The positive and negative overall sentiment scores both improve positive class accuracy. The positive sentiment score more than likely aids in classifying the negative class, reducing the error in the positive class. It's possible and likely that when an individual is found in an instance, the surround text cannot be scored using the sentiment algorithm due to the language quality. The compound sentiment, both overall and in proximity to a victim, significantly decreases accuracy for the same reasons as explained above.

The Dataset 3 results, Figure 4.12, show that the baseline feature of swearing found increases positive class accuracy. The number of words and capital letters improve the classifier as well, probably due to shorter instances and more emphasis applied with capital letters implying bullying and abuse. The ratio of victims to swear words used and the negative sentiment surrounding a found victim also increase the positive class average recall, despite other individual, swear word and sentiment based features performing worse. This implies that the dataset contains fluctuating usage of swear words, individual identifiers and sentiment across both classes. When specifically analysing the content surrounding a first or second person pronoun or common first name, the distinction can be made more accurate. The reason for this is due to the longer average instance length in comparison to the likes of Dataset 1. The compound sentiment features are the worst performing additions.

The results for Dataset 4 are similar to those of Dataset 3. Detailed in Figure 4.13, swearing, victim to swear word ratio, the number of swear words in the instance and the negative sentiment score near an individual all increase the accuracy of the positive class. The other features do not provide an improvement and are often significantly detrimental. The similarities across both classes in the remaining features cause a decrease in the positive class accuracy, most likely due to a blanket prediction of a negative label to test instances. The recurrence of the compound sentiment features as worst performing is not surprising, as it requires relatively perfect spelling and grammar in samples which is next to impossible to ensure when gathering the data.

4.2.5. Experiment 5

The results of this experiment show a steady decline in the positive class accuracy with each incremental size reduction applied using the Chi Square algorithm. The results, available in Figure 4.14, are not surprising given the relatively small number of overall features and their necessity. It is necessary to retain them as the majority are the sparse matrix text vectorisation, the terms that are relevant as determined by the results from Experiment 3. It therefore makes less sense to reduce that even further in the case of these datasets. For much

larger corpuses, and much larger feature sets, this type of feature reduction could provide significant advantages in both classification accuracy and run time and memory efficiency. However, in this project the datasets are quite small and due to this time and memory efficiency were at no stage an issue. This also resulted in a feature set that requires all available features after the document frequency pre-processing.

4.2.6. Experiment 6

Experiment 6 investigates the effect of term vectorisation matrix normalisation. The results in Figure 4.15 show large decreases in the positive class accuracy after applying the Euclidian distance based term frequency normalisation, which is popular normalisation [12, 20]. Vector normalisation was also tested and has a similar, stronger detrimental effect on classification. This normalisation attempts to represent terms in a single range. The count vectors could misrepresent the use of language and individual words when vastly varying lengths of text are vectorised. This “signal” of vectors is smoothed, effectively removing the large “spikes” caused by longer documents. However, this smoothing causes the spikes for words that are used in abusive content and bullying to be effectively normalised out of relativity. They are no longer the identifiable features for the positive class. This result was the most significant of the project, particularly as it is visible across all datasets.

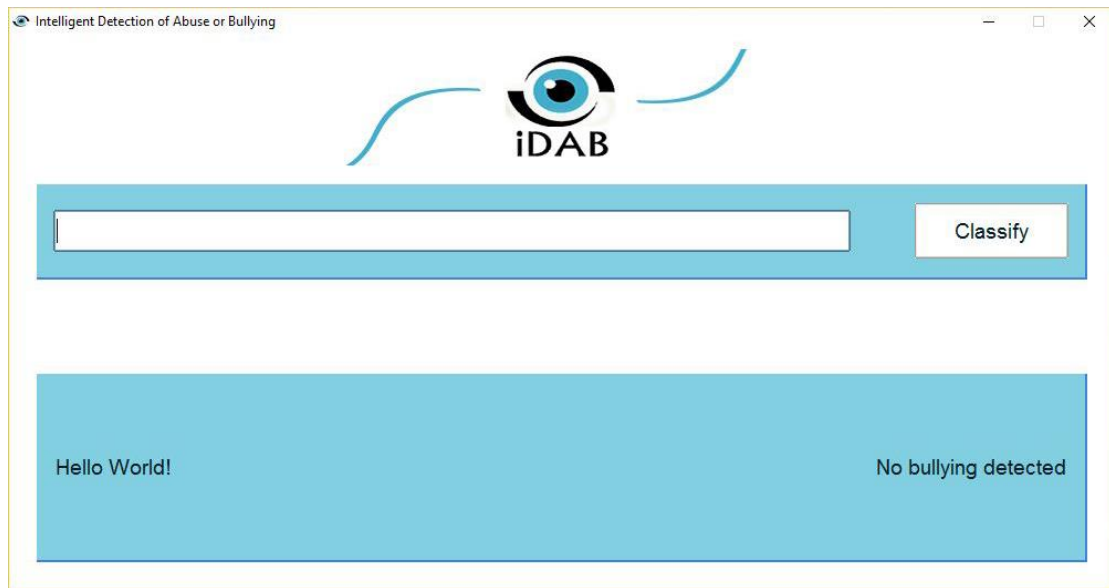
4.2.7. Discussion

New features can provide small individual increases in positive class accuracy even though the improvements are in some cases miniscule. The margins in accuracy increase and decrease are quite fine so any additional accuracy is encouraging. The quality of the individual instances in each dataset plays a significant role in the applicability of the newly constructed features and text representation. There can also be a case made for stricter labelling criteria during the data labelling phase of dataset construction to reduce subjectivity on behalf of the labellers. However, the subjective nature of the sample content means that there will always be a human presence necessary in conjunction with the implemented models. Without much larger corpuses that are balanced when collecting data, the models are rather undertrained. There are also large differences in the sample size and language across datasets. It may not be possible to define all-inclusive feature sets and models for social media analysis. Varying types of dataset require different angles of approach, so tailored solutions based on the social media source of the dataset are the most viable route.

4.2.8. Demonstration Interface

The final result of the experiments and the end product models is a basic interface for demonstration purpose. This was implemented in Python, and provides users with a simple front end in which they can define and pass their own text instances to the trained models. Users enter text and then receive a prediction from the model for their entered text. Their entered text essentially becomes the test data for the model. The working title for the application is “Intelligent Detection of Abuse or Bullying”, abbreviated to iDAB. The front end interface is displayed below in Figure 4.16.

FIGURE 4.16. THE INTELLIGENT DETECTION OF ABUSE OR BULLYING “iDAB” DEMONSTRATION INTERFACE SCREEN CAPTURE



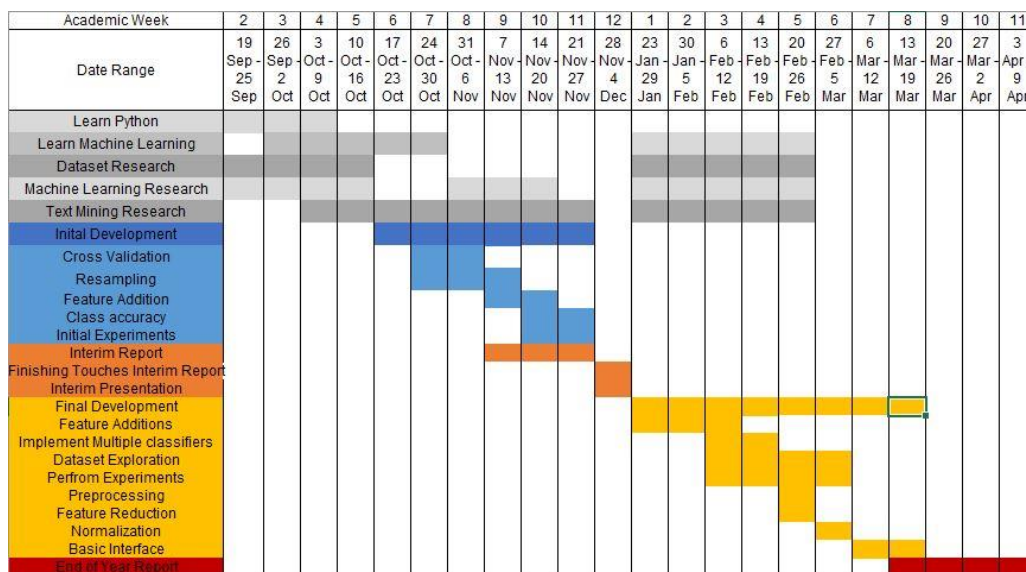
5. Project Management

5.1. Plan

Project planning was handled using a Gantt chart to determine tasks, estimating the length of time that each task would require and also as a log of work performed. The Gantt was separated on a weekly basis. The initial iteration was solely focused on the first project submission data, the Interim report on the 1st of December 2016 and the weeks leading up to this deadline in Semester 1. The goal for the first Semester deadline was to have a single dataset, Dataset 1, working in conjunction with a single classifier and some basic additional features implementing cross validation and the resampling required for the given dataset. Performing research into all aspects of the project was necessary to complete the developmental tasks. It was also a necessary goal to achieve for reference material in the Interim report document and this document. Most of the work performed in this semester was in order to learn Python and the necessary machine learning and text mining techniques needed to complete the project. The Semester 1 tasks are coloured grey blue and orange as seen in Figure 5.1.

The second iteration of the Gantt chart was focused on completing the project throughout Semester 2. Included in the Semester 2 revision are tasks for applying multiple classifiers, additional features, pre-processing, normalization and feature reduction, applied to all datasets. The Gantt chart is a very rough initial plan, and was supplemented by weekly meetings with the project supervisor in which work from the previous week was discussed. In each meeting the tasks for the next week were discussed in much greater detail and more specific sub tasks were decided upon. This stemmed from the overall project plan, with less refined tasks such as “Text Mining Research” being clearly defined as “Research VADER sentiment analysis” and “Feature Additions” refined to “Lexical additional features - structure and syntax”. This naturally led to a sprint-like development style, with incremental additions to the implementations made weekly, although it was much more flexible than strictly defined weekly tasks. The Semester 2 tasks are coloured grey, yellow and red as displayed in Figure 5.1.

FIGURE 5.1. PROJECT GANTT CHART



5.2. Outcome

The initial iteration of the Gantt chart proved to be very accurate in terms of time estimation and task ordering. The vast majority of developmental tasks were completed on time, if not sooner than had been anticipated. What was greatly underestimated was the research aspect of the project. The large amount of new concepts to learn in a relatively short space of time proved to be quite challenging. Text mining and machine learning are much vaster areas than initially considered. While the Gantt chart lists specific time periods during which these areas were to be researched and learned, it was an ongoing process with contiguous aspects. For those reasons, the second iteration of the Gantt that listed the second semester tasks includes various research and learning aspects across the whole of development.

Similar to the initial development in Semester 1, the implementation side of development in Semester 2 went relatively smoothly and the majority of tasks were completed on time or before time. However, the time taken to perform experiments was vastly underestimated. The volume of experiments, even across a single dataset, was unexpected, and took far longer than the initial estimates. The experiments were scheduled to begin relatively early in the development, which allowed for the extended time in executing them. Due to the other success in terms of implementation, there was no major effect on the overall timeline of the project.

5.3. Future Work

This project has many interesting and encouraging outcomes, so there is scope to continue in this particular area of research. However, the results are not universally conclusive and there are many facets that could be expanded upon.

The datasets implemented are very small in terms of overall sample size and the positive class sample size. For greater accuracy, particularly in text mining, many varied samples should be available to the classification algorithm. Future work should include collection of much larger datasets for use in predictive modelling that are specific to abuse or cyberbullying detection. While it would be a time-consuming process, the results would be much more universally conclusive, especially for the online source of the data. Other issues with the datasets used in this project could be addressed in the future and applied to other data. Many instances feature incorrect spelling, so implementing a spell check or spell correct could be a very advantageous addition to instance pre-processing.

As mentioned throughout this document, the issue of subjectivity when determining abusive and cyberbullying content is highly pertinent. This would most efficiently be addressed by increasing the number and diversity of the instance labellers, which would be very labour intensive and time consuming. This would again be very helpful in accurate classification, also in a much more universal manner. These future additions would all aid in the modelling, particularly from the term representation and sentiment analysis standpoints. The correctness of the terms themselves are of huge importance. Another, much larger scale future addition would be the inclusion of term or word lookup dictionaries to reduce colloquialisms and slang within instances. This could also allow for synonym term swapping in

order to allow for further accuracy in the term representation. This type of implementation would essentially find terms that are represented a relatively few number of times across a dataset and find their more abundantly represented synonyms. This would greatly reduce sparse feature columns while retaining the meaning of an instance.

Further research could be carried out in terms of the classifiers implemented. There are many more available classifiers that can be tested, and were not due to time constraints on this project. Some basic examples would be K Nearest Neighbour (KNN), Random Forest, applying other kernels to the non-Linear SVM classifier such as the “poly” or “sigmoid” kernels. A much more complex and interesting future possibility is the application of Neural Network classification algorithms and deep learning techniques to classify instances. This would require a vast amount of time, learning and even prior expertise in its application. This is due to the nature of deep learning, which is very much a black box scenario that creates its own, new intermediate variables based on the initial inputs during the learning stage. These intermediates could continue to grow before the final prediction is made, entirely unlike traditional machine learning that produces a prediction solely based on the input variables. However, this could provide a much greater scope for text classification compared to the traditional machine learning techniques.

6. Conclusion

The research performed throughout the project was inconclusive in many aspects, particularly for providing feature addition that would vastly improve any positive class prediction accuracy and recall. However, many of the tested features were promising for the specific datasets rather than all-inclusive results for all social media. Their results also point to features that should not be incorporated given the type of text samples available. The average recall for the final models is relatively quite high, particularly for Dataset 1 at 88% average accuracy, which can be considered quite successful. The results of the experiments performed and the potential hypotheses that can be derived from these results should be considered successful and considered in future research. Particularly note-worthy is the pre-processing and term frequency normalisation outcomes. While the research was the primary technical objective I am very pleased to have been able to also develop a front-end interface in which to demonstrate the predictive models. Although this interface is in no way complex, the ability to allow interaction in a simple and straightforward way is vital to user and public interest in the work.

In terms of personal and learning objectives, this project proved to have a great many more facets and aspects than I initially anticipated. This was due to my inexperience in the fields of machine learning, text and data mining, data analysis and academic style research. I had no prior knowledge in many of the areas applied other than a highest-level understanding of the most basic concepts. I had also never extensively used the Python language before, which also provided many challenges and very valuable learning opportunities. Python is a powerful language and nearing ubiquity in certain aspects of development. The learning outcomes from this project have resulted in a detailed understanding of specific aspects of machine learning and its application, text and data mining and data analysis. The personal insight this project has afforded me has greatly shaped my focus for future employment or possibly further education within the field of machine learning.

7. Bibliography

1. NSPCC (2014) Under Pressure Childline Review.
2. ditchthelabel.org (2013) The Annual Cyberbullying Survey.
3. Smith PK, Mahdavi J, Carvalho M, Tippet N (2006) An investigation into cyberbullying, its forms, awareness and impact, and the relationship between age and gender in cyberbullying. Res. Brief No RBX03-06 Lond. DfES
4. Bauman S (2013) Principles of cyberbullying research: Definitions, measures, and methodology. Routledge
5. Pang B, Lee L, Vaithyanathan S (2002) Thumbs up?: sentiment classification using machine learning techniques. In: Proc. ACL-02 Conf. Empir. Methods Nat. Lang. Process.-Vol. 10. Association for Computational Linguistics, pp 79–86
6. Chen Y, Zhou Y, Zhu S, Xu H (2012) Detecting offensive language in social media to protect adolescent online safety. In: Priv. Secur. Risk Trust PASSAT 2012 Int. Conf. 2012 Int. Confernece Soc. Comput. SocialCom. IEEE, pp 71–80
7. Mangaonkar A, Hayrapetian A, Raje R (2015) Collaborative detection of cyberbullying behavior in Twitter data. In: 2015 IEEE Int. Conf. ElectroInformation Technol. EIT. IEEE, pp 611–616
8. Dinakar K, Jones B, Havasi C, Lieberman H, Picard R (2012) Common sense reasoning for detection, prevention, and mitigation of cyberbullying. ACM Trans Interact Intell Syst TiiS 2:18
9. Yin D, Xue Z, Hong L, Davison BD, Kontostathis A, Edwards L (2009) Detection of harassment on web 2.0. Proc Content Anal WEB 2:1–7
10. Dadvar M, Trieschnigg D, de Jong F (2014) Experts and machines against bullies: A hybrid approach to detect cyberbullies. In: Can. Conf. Artif. Intell. Springer, pp 275–281
11. Reynolds K, Kontostathis A, Edwards L (2011) Using machine learning to detect cyberbullying. In: Mach. Learn. Appl. Workshop ICMLA 2011 10th Int. Conf. On. IEEE, pp 241–244
12. Chen H, McKeever S, Delany SJ (2017) Harnessing the Power of Text Mining for the Detection of Abusive Content in Social Media. In: Adv. Comput. Intell. Syst. Springer, pp 187–205
13. Hastie T, Tibshirani R, Friedman J (2009) Unsupervised Learning. In: Elem. Stat. Learn. Springer New York, pp 485–585
14. Olshausen BA (2004) Bayesian probability theory. Redw. Cent. Theor. Neurosci. Helen Wills Neurosci. Inst. Univ. Calif. Berkeley Berkeley CA
15. Kotsiantis SB, Zaharakis ID, Pintelas PE (2006) Machine learning: a review of classification and combining techniques. Artif Intell Rev 26:159–190

16. Tong S, Koller D (2001) Support vector machine active learning with applications to text classification. *J Mach Learn Res* 2:45–66
17. Abdi H (2004) Partial regression coefficients. *Encycl. Soc. Sci. Res. Methods* Thousand Oaks CA SAGE Publ.
18. Safavian SR, Landgrebe D (1991) A survey of decision tree classifier methodology. *IEEE Trans Syst Man Cybern* 21:660–674
19. 6. Learning to Classify Text. <http://www.nltk.org/book/ch06.html>. Accessed 27 Mar 2017
20. Burnap P, Williams ML (2015) Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy Internet* 7:223–242
21. Gilbert CHE (2014) VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth Int. Conf. Weblogs Soc. Media ICWSM-14 Available 200416 Httpcomp Soc. Gatech Edupapersicwsm14 Vader Hutto Pdf
22. nltk.sentiment.vader — NLTK 3.0 documentation. http://www.nltk.org/_modules/nltk/sentiment/vader.html. Accessed 25 Mar 2017
23. About Python™ | Python.org. <https://www.python.org/about/>. Accessed 29 Nov 2016
24. scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation. <http://scikit-learn.org/stable/>. Accessed 29 Nov 2016
25. Python Data Analysis Library — pandas: Python Data Analysis Library. <http://pandas.pydata.org/>. Accessed 29 Nov 2016
26. Natural Language Toolkit — NLTK 3.0 documentation. <http://www.nltk.org/>. Accessed 29 Nov 2016
27. SciPy.org — SciPy.org. <https://www.scipy.org/>. Accessed 29 Nov 2016
28. NumPy — NumPy. <http://www.numpy.org/>. Accessed 29 Nov 2016
29. TkInter - Python Wiki. <https://wiki.python.org/moin/TkInter>. Accessed 1 Apr 2017
30. 24.2. ttk — Tk themed widgets — Python 2.7.13 documentation. <https://docs.python.org/2/library/ttk.html>. Accessed 1 Apr 2017
31. Python Imaging Library (PIL). <http://www.pythonware.com/products/pil/>. Accessed 1 Apr 2017
32. Bayzick J, Kontostathis A, Edwards L (2011) Detecting the presence of cyberbullying using computer software.
33. Swear Word List, Dictionary, Filter, and API - NoSwearing.com. <http://www.noswearing.com/>. Accessed 29 Nov 2016

34. Bureau UC Frequently Occurring Surnames from Census 1990 – Names Files.
https://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html. Accessed 25 Mar 2017
35. Olson DL, Delen D (2008) Advanced data mining techniques. Springer Science & Business Media

8. Appendices

8.1 Table of Figures

FIGURE 2.1. AN EXAMPLE DECISION TREE.....	16
FIGURE 2.2. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 1 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	20
FIGURE 2.3. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 1 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	21
FIGURE 2.4. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 1 FOR BOTH CLASSES.....	21
FIGURE 2.5. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 2 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	22
FIGURE 2.6. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 2 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	23
FIGURE 2.7. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 2 FOR BOTH CLASSES.....	23
FIGURE 2.8. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 3 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	24
FIGURE 2.9. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 3 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	25
FIGURE 2.10. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 3 FOR BOTH CLASSES.....	25
FIGURE 2.11. COUNT OF POSITIVE CLASS INSTANCES IN DATASET 4 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	26
FIGURE 2.12. COUNT OF NEGATIVE CLASS INSTANCES IN DATASET 4 AND THE CORRESPONDING COUNTS AND PERCENTAGES OF THE INITIAL FEATURES "INDIVIDUAL FOUND" AND "SWEARING FOUND"	27
FIGURE 2.13. AVERAGE COUNTS OF CHARACTERS, CAPITAL LETTERS, PUNCTUATION MARKS, WORDS AND SWEAR WORDS IN DATASET 4 FOR BOTH CLASSES.....	27
FIGURE 2.14. A SAMPLE INSTANCE FROM DATASET 1	28
FIGURE 3.1. THE STRIP_LINKS FUNCTION.....	29
FIGURE 3.2. THE REPLACE_ALL_USERNAMES_AND_RTS FUNCTION	30
FIGURE 3.3. THE TEXT_INSTANCE_TO_WORDS FUNCTION.....	30
FIGURE 3.4. A FOR LOOP TO SEPARATE THE MOVING WINDOW INSTANCES IN DATASET 3 INTO INDIVIDUAL COMMENTS	30
FIGURE 3.5. ADDITIONAL FEATURE COLUMNS APPENDED TO THE PANDAS DATAFRAME FOR THE DATASET IN USE.....	31
FIGURE 3.6. THE COUNTINTEXT FUNCTION.....	32
FIGURE 3.7. THE FLAGFORVICTIM FUNCTION	34
FIGURE 3.8. THE FLAGFORSWEARING FUNCTION	35
FIGURE 3.9. THE RATIO OF POSITIVE (+) AND NEGATIVE (-) CLASS INSTANCES IN DATASETS 1, 2 AND 4 BEFORE AND AFTER RESAMPLING	36
FIGURE 3.10. PYTHON CODE IMPLEMENTED TO APPLY OVERSAMPLING TO REBALANCE A DATASET	36

FIGURE 3.11. A BAG OF WORDS COUNT VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON	37
FIGURE 3.12. A UNIGRAM TO BIGRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON	37
FIGURE 3.13. A UNIGRAM TO TRIGRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON	38
FIGURE 3.14. A 4 CHARACTER N-GRAM VECTORIZER APPLYING A MINIMUM AND MAXIMUM DOCUMENT FREQUENCY IN PYTHON	38
FIGURE 3.15. A TERM FREQUENCY VECTORIZER ACTING AS A BAG OF WORDS VECTORIZER APPLYING MINIMUM AND MAXIMUM DOCUMENT FREQUENCY AND EUCLIDIAN DISTANCE NORMALISATION	38
FIGURE 3.16. USING THE VALUES FUNCTION OF PANDAS DATAFRAMES AND SCIPY'S HSTACK FUNCTION TO RE-JOIN THE CONSTRUCTED FEATURES WITH THE VECTORISED TEXT	38
FIGURE 3.17. AN IMPLEMENTATION OF CHI SQUARE FEATURE SELECTION USING THE SELECTKBEST FUNCTION	39
FIGURE 3.18. INITIALISING AND TRAINING A NAÏVE BAYES CLASSIFIER IN PYTHON	39
FIGURE 3.19. INITIALISING AND TRAINING A DECISION TREE CLASSIFIER IN PYTHON.....	40
FIGURE 3.20. INITIALISING AND TRAINING A LOGISTIC REGRESSION CLASSIFIER IN PYTHON.....	40
FIGURE 3.21. INITIALISING AND TRAINING A LINEAR SUPPORT VECTOR MACHINE CLASSIFIER IN PYTHON.....	40
FIGURE 3.22. INITIALISING AND TRAINING A NON-LINEAR SUPPORT VECTOR MACHINE CLASSIFIER IN PYTHON.....	41
FIGURE 3.23. RETURNING THE RECALL AND STORING	41
FIGURE 3.24. MANUAL CROSS-VALIDATION IMPLEMENTATION	43
FIGURE 3.25. FINAL OVERALL SCORING OUTPUT	44
FIGURE 4.1. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 1	45
FIGURE 4.2. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 2	46
FIGURE 4.3. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 3	46
FIGURE 4.4. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH CLASSIFICATION ALGORITHM IN DATASET 4	47
FIGURE 4.5. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 1	48
FIGURE 4.6. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 2	48
FIGURE 4.7. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 3	49
FIGURE 4.8. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS, NEGATIVE CLASS AND THE AVERAGE OF BOTH CLASSES FOR EACH TEXT REPRESENTATION VECTORIZER IN DATASET 4	49
FIGURE 4.9. NUMBER OF FEATURES, POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING VARYING AMOUNTS OF DOCUMENT FREQUENCY THRESHOLDS FOR DATASETS 1 - 4	50
FIGURE 4.10. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 1.....	51
FIGURE 4.11. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 2.....	51
FIGURE 4.12. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 3.....	51
FIGURE 4.13. THE AVERAGE RECALL / ACCURACY FOR THE POSITIVE CLASS AND NEGATIVE CLASS AGAINST THE NEW FEATURE ADDED FOR DATASET 4.....	52

FIGURE 4.14. NUMBER OF FEATURES, POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING VARYING VALUES OF FEATURE REDUCTION FOR DATASETS 1 - 4	52
FIGURE 4.15. POSITIVE, NEGATIVE AND AVERAGE CLASS ACCURACIES WHEN APPLYING DIFFERENT NORMALISATION TO VECTORISED TEXT FOR DATASETS 1 - 4	53
FIGURE 4.16. THE INTELLIGENT DETECTION OF ABUSE OR BULLYING “IDAB” DEMONSTRATION INTERFACE SCREEN CAPTURE	57
FIGURE 5.1. PROJECT GANTT CHART	58