Institute of Technology Blanchardstown

Masters Thesis

Detection of Cyberbullying using Python and Text Mining

Author:
David Colton

Supervisor:
Dr. Markus Hofmann

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computing

at

Institute of Technology Blanchardstown
Dublin 15, Ireland

April 2020

Declaration of Authorship

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of **M.Sc. in Computing** in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed [David Colton]:

_____

Date:

_____

Abstract

Detection of Cyberbullying using Python and Text Mining

by David Colton

The internet technology boom has led to a proliferation of tablets, laptops and smart phones with high-speed internet access. This access, coupled with the advent of instant messaging, chat rooms and social media websites, has led to an internet generation who think nothing of posting selfies, mood updates, their relationship status or anything about their life on-line. The traditional bully was the kid in school, or office worker, who got pleasure from watching their victims suffer as they verbally abused them or perhaps made fun of them or maybe even threatened them with violence. At least the victim knew who the bully was and, although not a solution, could plan their day to avoid crossing paths with the bully and having to suffer further torment. However, the bully has also moved on-line. This cyberbully now has twenty-four hour access to a potentially unlimited number of victims. Through their mean and harassing posts and comments the consequences of their cyberbullying activity is too often read about in the papers following another tragic teen suicide. To prevent this new form of bullying, it is important that technology is used to detect these cyberbullying posts.

This dissertation shows that Python, together with the application of text mining techniques, can be successfully used in the automatic detection of cyberbullying text. The contributions of this paper are many. To begin, a comprehensive literature review of the current trends in cyberbullying detection is provided. A new classified cyberbullying dataset, including detailed descriptions of the criteria used in its classification, is generated. An in-depth analysis of several classifiers, including multiple approaches to tackle class imbalance, is undertaken before a novel way of determining the best overall classifier using the recall values of both the positive and negative class in addition to considering the execution time of the model and penalising any data manipulation performed when tackling class imbalance is suggested. Finally, an evaluation of the best models is performed by simulating their evolution as new, previously unseen, samples are classified and then included as training data for subsequent iterations.

Acknowledgements

"The only people with whom you should try to get even are those who have helped you."

John E. Southard

Niamh, Emma, Saoirse and Eva. Without your help, patience and the time you gave me, this masters would not have been possible. Time to get even.

"Some people talk in their sleep.
Lecturers talk while other people sleep."

Albert Camus

I'm very happy to say that this was not something that afflicted me over the last two years while completing this masters degree programme. First, a special thank you to Dr. Markus Hofmann who acted as my supervisor on this dissertation. This paper would not be what it is today without his thoughts and guidance from when the first green shoot of this work started showing, right up to the end. I must thank Geraldine Grey, Laura Keyes and Markus for their enthusiasm and passion for the subjects covered.

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Thesis Research Question . . . 3
- 1.2 Objectives . . . 3
- 1.3 Thesis Structure . . . 5

2 Background and Tooling 6
- 2.1 Introduction to Ask.fm . . . 6
  - 2.1.1 Creating an Ask.fm Account . . . 7
  - 2.1.2 Questions and Answers . . . 8
  - 2.1.3 Privacy Setting, Reporting Abuse and Deleting Questions . . . 8
  - 2.1.4 Deactivating an Account . . . 10
- 2.2 Python, NLTK and Scikit-Learn . . . 11
  - 2.2.1 Python . . . 11
  - 2.2.2 The Natural Language Toolkit . . . 11
  - 2.2.3 Scikit-Learn . . . 12
  - 2.2.4 Python Resources . . . 12
- 2.3 MySQL, SPSS and Excel . . . 12
- 2.4 Precision, Recall, Accuracy and G-Performance . . . 13
  - 2.4.1 G-Performance . . . 14

3 Literature Review 15
- 3.1 Cyberbullying: What is it and why should we care . . . 15
- 3.2 Mining for Cyberbullying Text . . . 18

3.3 Common Obstacles to Overcome . . . 29
3.3.1 Anonymity . . . 29
3.3.2 Data and Classification . . . 30
3.3.3 Class Imbalance . . . 30

4 Data Extraction, Exploration and Processing 33
4.1 Raw Data Extraction . . . 34
4.2 Initial Data Processing . . . 35
4.2.1 Extract data from HTML Files . . . 36
4.2.2 Splitting the data . . . 38
4.3 Classification . . . 40
4.3.1 Cyberbullying Criteria . . . 40
4.3.2 Dataset Classification . . . 43
4.4 Data Exploration . . . 46
4.4.1 Data Description . . . 46
4.4.2 Data Quality . . . 47
4.4.3 Data Exploration . . . 48
4.4.4 Data Visualisation . . . 56
4.5 Data Preparation . . . 59
4.5.1 Cleaning the Data . . . 59
4.5.2 Write Data in NLTK Corpus Format . . . 62
4.5.3 Analysis of Cleaning Steps . . . 63
4.5.4 Data Exploration Revisited . . . 65
4.5.5 Data Visualisation Revisited . . . 67
4.6 Summary and Conclusion . . . 68

5 Data Modelling 71
5.1 Natural Language Toolkit - Initial Modelling . . . 72
5.1.1 Model Development . . . 72
5.1.2 Model Execution and Performance . . . 75
5.1.3 Model Analysis . . . 77
5.1.4 Further Exploration . . . 77
5.2 Natural Language Toolkit - Class Imbalance . . . 80
5.2.1 Majority Class Under Sampling . . . 80
5.2.2 Minority Class Over Sampling . . . 81
5.2.3 Hybrid Approach . . . 83
5.2.4 Most Frequently Occurring Features . . . 85
5.3 Scikit-learn Modelling . . . 87
5.3.1 Model Development . . . 87
5.3.2 Model Execution and Performance . . . 89
5.3.3 Further Exploration . . . 90
5.4 Scikit-learn - Class Imbalance . . . 93
5.4.1 Pipeline and Grid Search . . . 93
5.4.2 Class Sampling . . . 95
5.5 Scikit-learn - Cost Sensitive Learning . . . 99
5.6 Choosing the Best Classifiers . . . 100
5.6.1 G-Performance . . . 100

5.6.2 Best Models . . . 101
5.6.3 Initial Models . . . 101
5.7 Applying the Best Classifiers . . . 103
5.7.1 Scenario Overview . . . 103
5.7.2 Data Preparation . . . 104
5.7.3 Initial Analysis . . . 104
5.7.4 Batch Processing of Samples . . . 107
5.7.5 Analysis of Batch Processing Results . . . 108
5.8 Summary and Conclusion . . . 112

6 Conclusions **115**
6.1 Summary of research activities . . . 116
6.2 Discussion . . . 118
6.3 Future Work . . . 119
6.4 Conclusion . . . 121

A Selected Python Scripts **122**
A.1 Parse Raw HTML Files from Ask.fm . . . 122
A.2 Clean data . . . 125
A.3 Replacer helper class . . . 126
A.4 Simple NLTK Naive Bayes Classifier . . . 128
A.5 NLTK Naive Bayes Classifier with Feature Selection . . . 131
A.6 Simple Scikit-Learn Classifier . . . 132
A.7 Scikit-Learn Grid Search Example . . . 134
A.8 Scikit-Learn Over Sampling Classifier . . . 136
A.9 Scikit-Learn Simulation Sample Script . . . 138

B Novel Method to Choose the Best Classifiers **141**
B.1 Choosing the Best Classifiers . . . 141
B.1.1 G-Performance . . . 142
B.1.2 Execution Time . . . 142
B.1.3 Data Manipulation . . . 143
B.1.4 Best Models . . . 144
B.2 Applying the Best Classifiers . . . 145
B.2.1 Scenario Overview . . . 145
B.2.2 Data Preparation . . . 146
B.2.3 Initial Analysis of Models . . . 146
B.2.4 Batch Processing of Samples . . . 148
B.2.5 Analysis of Batch Processing Results . . . 148

Bibliography **152**

List of Figures

2.1 Creating an Ask.fm Account . . . 7
2.2 Your Ask.fm Questions . . . 8
2.3 Asking a question when logged into your account . . . 8
2.4 Asking a question when not logged into your account . . . 8
2.5 Answering with text and an optional picture . . . 9
2.6 Configuring privacy settings . . . 9
2.7 Reporting a user or a question and answer . . . 9
2.8 Types of inappropriate behaviours to report . . . 10
2.9 Deleting questions and blocking the user that asked the question . . . 10
2.10 Blocking a user to prevent them asking further questions . . . 10
2.11 Deactivating an account . . . 11

3.1 Accuracies of sentence level offensiveness detection . . . 23
3.2 Roles taken in a cyberbullying incident . . . 24
3.3 Top ten profane words for male and female authors . . . 26
3.4 Weighted graph model identifying predators and victims . . . 27

4.1 Sample question and answer . . . 34
4.2 Simple process flow to extract data from HTML files . . . 36
4.3 Partitioning data using IBM SPSS . . . 39
4.4 Partition operator used generate a new attribute . . . 39
4.5 Select operator and the Filter operator settings . . . 40
4.6 Independent classification review results . . . 44
4.7 View of the thesis schema . . . 46
4.8 Distribution of user records . . . 49
4.9 Class Distribution of Sample Questions . . . 50
4.10 Sample Questions asked Anonymously . . . 50
4.11 Average question length per class . . . 51
4.12 Word cloud of not bullying words . . . 57
4.13 Word cloud of bullying words . . . 57
4.14 Filtered word cloud of not bullying words . . . 57
4.15 Filtered word cloud of bullying words . . . 58
4.16 Normalised distribution of word frequencies . . . 58
4.17 Remove non ASCII characters . . . 63
4.18 Remove punctuation characters . . . 64
4.19 Replace selected numerical ages with text then remove remaining digits . . . 64
4.20 Remove repeated characters and abbreviated words . . . 65
4.21 Normalised distribution of word frequencies . . . 67

4.22 Bi-gram word clouds 68
4.23 Tri-gram word clouds 69

5.1 The structure of the primary datset corpus 73
5.2 List of files for the bullying category 74
5.3 Sample positive features 74
5.4 NLTK model development process 75
5.5 Performance measurements for initial NLTK classifier 76
5.6 Initial NLTK model performance across datasets 76
5.7 Comparison of pre-generated and in model n-gram generation 79
5.8 NLTK model performance using under sampling 81
5.9 NLTK model performance using over sampling 83
5.10 NLTK model performance using hybrid sampling 84
5.11 Scikit-Learn model performance using default parameters 89
5.12 NLTK - Scikit-Learn runtime comparison 91
5.13 Scikit-Learn model performance with default parameters 92
5.14 Scikit-Learn comparing best manual results with grid search results . . . 95
5.15 Scikit-Learn sampling with grid search results 96
5.16 Best Model - G-Performance heat map 101
5.17 Percentage change in hold back dataset classification 110
5.18 Analysis of samples that changed classification 111
5.19 Sample analysis of samples that did not change classification 112

B.1 Best Model - G-Performance heat map 142
B.2 Best Model - Normalised execution heat map 143
B.3 Best Model - Data Manipulation Penalties 144
B.4 Best Model - Three best models identified 145
B.5 Processing time for batch simulation 149
B.6 Analysis of Hybrid Sampling on Hold Back Dataset 150
B.7 Analysis of Over Sampling on Hold Back Dataset 151

List of Tables

4.1 Description of raw_data table 47
4.2 Bullying questions that are not anonymous 51
4.3 Distinct word counts 52
4.4 Distinct word counts, no stop words 53
4.5 Distinct bi-gram word counts 54
4.6 Distinct tri-gram word counts 55
4.7 Unique tokens, total count and average frequency 56
4.8 Distinct n-gram word counts (cleaned datasets) 66
4.9 Unique tokens, total count and average frequency (cleaned datasets) . . . 67

5.1 N-gram generation increases execution time 78
5.2 Most informative feature comparison 79
5.3 Performance comparison of over sampling and hybrid sampling 86
5.4 Confusion matrices from the initial Scikit-Learn Naive Bayes model . . . 90
5.5 Confusion matrices from the initial Scikit Support Vector model 90
5.6 Grid search performance and confusion matrix comparison 96
5.7 Performance comparison of over sampling and hybrid sampling 97
5.8 Sampling parameter values returned by grid search 99
5.9 Top performing NLTK over sampling models 102
5.10 Top performing NLTK hybrid sampling models 102
5.11 Top performing Scikit-Learn models 103
5.12 Simple performance measurement of top 24 models 105
5.13 NLTK Models, analysis of samples predicted 108
5.14 Scikit-Learn Models, analysis of samples predicted 109

B.1 Penalties assigned for sampling 144

For Niamh, Emma, Saoirse and Eva

Chapter 1

Introduction

> ‘‘Bullying is any unwanted aggressive behaviour(s) by another youth or group of youths who are not siblings or current dating partners that involves an observed or perceived power imbalance and is repeated multiple times or is highly likely to be repeated. Bullying may inflict harm or distress on the targeted youth including physical, psychological, social, or educational harm.’’
> stopbullying.gov [1].

> ‘‘Cyberbullying is bullying that takes place using electronic technology. Electronic technology includes devices and equipment such as cell phones, computers, and tablets as well as communication tools including social media sites, text messages, chat, and websites.
>
> Examples of cyberbullying include mean text messages or emails, rumours sent by email or posted on social networking sites, and embarrassing pictures, videos, websites, or fake profiles.’’
> stopbullying.gov [2].

Bullying is not something new, some people might even consider it a rite of passage having either experienced it when growing up or knowing someone who was bullied in school or at work or when out playing or socialising with friends. Traditional bullying, though never a pleasant experience, can only be inflicted on the victim face to face. By steering clear of the bully, or by leaving the environment where the bullying was happening, the intended victim could avoid the pain and suffering imposed on them. Now, however, with the advent of instant messaging and social media, the bully has moved on-line with twenty-four hour access to their victims. This new on-line bullying is known as cyberbullying and, unfortunately, its sometimes tragic consequences are

plain to see with headlines like ‘‘Third suicide in weeks linked to cyberbullying’’[3], ‘‘Cyberbullies claimed lives of Five teens’’[4] and ‘‘Hanna Smith suicide fuels calls for action on Ask.fm cyberbullying’’[5] becoming an all too depressingly frequent occurrence.

Recent surveys have shown that the number of young children and teenagers who have access to the internet has soared. In September 2012 the Pew Research Center, a non-partisan American think tank based in Washington, D.C, found that 95% of all North American children and young adults aged between 12 and 19 had access to the internet and that 74% had access either through a smart phone or tablet [6]. In Europe a similar picture was painted by the 2012 Eurostat Internet use in households and individuals report [7], where it is reported that 93% of people aged 16 - 24 were regular users of the internet and that of these nearly 60% used the internet on the move accessing it either through smart phones or through portable computers such as laptops, notebooks or tablets.

When accessing the internet the Eurostat reports states that over 90% of 16-24 year olds use it to send and receive emails and 85% use it for posting messages to, and reading messages from, social media sites [7]. In North America, it is reported that 81% of teens use social media of some kind but that texting is still prominent with 63% saying they use texting to communicate with each other everyday [6]. In what the authors refer to as ‘‘The increasing privatisation of internet use’’ Mascheroni and Ólafsson [8] show that 55% of European children surveyed, aged from 9 to 16 years old, access the internet several times a day from a private location such as their bedroom and that the use of smart phones and tablets increases as the child grows leading to challenges for parents attempting to monitor and mediate internet use. It was also seen that teens are sharing more private and personal information about themselves [6]. Information such as their real name, date of birth, the town where they live, the school they attend and photographs of themselves and their friends.

In the 2014 Annual Bullying survey conducted by a prominent anti-bullying charity Ditch The Label, 45% of the respondents said they had experienced bullying of any sort and 55% of these said that they had experienced cyberbullying [9]. Another report found that 88% of social media using teens had witnessed other users being targeted by cruel or mean comments and that 67% also witnessed other users joining in with the harassment. 15% of users reported that they also had been harassed [10]. The same report also found that 41% of teens reported some negative outcomes resulting directly from their social media use including face to face confrontations that sometimes escalate to physical fights, stains placed on friendships and problems with parents or school. Some teens reported been nervous about going to school because of a social media incident.

Cyberbullying can negatively impact the quality of a teenagers life in many different ways. The victim of bullying can suffer physical stress and a range of emotional feelings including humiliation, isolation, powerlessness, feeling overwhelmed, depressed and even suicidal thoughts. These are feelings that the youth may not be emotionally mature enough to handle. This emotional turmoil can lead to a loss of appetite and an inability to sleep which can cause other more serious health problems. The perpetrator of the bullying could also be a victim of bullying or abuse, and their actions are a backlash against others for what they have experienced. However, it left unchecked, this bullying behaviour could escalate into other antisocial, abusive or criminal activities. By detecting and identifying the bully, intervention may be possible.

When considering cyberbullying and the identification of cyberbullying content, there are two distinct properties to consider. The act of posting a cyberbullying message and the content of the message. The act or delivery of a cyberbullying post can come in many different forms including flaming, exclusion, outing, flooding and masquerading to name a few. Because cyberbullying is often anonymous it is difficult to automatically detect these types of actions. However, the content of a cyberbullying post is a rich textual goldmine where the cruelty of intention, the insidious and harmful nature of the bullying or the hurtful and antagonising tone is plain to see. The content could be overtly sexual or a sexist attack against a persons sexual orientation, racially demeaning or disparaging against a persons race, nationality or skin colour, directly attack a persons appearance, weight or intelligence of their socio-economic status. Cyberstalking and grooming can both also be considered under the cyberbullying umbrella.

1.1 Thesis Research Question

The research question of this thesis is whether standard data mining techniques, for example n-grams, stop word removal, feature selection, term frequency inverse document frequency word vectors, can be used to develop a classifier in Python which can be used to predict whether or not unseen samples are bullying in nature.

1.2 Objectives

The primary objective of this thesis is to develop a model using Python that can be used to determine if samples from an unseen dataset should be classified as bullying in nature or classified as not bullying. To meet this objective the following data mining milestones must be achieved:

- **Construct a new cyberbullying dataset**
  The first objective is to create a new dataset for use in the development of a cyberbullying classifier. As will be seen in Chapter 3 the lack of standard cyberbullying dataset for use in a project like this is well lamented. In Colton and Hofmann [11] it was shown that there was good evidence of cyberbullying on the Ask.fm social networking site so using data scraped from this site a second, more recent, cyberbullying dataset will be generated.

- **Classify the new cyberbullying dataset**
  Once the raw data is sourced it will be manually classified. The criteria to determine whether a sample should be classified as bullying or not will be documented. Once these criteria are understood a classified dataset, that can be used in the development of a classifier, will be generated by manually classifying each of the samples in the dataset.

- **Develop multiple classifiers**
  Multiple different classifiers will be developed using standard text mining techniques such as n-grams, stop word removal, feature selection and term frequency inverse document frequency word vectors. Naive Bayes and Support Vector Machine learner algorithms will be used.

- **Address class imbalance**
  As seen in Colton and Hofmann [11] it is expected that there will be a significant class imbalance between the positive bullying class and the negative not bullying class. To address this class imbalance over, under and hybrid sampling will be explored in addition to investigating cost based classifiers.

- **Identify the top classifiers**
  Once modelling has been completed the top models developed will be identified using each models g-performance.

- **Evaluate top classifiers to determine the best**
  Once the top classifier models are identified each will be further evaluated with previously unseen samples in order to determine which classifier generalises best to new data. This testing attempts to simulate a real life scenario by iteratively classifying previously unseen samples before appending these newly classified records to the master training dataset. The model is then regenerated including these newly classified records before more unseen samples are classified. This process of classify, append, regenerate is repeated multiple times.

1.3 Thesis Structure

Chapter 2 Background and Tooling: This chapter gives a brief introduction to the tools and concepts used in this research. It also provides an overview of the Ask.fm website where the sample data was sourced.

Chapter 3 Literature Review: This chapter provides a critical analysis of research previously undertaken on the specific topics related to this dissertation. It identifies both relevant information and outlines existing knowledge in each of the areas.

Chapter 4 Background and Tooling: This chapter introduces the dataset used in this research. The goal of this chapter is to outline the steps undertaken to transform the raw HTML, scraped from the Ask.fm website, into a dataset that is a suitable starting point for this text mining project. From the initial processing and data analysis steps, through to the final data cleansing steps, the progression from unstructured data into structured text is shown.

Chapter 5 Data Modelling: The focus of this chapter is to describe the process followed to develop the best classifier model for predicting whether a question from the Ask.fm website is either bullying or not bullying. When a number of promising models have been developed, their performance is analysed with the top five chosen for further testing. Before going into the detail of the modelling approach there is a brief refresher of the available data and it structure.

Chapter 6 Conclusion: In the final chapter the objectives, the achievements and conclusions of the thesis are discussed including suggestions for future work.

Chapter 2

Background and Tooling

This chapter gives a brief introduction to the tools and concepts used in this research. It also provides an overview of the Ask.fm website where the sample data was sourced from. The following topics are discussed:

- Ask.fm
- Python, NLTK and Scikit-Learn
- MySQL, SPSS, Excel
- Recall, Precision, Accuracy and G-Performance

2.1 Introduction to Ask.fm

Ask.fm is a social networking website that uses a question and answer format to allow its users to interact. Based in Riga, Latvia, the sites ease of access and the anonymity offered means that it is increasingly being used as a means to communicate abusive, bullying and sexualised content [12]. A review of the Ask.fm ''Safety'' [13], ''Privacy'' [14] and ''Terms'' [15] policy pages in November 2013 revealed the following worrying disclosures. Firstly it was stated that the content of the site was not monitored. The terms of service clearly stated that ''the ask.fm service allows for anonymous content which ask.fm does not monitor'' and that users of the site do so at their own risk. Secondly users were also warned in the terms of service that they may ''encounter content that may be deemed objectionable, obscene or in poor taste''.

Another review of the site in July 2014 showed a more user-friendly and potentially safer site where a list of unacceptable behaviours are given. Filters have been put in place for

the detection of rude or offensive words [16]. Also given are a list of things to do and not to do in order to stay safe on the site and enjoy it [17], a frequently asked questions and answers section for concerned parents [18] and better documented instructions on how to manage your user account and block users or anonymous questions [19]. Probably the most important change seen was the appointment of Annie Mullins OBE as Lead Advisor on User Safety at Ask.fm [20]. Ms Mullins, a member of the UK Home secretary's Task Force on Child Protection on the Internet received an OBE for services to children and young people.

The data from the Ask.fm site used in this research project was extracted from the site the weekend of Friday March 14, 2014 to Monday March 17, 2014.

The following pages give an overview of some of the main features of Ask.fm that a user of the site would use on a regular basis.

2.1.1 Creating an Ask.fm Account

To create an account the user need only provide some standard information such as user name, password, email address and date of birth(Figure 2.1).

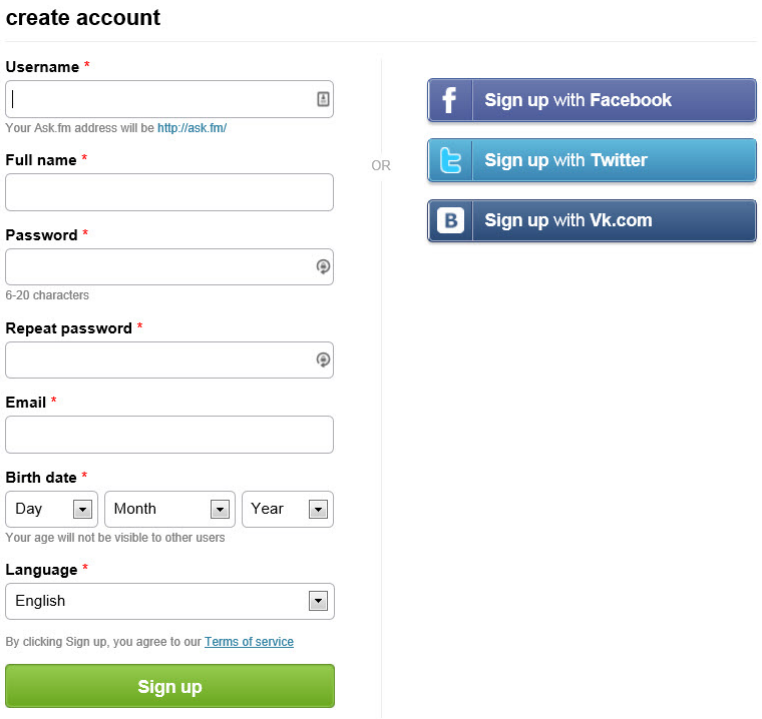

FIGURE 2.1: Creating an Ask.fm Account

There is no obvious validation of the user details entered though an account can be linked to the users Facebook, Twitter or Vk.com accounts.

2.1.2 Questions and Answers

Any unanswered questions the user has will appear on their questions page (Figure 2.2). Users will always receive the question of the day and also have the option of receiving a random question if they have not received questions from other users.

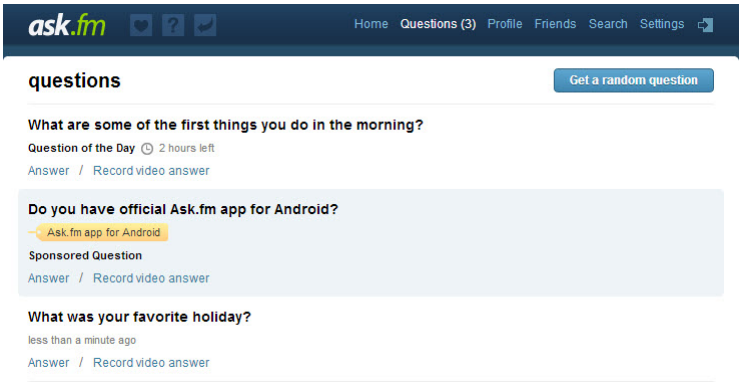

FIGURE 2.2: Your Ask.fm Questions

A question is a maximum of 300 characters long and a question can be asked whether the user has an Ask.fm account or not. Figure 2.3 shows a question being asked by a user who is logged in and notice that the option to ask a question anonymously is also available. It is also possible to ask a question without having an account (Figure 2.4).

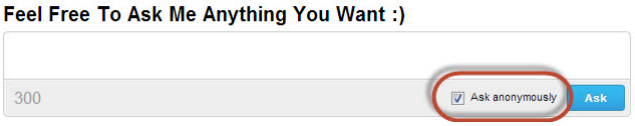

FIGURE 2.3: Asking a question when logged into your account

FIGURE 2.4: Asking a question when not logged into your account

Text, a combination of text and a picture (Figure 2.5) or video can be used to answer the question.

The option to share an answer using Twitter or Facebook is also available to the user.

2.1.3 Privacy Setting, Reporting Abuse and Deleting Questions

It should be highlighted that it is possible to prevent anonymous questions, to delete offensive questions before they are answered and also to report abuse.

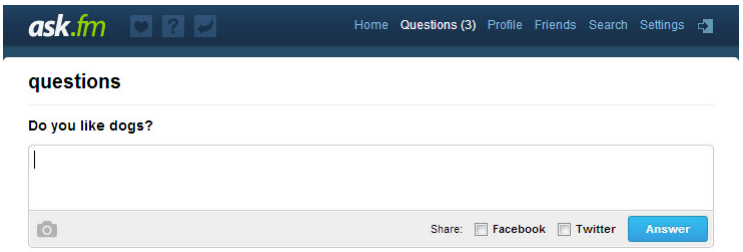

FIGURE 2.5: Answering with text and an optional picture

To configure an Ask.fm account so that anonymous questions are not allowed a user can select the "Do not allow anonymous questions" on their Settings - Privacy page. On this privacy page it is also possible to select not to display answers on the Ask.fm stream (Figure 2.6).

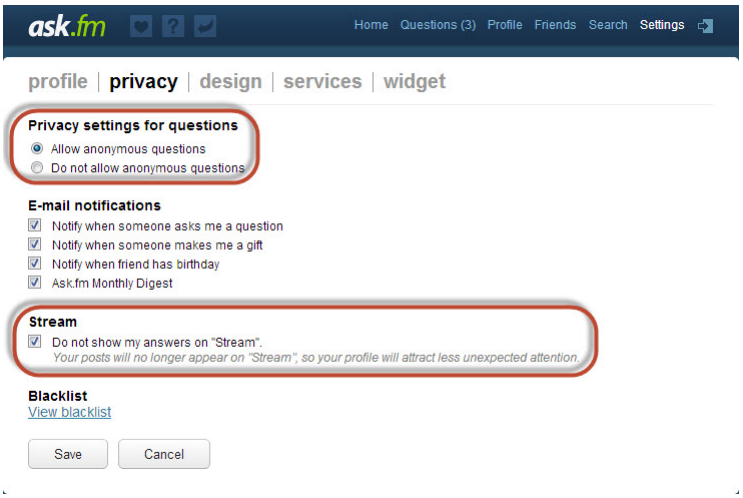

FIGURE 2.6: Configuring privacy settings

Both users and questions/answers can be reported. In the top right-hand corner of each question there is a report flag. The option to report a user is available at the top of the users questions and answers page (Figure 2.7). When the report button is clicked the type of inappropriate behaviour to report can be chosen (Figure 2.8).

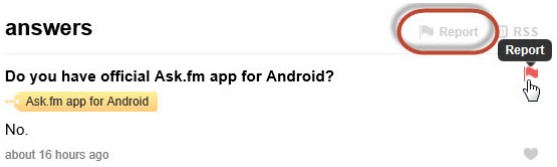

FIGURE 2.7: Reporting a user or a question and answer

It is possible to delete a single question by hovering over the top right-hand side corner of the question where a delete "X" icon will appear (Figure 2.9). Alternatively, it is possible to delete all unanswered questions. Finally, it is also possible to block the asker of the question using the "*Block*" icon. When a block is requested a new dialogue is

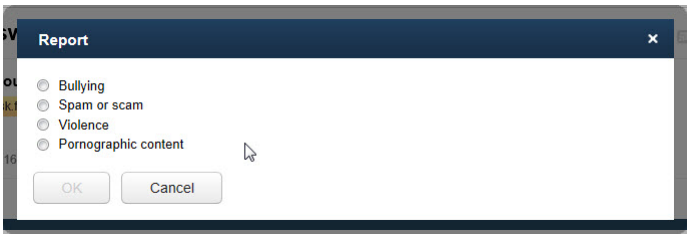

FIGURE 2.8: Types of inappropriate behaviours to report

displayed requesting the reason this user should be blocked (Figure 2.10). This option is presented even when the user has posted anonymously. Blocking a user will cause unanswered question from this user to be deleted.

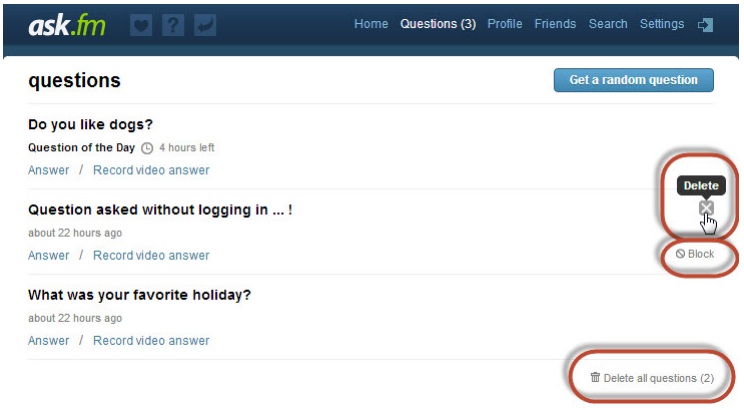

FIGURE 2.9: Deleting questions and blocking the user that asked the question

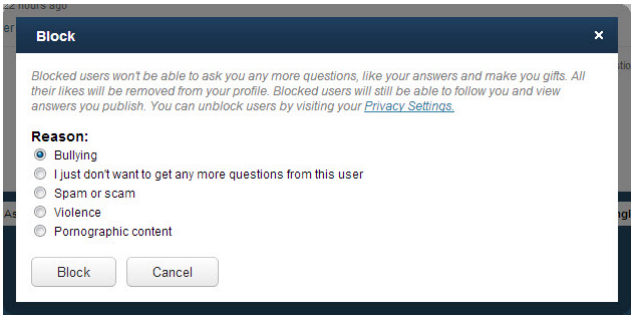

FIGURE 2.10: Blocking a user to prevent them asking further questions

2.1.4 Deactivating an Account

Finally, it is possible to deactivate an Ask.fm account from the users *Settings* - *Profile* page but the account can be reactivated at any time.

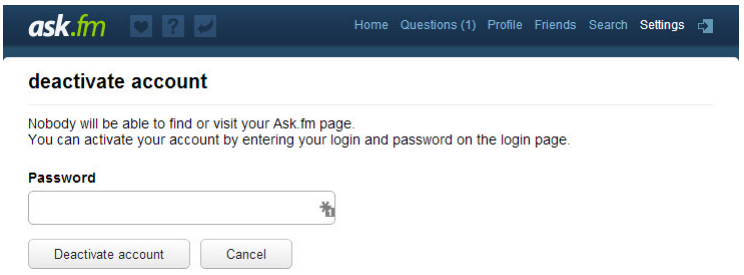

FIGURE 2.11: Deactivating an account

2.2 Python, NLTK and Scikit-Learn

In this section Python, the Natural Language Toolkit (NLTK) and Scikit-Learn are introduced. Python is the primary tool used to extract, manipulate and process the datasets used in this research. NLTK and Scikit-Learn are the packages that will generate the classifiers used to predict if a sample is either bullying or not bullying.

2.2.1 Python

Python is an open source high-level programming language developed under an OSI approved open source license. Although considered an excellent choice for the user who is only starting to learn how to program, Python offers enough features and functionality to meet the demands of the professional object-oriented programmer as well. It is highly scalable, portable, embeddable and suitable for development projects of all sizes. Another great advantage of Python is a very active community that enthusiastically develops, releases and supports commercial quality libraries, packages, tools and platforms that enhance and extend the core Python capabilities. Two of these, NLTK and Scikit-Learn are extensively used in this research project.

Python is extensively used at one of the worlds largest internet company, Google, where it used in the Google App Engine and YouTube to name but a few [21]. Currently, there are two main versions of Python supported, version 2.7.8 and version 3.4. Instructions how to download, install and configure Python can be found on the Python Language homepage [22].

2.2.2 The Natural Language Toolkit

The Natural Language Toolkit (NLTK) is a leading platform for building Python programs to work with human language data [23]. The NLTK provides a suite of libraries to handle text processing tasks such as corpora parsing, tokenisation, stemming and classification.

The NLTK also provides libraries that support part of speech tagging, which converts a sentence into a list of word / tag tuples, and chunk extraction that can be used to extract a short phrase from a sentence that has previously been part of speech tagged. The NLTK also provides easy access to WordNet [24], a large lexical database of English nouns, verbs, adjectives and adverbs, and access to over 50 corpora.

Like Python the NLTK is a free, open source project. It is community driven and is available for Linux, Windows and Mac OS machines. It is currently based on Python version 2.7 but support for version 3 of Python in planned.

2.2.3 Scikit-Learn

Scikit-Learn is a simple and efficient tool for data mining, data analysis and machine learning in Python [25]. It utilises other Python libraries such as NumPy, SciPy, and Matplotlib. Like Python and the NLTK, it is open source available under a BSD license. Scikit-Learn includes many supervised learning algorithms including Naive Bayes, Support Vector Machines, Random Forests and Decision Trees as well as many unsupervised learning techniques including clustering and hidden Markov models. As well as supporting cross-validation and providing comprehensive grid search capabilities for parameter optimization, Scikit-Learn also provides extensive model evaluation metrics and scoring options including, for example, precision, recall and accuracy as well easily accessible confusion matrices. Feature extraction including TF-IDF is also supported.

2.2.4 Python Resources

While writing this research thesis the following books provided invaluable support:

- Think Python [26]
- Python Text Processing with NLTK 2.0 Cookbook [27]
- Learning scikit-learn: Machine Learning in Python [28]
- Building Machine Learning Systems with Python [4]

2.3 MySQL, SPSS and Excel

MySQL, SPSS and Excel are other tools that were used to a lesser degree throughout the course of this research project.

MySQL is one of the world's most popular open source database. It was used to house the question records from the Ask.fm site before writing the data to disk in NLTK corpora format. SPSS is predictive analytics software from IBM. It includes a lot of tooling to allow the elegant manipulation of textual data including stratified sampling which was made use of in Chapter 4. Excel is a spreadsheet package from Microsoft that was used to generate all the charts seen in this dissertation.

2.4 Precision, Recall, Accuracy and G-Performance

When evaluating the performance of a model in predicting whether a question was correctly classified as bullying or not precision, recall and accuracy will be used. When predicting whether a question is bullying there are four possible outcomes:

1. **True Positive** (TP)
   A true positive is where the question is predicted as bullying and was also classified as bullying
2. **False Positive** (FP)
   A false positive is where the question is predicted as bullying but was also classified as not bullying
3. **False Negative** (FN)
   A false negative is where the question is predicted as not bullying but was also classified as bullying
4. **True Negative** (TN)
   A true negative is where the question is predicted as not bullying and was also classified as not bullying

The overall accuracy of a model is calculated as:

$$\frac{NumberofTruePositives + NumberofTrueNegatives}{TotalNumberofExamples} \tag{2.1}$$

Positive Class Precision is calculated as:

$$\frac{NumberofTruePositives}{NumberofTruePositives + NumberofFalsePositives} \tag{2.2}$$

Positive Class Recall is calculated as:

$$\frac{NumberofTruePositives}{NumberofTruePositives + NumberofFalseNegatives} \tag{2.3}$$

Negative Class Precision is calculated as:

$$\frac{NumberofTrueNegatives}{NumberofTrueNegatives + NumberofFalseNegatives} \tag{2.4}$$

Negative Class Recall is calculated as:

$$\frac{NumberofTrueNegatives}{NumberofTrueNegatives + NumberofFalsePositives} \tag{2.5}$$

Precision and recall are inversely related meaning that as precision increases recall decreases and inversely where recall increases precision decreases. When developing a classification model the critical decision is whether to seek to have high precision and low recall or to develop a model that delivers a low precision value but has high recall. Consider a scenario where we are trying to classify questions as bullying. High precision and low recall values suggest that a high percentage of questions predicted as bullying will be bullying. However, a significant number of bullying questions will not be correctly identified. A high recall value implies that a large percentage of bullying questions have been correctly identified but, as a consequence, a large number of not bullying questions would also be incorrectly identified as bullying yielding low precision. Usually a trade-off has to be made between precision and recall depending on the situation and the preferred outcomes.

2.4.1 G-Performance

In addition to standard classifier performance measures, for example accuracy, precision, recall and F-Measure Kubat and Matwin [29] describes another measure that uses the geometric mean of the accuracies measured separately on each class called the g-performance. The goal of this measure is to maximise the recall of both class but at the same time keeping them balanced such that a poor value for either the positive or negative class will give an overall poor performance for the classifier.

G-Performance is calculated as:

$$g = \sqrt{\frac{TruePositives}{TruePositives + FalseNegatives} \cdot \frac{TrueNegatives}{TrueNegatives + FalsePositives}} \tag{2.6}$$

Chapter 3

Literature Review

The primary focus of this research paper is the automated detection of cyberbullying using tools, techniques and processes from the world of text mining. However, before delving into the details of the approaches that could be used, and the issues presented by this task, it is important to understand cyberbullying and the adverse affects it has on the adolescents and teenagers of today.

3.1 Cyberbullying: What is it and why should we care

According to Nahar et al. [30] bullying has moved out of the school yard and is now causing concern online as cyberbullying. Dadvar et al. [31] highlight that in this age of digital communications you can have hundreds of virtual friends having never met them face to face. Kontostathis et al. [32] suggest that the internet and social media applications are being used, particularly by children and teenagers, as a new way to bully, to cyberbully.

We begin this review of the literature with an attempt to understand the phenomenon known as cyberbullying. The psychology of bullying and cyberbullying is well-covered elsewhere and is not within the scope of this paper. However, as the principal area targeted by this research is the automatic detection of cyberbullying, it is important first to understand how this new form of bullying is enacted on its victims and then also consider the impact of it on the victims.

The United States Department of Health and Human Services (DOH) [33] highlights that bullying exists where there is either a perceived or actual imbalance of power and there is repeated aggressive behaviour. These behaviours could be verbal, for example teasing or name calling, social, including spreading rumours and exclusionary acts or

threatened or actual physical violence. Xu et al. [34] concur with these descriptions of face to face encounters as being what Dadvar et al. [31] refer to as traditional bullying. Citing the work of others [35] [36] [37] bullying is described as taking many forms but the most common are physical or direct aggression, indirect aggressions like name calling and relational or social aggressions such as exclusion. Xu et al. [34] also highlights that this bullying can be more than an isolated single event and the intended victim can be repeatedly subjected to the abusive behaviour over time.

The DOH describes cyberbullying as bullying using communication tools such as instant messaging, chat sites and social networks using smart phones, computers or tablets. The DOH also highlights that cyberbullying can't be easily turned off and can happen twenty four hours a day seven days a week. To further exasperate the situation the bullying messages or texts can rapidly spread to a large on-line community. Also, the bully can be anonymous and difficult to trace and completely purging the internet of the offending text or image is next to impossible. The persistent nature of the world wide web and functionality provided by social media sites means that these abusive posts can quickly spread amongst a group and can subsequently be accessed again and again by both the victim and the perpetrator [31] [38] [39]. The implications of this is that the offending text or image can continuously reappear causing distress to the victim again and again. The effect of the cyberbullying can be traumatic on the victim leading to trouble sleeping, withdrawing from society, stress and more troubling mental health problems like anxiety, depression and suicidal thoughts.

The DOH is not alone in their views, there is much support for their views and opinions. For example Dadvar et al. [40] and Nahar et al. [38] are both of the opinion that cyberbullying is an electronic act, aggressive in nature, against a victim who is typically unable to defend themselves. Dinakar et al. [41] and Rybnicek et al. [42] describes these electronic acts as images or text messages posted to social media sites with the sole intent of hurting or embarrassing the victim through these repeated offensive postings.

It is also important to consider the tone and content of a cyberbullying incident and the various forms that it may take. Apart from the threat of physical violence, or the wishing harm on a person, Dinakar et al. [43] describe three main categorises of cyberbullying as sexual, racial and direct attacks against a person. Sexist attacks are typically against women or sexual minorities such as gay, lesbian, bi-sexual and transsexual individuals and groups. Racial or cultural attacks are typically against a cultural minority and its traditions. Direct personal cyberbullying attacks a persons intelligence or physical appearance for example their weight, height, appearance or IQ.

Chen et al. [44] describes cyberbullying as communications that disparage an individual or group on the basis of their nationality, ethnicity, colour or race, their gender, sexual

orientation or religion. Xu et al. [45] also includes socio-economic status as a cyberbullying category under which the victim can be targeted. Willard [46] [47] describes the many forms a cyberbullying attack can take as harassment, flaming, outing, exclusion, flooding, cyberstalking, impersonation or masquerading, trolling and denigration of their victims by the bullies. Ptaszynski, Michal et al. [48] bring to our attention and initiative from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) in Japan. Following several tragic suicides attributed to cyberbullying MEXT developed a manual to assist school teachers to identify cases of cyberbullying [49]. MEXT divided cyberbullying into two separate categories, cyberbullying posts appearing on blogs, forums and private profile sites and cyberbullying emails. These posts and emails can contain libellous or slanderous content, the unauthorised disclosure of sensitive personal data or posts intended to humiliate the victim.

The affects of cyberbullying on its victims can sometimes have tragic consequences as previously highlighted. News headlines like ‘‘Third suicide in weeks linked to cyber-bullying’’ [3], ‘‘Cyberbullies claimed lives of Five teens’’ [4] and ‘‘Hanna Smith suicide fuels calls for action on Ask.fm cyberbullying’’ [5] are now, unfortunately, becoming an all too depressingly frequent occurrence. However, behind these eye-catching headlines the daily torment and abuse suffered by the victims of cyberbullying are taking a dire psychological and emotional toll. The negative affects of posts that contain cyberbullying of a personal or sensitive nature can be internalised by children and young adults leading to significant and emotional psychological suffering says Dinakar et al. [41]. Apart from suicidal thoughts Xu et al. [34] say the affects of cyberbullying can include loneliness, anxiety, low self-worth and signs of depression. Other signs described in Xu et al. [45] are intra-personal problems, school absence or violence and physical complaints.

In their report ‘‘Cyberbullying Among 9-16 Year Olds in Ireland’’ O’Neill and Dinh [50] provide an overview of bullying and cyberbullying statistics in Ireland and Europe. The main findings of their report are that nearly one in four (23%) of all 9-16 year old children had experienced some bullying whether it was on-line or off-line. Although at 4% the overall number of Irish teens that experienced on-line bullying was lower than the European average of 6% it was noted that the more ‘‘experienced internet users’’, particularly users of internet social media sites, reported higher levels of bullying. The most commonly reported form of cyberbullying was being the target of hurtful or nasty messages. Dadvar et al. [39], however, warns that focusing solely on the text of a message may not provide sufficient evidence to conclude that the text is a cyberbullying post. The use of profanities, and what others would consider inappropriate and offensive language, may just be what the protagonists involved consider good banter.

Other types of unacceptable on-line activities that also target adolescents and teenagers include cyberstalking and grooming. Aggarwal et al. [51] define stalking as the unwanted and repeated observation, harassment, intimidation and intrusion of privacy of a person by a predator who wishes to acquire private information about their victim. They continue that the anonymity and wealth of new stalking opportunities offered by the internet has given the stalkers a feeling of relative safety as the technical ability and internet competency of the stalker typically exceeds that of the victim. Rybnicek et al. [42] say that the grooming of a child is typically performed by adults who approach children in order to have sexually explicit conversations, to exchange nude photographs or videos or to meet up in person resulting in the sexual abuse of the child. Elzinga et al. [52] say that there are seven stages of grooming. The first two stages are sweet greetings, where pet names like *''sweety''* are used to create an atmosphere of intimacy, and compliments, which is used to strengthen the intimacy sphere by complimenting the targets looks for example. Stages three and four are called intimate parts and sexual handling where the intimate parts or genitalia of the body are introduced to the conversation by using their popular names, for example *''boobs''* or *''willy''*, before the chat becomes more explicit. In stage five the groomer will use photographs and videos to show his private parts and attempt to get the intended victim to do likewise before in stages six and seven the location, where, and the date, when, the groomer can meet the victim is discussed

So how then can modern machine learning and computing help in the detection of cyberbullying? Some of the approaches used in the detection of cyberbullying are further explored next.

3.2 Mining for Cyberbullying Text

The application of text mining tools, techniques and processes to the automatic detection of cyberbullying text is still a relatively new research area [34] [45]. There appears to be a clear trend where text mining approaches are supplemented with other techniques such as the analysis of the social graph of the bully and the victim or by determining the role played by each party in a cyberbullying incident. When training a classifier to identify documents of different types a ''bag of words'' approach is a commonly used method that uses the frequency of the occurrence of each word in a document as a feature used to help determine a documents class. One of the first mentions of a bag of words in this linguistic context was in the 1953 ''Distributional Structure'' paper by Harris [53]. The bag of words approach is at the heart of many attempts to automatically detect cyberbullying.

Yin et al. [54], in ‘‘Detection of Harassment on Web 2.0’’ presents an approach to the detection of harassment that utilises content, sentiment and contextual features of documents. Using the Kongregate, Slashdot and MySpace datasets presented by Fundación Barcelona Media (FBM) for analysis at the CAW 2.0 workshop [55], the goal of this research was the identification of deprecating remarks. Three local features N-Grams, Foul words and Term Frequency - Inverse Document Frequency (TF-IDF) are used and local features are described as features that can be extracted directly from the text of the message.

N-Grams are a contiguous sequence of n items from an extract of text or speech and in this occurrence the n-grams are words. For example, given a sequence of words ‘‘The quick brown fox’’ the following tri-grams, or sequences of three words, can be formed ‘‘The quick brown’’ and ‘‘quick brown fox’’. TF-IDF is a statistical measure of how important a given word is in a collection of documents. Each document is represented as a vector of words and each word is represented in the vector by a value that is indicative of its importance in predicting the class of the document. The TD-IDF weight for a word i in document j is given as:

$$TFIDF_{ij} = TF_{ij} \cdot IDF_i \tag{3.1}$$

Where TF is a measure of a words importance in a document and is calculated as:

$$TF_{ij} = \frac{n_{ij}}{\Sigma_k n_{kj}} \tag{3.2}$$

The number of times a word i appears in a document j is represented by n_{ij} and $\Sigma_k n_{kj}$ is a count of all words in document j. This means that the more times a word appears in a document the larger its value for TF will get. The TF weighting of a word in a document shows its importance within that single document. IDF then shows the importance of a word within the entire collection of documents or corpus and is calculated as:

$$IDF_i = log \frac{\mid P \mid}{\mid \{p_j \ : \ t_i \, \epsilon \, p_j\} \mid} \tag{3.3}$$

Where $\mid P \mid$ is the total number of documents in the corpus or collection and $\mid \{p_j \ : \ t_i \, \epsilon \, p_j\} \mid$ represents the number of documents in which the word, or term, t_i

appears. The nature of the *IDF* value is such that terms which appear in a lot of documents will have a lower score or weight. This means terms that only appear in a single document, or in a small percentage of the documents, will receive a higher score. This higher score makes that word a good discriminator between posts [54].

In their research Yin et al. [54] used the libSVM [56] learner with a linear kernel as their classification tool with ten-fold cross validation. They found a combination of 1, 2 and 3 n-grams, foul or profane language and TF-IDF on their own did not perform very well. When applied separately to the three datasets F-Measure scores of less then 0.2 for n-grams and foul words and between 0.25 and 0.40 for TF-IDF were returned. When combined with sentiment features and contextual features a small improvement in F-Measure values was seen. When reviewing the dataset it was noticed that a lot of the posts classified as harassment contained foul language used in conjunction with pronouns, particularly second person personal pronouns such as ‘‘you’’, ‘‘your’’ and ‘‘yourself’’. To boost the effectiveness of these pronouns the sentiment features were captured together into three groups. The first group treated all second person pronouns together as a single term. The second group took all other pronouns together as a single term, for example, ‘‘he’’, ‘‘she’’, ‘‘her’’ etc. and the final group treated all foul words as a single term. The TF-IDF weighting of the combined terms was calculated for each group. It was identified, for contextual features, that the posts classified as harassment were ‘‘different’’ to their neighbouring posts so a cosine similarity between posts was calculated to assist in the identification of the bullying posts. The authors concluded that combining these sentiment and context feature to the earlier content features lead to an increase in F-Measure values to between approximately 0.3 and 0.44.

In ‘‘Modelling the Detection of Textual Cyberbullying’’, Dinakar et al. [41], a set of binary and multi-class classifiers, developed using a bag of words approach, are proposed for the detection of textual cyberbullying in a corpus of 50,000 YouTube comments. Unlike Yin et al. [54] a set of preprocessing steps were used on the comments including stemming, stop word removal and the removal of repeated characters for example ‘‘lolllllll’’. Stemming is a process by which derived or inflected words are reduced to their stem, sometimes also called the base or root. Using the words stemming and stemmed as examples, these are both based on the word stem. The first algorithm proposed for a stemmer was by Lovins [57] in 1968. When processing natural language, or text, stop words are words which are filtered out of the corpus typically early on in the process [58]. Stop word are usually what are called short function words or words with little or ambiguous meaning. Example of English stop words include ‘‘the’’, ‘‘and’’, ‘‘at’’, ‘‘which’’ etc. After preprocessing, the YouTube comments were divided into several clusters representing different types of harassment, for example, physical appearance, sexuality, race and cultural and intelligence. From each cluster 1,500 comments were

hand-annotated to verify that labels were correctly applied to each cluster. Comments that did not fit in any of the clusters were given a neutral label. Like Yin et al. [54] some general features of the text were used. Both TF-IDF weighted uni-grams and a list of profane words were used. In addition, the Ortony lexicon [59] of words donating negative connotation, the positive words were stripped beforehand, and frequently occurring part of speech (POS) bi-grams were utilised.

In their experiment set-up Dinakar et al. [41] divided their datasets 50% for training, 30% for testing and 20% for validation. A Naive Bayes classifier, as well as three supervised learning methods, were used to train a model. The supervised learner were Support Vector Machines (SVM), J48 which is a C4.5 based decision tree classifier and JRip a propositional rule learner. In the first experiment the models were trained on each dataset separately i.e. a binary classification task. In the second experiment the datasets were combined into a single dataset and the models were trained as a multi-class classifier. Rather than using accuracy alone in the assessment of the performance of the models developed Cohen's kappa statistic was also used. It was found that the binary classifiers developed performed much better than the multi-class classifier. It was also observed that whilst the JRip rule-based model gave the best accuracy with values as high as 80% the SVM gave the best kappa statistic values ranging from 0.72 to 0.79 for the binary classifiers. Kontostathis et al. [60] also utilises the J48 classifier from Weka [61] when attempting to categorise predators and victims in grooming chat transcripts from Perverted Justice [62] and ChatTrack [63].

Reynolds et al. [64] explicitly state that they wanted to avoid a bag of words approach. This was because the feature space of the bag of words approach can become very large, they wished to be able to produce the model in code and they wanted to be able to fully understand the reason a post was considered as cyberbullying. Having reached the conclusion that ''bad'' words, for example, swear words, profanities and offensive words, were a good indicator of cyberbullying a weighted dictionary of terms, built from the terms on www.noswearing.com, was generated. The weighting ranged from 100 to 500 depending on the deemed severity of the word. The dataset used was scrapped from the www.formspring.me website, a questions and answers style website, and labelled using Amazon Mechanical Turks. Using the bad word dictionary the contents of each message was then scored in several ways. Two datasets were created and in the first the count of bad words contained in each post, *NUM*, was calculated and in the second a normalised count of the words, *NORM*, was calculated to reflect the density of the bad words in each post. When calculating the normalised value the severity weight of each bad word was also considered. In additional *SUM* and *TOTAL* values were also calculated were SUM is the overall weighted average badness of a post taking into account the severity weighting of each word and TOTAL is the total number of words. In additional to

the J48, JRip and SVM learners, which were also used in [41], IBK an instance based algorithm was also used. Using ten-fold cross validation the authors report that the NORM dataset outperformed the NUM dataset suggesting that the percentage of bad words in a post is more indicative of cyberbullying than a simple count of bad words. Overall the learners used, with the exception of SVM, were able to correctly identify 78.5% of cyberbullying posts.

Chen et al. [44] presents a two phase Lexical Syntactic Feature (LSF) based framework for the detection of offensive content and users. Their proposed system presents a new method of sentence offensiveness prediction based on the overall offensiveness of the words in the sentence. The offensive is measured using a lexicon constructed using Xu and Sencun Zhu [65], the Urban Dictionary, and a syntactic intensifier that adjusts a words offensiveness based on the context in the sentence. Words defined as being either strongly offensive and given a larger weighting, for example, profanities such as ''fuck''. Weakly offensive words, such as ''stupid'' or ''liar'', were given a smaller weighting. Words that were not offensive were given a neutral weighting. When a pejorative or obscenity is directed at a user, or semantically associated with another pejorative or obscenity, it is suggested that they become more offensive. For example, the word stupid would be considered more offensive when directed at a person, ''are you stupid'', opposed to referring to the latest console game ''it's a stupid game'' giving the word a larger intensifier value. Using this intensifier value the offensive value of a word is adjusted accordingly giving an overall sentence offensive value O_s of:

$$O_s = \sum o_w I_w \tag{3.4}$$

The offensiveness of a user is a combination of the aggregate of the sentence's offensiveness and the extraction of some of the users language styles. The dataset used was a corpus of over two million YouTube comments with minimal preprocessing. Six approaches were used to determine each sentence's offensive values including bag of words, 2-gram, 3-gram, 5-gram, an appraisal approach [66] and the newly proposed LSF approach. Recall, precision and F-Score, F-Measure, were used to evaluate each of the approaches. Also of interest to the authors was a measure of false positives, sentences erroneously scored as offensive, and false negatives, sentences that should have been scored as offensive but were not. A summary of how each approach performed is shown in Figure 3.1 where the Proposed LSF approach performed the best.

The calculation of an individual users offensiveness was a more complicated matter. A number of users, whose sentence offensiveness levels as calculated in phase one was

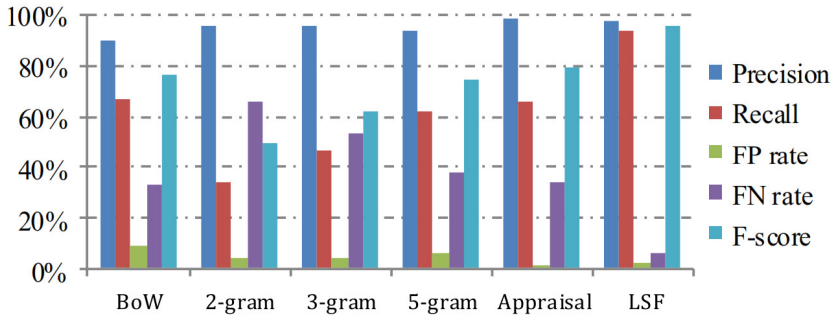

FIGURE 3.1: Accuracies of sentence level offensiveness detection Chen et al. [44]

uniformly distributed, next had their comments manually annotated as either offensive or not. Using Naive Bayes and SVM classifiers and ten-fold cross validation a number of experiments were then run to determine a user offensiveness estimation. The authors found that using both strongly offensive and weakly offensive words in the calculation achieved better results than the proposed LSH. However, on removing the strongly offensive words LSF proved a better predictor. In both cases, the learner that performed the best was SVM.

Like some of the previous papers reviewed Dadvar et al. [40] also used the content features of manually annotated comments collected from YouTube as well as cyberbullying features and user based features of the text. However, rather than use a bag of words and a TDIDF approach the numerical values for each feature was calculated. The first two content features used we have seen before. These are the number of profane words, taken from a dictionary of 414 words and the number of first and second person pronouns. Both of these features were normalised on the overall length of the comment. The third feature was a boolean feature called the profanity window which identified where a profanity was within a given window size, for example, two to five words, of a second person pronoun. The fourth feature was a normalised count of the number of emoticons. The final feature was the ratio of capital letters in the comment, to capture perceived shouting. The cyberbullying features were the normalised number of bullying words based on a manually compiled dictionary of bullying words and also the length of the comment. It was proposed that bullying comments are typically shorter in length than not bullying comments. The final group of features used are user features. To exploit the knowledge already known about each user their historical content features were averaged to determine if there was a history of offensive language use. All the content features listed were considered in this calculation as well as the users age. Before processing stop word removal and stemming was applied. A SVM was used to classify the comments and precision, recall and F-Measure values were used to rate the performance of the model. The content based features were first used on their own to obtain a baseline. The content based features get an F-Measure value of 0.55. Next the cyberbullying features

were also included and their inclusion saw an increase in the F-Measure value to 0.60. The user features were then added and this resulted in an F-Measure value of 0.64 which led the authors to conclude that the inclusion of user features lead to an increase in the detection of cyberbullying accuracy.

Xu et al. [45] say that the computational study of cyberbullying is, except for a few exceptions, largely unexplored. Using simple logic based on manual inspection of a Twitter dataset they calculate that there are probably upwards of 50,000 English bullying tweets a day. They outline three approaches that could be used in the detection of cyberbullying. The first is a Natural Language Processing (NLP) task which uses a sample dataset of tweets that has been keyword filtered to contain the terms ''bully'', ''bullied'' and ''bullying''. 1,762 sample tweets were hand annotated and no pre-processing, such as stop word removal or stemming, was performed but the data was case folded [67], user names were anonymised, and URLs were replaced with the ''HYPERLINK'' token. Emoticons and Hashtags were also treated as tokens. Following this processing three feature representations, uni-grams (1g), uni-grams and bi-grams(1g2g) and part of speech (POS) coloured uni-grams and bi-grams (1g2gPOS) were created. Four common text classifiers, Näive Bayes, SVM with linear kernel, SVM with RBF kernel and Logistic Regression, were trained on dataset sizes ranging from 100 to 1,500 tweet and then tested using tweets that were held back. Although SVM(linear) + 1g2gPOS achieved an accuracy of 81.6% it was decided that SVM(linear) + 1g2g, 81.3% accuracy on the training data and F-measure 0.77 on the test data, was preferred due to its simplicity. Of all the classifiers used, Naive Bayes performed the worst.

The second approach suggested by Xu et al. [45] is based around the role a user takes in a cyberbullying incident. These roles, and their relationship, is shown in Figure 3.2:

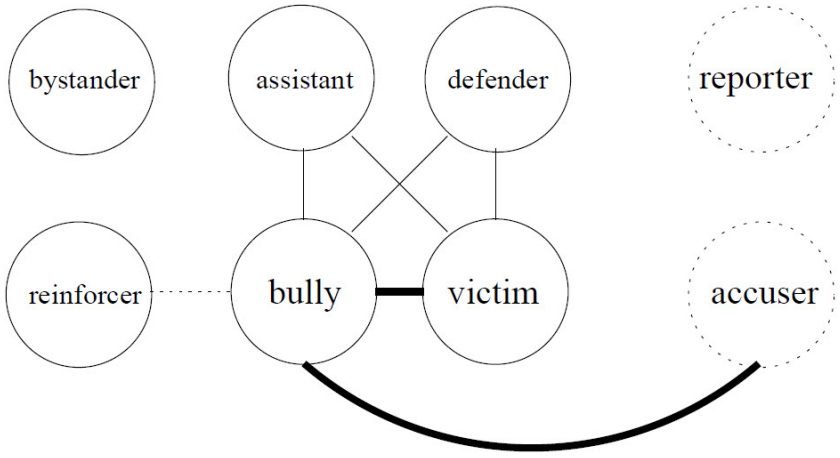

FIGURE 3.2: Roles taken in a cyberbullying incident Xu et al. [45]

The role taken by the bully and the victim are apparent and need not be examined further. The bullying role is supported by the assistant role and reinforcer role. Though

they did not start the bullying the assistant joins in and adds further bullying comments The reinforcer is seen to approve, maybe by liking the bully's post, but does not directly join in. The defender, reporter and accuser roles support the victim. The defender will stand up for the victim, maybe by adding positive comments and also by standing up for the victim. The accuser, however, will go a step further and might confront the bully about the offending comment and point out that they are bullying the victim. The reporter brings the bullying content to the attention of a parent or other authoritative figure. The bystander sees the post and does nothing. The primary interaction is between the bully and the victim but the strength of the line from the accuser to bully is indicative that the confrontation of the bully by the accuser is equally important.

Four of the most frequently observed roles are targeted Accuser (A), Bully (B), Reporter (R) and Victim (V). The other roles were merged into a single generic role of Other (O). Two tasks were then defined. The first is to identify the author of the tweet which is held in an AUTHOR token. The second is to label each person mentioned in the tweet with one of the bullying roles. Following a manual annotation exercise and using the same classifiers and training techniques as before SVM(linear) + 1g2g gave the best results in the first task. The second task compared the performance of a Conditional Random Field (CRF) classifier against the SVM(linear) to correctly identify the various roles. Once again the data was manually annotated and it was found that for this task the CRF outperformed SVM with an accuracy of 87% and an overall F-Measure of 47%. It was discussed that the performance values for precision and recall were low because the tweets were short in length and considered noisy.

Sentiment analysis was the next task undertaken. It was decided to focus on whether a tweet, previously identified as bullying, could be considered as teasing. Hand annotating the tweets from the initial text categorisation task and using the same feature representations and classifiers SVM(linear) + 1g2g again gave the best accuracy performance of 89%. However, overall, nearly half of the teasing example were misclassified. The lack of emoticons or tokens, and the inconsistency of their use, where they are used indiscriminately in both bullying and not bullying tweets was suggested as a possible reason why the overall performance was poor. Another reason suggested was the lack of standardisation in the spelling of the token or the representation of the emoticon.

Finally in an effort to understand the main topics from the cyberbullying tweets a collapsed Gibbs sampling implementation of Latent Dirichlet Allocation (LDA) [68] was run. Six topics of interest highlighted by the authors are feelings, suicide, family, school, verbal bullying and physical bullying.

A novel approach to the detection of cyberbullying by using gender information is suggested by Dadvar et al. [31] . Referring to work by Argamon et al. [69] and Chisholm

[70] the authors investigated the differing bullying vocabularies of males, who tend to use profanities and threatening behaviour, and females, who use a more relational aggression such as social exclusion or by ganging up on an individual. This was shown to be true as seen in Figure 3.3. Using supervised learning on a corpus where the gender of the author was known, the dataset was separated into male and female subsets and the features of each set of posts are extracted. Profane words are treated by calculating their ratio in each post and then normalising. Second person pronouns are again considered important and treated separately to all other pronouns. A TF-IDF of all the words in the post was also calculated. The authors of the paper found that incorporating gender specific features achieved better overall detection accuracy with a 39% increase in precision, 6% increase in recall and 15% improvement in the F-Measure.

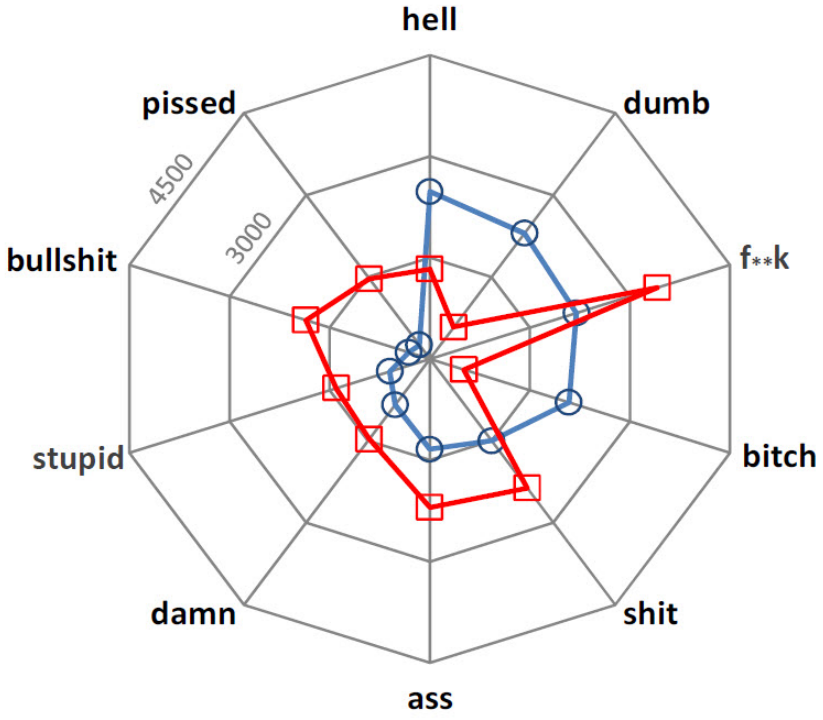

FIGURE 3.3: Top ten profane words for male (square) and female (circle) authors Dadvar et al. [31]

Nahar et al. [38] suggests a two phase approach to identify cyberbullying instigators and victims by graphing the cyberbullying network and by using a ranking algorithm. In phase one harmful messages are detected using a weighted TF-IDF scheme on bullying features, like a bad word dictionary, and an LDA topic modelling approach which, as previously seen, can be used to understand the underlying semantic topics (for example bullying). The second phase involves building a weight directed graph model of the users social networks to determine who the victims are and who the predators are. Each user is given a predator and victim rank based on the number of harmful messages sent or received. A predator is a user that sends messages with harmful words to users with

a higher victim score, a victim is a user that receives messages from users with higher predator scores. Using the users predator and victim ranking in conjunction with the message analysis results from the first phase it was possible to more accurately predict real instances of cyberbullying. Figure 3.4 shows an example weighted graph where *u1* has been identified as a predator and *u2* a victim based on the number of harmful messages sent or received.

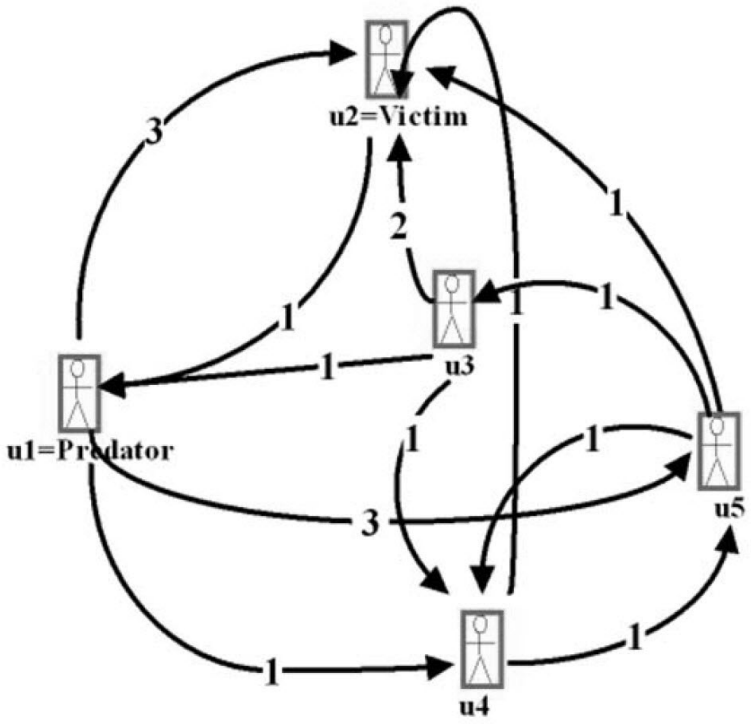

FIGURE 3.4: Weighted graph model identifying predators and victims Nahar et al. [38]

The corpora used were the FBM MySpace, Kongregate and Slashdot datasets from CAW 2009 [55]. The data was deemed to be noisy and was preprocessed to increase the quality and the effectiveness of the analytical steps. The text was converted to lower case then stemming and stop words removal was applied before hyperlinks and extra characters were removed. As seen in other works, three types of semantic features, all second person pronouns, all other pronouns and foul words were defined. A weighted TF-IDF scheme was used with the foul words weighted by a factor by two. Finally, an LDA model was used, and the top features generated were selected. The words generated under the bullying, and not bullying topics were extracted and ranked. A linear kernel SVM classifier was used to generate the model. The weighted TF-IDF results were compared against the previous research baseline values, by the same authors [71], and also against the LDA features. The weighted TF-IDF outperformed the baseline significantly. On the Slashdot an MySpace datasets it was nearly by a factor of three with an F-Measures of 0.30 and 0.31 for the former and 0.87 and 0.92 for the latter. Overall it was considered that weighted TF-IDF outperformed all other methods.

The final paper to be reviewed in this section was Kontostathis et al. [32] where a number of experiments are performed to identify terms that are indicative of cyberbullying. The dataset used was one that was manually scrapped from the www.formspring.me social media site where users interact using a question and answer model. The dataset was labelled using the Amazon Mechanical Turk service with posts identified as cyberbullying labelled as yes and not bullying posts labelled as no. Approximately 11% of posts, or 1,185 out of 10,685, were identified as containing content that could be considered cyberbullying. The first experiments utilised a bag of words approach that the authors consider as simple and transparent. The only pre-processing performed was the conversion of all letters to lower case and the removal of all numeric characters and special characters. It was identified that this could lead to the loss of the emphasis as capital letters can be used to show displeasure. The removal of the special characters would also lead to emoticons being removed which are used to show emotions. No other pre-processing was performed giving 15,713 unique terms which gave a term by document matrix of 10,685 x 15,713 where the intersection of rows and columns represented the number of times a term appeared in a particular post. As discussed previously it was determined that ‘‘bad’’ words are a likely predictor of cyberbullying. A dictionary of bad words was constructed using the terms on the www.noswearing.com website. Each term in the bad word dictionary was then used to query the 10,685 posts returning the number of times each word appears in each post. After all the words in the dictionary had been queried any posts that returned zero bad words were discarded. The remaining posts were sorted by their overall bad word occurrence count. The number of true positive posts in the top 10, 20, 30, 100 and 500 sorted posts was also noted to identify the terms that yielded the highest top scores. Another similar round of context based experiments was also run using the 1,185 identified bullying posts.

The outcome from the first set of queries was to assist in developing another set of query terms that could be used to predict cyberbullying. Precision, recall and f-measure were used to determine the performance of this first set of queries. The result of this analysis was four groups of terms from the bad word dictionary. The first set of words were representative of the terms that results in a precision value of greater than 0.75 and it was found that twenty five terms met this criteria. The next group was made up of 39 terms that achieved a true positive rate for recall of greater than 0.50. The third group of term were representative of words that had at least five posts in the top ten list of posts when sorted by bad word rating of which there were 48 terms. The final content based group consisted of all the terms in the bad word dictionary. A similar analysis was performed on the results of the context queries. There were seven posts with a precision of greater than 0.75 and the words from these posts were used to form the first context based group of words. The second group of context based words was made up of the

terms in the top ten queries that had the highest number of true positives. the final context group contained the words of posts that had at least nine true positives in the top ten ranked documents. These four content based and three context based queries were then run against the data and their results were analysed. The authors found that the bad words content queries, F-Measure 0.49 to 0.57, outperformed the context queries, F-Measure 0.28 - 0.45, even though the context queries contained helper and filler words which it was assumed would help. Whilst it was found that precision values were high, recall values did suffer and while the longer content based queries were effective this was at the cost of performance.

This completes the review of approaches that can be used to mine text posted on social network sites for cyberbullying content. During the course of this review it was clear that a number of issues or obstacles reappeared again and again throughout all the papers reviewed. The next section will give a brief overview of these issues and some of the suggested solutions. The topics covered include anonymity, the availability of suitable datasets, classification or labelling and handling class imbalance.

3.3 Common Obstacles to Overcome

It was mentioned at the end of the previous section that a number of issues or obstacles reappeared again and again throughout the literature review. This next section will give a brief overview of these issues and, where offered, some of the suggested solutions. The topics include anonymity, the availability of suitable datasets, classification or labelling, and class imbalance.

3.3.1 Anonymity

Dinakar et al. [41] identifies that the ability to anonymously post hurtful content online that targets another person and the absence of meaningful supervision of the electronic medium as two of the factors contributing to the increasing social menace of cyberbullying. Yin et al. [54] suggests that the anonymity offered in on-line communities gives the bully the perception of a ‘‘safer’’ environment where their real identity is hidden implying that they are free to bully without fear of discovery. Nahar et al. [38] highlights that an anonymous cyberbullying post can create a heightened sense of fear in the victim as they are confronted with the unknown. The anonymity gives the bully a feeling of power and control over their victim. Reynolds et al. [64] and Kontostathis et al. [32] both agree that www.formstring.me, a questions and answers format social media site, which permits the posting of an anonymous question to a user of the site, is prone to cyberbullying

because of this offer of anonymity. One interesting solution to the anonymous posting of content online, the Real Name Verification Law, is described in Cho et al. [72]. This law, introduced in South Korea in 2007, meant that before users could post on popular sites they first had to verify their real name. The Real Name Verification Law was overturned in 2012 [73].

3.3.2 Data and Classification

Nearly every major paper reviewed expressed frustration at the lack of a standard labelled dataset that could be used in the research of cyberbullying detection. Yin et al. [54] and Nahar et al. [38] both used the MySpace, Kongragate and Slashdot datasets from Fundación Barcelona Media datasets provided for the CAW 2.0 Workshop [55]. Nahar et al. [38] also used the manually labelled dataset from Yin et al. [54] as their ground truth. Dinakar et al. [41] used comments from YouTube videos as their data, treating each comment as a stand-alone comment. Dadvar et al. [40] also used YouTube video comments but Xu et al. [34] and Xu et al. [45] used Twitter tweets as their dataset. Kontostathis et al. [32] and Reynolds et al. [64] both used data that was scraped from the www.formspring.me website.

Once the data had been acquired the next issue was the classification or labelling of the data to identify those posts which were considered to contain cyberbullying content and posts that did not. In most cases, the data was manually classified using a simple binary label [39] [40] [41] [54] by annotators known to the authors but the Mechanical Turk service offered by Amazon was also used on occasion [32] [64]. Xu et al. [34] and Xu et al. [45] used Twitter tweets for their dataset and rather than classifying by hand an enriched dataset was created by automatically filtering on tweets that contained the ‘‘bully’’, ‘‘bullying’’ and ‘‘bullied’’ keywords. In Chen et al. [44] a semi-automated process was used where words were automatically identified as profane using a dictionary and weighted according to their offensiveness.

3.3.3 Class Imbalance

It was also recognised that there existed a class imbalance between the positive bullying class, and the negative class [39] [44]. Reynolds et al. [64] found that only 7.2% of their training dataset was given a positive classification. As a result of this imbalance, it was observed that the chosen learner could achieve accuracy figures of over 90% by ignoring the cyberbullying cases and by just labelling everything as not containing cyberbullying. Yin et al. [54] and Reynolds et al. [64] address this issue by replicating the positive classes.

Weiss et al. [74] suggests that most classifiers are designed to maximise accuracy implying that when used to label a highly imbalanced dataset the more frequently occurring class will be predicted. However, when labelling such a highly skewed dataset it is usually the case that it is the less frequently occurring class that is of interest, for example in medical diagnostics and fraud detection. Three methods were evaluated to handle class imbalance. The first method uses Randon Under Sampling (RUS) or the majority class. The second uses Random Over Sampling (ROS) of the minority class. The third method is a cost based method where the cost of misclassification is built into the learner. Under and over sampling are not without their disadvantages. Under sampling runs the risk of discarding potentially useful data whilst over sampling, besides increasing the learning time required, can lead to over fitting or the generation of a rule specifically for the replicated data. In their experiments a number of datasets were used with the C5.0 cost sensitive decision tree learner [75], an enhanced version of C4.5 [76] learner. For the experiments, the unbalanced datasets were submitted to the C5.0 learner with a variety of costs for misclassification. Each dataset was then rebalanced using both over and under sampling to replicate a ratio that mimicked the cost of misclassification and submitted to the C5.0 learner without a cost. Though the overall result was inconclusive, by measuring the total cost of misclassification at the various costs / ratios it was found that for larger datasets, greater than 10,000 examples, the cost sensitive algorithm outperformed both over and under sampling methods. Over sampling was found to perform the best on smaller datasets, in the experiments this was on datasets of less than 250 examples.

Kubat and Matwin [29] refers to the problem when one dataset is significantly larger than the other as ‘‘Addressing the Curse of Imbalanced Training Sets’’. They identify that it is usually the minority class that is of interest and point out that a classifier that achieves 99.8% accuracy where the minority positive class only consists of 0.2% of the samples is, in fact, of no use at all if it the presence of the samples of interest are completely ignored. In addition to standard classifier performance measures, for example accuracy, precision, recall and F-Measure Kubat and Matwin describe another measure that uses the geometric mean of the accuracies measured separately on each class as $g = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$. The goal of this measure is to maximise the accuracies of both classes but at the same time keeping them balanced such that a poor value for either the positive, or negative class, will give an overall poor performance for the classifier.

There are other papers that suggest alternative approaches to tackle class imbalances. Chawla et al. [77] uses a combination of under sampling of the majority class and over sampling of the minority class using their novel algorithm SMOTE, Synthetic Minority Over-sampling Technique, to create synthetic minority class examples.Cardie [78] presents two case based learning frameworks in a natural language environment that

uses information gained from analysing a baseline case to determine the appropriate class weighting. In Chan and Stolfo [79] the dollar value cost of fraudulent credit card transactions are incorporated into the model as a cost for making an incorrect prediction. A multi classifier meta learning approach is used to handle the non uniform distribution of the samples. García et al. [80] provide an in-depth review of the important research in the area of class imbalance in pattern learning and classification.

Chapter 4

Data Extraction, Exploration and Processing

This chapter introduces the dataset used in this research project. The goal of this chapter is to outline the steps undertaken to transform the raw HTML, scraped from the Ask.fm website, into a dataset that is a suitable starting point to undertake a text mining research project. From the initial processing and data analysing steps, through to the final data cleansing steps, the progression from unstructured data into structured text is shown.

The chapter begins with Section 4.1 describing how the raw data was extracted from the Ask.fm website as well as the selection criteria used to determine which user accounts would be suitable for scraping. This raw data from Ask.fm, initially held in a single HTML file per user account, is then processed in order to extract the information contained within. An overview of these initial data processing steps are given in Section 4.2. This section includes details of the python script used to extract the data, the MySQL database that was used to store the data and also the SPSS process used to partition the data into three separate data blocks.

Section 4.3 gives an overview of the process used in the classification of each question as either bullying or not bullying and also includes detailed descriptions of the various types and categories of cyberbullying to be identified. Also provided are samples of the types of cyberbullying uncovered. Section 4.4 is an exploration of the primary dataset before Section 4.5 gives an in-depth analysis of the processing performed on the data to prepare it for modelling.

4.1 Raw Data Extraction

The first task is the extraction of user data from the Ask.fm website. Before discussing the user data it is important to understand the criteria used to select which users data will be scraped. The most significant criteria is that the predominant language used is English. It would have been outside the scope of this research to consider languages other than English. The second criteria was that the user had at least one thousand, but no more than five thousand, question and answers. There were no other restrictions placed on which users data were eligible for selection.

Each users entire dataset of questions and answers is available from a single URL in the format `http://ask.fm/user_id` where the user id is a unique identifier for each account on the website. It is worth looking again at the layout of the individual questions as they are presented on the screen before then looking under the covers at the structure of the underlying HTML.

Figure 4.1 shows a sample question and answer from the Ask.fm site and the parts numbered one through six can be identified as follows:

1. This is the question that was asked.
2. The answer given to the question.
3. A time indication of when the question was answered.
4. The identity of the user that asked the question. Nothing is shown when the question is asked anonymously.
5. The report flag to report the question and or answer.
6. The number of likes that this question and answer has received. The heart icon can be used by a user of the site to show that they like this question and answer.

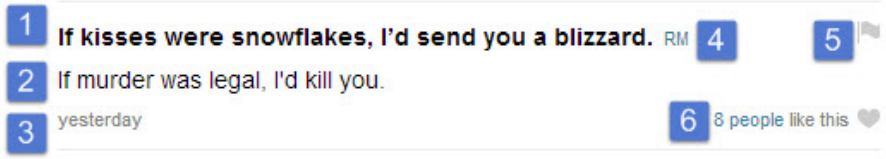

FIGURE 4.1: Sample question and answer from ask.fm showing the question, the answer, when the question was answered, the user that asked the question, the report question flag and the number of likes that this question and answer has.

All questions and answers belonging to a single user can be displayed on a single web page. Initially, only the first 25 are displayed but a "View More" button makes it extremely easy to access all of a users questions and answers without the need for complicated scraping rules. Once all the questions and answers belonging to a user are displayed it is just a simple matter of saving the web page to a HTML file on disk.

Once the criteria to select potential users and the method to extract the users raw data was determined the final action was to download a suitably sized dataset. Having considered the downloading process in advance the goals were to obtain at least 100,000 questions and answers and to try and spread the distribution of the users as much as possible across the globe. It was estimated that there would be an average of approximately 2,000 questions and answers per user account. To meet the first goal at least fifty accounts would be downloaded. To achieve the second goal the data was downloaded in three separate sessions each spaced over a twenty-four hour period. The first set was downloaded early in the morning local time at around 09:00. The second set was download in the early evening around 16:00. The final set was downloaded at 23:00 at night. The final action was to determine which users data to download.

The stream page displays the most recently submitted questions and answers so this page was used to identify potential user accounts. Starting at the top, and working through all twenty five questions on the page, each users account was examined to determine if it met the criteria for selection. If it did then the users complete set of questions and answers were download. If the criteria were not met then the next users questions were examined until all questions on the stream had been reviewed. When there were no further users questions to be examined the stream was refreshed. The process was repeated until a proportional amount of user data had been downloaded at each time slot.

Upon completion, data from 57 accounts had been downloaded yielding 109,312 questions and answers. A cursory examination of the users showed that there was representation from North America, Australia / New Zealand and Europe.

4.2 Initial Data Processing

The next phase was to extract the individual elements from each question and answers combination. The questions and answers from 57 users were downloaded with each users data stored in a single HTML file. To extract the data from these files Python and the Beautiful Soup HTML parsing library were used. After extracting the data from

each file, it was written to a table in a MySQL database. The entire process can be summarised as shown in Figure 4.2.

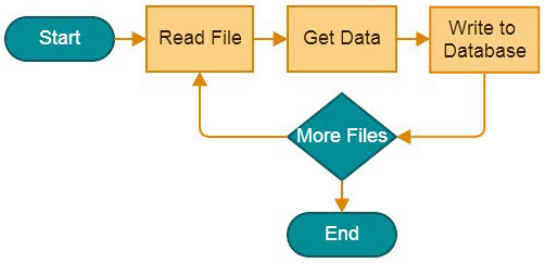

FIGURE 4.2: Simple process flow to extract data from the HTML files downloaded from the Ask.fm website.

4.2.1 Extract data from HTML Files

Access to each HTML file is done using the glob and os packages. Each file is then parsed using the Beautiful Soup HTML parser (line 38). The script used to extract the data from the raw HTML files is included in Appendix A.1.

The individual data elements associated with each question and answer combination are contained within a HTML division element with a class attribute of *"questionBox"*. A sample question box element is shown in Listing 4.1:

Listing 4.1: Sample question and answer HTML

```
1  <div class='questionBox' id='question_box_107612243263'>
2   <div class='question' dir='ltr'>
3    <span>
4     <span>I miss you</span>
5    </span>
6    <span class='author nowrap'>
7     <a href='http://ask.fm/askerid'>
8      askername
9     </a>
10   </span>
11  </div>
12  <div class='reportFlagBox '>
13   <a href='http://ask.fm/userid/questions/107612243263'/>
14  </div>
15  <div class='answer' dir='ltr'>
16   Text me?
17  </div>
18  <div class='time'>
19   <a href='http://ask.fm/userid/answer/107612243263'>
20    about 1 month ago
21   </a>
22  </div>
23  <div class='likeList people-like-block'>
24   <a href='http://ask.fm/likes/userid/question/107612243263'
25    1 person
```

```
26    </a> likes this
27   </div>
28 </div>
```

These data elements are examined in further detail later in this paper.

It is worth considering further the python in Appendix A.1 to understand how the data is extracted from the HTML files. As previously mentioned, each HTML file was parsed using the beautiful soup package resulting in an object called `soup` that encapsulated all the file contents. From an earlier examination of the HTML it was known that each block of data was contained in a HTML division element that had a class attribute with a value of `questionBox`. Line 55 shows how the `soup.findAll()` method is applied to find all `div` elements with a `class` attribute with a value of `questionBox` creating a list containing all question objects. Each question object is then individually retrieved for processing.

```
55 for a_question in soup.findAll("div", {"class": "questionBox"}):
```

Each of the seven data elements can then be extracted from this question object. Taking the question id first, it is extracted from the `id` attribute of the root question box division element. The value of the id attribute is split on the `_` character and the question id is the third element of the array resulting from the split. In the sample HTML shown in Listing 4.1 the question id is `107612243263`. The python for this is shown on line 63. The `strip` method simply removes any leading or trailing whitespace characters that may have been included in the extracted string.

```
63 question_id = a_question["id"].split("_")[2].strip()
```

The identification of the asker of the question and the number of likes that the question and answer have received are two more complicated examples because values for these elements may not exist so a check must be first performed to ascertain if the elements exist or not. To do this, a `try` block is used. Taking the identification of the asker of the question as an example, the `find` method is used on the question object to search for an element that has an `author nowrap` attribute with a child `a` element. If so then the asker identification is in contained in the `href` attribute. As the value of this attribute is a URL the string is split on the `/` character with the fourth value of the resulting array containing the required value. If the required elements do not exist in the question object, that is the question was asked anonymously, an exception is caught and the data element is set to an empty value:

```
78 try:
79     if a_question.find(attrs="author nowrap").          \
80         a["href"].split("/")[3]:
81       ask_id = a_question.find(attrs="author nowrap").  \
82       a["href"].split("/")[3].strip()
```

```
except (AttributeError, KeyError):
    ask_id = ""
```

The final step is to write the data extracted from the HTML files to a database. For this purpose a MySQL database called `thesis` was created as well as a table called `raw_data` to house the question and answer data elements. In the Python script the first step is to create a database connection and then using this connection create a cursor object. When the data elements for each question and answer block have been extracted the required SQL code is then executed to write the data to the database. All database related code is executed with a try block in case an exception is thrown. In Appendix A.1 lines 41 - 50 show where the database connection is created and lines 105 - 124 where each record extracted is written to the database. This is just a brief summary of some of the main functionality of the python script used. Other functionality not discussed includes making the users identification and the question number anonymous.

4.2.2 Splitting the data

The final part of the initial processing performed on the data was to split the 109,312 samples into three blocks of data. It was decided to divide the dataset into blocks of 10% for initial classification, model training and test, 80% to later simulate a stream of data arriving at the Ask.fm website and the final 10% for validation purposes. The details of how the block of 80% of the data and the second 10% validation block will be explained later.

Before splitting the dataset, it was decided that the only criteria imposed would be that the proportions of questions per user should be maintained. For example, if user `X` has 2,400 questions, after the split this user would have approximately 240 questions in the first block, 1,920 in the second block and 240 in the third.

Although there are many ways to split data in the manner required, called stratified sampling, it was decided to use IBM SPSS Modeler. The model used is shown in Figure 4.3.

Using a `SQL` operator a connection is created to the database and the raw data is extracted. It should be noted that this operator uses an ODBC connection to the database that must be created is advance. A `Partition` operator is then used to split the data as required on the anonymous user id. The details of this operator are shown in Figure 4.4. The required split into three groups, based on the anonymous user identification field `u_id` field, can be clearly seen. In the Figure 4.4 it can also be seen that a new partition attribute is added to the data to help identify which group each record belongs to. This

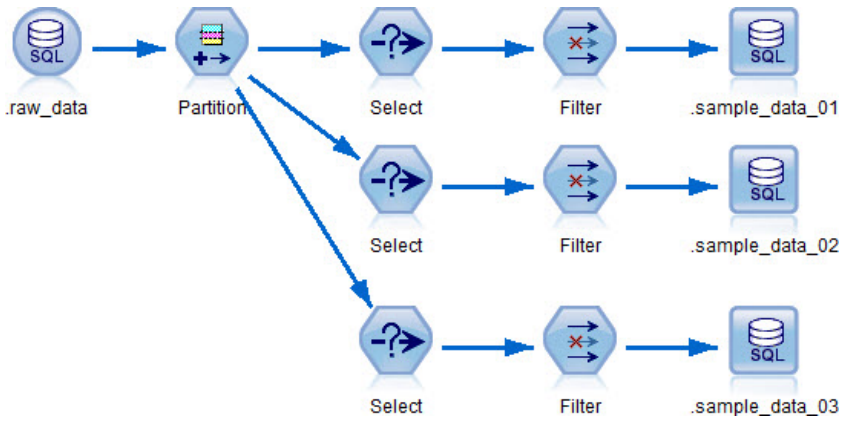

FIGURE 4.3: IBM SPSS Modeler process to split the dataset into three groups.

new attribute will be used in the next operator to split the data into the three required groups.

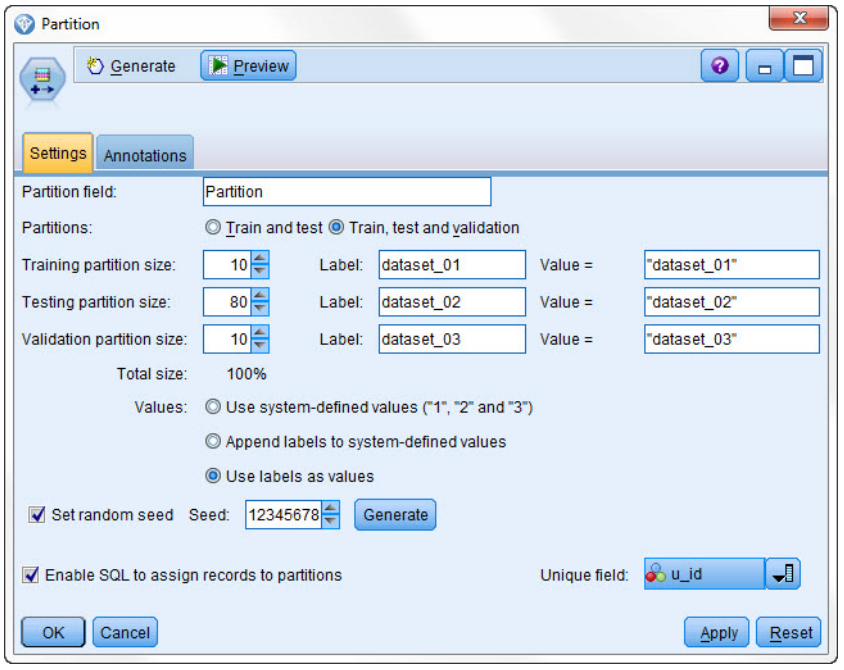

FIGURE 4.4: The Partition operator that is used generate a new partition attribute on which the data will be split.

The output of the `Partition` operator is then used as input into three separate `Select` operators. The input to each of these operators is the same, the complete dataset, but as can be seen in Figure 4.5 only records that match the given condition, in this case `Partition = "dataset_01"`, are passed through to the next operator which is the `Filter` operator. Although it would be possible at this point to write the entire record of the question to the database, this would represent significant data repetition and redundancy. Using views in the database means that it is only necessary to write the anonymous question id of each record to the database. Once the question id is know

then it will be possible to access any other attributes or columns as required using SQL. Only the anonymous question id, `q_id`, is allowed pass through to the final operator where the anonymous question ids of the three blocks of data are each written to their respective tables created solely for this purpose.

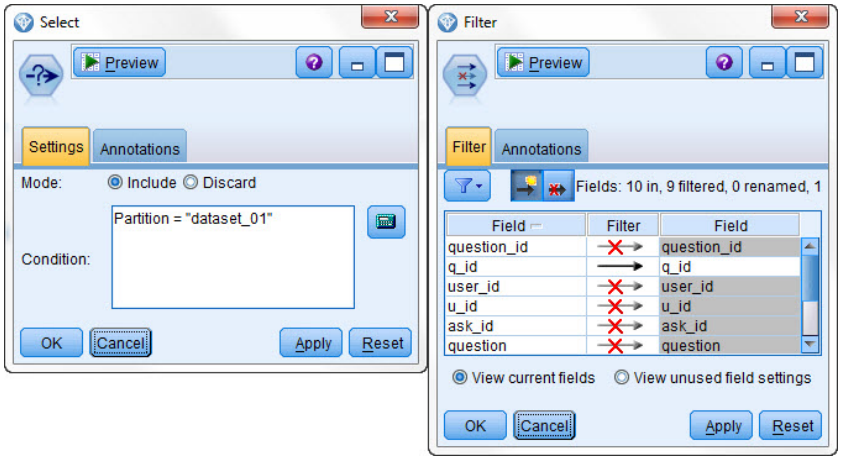

FIGURE 4.5: The Select operator and the Filter operator settings.

Further details about the format, structure and contents of the data and the database housing the data at this point in the process is provided in the Section 4.4.

4.3 Classification

The next task was to analyse the data selected for use in the development and training of a model and then classify each questions as either bullying or not bullying. As described in Chapter 3, cyberbullying can take many forms such as flaming, outing, exclusion, flooding, cyberstalking, impersonation or masquerading, trolling and denigration of their victims. These attacks could then be categorised as sexual, racial, cultural or against the intelligence or physical appearance of a person or their socio-economic status to name but a few. The types and categories of cyberbullying are further examined and analysed to develop a sense of the criteria against which to evaluate each question. Once evaluated, each question will then be classified as bullying or not bullying.

4.3.1 Cyberbullying Criteria

This section provides more details about the type of on-line post, or in this case question, that should be considered as cyberbullying.

''Flaming'' is usually considered as off topic insults or attacks against an individual or group and are more typically seen in a bulletin board or discussion type forums. A classic

example of flaming is where a discussion comparing the virtues of the Linux, Microsoft Windows and Apple Mac operating systems disintegrates into a series of posts where each side questions the intelligence of the other two groups for using such an obviously inferior product. In the context of the ask.fm dataset, it is unlikely that a series of posts of this type would be obviously detectable. The nature of the question and answer format, the order of the answers given, and their intermingling with other question would prevent it. Some questions that are obviously flaming made even be deleted or blocked. However, even if a series of questions that are flaming in their nature, either against an individual or a number of individuals, cannot be specifically identified as flaming their hostility, aggressiveness or derogatory sentiments would probably guarantee that they are identified as bullying. A user that purposefully starts, encourages or engages in such flaming activities is sometimes known as a *''Troll''* and by denigrating or annoying people in this manner they are said to be *''Trolling''*

The term *''outing''* is typically used to describe the disclosure of a persons sexual orientation, for example, revealing for the first time that a person is gay. As the users of the Ask.fm site are mostly teenagers and young adolescents, and because of the amount of sarcasm and black humour observed, it would be difficult reading isolated questions to determine whether a genuine outing has occurred. There is no facility to validate any such outing claim, or to determine if the question is just a slanderous attack. Regardless of these uncertainties, any post that appears to out someone or to question their sexual orientation must be considered as bullying.

Cyber *''exclusion''* can be considered as any type of activity that purposefully excludes an individual or group from an event, a party, shopping trip or holiday, for example. Alternatively, perhaps, they are prevented from joining a sports team or social group. This exclusion could take the form of inviting everyone to a party but then, explicitly, uninviting the excluded person in a public and humiliating manner. Another scenario could be to openly mock or taunt someone because they were prevented from joining a popular social clique in school. Any such obvious exclusionary activity seen in the Ask.fm dataset would be highlighted as bullying.

In some ways *''flooding''* could be seen as similar in nature to a Denial of Service (DoS) attack on a website. A DoS attack is an attempt to interrupt the normal functioning of a website or make it unavailable to users by bombarding the site with so many requests that the sites response time to legitimate traffic is extremely slow or non-existent, rendering the site unusable. In a chat room, flooding could be seen as repeatedly, and in very quick succession, posting empty messages or the same message again and again to prevent other users in the chat room from voicing their opinion. Alternatively, in the context of the Ask.fm dataset, flooding could be considered as the repeated asking of the same

question or requests for information. Such flooding activity, if seen, should be considered bullying. However, as the data has been both anonymised and randomised, or instances of flooding may already have been deleted by the targeted user, it may be difficult to identify.

Stalking is seen as the unwanted, obsessive and criminal intrusion into the personal life of an individual by another person. When computers or other electronic equipment is used it can be considered as *''cyberstalking''*. The actions of a stalker can include activities such as a desire to control their victim, to subversively gather information about them, monitoring and anonymously commenting on their on-line activity and making claims of false victimisation against the target and other false accusations such as defamation and libel. Other behaviours exhibited by the stalker may include flooding the victims with constant requests for personal information or for requests to meet in person. Any questions that are malicious in nature, from their content or tone, should be considered bullying. However, some questions, which may at first appear innocuous, for example ''how old are you'', ''where were you last night'' or ''send me a selfie'' should also be classified as bullying as they are requesting personal information which could be used by a potential stalker.

''Impersonation'' or *''Masquerading''* is where a person creates a false identity, or assumes the identity of someone else, when posting or creating on-line content. Typically the reason for posting content using this assumed or false identity is to harm the reputation of the person they are pretending to be, by making what appear to be personally humiliating comments or by making harmful, harassing, confrontational or bullying remarks against another person. Alternatively the owner of a false account could use it solely for the purpose of anonymously posting harmful content while observing on-line etiquette with their real name account. Due to the anonymous nature of the content on the Ask.fm site and the lack of any validation of a persons true identity it would be difficult if not impossible to ascertain whether a question is asked by a user masquerading or impersonating another person. However, it could also be argued that regardless of who posted the question if it meets any of the criteria that identify cyberbullying then it should be marked as such.

It is also important to consider the tone and content of a cyberbullying incident as well as the various forms that it may take. As well as threats of physical violence, the wishing of harm on a person, or by goading a perform to harm themselves, the tone of a cyberbullying attack can be categorised in many ways including gender, sexist, racial, cultural, nationality, ethnicity, colour or race, intellectual, appearance, religious or socio-economic. Questions that could be considered confrontational, libellous, defamatory

or make unfounded accusations that are intended to be hurtful or that could result in unstated repercussions should be considered as bullying.

Gender or sexist attacks would typically target women or sexual minorities, for example gay, lesbian, bisexual or transgender orientation. Simple examples of this type of cyberbullying would be name calling such as referring to a person as a *''bitch''*, *''faggot''* or *''dyke''* or questions relating to sexual experience. Other more sinister types of these attacks would include unwanted sexual advances or invitations to engage in or descriptions of unsolicited sexual activity. An extreme case would be the threat or implication of rape. Racial or cultural bullying includes slurs or attacks against a cultural minority and its traditions, or direct attacks against a person based solely on their skin colour, ethnicity, race or nationality. Questions asking someone in a derogatory way if they are mentally or physically handicapped or questions that are hurtful or mean about their weight, height or general appearance are all bullying.

Though difficult to identify in isolation, text that may be considered grooming in nature should also be identified. There are multiple different categories of grooming approaches including Approach, Communicative Desensitisation, Compliment, Isolation and Reframing. Samples of each type of categories are *''i just want to meet''*, *''how cum''*, *''you are a really cute girl''*, *''are you alone''* and *''let's have fun together''* [60]

It should be noted that profanity, without any of the other criteria listed here, would not, for the purposes of this research, be considered as cyberbullying.

The criteria for identifying cyberbullying can be summarised as follows:

1. Can readily be identified as being attempts at flaming or of flooding.
2. Is an obvious attempt to exclude or isolate an individual from a group or event.
3. Could be perceived as stalking by overtly asking for personal information or photos
4. May be considered grooming where the question could be seen as an approach, an attempt to isolate or any of the other categories defined.
5. Unnecessarily sexually explicit references or unsolicited invitations or by outing an individual whether real or perceived.
6. Derogatory comments about race, culture, nationality, ethnicity or religion.

4.3.2 Dataset Classification

Once the criteria to be used to identify a question as bullying had been determined the next step was to classify the block of questions that were to be used to train and

test the initial model. Classification was a two step process. First all questions were reviewed and classified as either bullying, not bullying or discard. In total 10,914 question were reviewed. 1,644 were classified as bullying, 78 were classified as discard and the remainder as not bullying. Questions that were classified as discard contained only digits, emoticons or what appeared to be random characters.

To validate that the classification criteria specified are both meaningful and understandable and that the original classification was repeatable, accurate and consistent, a random selection of 500 questions, 250 bullying and 250 not bullying, were independently classified by two other reviewers. Each reviewer was given the classification criteria described here and a print out of the original unedited questions. They were then asked to classify each sample question as either bullying or not bullying. The results from each reviewer is shown in Figure 4.6

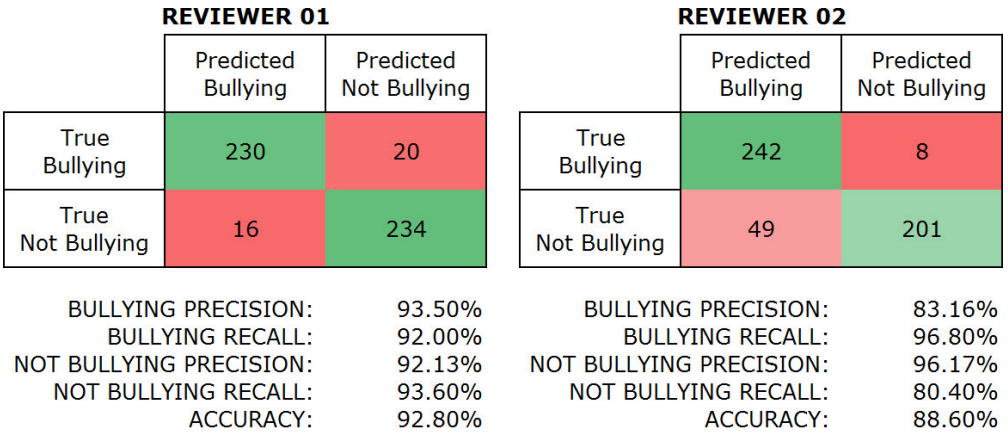

FIGURE 4.6: Matrix showing the number of questions classified as bullying and as not bullying by two independent reviewers

In total, the first reviewer classified 246 questions as bullying of which 230 were originally identified as bullying. The second reviewer predicted 291 questions as bullying, of which 242 were originally identified as bullying. In total 224 of 250 sample questions, or just under 90%, were identified by all three classifiers as bullying. However, 248 out of 250 sample bullying questions, or 99.2%, were identified as bullying by two out of the three classifiers. Of note, 14 sample questions, that were originally classified as not bullying, were classified as bullying by both the independent reviewers. At just over 5.5% this is not hugely significant but important to highlight. Overall, however, this was a very satisfactory validation of the both the classification criteria and the original classification of the complete dataset.

Before leaving this section, it is worth briefly examining some of the questions that were classified as bullying and understand why they were so classified. All the samples listed here are genuine examples from the Ask.fm dataset that was classified. Some of the samples have been abbreviated so as to highlight the relevant part of the question.

- Asking someone their age.
  Asking how old they are could be very innocent and a genuine friendly request. However, it could just as easily be considered as stalking or as grooming.
  - `Age?`
  - `How old are you?`
- Asking for information of an obviously private nature
  - `What Bra size are you?`
  - `can i have ur number?`
  - `How many ladies have you slept with?`
- Asking someone to post a picture or video of themselves often requesting specific parts of the body, clothes or naked.
  - `Puberty progress photo?`
  - `bikini picture?`
  - `Pap of you right now?`
- Asking someone about their sexual experience or preferences
  - `Are you a virgin?`
  - `Are you good at giving head ?`
  - `butt or boobs`
- Threatening physical violence or goading others to hurt themselves
  - `...Cut your self ugly saggy ass hoe`
  - `...I'm gonna beat your trashy little twig of an ass in so far you'll puke it out ...`
  - `...i will slap you across your f**king mouths`

Before leaving this section it is important to note that these are only a small example of the questions that were classified as bullying. The content of a large proportion of the question would be considered crude, vulgar and too offensive to list here.

4.4 Data Exploration

At this point the data from the ask.fm site has been analysed, extracted and classified. The next step is to explore the data and get a better understanding of its contents and quality. It could be argued that an exploration of the data would have been better performed before it was classified. However, because of the simplicity of the structure of the data used in this research project it was decided that there was nothing to be gained in performing the exploration first. In this section, the structure of the data is briefly described before giving details of the exploration performed and also the finding. Finally the quality of the data is reviewed.

4.4.1 Data Description

In the previous section the classification process was described where each question was classified as bullying, not bullying or discard. This classification attribute, along with the anonymous question identify was imported back into the MySQL database to allow further exploration of the data and for future access. A simple view of the database schema is shown in Figure 4.7. This is the view of the schema automatically generated by the MySQL Workbench Tool.

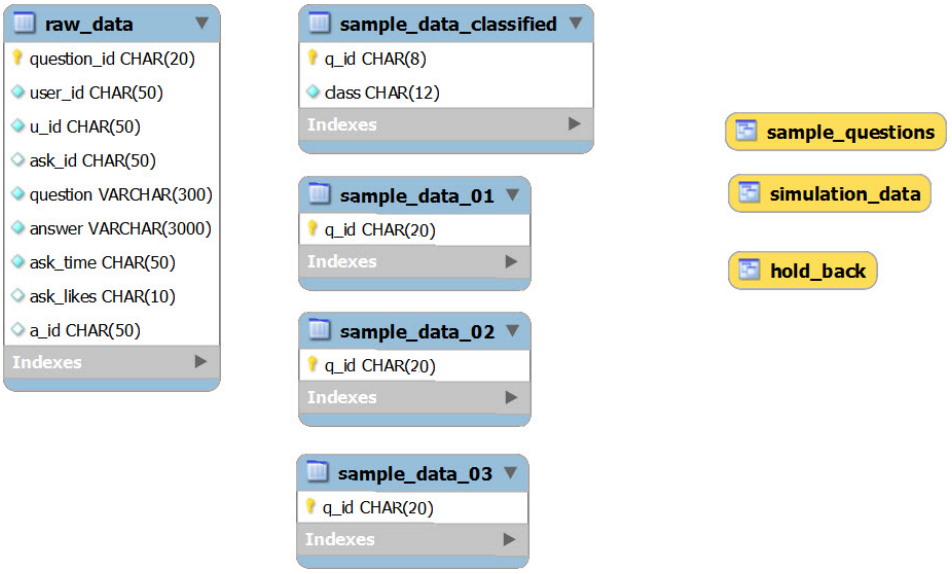

FIGURE 4.7: A simple representation of the thesis schema.

To recap, the `raw_data` table is the all of data as extracted from the HTML files of each of users whose questions were scraped earlier. The `sample_data_01` table contains the anonymous question ids, representing 10% of all questions, to be used for generation of the model and the `sample_data_classified` table is the same list of the anonymous question ids and their classification label. From these tables the `sample_questions`

view was created to allow easy access to the primary data attributes. Over the course of the data exploration exercise many views of the data were created. Table 4.1 provides a brief description of each of the attributes in the `raw_data` table. The only other attribute shown that is not included in this table is the `class` attribute which holds the classification of whether the question is bullying or not bullying.

TABLE 4.1: Table describing the purpose of each attribute in the raw_data table.

Attribute	**Description**
question_id	The ask.fm question id
q_id	The anonymous question id
user_id	The ask.fm id of the user
u_id	The anonymous id of the user
ask_id	The ask.fm id of the asker
a_id	The anonymous id of the asker
question	The question
answer	The answer
ask_time	When the question was asked
ask_likes	The number of likes received

4.4.2 Data Quality

Upon review of the data in the `raw_data` table, it was immediately apparent that there were some minor issues with the text in the question and answers field. As initially only the questions would be used this was where effort was concentrated when determining data quality.

- Questions with only images as answers appeared empty
  These are of no use in the automated classification of the question. There is no feasible way to replace a picture with text so these records will be removed from the dataset.

- Records with only numerical answers
  These will be of little value when trying to establish what features help identify text as bullying so will also be removed.

- Records where part or all of the text was not English.
  Every effort was made though to only select English speaking users. There was little evidence of questions in languages other than English so no further investigation or processing is required.

- Records where non ASCII characters were used
  There was some evidence of the use of non ASCII characters. To simplify and

standardise word tokens non ASCII characters will either be removed or replaced with their ASCII equivalent.

- Slang words, emoticons and abbreviations
  Slang words are usually very descriptive in nature and will be useful when developing the classification model. There is no need to attempt to identify, remove or replace these words. Emoticons are typically inconsistently used and often the same emoticon sentiment can be characterised in many different ways. Emoticons will be removed. To preserve sentence integrity, abbreviations should be replaced with the correct word token.

- Repeated characters
  Sometimes certain letters in words are repeated to emphasise the word. Where appropriate these repeated letters should be removed to reduce the number of distinct word tokens in the corpus.

These issues will further investigated and handled in Section 4.5.

4.4.3 Data Exploration

Now that there is a basic understanding of the data it will be explored in more detail by visualising some of the key attributes. There is not actually much exploration required as the dataset is so simple. The first aspect to be examined is the number of records, questions, each user contributed to the total dataset and to ensure that when this total number of records was partitioned into the three separate data blocks that the approximate user to question ratios were maintained. It is also important to ensure that an average of 10% of each users records were allocated to the block of data to be used for modelling. The total number of records per user is shown in the top chart in Figure 4.8. The criteria for user selection was that each user should have answered at least 1,000 question and no more than approximately 4,000. As can be seen there is a good distribution of the numbers of questions and, although the top three number of questions answered are greater than 4000, it can be said that this criterion has been met. The middle and bottom charts show the count of records each user contributed to the training dataset and the percentage of their total number of questions that this represents respectively. Although it can be seen that the percentages, at the extremes, vary between approximately 8.3% and 11.7%, at an overall average value of 9.92% and a standard deviation of 0.7% this distribution is acceptable.

Next the sample questions that will be used for modelling are further explored. The first two attributes to be examined are the distribution of the questions, between the bullying

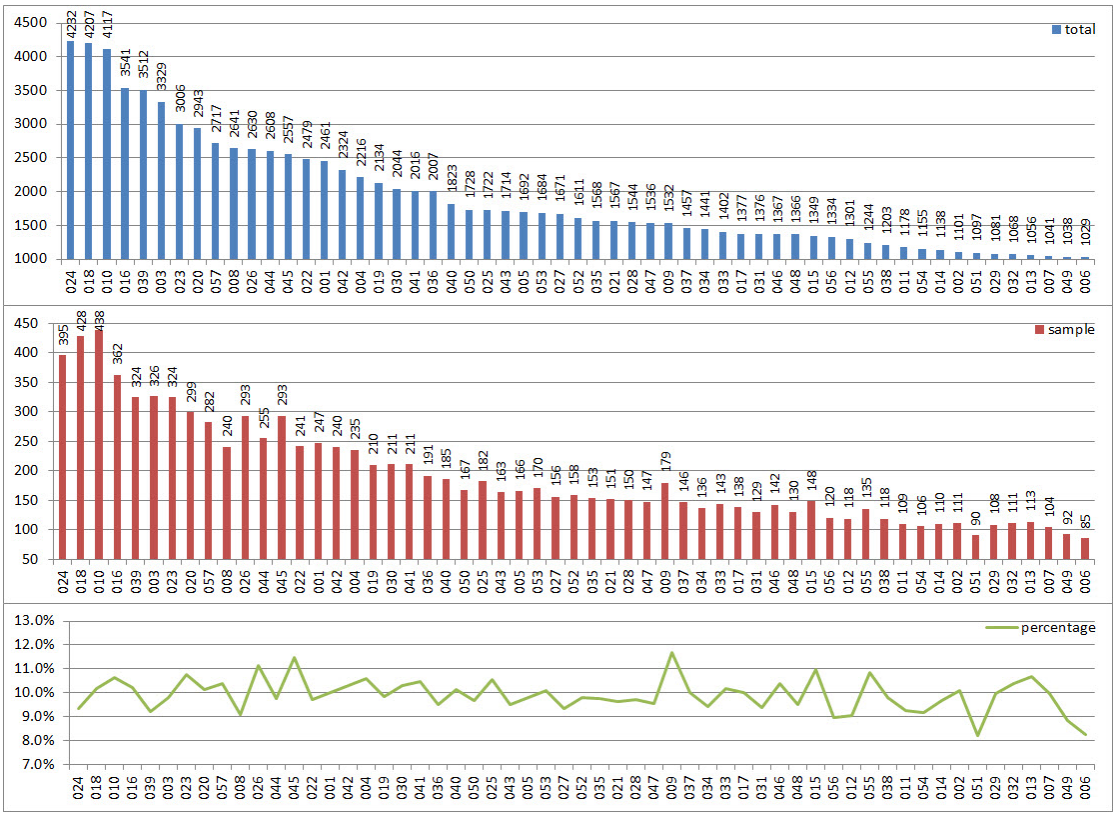

FIGURE 4.8: Distribution of users records showing the total number per user and the number and percentage of each users records in the training dataset

and not bullying classes, and an analysis of the number of anonymous questions that are asked. As shown in Figure 4.9 approximately 1 question in every 7 was classified as bullying and that slightly less than 1% of all question were identified as being not suitable for inclusion and could be discarded. The ratio of bullying question to not bullying questions is in line with figures obtained from other research projects in this area. In total 1,644 questions were classified as bullying.

The number of questions that were asked anonymously, and also the number of questions classified as bullying that were asked anonymously was also examined. There are 10,914 records in the sample questions dataset and it was noticed immediately that the percentage of anonymous questions that are asked was extremely high at 84%. Unsurprisingly the number of anonymous questions that were classified as bullying was also very high at 86% as shown on the right hand side of Figure 4.10. Of the 1,644 questions that were classified as bullying 1,420 were asked anonymously and 224 were not asked anonymously.

Table 4.2 takes a closer look at the 224 question that were not asked anonymously. Two-thirds of the questions that were not asked anonymously were single questions asked by a user. Including instances where an identifiable user asked two questions

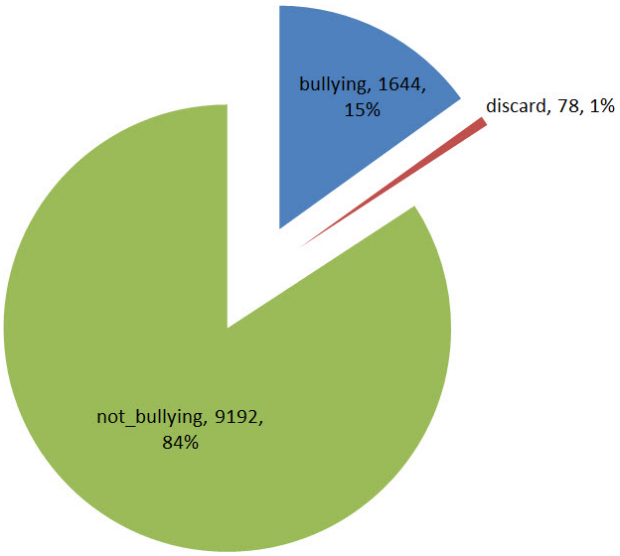

FIGURE 4.9: Class Distribution of Sample Questions

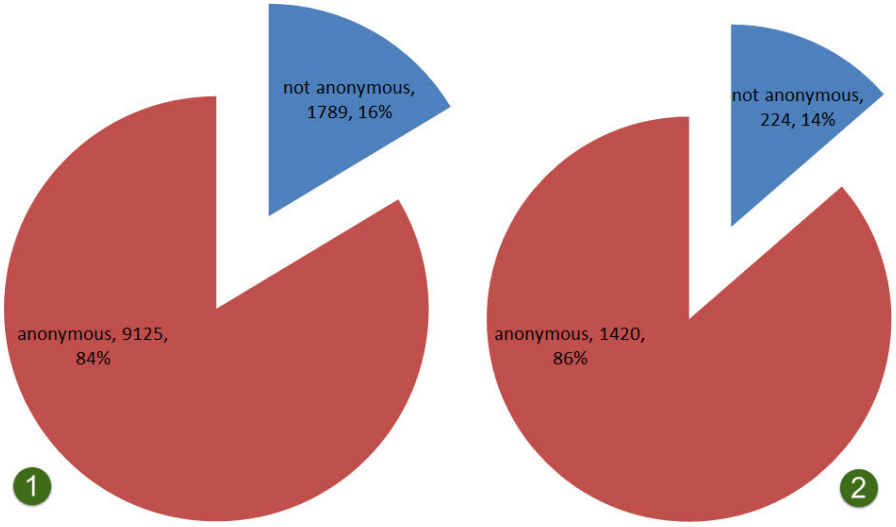

FIGURE 4.10: Ratio of all questions asked anonymously (1) and ratio of questions classified as bullying asked anonymously (2)

that were classified as bullying, nearly 91% of all bullying questions that were not asked anonymously were asked by a user who only asked one or two bullying questions. In total 5 users asked three or more anonymous bullying questions. In each case the bullying questions were asked to the same user. For example, the user identified by id `4145` asked user `57` seven questions, out of a total of thirty seven, that were classified as bullying. This suggests a familiarity between the two users and perhaps the questions are not bullying but just banter between two friends. However, as the total number of questions in this category are low, and the primary goal of this research is solely the identification of bullying posts, such considerations are not in scope now but could possibly be included in any future work.

The average length of questions classified as bullying and those classified as not bullying

TABLE 4.2: Table showing the break down of questions that were not asked anonymously by the id of the asker.

Questions Asked	# Users	# Questions	Percentage
7	1	7	3.1%
4	2	8	3.6%
3	2	6	2.7%
2	27	54	24.1%
1	149	149	66.5%

was analysed. As shown in Figure 4.11 the average length of a question that was classified as bullying is nearly 19% longer than a question that is not. However with an overall average length of 42.3 with a standard deviation of 42.2 this difference in question lengths does not appear to be of great importance.

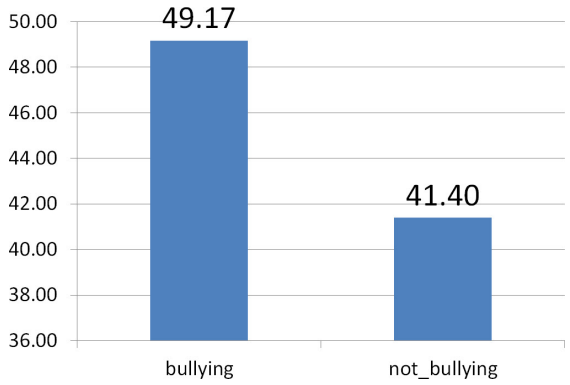

FIGURE 4.11: The average question length for questions classed as bullying and those that are not.

The corpus of distinct words, and the count of the number of times each word was used, was then examined. A simple Python script was developed to identify all distinct words in the corpus and also count the number of occurrences of each word. Unique words, or tokens, that were at least 2 characters long were considered. There were 12,162 distinct string tokens that met this criterion with 85988 tokens in total. The average word occurrence frequency was 7.07 but with a massive standard deviation of 69.38. The questions were then divided into not bullying and bullying corpora, and the analysis performed again. Table 4.3 shows the twenty most frequently occurring words for all questions, not bullying questions and bullying questions. A quick glance at the table shows that when these words are considered there is no obvious discernible difference between the words in the not bullying questions corpus and the words in the bullying questions corpus. When comparing the top twenty words in the not bullying and bullying classes it can be seen that four in every five words, or 80%, appear in both lists.

This word count exploration was performed again. In addition to only selecting words that were at least 2 characters long, all English stop words defined in the Python Natural

TABLE 4.3: Table showing distinct word counts for all questions and separately for not bullying and bullying questions

Total		**Not Bullying**		**Bullying**	
Word	**Count**	**Word**	**Count**	**Word**	**Count**
you	5399	you	4492	you	907
your	1759	what	1520	your	370
the	1716	the	1488	and	313
to	1701	do	1422	to	283
what	1657	to	1418	the	228
do	1573	your	1389	are	223
and	1345	and	1032	of	172
is	1114	is	986	have	158
are	1076	are	853	do	151
like	895	like	753	like	142
in	816	in	687	so	137
of	772	on	612	what	137
have	726	of	600	in	129
on	714	would	571	is	128
would	650	have	568	with	127
so	632	that	517	me	126
that	610	if	497	on	102
it	569	so	495	ur	102
now	561	who	489	how	97
who	555	how	464	that	93

Language Tool Kit were removed. Now, whilst there are still 12,044 distinct tokens, overall there are only 49,942 token instances, a 42% decrease, with an average word occurrence of only 4.15. The results of this word count analysis is shown in Table 4.4. Although there is now some better differentiation between the not bullying and bullying questions, there are still 12 common words in the top twenty words of each category. This suggests that building a model to predict the class of questions using word counts alone will not be predictive enough and that maybe additional strategies will need to be considered.

A common strategy used in text mining classification is the generation of term n-grams. N-grams have the effect of concatenating together two or more tokens to form new hopefully more predictive tokens. For example the quote *''to be or not to be''* produces the following tri-grams:

- to_be_or
- be_or_not
- or_not_to

TABLE 4.4: Table showing distinct word counts for all questions and separately for not bullying and bullying questions with stop words removed

Total		**Not Bullying**		**Bullying**	
Word	**Count**	**Word**	**Count**	**Word**	**Count**
like	895	like	753	like	142
would	650	would	571	ur	102
don't	349	think	307	would	79
think	344	don't	276	don't	73
know	327	know	260	pic	69
ur	323	love	258	know	67
love	302	:)	252	get	58
:)	294	favorite	245	post	56
you're	292	you're	238	f**k	56
get	285	get	227	you're	54
really	261	best	222	want	53
one	253	ur	221	i'm	52
favourite	250	one	220	really	50
i'm	238	people	213	see	50
best	238	really	211	go	50
go	238	what's	208	love	44
people	235	ever	194	old	42
ever	233	thing	189	bitch	42
what's	226	go	188	right	40
want	224	i'm	186	ever	39

- not_to_be

Bi-grams, two tokens, and tri-grams, three tokens, word lists were generated from the bullying and not bullying questions examined earlier. As before only words or tokens that were at least two characters long were considered. Word or token frequencies were determined and these results are shown in Tables 4.5 and 4.6.

Examining the top ten most frequently occurring bi-gram tokens some features are immediately apparent. The first most obvious thing to notice is that where stop words are not removed only three out of the ten not bullying tokens, 30%, also appear in the bullying list. When compared to the top ten uni-grams, single words tokens, where 80% of the top ten tokens appears in both lists this represents a possible indicator that bi-grams would be of more use in the classification of unseen questions. When stop words are removed the number of common words fell from six out of ten repeated uni-grams, 60%, to only two out of 10 or 20% bi-grams. Also of note is the use of `you`, the second person pronoun. When stop words are not removed `you` is prominent, appearing in six out of the top ten bi-grams in both the not bullying and word bullying lists. Another pronoun, `your`, also appears twice in the not bullying top ten list. However, when stop

Table 4.5: Table showing distinct counts of bi-gram tokens for not bullying and bullying questions with and without stop words

Not Bullying				**Bullying**			
With Stopwords		**No Stopwords**		**With Stopwords**		**No Stopwords**	
do_you	959	would_like	74	are_you	93	right_now	28
what_is	344	would_be	47	do_you	72	wearing_right	20
are_you	342	don't_know	42	you_have	42	post_pic	16
would_you	318	one_thing	41	you_are	40	don't_know	15
if_you	259	last_time	40	what_are	40	old_you	14
you_like	246	what's_favorite	40	how_old	37	cut_cut	8
is_the	215	last_person	33	want_to	31	post_picture	7
is_your	198	right_now	32	if_you	30	would_go	6
your_favorite	184	best_friends	27	of_you	30	dont_like	6
have_you	179	best_friend	25	pic_of	29	look_like	6

words are removed `you` does not appear at all in the not bullying top ten and only once in the bullying top ten. This observation flies somewhat in the face of Yin et al. [54] which mentioned that when reviewing their dataset it was noticed that a lot of the posts classified as harassment contained foul language used in conjunction with pronouns, particularly second person personal pronouns such as ''you'', ''your'' and ''yourself''. An examination of the data from Ask.fm does not, at first sight, corroborate these findings.

An examination of the tri-gram tokens in all categories showed that the top ten most frequently occurring tokens were all unique. Expanding this type of comparison to the top fifty tokens, it was found that when stop words were included three out of every ten tokens, or 30%, appeared on both the bullying and not bullying lists. Expanding this to the top 100 tokens the repetition fell further to between 24% and 20% respectively. Examining next the tri-gram tokens where the stop words had been removed was even more revealing. Only two of the top one hundred not bullying tokens appeared anywhere in the bullying list. In the other direction none of the top 67 bullying tokens appeared anywhere in the top one hundred not bullying tokens. The reason the top 67 bullying tokens were chosen is because after these all tokens only had a frequency of one. Expanding the search to include all not bullying and bullying tri-grams, where the stop words had been removed, showed that only 57 tokens appeared in both lists. This represents 57 out of 21031 not bullying tri-grams, approximately 0.27%, and 57 out of 5858 bullying tri-grams or 0.97%. This finding would further support the suggestion that tri-gram tokens should prove to be very good predictors when it comes to developing the models.

One final set of measurements worth considering is the count of distinct tokens in each word list generated and also the average number of times each occurred. This data

TABLE 4.6: Table showing distinct counts of tri-gram tokens for not bullying and bullying questions with and without stop words

Not Bullying			
With Stopwords		**No Stopwords**	
what_is_your	154	would_like_meet	74
what_do_you	153	people_think_you	47
do_you_think	133	favorite_month_year	42
what_is_the	132	many_times_day	41
do_you_like	128	gift_would_like	40
is_your_favorite	110	think_people_think	40
if_you_could	94	what's_longest_you've	33
what_would_you	86	last_thing_made	32
was_the_last	79	anyone_right_now	27
you_like_to	76	made_happy_today	25
Bullying			
With Stopwords		**No Stopwords**	
what_are_you	30	wearing_right_now	20
how_old_are	24	cut_cut_cut	7
are_you_wearing	21	last_person_kissed?	5
wearing_right_now	20	whoever_likes_thinks	4
you_wearing_right	20	see_window_post	4
old_are_you	13	window_post_pic	4
have_you_ever	11	slept_ex_bf	3
of_you_and	10	shannon_slept_ex	3
post_pic_of	10	likes_thinks_you're	3
do_you_have	9	ex_bf_max	3

is summarised in Table 4.7. The data is divided into three sections. The first section describes the dataset when all the tokens are considered together. The second describes the not bullying data and the third the bullying data. As previously mentioned there were 12,162 unique uni-grams with 10,514 occurring in the not bullying dataset and 3,598 occurring in the bullying dataset. Delving a little deeper into this information it was discovered that of the 10,514 not bullying tokens 8,238 of these only occurred in the not bullying set showing that over 78% of these tokens would be good predictors of a not bullying question. Taking the 3,598 bullying token it was seen that 1,575 of these tokens, or approximately 44%, only occurred in the bullying list. Taking bi-gram tokens next, the percentages of distinct tokens used in either the not bullying dataset or the bullying data set rise to 97% and 73% respectively. This diverging of the common words between the two datasets continues all the way up to the tri-gram tokens with stop words removed where, as previously seen, over 99% of the tokens in each dataset are unique.

Next, the distribution of the tokens is further examined and visualised but from Table 4.7

TABLE 4.7: Table showing the number of distinct tokens in each dataset as well as the total token counts and average frequencies

Token description	**Unique Tokens**	**Total Count**	**Average Frequency**
All Words			
Uni-Grams	12162	85988	7.1
Bi-Grams	41012	75128	1.8
Tri-Grams	51111	64983	1.3
Uni-Grams No Stop Words	12044	49942	4.1
Bi-Grams No Stop Words	31370	39125	1.2
Tri-Grams No Stop Words	27106	29567	1.1
Not Bullying			
Uni-Grams	10514	70722	6.7
Bi-Grams	33665	61557	1.8
Tri-Grams	40913	52981	1.3
Uni-Grams No Stop Words	10397	40831	3.9
Bi-Grams No Stop Words	25204	31706	1.3
Tri-Grams No Stop Words	21391	23604	1.1
Bullying			
Uni-Grams	3598	15208	4.2
Bi-Grams	9979	13565	1.4
Tri-Grams	11086	11998	1.1
Uni-grams No Stop Words	3489	9053	2.6
Bi-Grams No Stop Words	6847	7413	1.1
Tri-Grams No Stop Words	5858	5959	1.0

it is clear to see that the average frequency of each token falls from 7.1 for all uni-grams tokens to 4.1 for uni-grams where stop words have been removed all the way down to nearly a 1:1 ratio for bullying tri-grams where stop words are removed. This shows that very few tokens appear more than once in this tri-gram dataset. This observation suggests that a model based solely on these tri-gram tokens may, in fact, be over trained and would not generalise well to unseen data. This contradicts the earlier suggestion that tri-grams would be good predictors and must be taken into consideration and examined when models are being developed.

4.4.4 Data Visualisation

To visual the not bullying and bullying classes, word clouds, generated using the IBM Many Eyes tool, were used. Uni-grams were considered first and the results of these word clouds can be seen in Figures 4.12, 4.13. When generating the visualisations only

the fifty most frequently occurring words were used. It is very clearly seen that `you` dominates both word clouds.

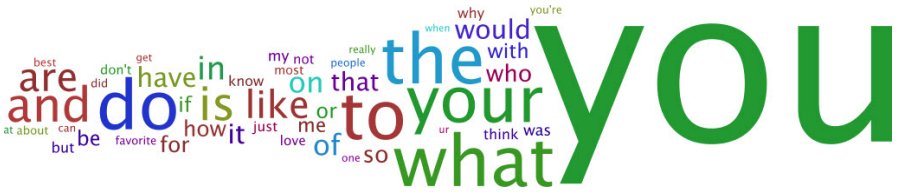

FIGURE 4.12: Word cloud of the 50 most frequently occurring words in questions classified as not bullying

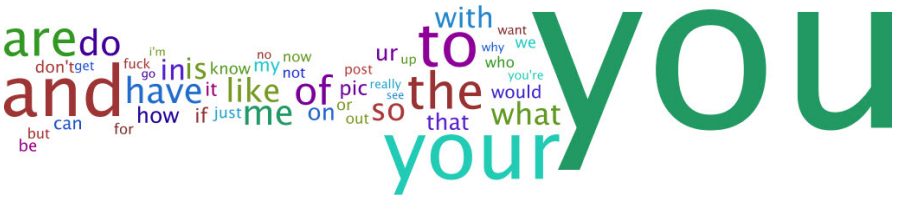

FIGURE 4.13: Word cloud of the 50 most frequently occurring words in questions classified as bullying

The exercise was then repeated with the uni-gram tokens where the NLTK stop words had been removed. The two word lists were loaded onto the Many Eyes website and visualised using the same settings as before. Figures 4.14 and 4.15 show the results. It is immediately obvious that the words maps now appear more balanced with a better distribution of word weights across the images. Even so some words, for example `like`, `would`, `don't` and `know`, are still prominent in both classes. It was also observed that some stop words, for example `you`, were still appearing in the word clouds when the token should have been removed. After some investigation it was discovered that the NLTK did not removed stop words where there was an adjacent punctuation character, for example `you!`. These punctuation characters appear to have been striped when the data was loaded into Many Eyes explaining why they are still seen.

FIGURE 4.14: Filtered word cloud of the 50 most frequently occurring words in questions classified as not bullying with stop words filtered out

Whilst the word clouds present a nice way of visualising the frequencies of words in each of the categories, to generate a cloud for all the permutations already highlighted would

FIGURE 4.15: Filtered word cloud of the 50 most frequently occurring words in questions classified as bullying with stop words filtered out

be very time consuming. Also, any analysis performed on them would be subjective. For example, in Figure 4.15 which word is more prominent `ur` or `pick`. A better way to visualise how the frequencies of words, or tokens, are distributed across each dataset is shown in Figure 4.16. These three graphs shown the normalised word frequencies distributions for the 250 most frequently tokens for all tokens, labelled (1) in the diagram, all not bullying tokens, labelled (2) and all bullying tokens. For brevity, in the legend in each graph *A* represents all tokens, *NB* not bullying and *B* bullying. Similarly *BI* represents bi-grams, *TRI* trigrams and *NO* to show that stop words have been removed. So *B_TRI_NO* is used to represent bullying tri-grams with stop words removed.

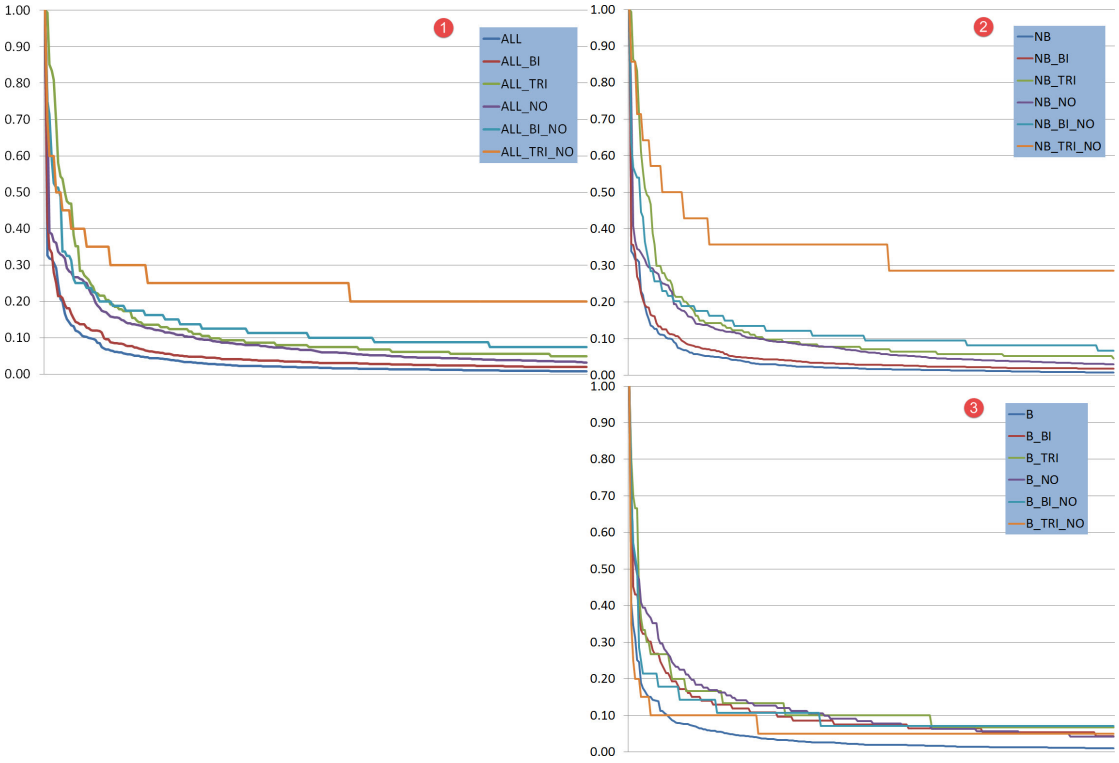

FIGURE 4.16: Graphs showing the normalised word distribution of the different token configurations examined

Analysing these charts some observations can be immediately noted. Firstly the frequency distribution of tri-gram tokens for all words and for not bullying words, both with stop words removed, have a significantly different shape to all other frequencies. The frequency for not bullying bi-grams is particularly distinctive suggesting maybe that this list is more evenly distributed than the others. It was actually expected that the bullying trigrams would have proven the most evenly distributed dataset. However, the counts of the two most frequency occurring tokens, 20 and 7, when compared to the other frequency counts of 5 or less, meant that the distribution graph for this category fell off faster than any other.

4.5 Data Preparation

With a better understanding of the data following its thorough exploration, the final step before modelling begins is to prepare the data. It would possible to include some of the processing of the data described here within the modelling process, for example converting case of all characters to lower case. However, it was decided that performing these steps in advance would allow better control and give greater transparency of the data submitted to classifier models. In this section, the processing or transformations performed are each described before discussing the reasons for each of them.

4.5.1 Cleaning the Data

In total eight transformation steps are applied to the data:

1. Convert to lower case
2. Convert to ASCII
3. Remove URLs
4. Remove all punctuation
5. Replace numeric values 11 to 25 with text values
6. Remove any digits that remain
7. Remove any repeating characters
8. Fix common abbreviations and replace with full word

Python is once again the language of choice to perform these transformations. The data was read from the MySQL database in two blocks, the block of questions that were identified as bullying and the non bullying block. Instead of writing the final questions back to the database the questions were written to a file on disk for later processing by the Python Natural Language Toolkit (NLTK) [81] as described in Chapter 5.

The first four transformations are conversion to lower case, the removal of non ASCII characters and URLs and the removal of any remaining punctuation characters. The complete scripts showing the details of how these string manipulations were performed are given in Appendix A.2. The conversion of all text to lower case is a simple invocation of the `lower` method on the question string which needs no further explanation. During data exploration, it was noticed that there were several characters that were not in the ASCII code set. The `unicodedata.normalize` method shown on lines 28 to 32 of Appendix A.2 will return the normalised version of the passed string. The first parameter passed defines the decomposition to be applied to the Unicode string. Normal form KD (NFKD) will replace all compatibility characters with their equivalents. The string returned is encoded as ASCII. Next a standard regular expression representing a URL is used to remove all URLs by replacing them with an empty string, lines 35 to 38.

Any remaining punctuation characters are removed next as shown on lines 41 to 45. The `translate` string method takes two arguments. The first is a table of characters to translate, and the second is the list of characters to delete. `string.maketrans` is used to create the table of characters to translate but when used as shown an empty table is created. The list of punctuation characters is passed as the second parameter. The effect of passing an empty table of translation characters and the list of punctuation characters is that the punctuation characters are just deleted which is the desired effect. The sixth transformation uses a slightly differently formatted version of the translate method but is used in the same manner to remove any remaining digits as shown on lines 52 and 53. Rather than passing the punctuation characters in this instance the lists of digits are passed, and `None` represents an empty value in this case an empty table of translation characters.

The other three string manipulation routines replace selected numeric values with text, removes repeating characters and fix an identified number of abbreviations with whole words. From the data exploration, it was noticed that there were a number of bullying questions that requested sensitive personal information such as a persons age. To capture this important information, for use during the modelling process, a new class was developed to allow such a replacement. The details of this class are discussed shortly but the code to execute this substitution is given in lines 48 and 49 of Appendix A.2. An instance of the `RegexReplacerAge` is created and then the replace method is called on

the passed string returning a new string where the required values have been replaced. The `RegexReplacer` class used to replace common word abbreviations uses the same logic taking a string and returning the same string where occurrences of defined abbreviated words are expanded to their full representation.

The final piece of code to be examined is the code that removes repeating characters from the text of the questions. A `RepeatReplacer` object is created as well as any empty array. Then for each word in the question, obtained using the split method, the replace method of the class is called returning a valid word without the repeating characters. The array of words created is then converted back to a string.

The last three manipulation operations described rely on classes imported from a package in the same folder called *''replacers.py''*.

```
1 from .replacers import RegexpReplacer
2 from .replacers import RegexpReplacerAge
3 from .replacers import RepeatReplacer
```

This replacer package is based on a package of the same name described in Python Text Processing with NLTK 2.0 Cookbook (Perkins [27]). The first class to consider is the `RegexpReplacer` class. The core of this class is the `replace()` method which takes a word and returns what is considered a more correct version. Before this class can be utilised an array of replacement patterns, or words, must first be defined. As shown in and Appendix A.3 the first three patterns are used to replace *u* with the full word *you*. For example `(r'(\su\s)', ' you ')` searches the text for any occurrence of the letter u with white space (`\s`) on both sides and replaces this with a space followed by the word ''you'' followed by another space. The two edge cases, where u is the first word in a question or the last word of the question are also handled. The `$` symbol is used to identify the end of the text of the question and the `^` symbol to identify the start of the question.

The two most important concepts encapsulated in the class are performed by the `re.compile` method and the `re.subn` method. The compile method compiles the regular expression patterns, for example `\su\s`, into a regular expression object. The subn method takes three parameters, the compiled regular expression object, the replacement text value and the string on which to perform the regular expression substitution. The resulting string and the number of substitutions made are returned.

The `RegexpReplacerAge` class is identical to the `RegexpReplacer` class except that instead of taking text based patterns it is looking to replace digits that are representative of peoples ages.

Just like the `RegexpReplacer` class, the purpose of the `RepeatReplacer` class, lines 115-130, is to return a more correct version of the word passed for processing. However, instead of replacing abbreviated words with the correct fully expanded word, the goal here is to remove unnecessary repeating characters. This class makes very good use of backreferences to remove these repeating characters. The `repeat_regexp` regular expression will match zero or more characters at the start of the string and these are referenced as `\1`. Next a single character is matched but it must be immediately followed by another instance of itself, referenced as `\2`. Finally zero or more characters at the end of the string and these are referenced as `\3`. Then `repl` simply references `\1 \2 \3` recombined without the single repeated character which is not included as it was not within the grouping parentheses of the second reference. The `replace()` method is then called recursively to remove multiple repeated characters but makes use of a WordNet lookup as a stopping mechanism when a valid word is identified.

4.5.2 Write Data in NLTK Corpus Format

The Python script described in the previous section, shown in Appendix A.2, was executed separately for questions classified as bullying and for questions classified as not bullying, the SQL on lines 18 and 19 was changed to select each class separately. Processing each class separately in this manner allowed each question to be written individually to a file in a folder named in accordance with the class type, for example bullying questions were written into a folder named `bullying` and not bully question were written to a folder called `not_bullying`. The parent folder for each was called `01` as the dataset produced as a result of this processing will be called `cyberbully_01` in later sections, the original raw data was written to disk in a similar manner without processing to a folder named `00`. The name of the folder that the files were written to, lines 70 and 71, was changed to reflect the class that was being processed. Each question was written to a separate file that was anonymously named in the range `00001` to `99999` so that the question could not be traced back to the original question on the Ask.fm website.

Once this `01` dataset had been written to disk five further datasets were generated from it. Data `02` and `03` were the bi-grams and tri-grams datasets derived from the original `01` dataset. Next, NLTK stop words were stripped from the `01` dataset producing the `11` dataset from which bi-grams and tri-grams were again generated producing datasets `12` and `13` respectively. The final structure of the NLTK corpora developed here can be seen in Figure 5.1. All dataset manipulation was achieved using Python.

4.5.3 Analysis of Cleaning Steps

It is worth examining some of the changes made to the text of the questions by the processing described here, and explain the relevance of each step.

The conversion of all characters to lower case is a standard step to perform in text mining. The goal is to reduce the number of tokens that will later be presented to the model. For example, whilst the reader understands that *''Why''*, *''WHY''* and *''why''* are all different representations of the same word. If presented to the text mining model in this manner, three distinct tokens will be identified. Interpreting words with different case representations as separate tokens increases the size of the final word vector and also may have the unwanted side effect of either incorrectly diluting or multiplying the impact of a word in the generation of the classification model. Though capitalisation of words can be used to infer shouting or displeasure, for the purposes of this study it is acceptable that all words are converted to lower case.

Removal of non ASCII characters was performed to clean up the text of the question and remove extra characters added to ''beautify'' the question. It was also to ensure that all text was represented in its simplest form. This processing had the beneficial side effect, like the conversion to lover case, of standardising tokens. Examples of both the removal of unwanted characters and the standardising of text is shown in Figure 4.17. The first example shows how the non ASCII characters that spelt out the word `beautiful` were converted into normal ASCII characters and the decorative symbols after the word were removed. The second example converted the phrase `Love you Forever` to ASCII as well as removing the heart symbol.

```
b e a u t i f u l ▽ς•ﻌ•ʔ
beautiful
hey! 𝓛𝓸𝓿𝓮 𝔂𝓸𝓾 𝓕𝓸𝓻𝓮𝓿𝓮𝓻 ♥ how are you? #muchlove (send me a gift? thanks)
hey! Love you Forever  how are you? #muchlove (send me a gift? thanks)
```

FIGURE 4.17: Samples of where non ASCII characters were either removed or replaced with their ASCII equivalent

It was decided that URL strings where so unique that they would not add any significant benefit if included in the model. They were stripped out wherever they were found. The removal of punctuation characters served at least two purposes. The first thing that stripping punctuation characters achieved was to remove all emoticons from the text of the questions. Although it could be argued that the emoticons are representative of the tone or sentiment of the question, it was observed that their use was inconsistent and there were multiple different representations of several emoticons. This all meant that the intended meaning of some emoticons were not obviously clear. In the first two examples shown in Figure 4.18 it can be seen how removing the punctuation greatly

cleans up the presentation of the text. The third examples shows the removal of two emoticons.

```
so your over that girl that's dating what's her face... (truth question)
so your over that girl thats dating whats her face truth question
do you like melanie?
do you like melanie
kush ;) :*
kush
```

FIGURE 4.18: Samples of where punctuation has been removed showing where emoticons are removed

Next it was decided to remove all numeric characters. However, it was noticed that there were a significant number of references to peoples ages and that most of these related to teenagers or early twenties. This information was considered useful and worth saving so before all other digits were removed the numbers from *''11''* to *''25''* inclusive were replaced with their text equivalent. After these numbers were substituted, all other digits were removed. A sample of an age being replaced with the text value is shown in 4.19 as well as the removal of all other digits that are deemed not to represent ages.

```
text me 9282308338
text me
what did u do u in the last 10 mins
what did u do u in the last  mins
bed at 930 hahawhay are you 12 oh wait yeah you actually are 12
bed at  hahawhay are you twelve oh wait yeah you actually are twelve
```

FIGURE 4.19: Samples of where certain numbers have be substituted with text values and all other digits are removed

As discussed earlier, the nature of social media means that text is often unstructured. Words are sometimes accentuated with the addition of extra repeated characters or abbreviated to their simplest forms. For example, *''looove yoooou''* or *''love u''* both mean the same thing but if left in this format extra unnecessary tokens will be added to the word vector diluting the effectiveness of the model. Alternatively, shorter tokens will be stripped because they do not meet a minimum token length criterion. In the first case the repeated characters are removed returning a valid word. In the second, a number of abbreviated words and their full text representations were defined. Sample of both these word manipulations can be seen in Figure 4.20. The first two examples show the successful removal of repeated characters whilst the third example shows both repeated characters removed and the replacement of an abbreviated word with the correct word.

```
im really nervous to kik you coz i liiiiiike you heaps
im really nervous to kik you coz i like you heaps
selfieee
selfie
wow u like alot of boyyyysss
wow u like alot of boys
wow you like alot of boys
```

FIGURE 4.20: Samples of where repeated characters have been removed and abbreviated words are replaced with correct word

4.5.4 Data Exploration Revisited

Now that *''cleansed''* datasets are available for the not bullying and bullying classes it is worthwhile taking a brief step backwards and analyse these dataset using the same techniques described in Section 4.4. For brevity only the not bullying and bullying datasets are considered and only certain measures.

The top ten distinct token frequencies of bi-grams and tri-grams from the bullying dataset is shown in Table 4.8. Looking first at bi-grams with stop words, eight out of the top ten tokens have not changed but what is noticeable is that overall the frequencies have increased. Looking at the bullying bi-grams with stop words removed only six of the original top ten are still present. Once again the frequencies have increased albeit more modestly. What is significant is that the previous number one most frequently occurring bi-gram, `right_now`, is now no longer included. Initially it was thought that this was an error, however, upon a more detailed examination it was discovered that the original two tokens were `right` and `now?`. The question mark character prevented `now` from being identified as a stop word. It would appear that the cleaning process has impacted on bullying bi-grams. Whether this impact is positive or negative remains to be seen.

When the bullying tri-grams that included stop words were re-examined, eight of the top ten remained the same with high count frequencies evident. Surprisingly not only were nine of the top ten tri-grams where stop words had been removed the same as before the dataset was cleaned, but these nine tokens also had the same frequencies. The only token missing from the list was `wearing_right_now` and as previously highlighted with the bi-grams this token is no longer occurring in the list because the genuine stop word `now` is no longer available for consideration. It should be noted that further examination showed that this was not the only such occurrence of where a stop word had not been removed from the dataset because it was attached to a punctuation mark. It was again noticed that where stop words were not removed, personal pronouns were again a very common occurrence in the top ten list.

TABLE 4.8: Table showing distinct counts of bi-gram and tri-gram tokens for the cleaned bullying dataset and also with and without stop words

Bullying Bi-Grams			
With Stopwords		**No Stopwords**	
are_you	155	post_pic	25
do_you	106	dont_know	20
you_have	65	wearing_right	20
you_are	54	dont_like	13
of_you	48	cut_cut	9
you_and	47	really_pretty	9
what_are	44	post_selfie	8
if_you	42	post_picture	8
how_old	37	dont_even	8
would_you	36	bra_size	7

Bullying Tri-Grams			
With Stopwords		**No Stopwords**	
what_are_you	36	cut_cut_cut	7
old_are_you	32	last_person_kissed	5
how_old_are	32	whoever_likes_thinks	4
are_you_wearing	22	see_window_post	4
you_wearing_right	20	seem_really_nice	4
wearing_right_now	20	window_post_pic	4
do_you_have	17	slept_ex_bf	3
are_you_single	17	shannon_slept_ex	3
of_you_and	16	likes_thinks_you're	3
are_you_doing	14	ex_bf_max	3

Table 4.9 shows the number of unique tokens in each of the cleaned datasets as well as the total count of tokens, and their average frequencies. The numbers in brackets show the percentage increase or decrease when compared with the original data. It is clear to see that nearly all counts are down and that, in general, the average frequencies have either increased or stayed the same. The general decrease in the number of tokens can be explained by the cleaning process where it was seen that many numerical tokens were removed, and multiple variations of the same token were standardised. The slight increase in the number of bullying n-grams with stop words is down to common abbreviations being standardised.

Finally, in Figure 4.21, the normalised word frequencies for the cleaned datasets are shown. When compared to the earlier graph in Figure 4.16 differences are not immediately obvious. On closer examination of the not bullying graph, labelled (1), it is clear that the distribution of the bi-grams is more uniformly spread out and that the tri-gram distribution is closer to the other datasets. The most obvious difference in the bullying

TABLE 4.9: Table showing the number of distinct tokens in each dataset as well as the total token counts and average frequencies

Token description	Unique Tokens	Total Count	Average Frequency
Not Bullying			
Uni-Grams	6579 (-37%)	61293 (-13%)	9.1 (+39%)
Bi-Grams	29149 (-13%)	53436 (-13%)	1.8 (same)
Tri-Grams	38701 (-5%)	46120 (-13%)	1.2 (-8%)
Uni-Grams No Stop Words	6462 (-38%)	33156 (-19%)	5.1 (+31%)
Bi-Grams No Stop Words	21172 (-16%)	25394 (-20%)	1.2 (-8%)
Tri-Grams No Stop Words	18311 (-14%)	18746 (-21%)	1.0 (-9%)
Bullying			
Uni-Grams	2568 (-29%)	15323 (+1%)	6.0 (+43%)
Bi-Grams	9039 (-9%)	13679 (+1%)	1.5 (+7%)
Tri-Grams	10809 (+3%)	12112 (+1%)	1.1 (same)
Uni-grams No Stop Words	2459 (-30%)	8312 (-8%)	3.1 (+31%)
Bi-Grams No Stop Words	5993 (-12%)	6676 (-10%)	1.1 (same)
Tri-Grams No Stop Words	5190 (-11%)	5282 (-11%)	1.0 (same)

graph, (2), is the affect on the tri-gram distribution following the removal of the most frequently occurring token.

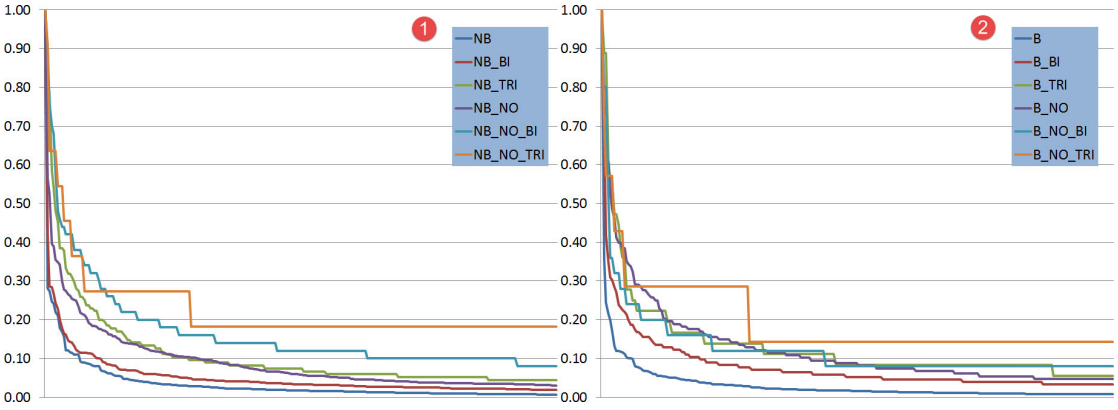

FIGURE 4.21: Graphs showing the normalised word distribution of the different token configurations examined

4.5.5 Data Visualisation Revisited

Word clouds were also generated for bi-grams and tri-grams and the results can be seen in Figure 4.22 and Figure 4.23. The many Eyes website was used once again with the same settings as before.

As was seen with uni-grams the word clouds are dominated by stop words, especially the personal pronoun you. Looking at bi-grams first in Figure 4.22, the most frequently

FIGURE 4.22: Word clouds showing bullying bi-grams (a), bullying bi-grams with stop words removed (b), not bullying bi-grams (c) and not bullying bi-grams with stop words removed (d)

occurring bullying and not bullying tokens, *"are_you"* and *"do_you"*, are easily seen in section A and section C. Even after stop word removal, section B and section D, the bi-grams with the highest frequency occurrences are still clearly visible. Examining the tri-gram word clouds in Figure 4.23 it is obvious that the tokens are now more evenly distributed. Even though the frequency of these bullying and not bullying tokens are more evenly distributed, when compared to uni-grams and bi-gram clouds, there are still a number prominent tokens. It is hoped that these tokens would be good predictors for both class types.

4.6 Summary and Conclusion

This chapter focused on the data. In the next chapter this data will be mined to develop a model to classify a question as either bullying or not bullying.

FIGURE 4.23: Word clouds showing bullying tri-grams (a), bullying tri-grams with stop words removed (b), not bullying tri-grams (c) and not bullying tri-grams with stop words removed (d)

In Section 4.1 an overview of how the data was obtained from the Ask.fm site was given. A sample block of data, the question, answer and other meta data was shown and explained. The criteria used to determine which data was to be scraped, and the method used to extract it, was given. The data was then obtained over a twenty-four hour period in an attempt to make sure that the users from which the data was scraped covered as many geographical areas as possible and were not, for example, all from a single time zone. Upon completion of this task 110,000 data examples had been obtained.

Section 4.2 described how the raw data extracted from the Ask.fm website was transformed into text more suitable for text mining. The original HTML format of the data was given and then the Python used to transform this into text was detailed. The Python `glob` and `os` packages were used to access the HTML files and Beautiful Soup, a Python library for parsing HTML documents, was used to parse these files so that the individual data elements described could be extracted. In total seven attributes were extracted per record. The data was loaded into a MySQL database. To complete the preprocessing the data was divided into three separate groups. 10% for classification and model training and test, 80% to simulate a stream of data arriving at the ask.fm website and a final block of 10% for validation purposes.

Section 4.3 gave an overview of the process used in the classification of each question as being either bullying or not bullying and also included detailed descriptions of the various types and categories of cyberbullying to be identified. Samples of some of the questions classified as bullying were also shown. Just under 11,000 sample questions were classified.

In Section 4.4 a more detailed description of the data was given. It was noted that the structure of the data is relatively simple. Next some data quality issues were highlighted including non ASCII characters, slang words, abbreviations, repeated characters and emoticons. The section also included a detailed exploration of the dataset and also some word cloud visualisations and word, or token, frequency analysis.

In the final section, Section 4.5, the data underwent a series of transformations in order to clean the data in preparation for modelling. As before Python was used to process the text of the dataset and in all eight cleansing steps were performed. The data was then written out to files in NLTK corpora format.

Chapter 5

Data Modelling

The focus of this chapter is to describe the process to develop the best model for predicting whether a question from the Ask.fm website could be considered as either bullying or not bullying. The modelling approach is given, and it also highlights any alterations that were required to be made to this initial plan. However, before going into the detail of the modelling approach there is a brief refresher of the data available and its structure.

At the completion of Chapter 4, where the data extracted from the Ask.fm website was explored and processed, a number of datasets were now available for modelling. In total seven datasets were generated:

1. The original raw question data as extracted from Ask.fm [dataset 00]
2. The cleaned version of the original dataset [dataset 01] (considered the primary dataset)
3. Bi-grams created from the cleaned dataset [dataset 02]
4. Tri-grams created from the cleaned dataset [dataset 03]
5. The cleaned dataset with all NLKT stop words removed [dataset 11]
6. Bi-grams created from the cleaned dataset with all NLKT stop words removed [dataset 12]
7. Tri-grams created from the cleaned dataset with all NLKT stop words removed [dataset 13]

All seven datasets were written to the NLTK data folder as separate corpora.

A description of the initial first modelling attempts using the Natural Language Toolkit (NLTK) are given in Section 5.1. Here a Naive Bayes classifier is used to gain insight into the potential predictive performance of a model when classifying the text. As well as examining the performance of each of the datasets generated in Chapter 4, this section also analyses whether the N-gram generation and stop word removal functionality natively available in the NLTK performs any better or worse.

Next, in Section 5.2, ways to address the minority positive class and majority negative class imbalance are explored. Techniques utilised include under sampling of the majority class, over sampling of the minority class and a hybrid approach that uses both under and over sampling. Also explored is whether only selecting a percentage of the most frequently occurring tokens in each class makes a difference to the performance of any models generated. Once again a basic NLTK Naive Bayes model is applied.

Sections 5.3 and 5.4 then repeat the modelling processes undertaken in Sections 5.1 and 5.2 but using the Scikit-Learn Python libraries. In these sections a Linear Support Vector classifier is used in addition to a Naive Bayes. Section 5.5 investigates the possible use of a cost sensitive approach to modelling before the best models developed are identified in Section 5.6. Finally in Section 5.7 the three best models identified are further tested using previously unseen data. Using the top three models a hold back dataset is classified. The models then evolve using a third dataset and the hold back dataset is classified and the results analysed.

5.1 Natural Language Toolkit - Initial Modelling

The first model was developed using the primary dataset, data that has been cleansed but stop words have not been removed, and the Python NLTK. A Naive Bayes learner using a feature based bag of words approach was chosen. The data was stored in the NLTK corpus format where each question was individually written to a file in a folder that represented whether it was classified as bullying or not bullying. For reference, this structure is shown in Figure 5.1

5.1.1 Model Development

The development of a NLTK classifier model is a relatively straight forward task. The first step is to create a categorised plain text corpus reader. The reader takes three parameters. The first parameter is the root folder for the corpus, the second is a regular expression used to identify the files in the corpus, the third is a regular expression used

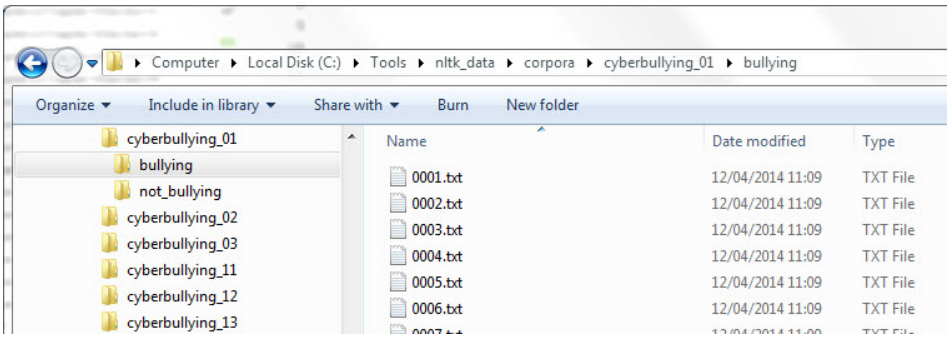

FIGURE 5.1: The structure of the primary dataset is shown with the top level folder containing a folder for the bullying questions and the not bullying questions. A file per question is created.

to determine the category names which are derived from the folder names within the corpus. In the code snippet shown in Listing 5.1, lines 1 to 5, the corpus to be read is `cyberbullying_01` and all files with a `.txt` extension are included. Now that the reader is initialised the next step is to identify the file ids of each category type.

Listing 5.1: Creating the categorised plain text corpus reader

```
1 # Set the working folder to the nltk_data corpora folder
2 os.chdir ('c:/Tools/nltk_data/corpora')
3 reader = CategorizedPlaintextCorpusReader('./cyberbullying_01',
4                                           r'.*\.txt',
5                                           cat_pattern=r'(\w+)/*')
```

Identifying the file ids of each category is just a matter of calling the `fileids` method passing the category name of the required file ids. The code snippet in Listing 5.2 returns two file lists the first, `pos_ids` is a list of all the files, including each files relative path, in the positive or bullying category while the `neg_ids` is the list of all files in the negative or not bullying category.

Listing 5.2: Identifying the positive and negative file ids

```
1 # The questions containing bullying are the positive tests
2 pos_ids = reader.fileids('bullying')
3
4 # The questions not containing bullying are the negative tests
5 neg_ids = reader.fileids('not_bullying')
```

Figure 5.2 is a screen capture showing the first nine files in the positive, bullying, category and that there are 1,644 files in this category. There are 7,899 files in the negative or not bullying category.

The NLTK uses the concept of a feature, in this case a word or token, and the Naive Bayes Classifier in the NLTK operates solely on whether a feature is present in a sample or not. The code in Listing 5.3 shows how the positive and negative feature dictionaries

```
>>> pos_ids[:9]
['bullying/0001.txt', 'bullying/0002.txt', 'bullying/0003.txt', 'bullying/0004.txt',
 'bullying/0005.txt', 'bullying/0006.txt', 'bullying/0007.txt', 'bullying/0008.txt',
 'bullying/0009.txt']
>>> len(pos_ids)
1644
```

FIGURE 5.2: Screen capture showing the first nine files in the positive, bullying, category and also that there are 1,644 files in this category

are generated. The first three positive features are shown in Figure 5.3 with the second sample highlighted to help distinguish each sample. Examining this second sample further, it can be seen that it is a Python dictionary containing fourteen words where each value has been given a value of true to signify their presence in the sample. Each sample is also categorised as bullying. It should be noted that all sentence word order has been lost as would be expected when using a bag of words approach. A similar process is performed to created the negative features.

Listing 5.3: Generation of the positive and negative feature dictionaries

```
1  def word_feats(words):
2      return dict([(word, True) for word in words])
3
4  pos_feat = [(word_feats(reader.words(fileids=[f])),
5               'bullying')
6              for f in pos_ids]
7
8  neg_feat = [(word_feats(reader.words(fileids=[f])),
9               'not_bullying')
10             for f in neg_ids]
```

```
>>> pos_feat [:3]
[({'about': True, 'weed': True, 'thought': True, 'quiting': True, 'have': True, 'you
': True, 'ever': True}, 'bullying'), ({'be': True, 'life': True, 'love': True, 'gona
': True, 'that': True, 'always': True, 'promises': True, 'are': True, 'of': True, 't
he': True, 'my': True, 'your': True, 'best': True, 'remember': True}, 'bullying'), (
{'age': True, 'otd': True, 'selfie': True}, 'bullying')]
```

FIGURE 5.3: Screen capture showing the first three positive features

Next, the Naive Bayes learner is trained and it's performance calculated. Unfortunately, NLTK does not natively support cross validation. This meant that ten-fold cross validation had to be manually implemented in code. Full details of the implementation of the cross validation can be seen in Appendix A.4. Using a simple cross validation solution, the positive and negative samples were divided into ten parts. Nine of the positive and negative parts were joined together to create the training dataset and the remaining positive and negative parts joined to create the test dataset. A Naive Bayes classifier was then created using the training dataset as shown in Listing 5.4. Also shown, lines 4 - 8, is the generation of a reference set and test set to be used to calculate the performance of the classifier. To achieve this the actual classification of each of the test samples is loaded into the reference set and the value for the sample returned from

the classifier is loaded into the test set. Then, using the NLTK metrics package, the performance of the classifier is calculated. For example, line 12 shows the calculation of the precision for the positive bullying samples. Finally line 18 will display the top 5 most informative features of the classifier.

Listing 5.4: Generation of the positive and negative feature dictionaries

```
1  # Create the classifier using the train dataset
2  classifier = NaiveBayesClassifier.train(train)
3
4  #Populate the metrics sets with values
5  for i, (feats, label) in enumerate(test):
6      refsets[label].add(i)
7      observed = classifier.classify(feats)
8      testsets[observed].add(i)
9
10 # Calculate precision. recall, f-measure
11 #   and accuracy
12 pos_pre  += nltk.metrics.precision(refsets['bullying'],
13                                    testsets['bullying'])
14 ...
15 # Show the top 5 most informative features
16 classifier.show_most_informative_features(5)
```

Once the cross validation testing has been completed then the average performance measures are calculated. A simplified visual representation of the process used to develop the first model using the NLTK is shown in Figure 5.4

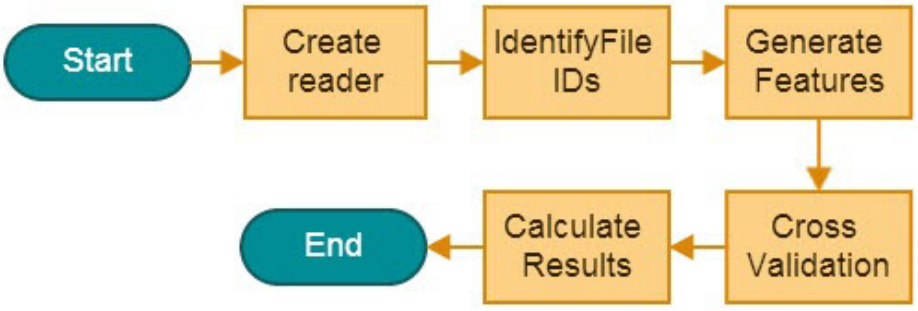

FIGURE 5.4: Simplified visual representation of the process used to develop the first model using the NLTK

5.1.2 Model Execution and Performance

The NLTK Naive Bayes classifier described is then run on the primary dataset. The performance results and top five most informative features of the tenth cross validation run are shown in the screen capture in Figure 5.5

Considering the top five most informative features, it is clear that these tokens, real words in this example, could easily be included in a question that was bullying in nature.

```
Most Informative Features
                  selfie = True           bullyi : not_bu =    113.6 : 1.0
                     pap = True           bullyi : not_bu =     54.7 : 1.0
                    dick = True           bullyi : not_bu =     40.0 : 1.0
                    suck = True           bullyi : not_bu =     33.6 : 1.0
                     bra = True           bullyi : not_bu =     33.6 : 1.0
pos precision:  0.385960809818
pos recall:     0.753048780488
pos F-measure:  0.509514973012
neg precision:  0.935880644696
neg recall:     0.748669201521
neg F-measure:  0.831482899433
model accuracy: 0.749422875131
```

FIGURE 5.5: Screen capture of the top five most informative features from the tenth cross validation run and also the overall performance measurements for the NLTK classifier

The overall average accuracy of the model, just under 75%, could suggest that this was a very successful first attempt at developing a model to predict bullying. However, looking closely at the precision and the recall for the different classes this is clearly not the case. The negative class precision, 93.6%, shows that nearly every sample classified as not bullying was an actual not bullying sample. Add to this a negative class recall value of 74.8% and the model correctly identified nearly three out of every four not bullying questions. Examining the positive class next a recall value is seen which, at 75.3%, shows that three-quarters of all bullying questions were correctly identified. However, a precision value of just 38.6% shows that only two out of every five samples classified as bullying were bullying questions implying that three out of every five samples classified as bullying were, in fact, not bullying questions.

The model developed was then run on all seven datasets. The performance results for each dataset are shown in Figure 5.6.

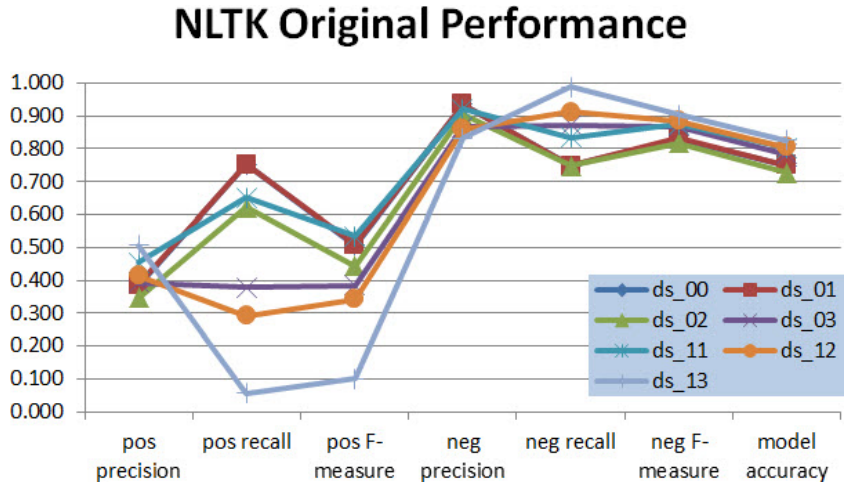

FIGURE 5.6: Graph showing the performance of the initial NLTK Naive Bayes model across the seven manually crafted datasets

5.1.3 Model Analysis

The first observation from an analysis of the data is that the precision of the model in predicting the positive class, the bullying questions, is poor when compared to the values obtained for the negative class with an average value of 41% compared to an average value of 89%. Although the average performance for recall for the positive class is 50% this figure is not representative because of the value achieved for `ds_13`, tri-grams with stop words removed, which is less than 10%.

Taking all performance measures into account, it could be said that the best performance achieved by this model was with `ds_11`, uni-grams with stop words removed. However, with a precision value for the positive class of 45.3% and a recall value of 65.2% the results are far from satisfactory. This was just an initial modelling attempt and there are many changes that could be made to this model to improve its performance. Addressing class imbalance, only selecting the features that have the highest impact and by changing from the feature based approach offered by NLTK to the more statistical TF-IDF implementation offered by the Scikit-Learn package are some of the options to consider. It should also be noted that the Scikit-Learn package comes with its own cross validation implementation which will lead to a simplification of the code used.

5.1.4 Further Exploration

Before leaving this first simple model it was decided to determine if there was any advantage gained, or different results achieved, by generating the bi-grams and tri-grams up front. To test this, a simple change was needed to the `word_feats` function to return bi-grams or tri-grams. Returning bi-grams is shown in Listing 5.5. Four new models were created using the bi-grams and tri-grams datasets generated from `dataset_01` and `dataset_11`.

Listing 5.5: Generation of the positive and negative feature dictionaries

```
1 from nltk import bigrams
2 from nltk import trigrams
3 ...
4 def word_feats(words):
5     return dict([(word, True) for word in bigrams(words)])
```

The results generated by these new models were then compared to the original models where the bi-gram and tri-gram tokens were manually created beforehand. The different models were compared in three areas. The first, and easiest to compare, was average execution time. The average run time for each model was determined and it was found that generating the bi-grams or tri-grams during model execution did add to the overall

execution time as can be seen in Table 5.1. With a maximum time of 0.1 seconds to generate all tri-grams in advance, the longest to create, it is clear that there is a time advantage to creating all n-grams upfront. In Table 5.1 the first column is the model execution where the n-grams were generated before hand. Thee second column is the time for n-grams created during model execution. Column three is the time difference in seconds and the fourth column is the percentage increase in time.

Table 5.1: Table showing average model execution time increase when n-grams are generated as part of the modelling process

	orig	**new**	**diff**	**percent**
bi-gram	19.399	20.549	1.150	5.9%
tri-gram	23.116	24.647	1.531	6.6%
bi-gram ns	14.357	15.685	1.328	9.3%
tri-gram ns	11.477	14.225	2.748	23.9%

The second comparison is the performance of the models. The results of this analysis is shown in Figure 5.7. A cursory look at these charts shows that the generation of the n-grams during the model execution did not improve the overall performance of the model. It may, it fact, have had a slightly negative impact which can be seen in the positive class recall and f-measure values. In all charts the top line in the legend represents the performance of the original model. For example, in the chart labelled ''Bi-grams no stop words'', the `01_12` series, represented by the blue line shows, the performance values of the original model, `01`, using bi-grams generated from the dataset where stop words have already been removed, `_12`. The orange line, `02_12`, is from the second model where bi-grams were generated during model execution.

The final analysis performed was a comparison of the most informative features returned during each model execution. Ten-fold cross validation was used and the top five most informative features were returned. The results of the number of most informative features comparison is summarised in Table 5.2.

Whilst initially it appeared that there were a number of significant differences, after an investigation it was found that sometimes the most informative features with the same ratio values were sorted differently between the two model types. The other cause of the differences was the way the datasets were processed when the n-grams were generated during execution. When the tri-grams were generated up front, samples that had only two words never got included in the datasets submitted to the modelling process. This meant that there were only 1,049 positive and 4,915 negative samples for tri-grams with no stop words. However, there are 1,637 and 7,800 samples respectively in the uni-grams with no stop words dataset. As the ten-fold cross validation datasets were created before

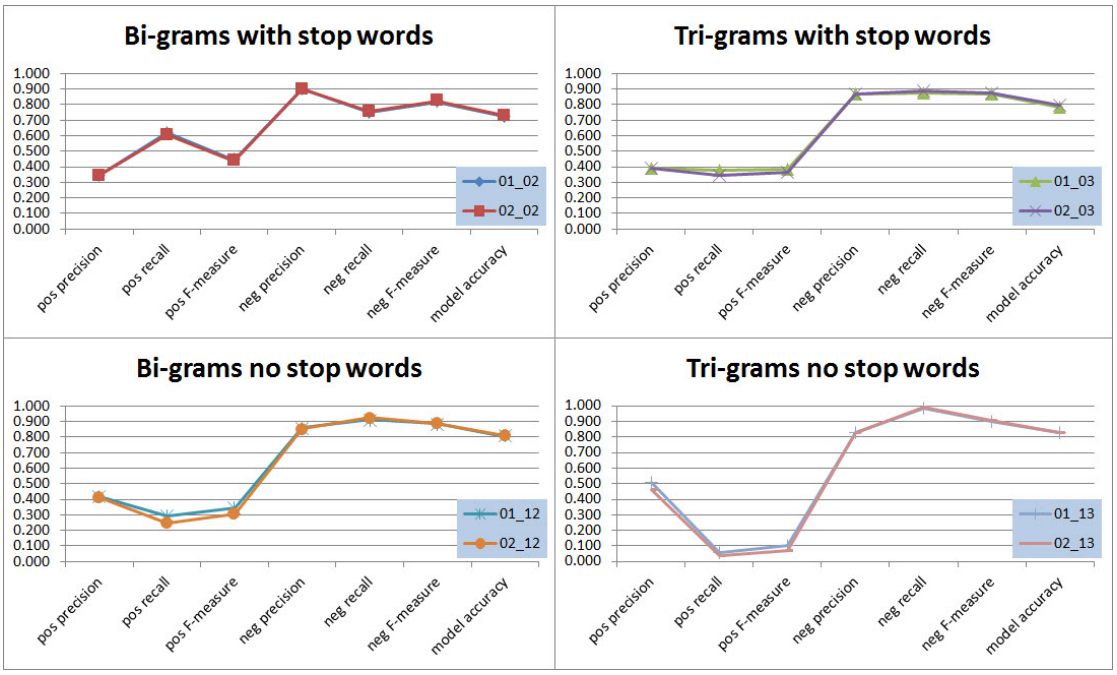

FIGURE 5.7: Graphs showing a comparison or model performance when n-grams are pre-generated or generated during model execution

the n-grams were generated, this lead to a different split of the samples in the two models being compared.

TABLE 5.2: Table showing the number of the most informative features that were different when n-grams were generated during model execution using the NLTK compared to being generated beforehand

Model Type	Number Differences
Bi-grams	5
Tri-grams	2
Bi-grams no stop words	3
Tri-grams no stop words	15

Based on the analysis given it was decided that for the simple NLTK model developed here it did not matter that the n-gram datasets were generated beforehand. It should be noted, however, that the models generated used all the classified data available and they were not tested on unseen data. This is not considered an issue as the performance results achieved here will be considered as a baseline against which all subsequent models could be compared.

5.2 Natural Language Toolkit - Class Imbalance

It was observed that there was a class imbalance between the positive and negative samples. In this section, it will be investigated if over sampling the minority positive class or under sample the majority negative class affects the performance of the model. A compromise solution where the positive class is partially over sampled and the negative class is partially under sampled is also examined. More advanced methods for handling class imbalance such as cost based learners or the Synthetic Minority Over-sampling Technique (SMOTE) are not suitable for consideration at this time using the NLTK datasets. Although not strictly a class imbalance solution, sampling using the most frequently observed features is also explored.

5.2.1 Majority Class Under Sampling

Under sampling of the majority negative class was first to be explored. In all NLTK datasets the ratio of negative to positive classes is between 4:1 and 5:1. To determine how different negative class to positive class ratios affect the model performance, under sampling of the majority class at ratios of 3:1, 2:1 and 1:1 to the minority class will be examined.

To implement the desired ratios, the Python script described in Section 5.1 requires one minor change as shown in Listing 5.6. The file ids of the not bullying, or negative class, are read into the `all_neg_ids` list. The order of the file ids are then shuffled and the required ratio is achieved by only taking multiples of the number of positive ids as required. In line 6 the ratio of negative to positive samples is 3:1.

Listing 5.6: Adjust the postive to negative class ratio

```
1 # The questions not containing bullying are the negative tests
2 all_neg_ids = reader.fileids('not_bullying')
3
4 # Shuffle the file ids of the negative file
5 random.shuffle(all_neg_ids)
6 neg_ids = all_neg_ids[:len(pos_ids)*3]
```

The NLTK Naive Bayes model was generated five times for each dataset and average performance values calculated. Because the negative samples are randomly chosen, it was important to run multiple executions for each dataset to allow for any possible variance or imbalance in the samples selected. It was found that five executions were enough to show that all results were similar and consistent. The performance results achieved for each dataset at ratios of 3:1, 2:1, 1:1, and the original performance measures from Section 5.1, are shown in Figure 5.8.

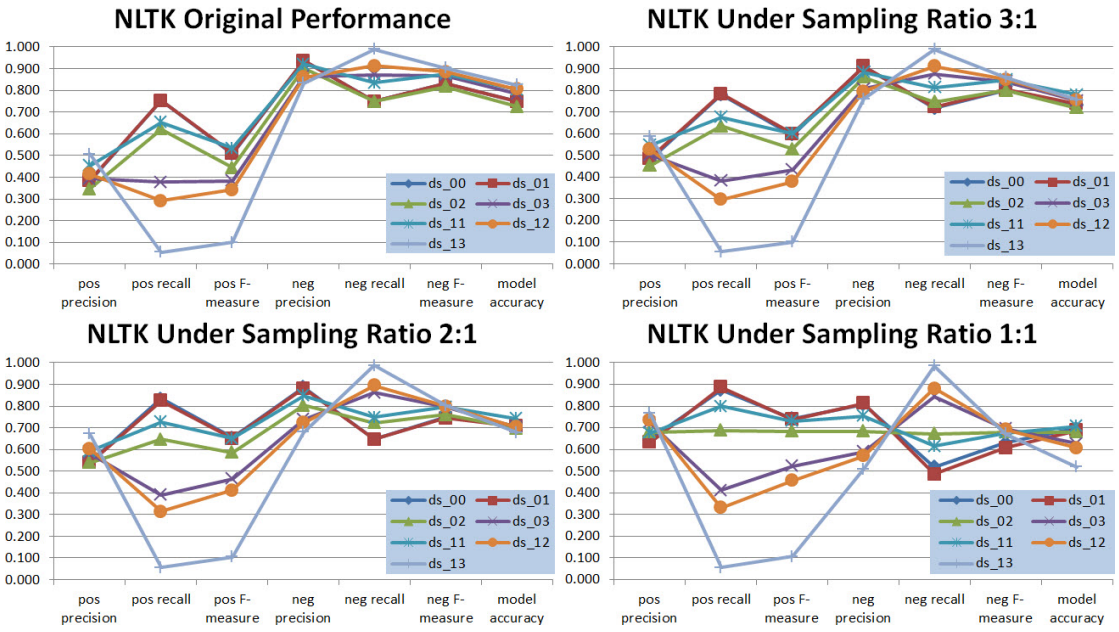

FIGURE 5.8: Graph showing the performance of the NLTK Naive Bayes model for ratios 3:1, 2:1, 1:1 and the original baseline performance measures

It is clear that the performance of the positive class prediction is increasing as the ratio of samples approaches 1:1. The positive recall values for some of the datasets also show modest improvement, however, the values do not show the same improvement for `dataset_03` tri-grams, `dataset_12` bi-grams no stop words and `dataset_13` tri-grams no stop words. This lack of improvement could be attributed to the uniqueness of the bi-grams and tri-grams in the datasets. It was seen in Section 4.4 that the average frequency of n-grams in these datasets were very close to 1. Also of note is that as the ratio of classes approaches 1:1 the negative class precision and recall values are decreasing.

At a ratio of 1:1 the overall performance of `dataset_02`, bi-grams including stop words, could, at this early stage of development, be considered very satisfactory. With performance values in all categories of just under 70% the model is equally accurate predicting both positive and negative classes. If the ability to solely predict the positive class was the main driving force, than both uni-grams models, with a sample ratio of 1:1, offer a better solution but at the cost of over predicting samples as positive. It must be kept in mind that one of the major drawbacks of under sampling in this manner is the risk of discarding samples that may, in fact, be very representative of the general population.

5.2.2 Minority Class Over Sampling

Over sampling of the positive minority class was explored next. Ratios of 3:1, 2:1 and 1:1 were again simulated, but this time, instead of reducing the number of negative samples

in order to achieve these ratios the number of positive samples was increased. In line with all testing to this point the method used to increase the number of positive samples was as simple as possible. The implementation, in Python, is shown in Listing 5.7 where a ratio of 2:1 is created.

Listing 5.7: Adjust the postive to negative class ratio

```
1  # The questions containing bullying are the positive tests
2  all_pos_ids = reader.fileids('bullying')
3  random.shuffle(all_pos_ids)
4
5  # Ratio 2:1 required
6  multi = (len(neg_ids)/2) / len(all_pos_ids)
7  modul = (len(neg_ids)/2) % len(all_pos_ids)
8
9  pos_ids = []
10
11 for n in range(multi):
12     pos_ids = pos_ids + all_pos_ids
13
14 pos_ids = pos_ids + all_pos_ids[:modul]
```

When under sampling the negative class was randomly shuffled. This time the positive class is shuffled and the number of times the positive samples need to be replicated, to achieve the desired ratio, is calculated in lines 6 and 7. Taking `dataset_13`, 1049 positive sample and 4915 negative, as an example, the number of times the full set of negative samples need to be replicated is $((4915/2)/1049) = 2$. Then using the modulo operator the number of additional samples required is calculated $((4915/2)\%1049) = 359$. So to create the test dataset two complete copies of the positive samples are required and additional 359 samples. As before, the model was generated five time to ensure that an average performance was achieved. Figure 5.9 compares the performance results from the models generated using under sampling of the majority class with the models generated using over sampling.

It is clear that over sampling of the minority positive class has significantly improved the performance of all models. Bi-gram and tri-gram tokens, with stop words removed, nearly achieving perfection with 99.8% and 100% positive sample recall and 88.4% and 98.4% negative sample recall respectively. Inversely though, it was also observed that as the ratio of the classes approaches 1:1 that the recall performance of the model predicting the negative class decreased significantly, particularly for all the uni-gram datasets.

Overall though, the results achieved using over sampling of the minority class outperform the models developed using under sampling of the majority class. It must be questioned though whether this gain in performance has been achieved by over fitting the models to the repeated samples? This over fitting to replicated data is a known issue when over sampling is applied.

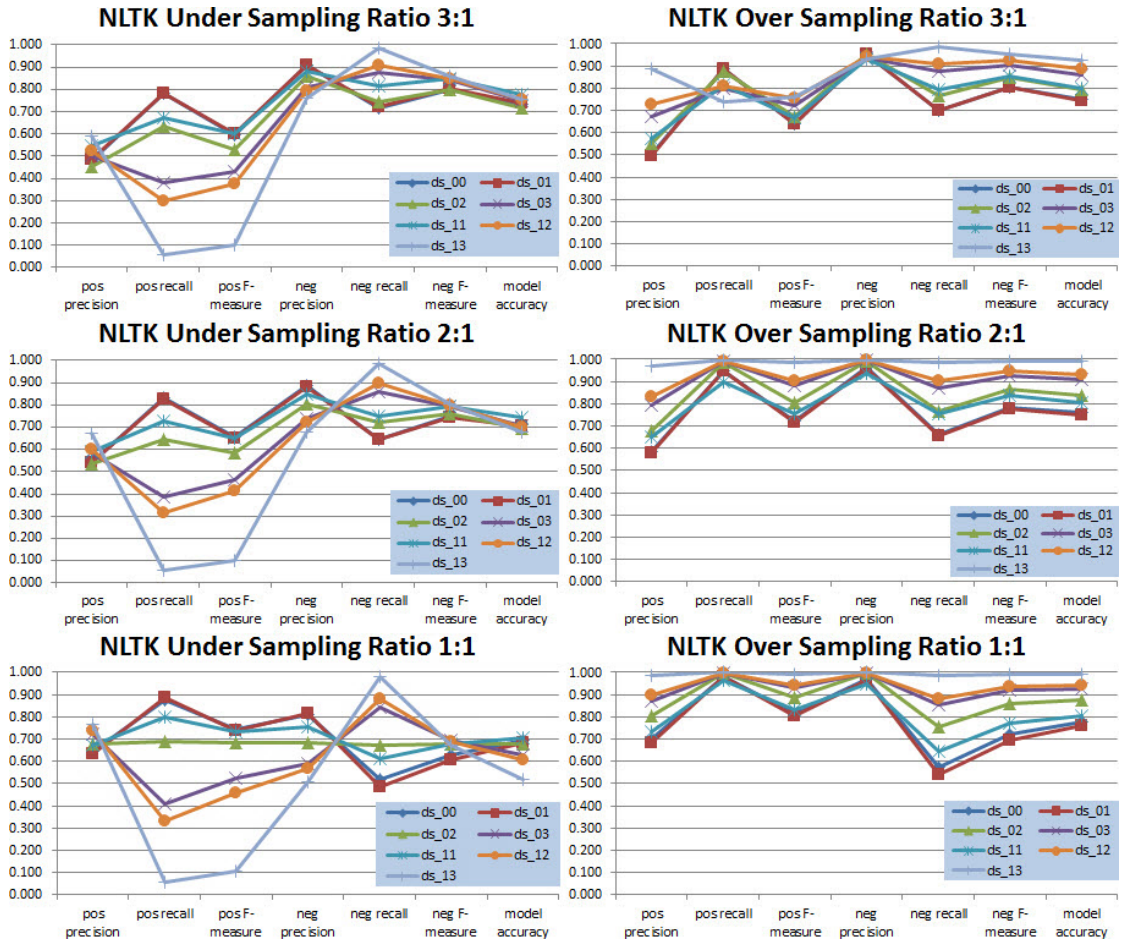

FIGURE 5.9: Graph showing the performance of the NLTK Naive Bayes model for over sampling ratios of 3:1, 2:1, 1:1 and the original baseline performance measures

5.2.3 Hybrid Approach

The third option used to tackle the imbalance of the positive and negative classes was a hybrid approach that uses both under sampling of the majority class and over sampling of the minority class. In this hybrid method, the normal approach is to take the total number of examples and then divide by the number of classes to get the target sample number. For example, in a two class scenario with 800 positive and 400 negative samples, the target number of samples for each class would be 600 if a ratio of 50:50 was the desired ratio. In this section, both classes are over or under sampled to half the total number of examples as just described. However, as the negative class is significantly superior to the negative sample, ratios of 60:40 and 70:30 are also explored. The additional python to achieve the required ratio of samples is shown in Listing 5.8.

Listing 5.8: Hybrid approach using both over and under sampling

```
1 # Calculate the number of pos and neg samples
2 pos_sample = int(float((len(all_neg_ids) + len(all_pos_ids)))
3                 * 0.4)
```

```
4  neg_sample = int(float((len(all_neg_ids) + len(all_pos_ids)))
5                   * 0.6)
6  # Ratio 50:50 required
7  multi_pos = pos_sample / len(all_pos_ids)
8  modul_pos = pos_sample % len(all_pos_ids)
9
10 modul_neg = neg_sample % len(all_neg_ids)
```

In this model both the positive file ids and the negative file ids are shuffled, not shown, and Listing 5.8 lines 2 to 4 shows how the total number of positive and negative samples are calculated, in this case to a ratio of 60:40. The same sampling technique using `modul` and `multi`, as used in the over sampling model, is used to generate the sample datasets. As the negative class will never be over sampled only the `modul` value needs to be calculated. Figure 5.10 shows the result of the model execution using the hybrid sampling ratios.

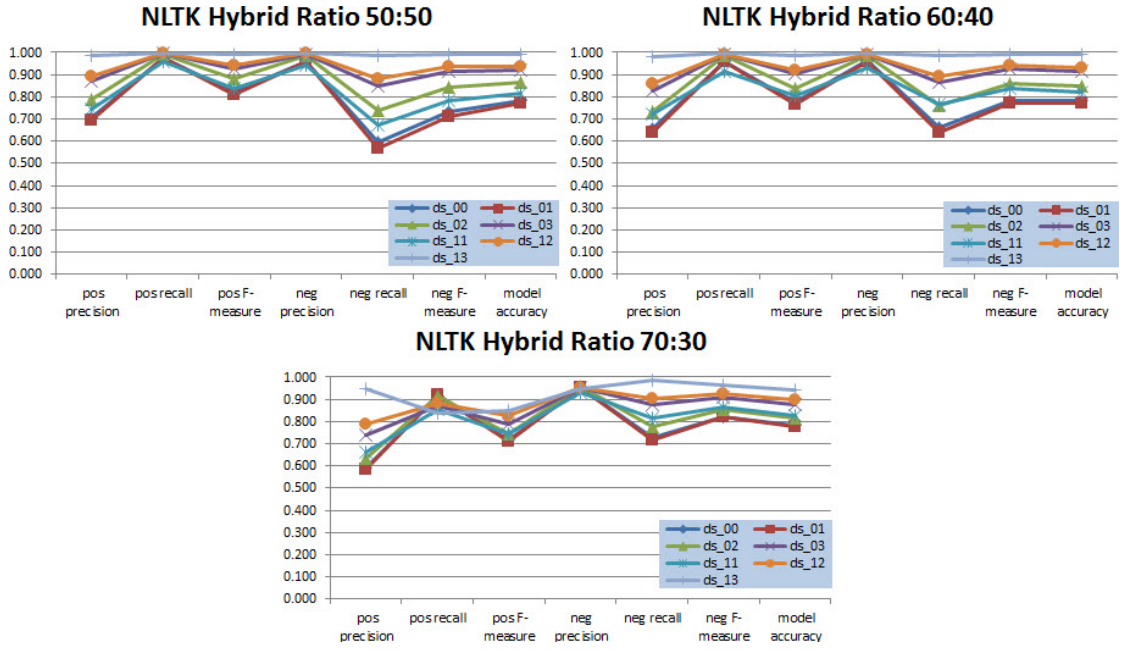

FIGURE 5.10: Graph showing the performance of the NLTK Naive Bayes model for hybrid sampling ratios of 50:50, 60:40 and 70:30

Comparing Figures 5.9 and 5.10 it is clear that the results achieved by over sampling to ratios of 1:1 and 2:1 are very similar to the results achieved by the 50:50 and 60:40 hybrid approach. This fact is highlighted by Table 5.3. The top part of this table gives the results for the over sampling model where a ratio of 1:1 was used. The middle part of the table gives the performance results for the model where hybrid sampling with a ratio of 50:50 was used. The bottom part of the table gives the percentage change between the oversampling model and the hybrid sampling model. Excluding run time measurement, the majority of all performance measures between the two models are within 1% of each other. Considering all performance measures, again excluding run

time, then the total average performance difference between all the models of each type is just 0.18%. Although the difference between the 2:1 over sampling model and the 60:40 hybrid model is slightly more obvious again the total average difference between the two models is just 1.78%.

The final thing to consider is the run time. On average, the hybrid model is approximately 35% faster. This is a significant time saving considering that the performance of the models are so similar. This variance in run time can be explained by the difference in the dataset sizes used. The over sampling dataset had, on average, just over 14,000 samples whereas the hybrid sampling dataset had an average size of just over 8,500. In this direct comparison of two models the obvious choice would be the hybrid approach. This difference in execution time could be pivotal is selecting the best models.

5.2.4 Most Frequently Occurring Features

Where over and under sampling tackles the class imbalance problem by attempting to equalise the number of positive and negative samples, another approach that was explored was to determine if the class imbalance problem could be solved by feature selection. Using a frequency distribution of all positive and negative tokens, the top $x\%$ most frequently occurring tokens in each class are chosen. Alternatively, choosing the same top n positive and negative samples is also an option.

To achieve this feature sampling required significant changes to the original NLTK Python script described in Section 5.1. A partial listing of the Python Script, where it is different from the original script, is given in Appendix A.5. The highlights are shown in Listing 5.9 where the additional functionality required to extract to most frequently tokens for the positive class is shown.

The first difference is that the list of all words for the positive class is generated, lines 5 to 8. Next the frequency distribution is created and the top percentage is extracted. In line 12 the top 25% most frequently occurring tokens are chosen and the comment in line 13 shows how the top 500 tokens would be chosen. Finally, the dictionary describing whether the top features are present or not in the samples is created. As shown in lines 1 to 3 this function is slightly different in that it is checked whether each token is one of the top tokens and, if it is, only then is it included as a feature.

Listing 5.9: Selecting the most frequently occuring tokens

```
1 def pos_word_feats(words):
2     return dict([(word, True) for word in words
3                  if word in top_pw])
4
```

TABLE 5.3: Table giving the performance measured for over sampling at ratio 1:1 and hybrid sampling at 50:50 and the percentage differences

Over Sampling Ratio 1:1							
	ds_00	**ds_01**	**ds_02**	**ds_03**	**ds_11**	**ds_12**	**ds_13**
pos precision	0.698	0.682	0.804	0.872	0.733	0.896	0.984
pos recall	0.981	0.981	0.997	0.998	0.965	0.998	1.000
pos F-measure	0.815	0.804	0.890	0.930	0.833	0.944	0.992
neg precision	0.968	0.966	0.996	0.998	0.949	0.997	1.000
neg recall	0.574	0.542	0.756	0.853	0.647	0.884	0.984
neg F-measure	0.720	0.693	0.859	0.919	0.769	0.937	0.992
model accuracy	0.777	0.761	0.876	0.925	0.806	0.941	0.992
run time	17.538	16.376	24.878	27.112	14.181	18.385	14.658
Hybrid Sampling Ratio 50:50							
	ds_00	**ds_01**	**ds_02**	**ds_03**	**ds_11**	**ds_12**	**ds_13**
pos precision	0.707	0.692	0.791	0.870	0.744	0.894	0.984
pos recall	0.975	0.975	0.992	0.995	0.957	0.996	1.000
pos F-measure	0.819	0.809	0.880	0.928	0.837	0.942	0.992
neg precision	0.959	0.957	0.989	0.994	0.940	0.996	1.000
neg recall	0.595	0.566	0.737	0.850	0.670	0.881	0.984
neg F-measure	0.734	0.711	0.844	0.917	0.782	0.935	0.992
model accuracy	0.785	0.770	0.864	0.923	0.813	0.939	0.992
run time	11.036	10.405	16.327	18.245	9.040	11.817	9.325
Percentage Difference							
	ds_00	**ds_01**	**ds_02**	**ds_03**	**ds_11**	**ds_12**	**ds_13**
pos precision	1.28	1.49	-1.64	-0.23	1.51	-0.24	0
pos recall	-0.64	-0.64	-0.51	-0.30	-0.83	-0.13	0
pos F-measure	0.49	0.61	-1.14	-0.26	0.49	-0.19	0
neg precision	-0.89	-0.90	-0.69	-0.34	-0.93	-0.15	0
neg recall	3.71	4.47	-2.55	-0.26	3.51	-0.31	0
neg F-measure	2.04	2.53	-1.76	-0.30	1.69	-0.24	0
model accuracy	0.96	1.18	-1.39	-0.28	0.91	-0.21	0
run time	-37.07	-36.46	-34.37	-32.70	-36.25	-35.72	-36.38

```
5  pos_words = []
6
7  for fileid in pos_ids:
8      pos_words += [word for word in (reader.words(fileid))]
9
10 pw_dist =  nltk.FreqDist(pos_words)
11
12 top_pw = pw_dist.keys()[:int(len(pw_dist.keys())*.25)]
13 # top_pw = pw_dist.keys()[:500]
14
15 pos_feat = [(pos_word_feats(reader.words(fileids=[f])),
16               'bullying')
17              for f in pos_ids]
```

The performance measures returned using the most frequently occurring tokens was very disappointing and they did not show any improvement over the baseline from Section 5.1. Feature selection combined with over, under and hybrid sampling actually returned worse performance measures than the sampling techniques did on their own.

5.3 Scikit-learn Modelling

The Natural Language Toolkit produced some promising models and was quite easy to use. It did not, however, offer much choice or scope for different lines of investigation during model development. Although additional functionality is made available to the developer through a wrapper for Scikit-Learn [25], in this section the NLTK is left behind and focus, instead, turns to the use of Scikit-Learn. Initially, some simple models are quickly developed and explored before attention is given to the more advanced fine tuning operations that are available.

5.3.1 Model Development

Whilst the development of the Scikit-Learn model follows a very similar flow to that utilised for the NLTK model, Scikit-Learn could be considered as a much more advanced toolkit where built-in methods provide a lot of the required data manipulation functionality. As well as this data manipulation functionality Scikit-Learn also provides a better selection of learner algorithms. For the initial investigation, a Naive Bayes and a Support Vertor Machine learner were considered.

The first step in the process is to access the data. Whilst the data could have been accessed directly from the NLTK corpora location, it was decided to pre-process the dataset such that each corpus was consolidated into a single comma separated file with two attributes. The first attribute was the class where `0` was used to represent a bullying sample and `1` not bullying sample. The second attribute was the text of the question. For example:

```
1,"your really cute"
0,"could you just like not breathe"
```

A comma separated versions reader was then used to load the data. The class attribute was loaded into an array call target, class is a reserved word in Python, and the text of the question was loaded into an array named data.

Previously, when using the NLTK, the models developed were not evaluated using unseen data. However, as the performance of the models developed using Scikit-Learn will be evaluated against unseen data, the next step was to separate the data into a training dataset and a testing dataset. Rather than having to manually split the data, and ensure that the division is representative, Scikit-Learn provides a `cross_validation.train_test_split` operator that will automatically split the dataset into random training and testing subsets. The code snippet in Listing 5.10 shows how this function is used.

Listing 5.10: Using the cross validation train test split function

```
1 X_train, X_test, y_train, y_test = \
2      cross_validation.train_test_split(
3          data, target, test_size=0.2,
4          random_state=(random.randrange(1,100)))
```

The data and target arrays are the first two parameters passed. The third parameter passed is the proportion of the dataset to include in the test set. In this case 20% of the data will be held back for the test dataset. The final parameter is the pseudo-random number generator state used for random sampling. A random number in the range 1 to 100 is used for each invocation. By convention the returned arrays are named `X_train, X_test, y_train, y_test` where `X_train, X_test` are the datasets representing the split data array and `y_train, y_test` are the datasets representing the split target array.

The next step is to create an instance of a `TfidfVectorizer` which performs two main tasks. The first is to convert a collection of text documents, in this case the array of training questions, into a matrix that is a sparse representation of token counts. The second tasks transforms the count matrix into a term frequency inverse document frequency (TF-IDF) representation. There are many options that can be specified when creating the `td-idf` object which will be explored later. Whereas the training data is transformed into a document / token matrix and the TD-IDF of each token is calculated, the test data is only transformed into the sparse matrix format for use later.

Listing 5.11: Create TfidfVectorizer and transform training and test data

```
1 tfidf   = TfidfVectorizer()
2 X_train = tfidf.fit_transform(X_train)
3 X_test  = tfidf.transform(X_test)
```

The final step, before evaluating the model, is to fit the model to the training data and then predict the class of testing data. Listing 5.12 shows the creation of a Multinomial Naive Bayes model and its fitting to the training data, followed by the class prediction for the training dataset. Also shown in brackets is the generation of the second model that will be examined, the Linear Support Vector Classifier. As well as the performance

measurements generated for the NLTK models scikit-learn also gives the resulting confusion matrix.

Listing 5.12: Create TfidfVectorizer and transform training and test data

```
1 model = MultinomialNB().fit(X_train, y_train)
2 # model = LinearSVC().fit(X_train, y_train)
3
4 y_pred = model.predict(X_test)
```

The complete listing for this simple Scikit-Learn learner is given in Appendix A.6.

5.3.2 Model Execution and Performance

Once developed, the model was executed for each of the seven datasets previously defined. It was also run for the Multinomial Naive Bayes learner and the Linear Support Vector learner using the default parameters (in both cases passing no parameters uses all default parameters options). Figure 5.11 shows the performance of these two models.

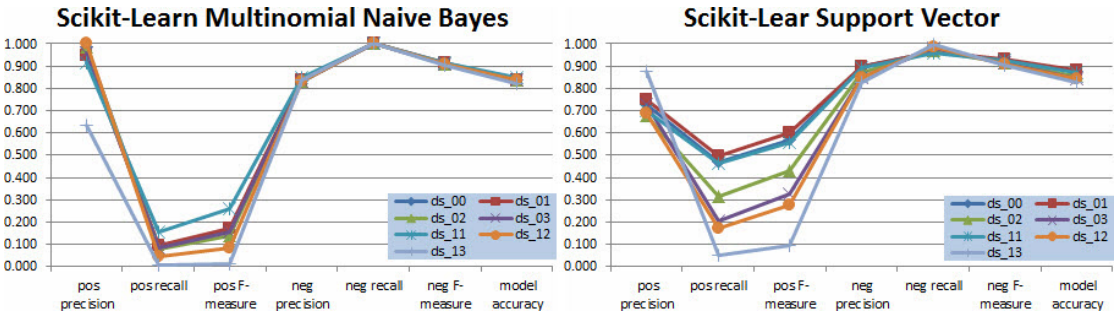

FIGURE 5.11: Graph showing the performance of the Scikit-Learn Multinomial Naive Bayes and Linear Support Vector learners using the default parameters

It is immediately apparent that neither learner has performed well when predicting the positive class, but it is clear that the Naive Bayes classifier has performed particularly badly. Both models returned good results predicting the negative class with the Support Vector classifier performing the better of the two. However, with the use of Scikit-Learn it is also possible to dig deeper into these performances by analysing the confusion matrices for each of the datasets and models. Table 5.4 and Table 5.5 both give sample matrices, and it can be seen that the Naive Bayes model nearly exclusively predicted all samples as negative and although the Support Vector model managed to achieve better results it still favoured the negative class. The extreme examples are the performance of the models on dataset 13. In total, the Support Vector model only predicted 17 positive samples and the Naive Bayes model only predicted 3 positive samples or less than 1.5% of the total possible number. It was assumed that the first simple NLTK models were also only predicting negative classes so Scikit-Learn providing this extra information is an advantage.

As five-fold cross validation was used this meant that five confusion matrices, one for each execution, were generated. It should be noted that the confusion matrices show in Tables 5.4 and Table 5.5 are actual numbers from one of the models and not averages as calculating averages would not make any sense.

Table 5.4: Table giving summary confusion matrix results for each test dataset from the initial Scikit-Learn Naive Bayes model

	ds_00	**ds_01**	**ds_02**	**ds_03**	**ds_11**	**ds_12**	**ds_13**
Pos Samples	335	328	320	309	352	276	211
Neg Samples	1580	1581	1466	1327	1536	1333	982
Pred Pos True Pos	35	34	30	28	59	21	2
Pred Pos True Neg	0	5	0	0	7	0	1
Pred Neg True Neg	1580	1576	1466	1327	1529	1333	981
Pred Neg True Pos	310	294	290	281	293	255	209

Table 5.5: Table giving summary confusion matrix results for each test dataset from the initial Scikit Support Vector model

	ds_00	**ds_01**	**ds_02**	**ds_03**	**ds_11**	**ds_12**	**ds_13**
Pos Samples	353	335	306	295	361	292	212
Neg Samples	1562	1574	1471	1341	1527	1317	981
Pred Pos True Pos	172	173	104	64	163	61	12
Pred Pos True Neg	67	49	66	22	56	19	5
Pred Neg True Neg	1495	1525	1405	1319	1471	1298	976
Pred Neg True Pos	181	162	202	231	198	231	200

A very interesting performance measure, that has only been briefly mentioned thus far, is the length of time each model takes to run. An examination of the run times for the first NLTK model and these first Scikit-Learn models showed that the Scikit-Learn models were between 87.5% and 90.5% faster with executions times around 1 second compared to 10 seconds for the NLTK models. This is a significant discovery. Given a choice between two models that have similar performance measurements, but one is ten times faster, the model chosen would always be the faster model. Hopefully this runtime advantage is maintained as more complexity is added to the Scikit-Learn models and the other performance measures are on a par with or better than the NLTK models. The time comparisons are shown in Figure 5.12.

5.3.3 Further Exploration

As with the NLTK model it is important to now examine some of the additional options offered by the Scikit-Learn TF-IDF implementation and also the Naive Bayes and Support

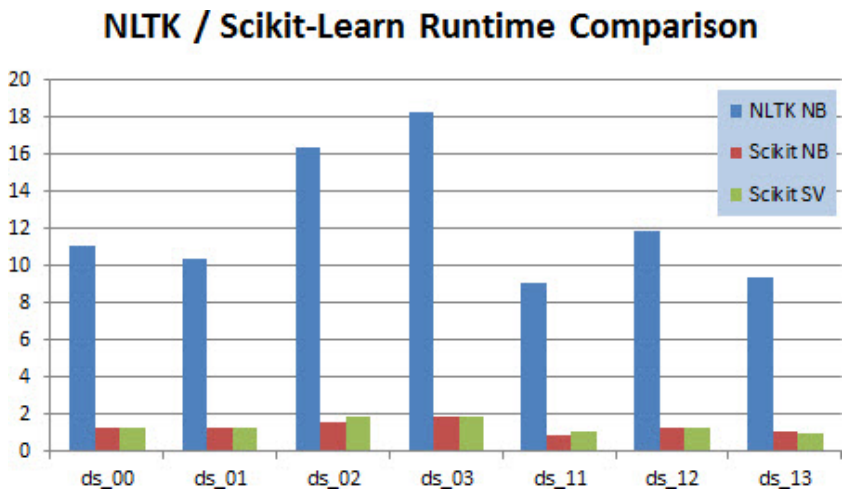

FIGURE 5.12: Graph showing the difference in runtime performance between the NLTK and Scikit-Learn

Vector classifiers to determine if the model performance can be further improved before addressing the class imbalance. First the TF-IDF implementation will be examined.

The `TfidfVectorizer` object performs two main tasks. The first is to convert a collection of text documents, in this case the array of training questions, into a matrix that is a sparse representation of token counts. It then transforms this count matrix into a term frequency inverse document frequency (TF-IDF) representation. In all, there are over twenty parameters that can be used to customise and fine tune the performance of this operator but for this research project only the following will be examined [25]:

1. **ngram_range**: tuple (min_n, max_n)
   The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $min_n <= n <= max_n$ are used.

2. **stop_words**: string 'english', list, or None (default)
   `english` is currently the only supported string value and this has the affect of removing English stop words.

3. **max_df**: float in range [0.0, 1.0] or int, optional, 1.0 by default
   When building the vocabulary ignore terms that have a term frequency strictly higher than the given threshold.

4. **norm**: 'l1', 'l2' or None, optional
   Norm used to normalize term vectors where `l1` is the Manhattan Norm, $\|x\| = \sum_i |x_i|$, and `l2` is the Euclidean Norm, $\|x\| = \sqrt{\sum_i |x_i^2|}$. None is for no normalization.

Other parameters allow preprocessing of the corpus, similar to that described in Chapter 4, and more advanced control of the TF-IDF calculation, but these were not considered.

In total six models of interest were examined, all of which were based on `dataset_01`, uni-grams without stop words removed. Three were based on the Naive Bayes classifier and three on the Support Vector classifier. The first model of each classifier type used the `stop_words` parameter to remove frequently occurring tokens and these execution runs were called `run_01` and `run_04`. The second and third iteration of each model used the `ngram_range` parameter to specify the use of bi-grams,`run_02` and `run_03`, and tri-grams `run_05` and `run_06`. What is different with the Scikit-Learn N-Gram parameter, when compared with the NLTK version, is that the Scikit-Learn retains all the n-grams in the range specified. For example, when `ngram_range(1, 3)` is specified, this means that all uni-grams, bi-grams and tri-grams will be returned. The performance results from the execution of these six models is shown in Figure 5.13.

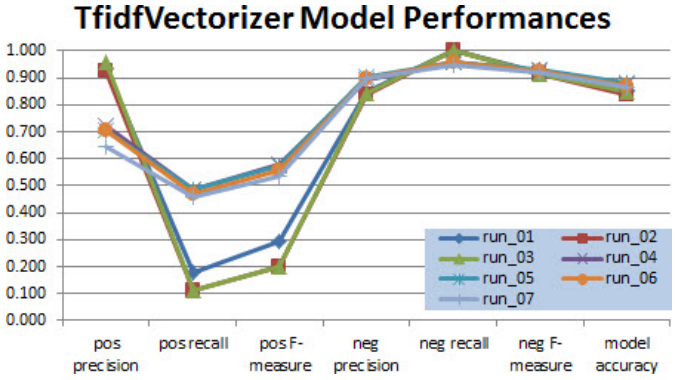

FIGURE 5.13: Graph showing the performance of the Scikit-Learn Multinomial Naive Bayes and Linear Support Vector learners using the default parameters

Comparing Figures 5.11 and 5.13, and from an analysis of the details of the results, it is clear that there was an overall improvement in the models. Though this improvement was especially seen in the models that utilised bi-grams and tri-grams, models `run_02, 03, 05 & 06`, the improvement was attributed to the inclusion of uni-grams. This fact was borne out by testing these models again without including uni-grams, `ngram_range(2, 3)`. This yielded similarly disappointing performance results to the models utilising the original bi-grams and tri-grams only datasets. The inclusion or the `norm` parameter did show a very slight improvement of less than 1% in the performance of the models but the `max_df` appeared to have little or no affect at all. The `TfidfVectorizer` is now as shown in Listing 5.13.

Listing 5.13: TfidfVectorizer parameters

```
1 tfidf = TfidfVectorizer(
2       stop_words='english'
3     , ngram_range=(1, 3)
4     , norm='l2'
```

```
5 )
```

Looking next at the Multinomial Naive Bayes classifier the only obvious parameter to consider at this stage is the `alpha` parameter which is an additive Laplace smoothing parameter. Values of `alpha` of 0.1, 0.01, 0.001 and 0.0001 were tested. Although all values returned improved results when compared with the default value of 1 it appeared that 0.001 gave slightly better results than the others. The performance measures returned when the value of `alpha` was set to 0.001 is also shown in Figure 5.13 as `run_07`. It is clearly seen that introducing the `alpha` parameter has significantly improved the predictive performance of the Naive Bayes classifier bringing its results in line with the Support Vector learner. Whilst there are Support Vector parameters that could be modified to possibly improve the performance of the model they are more associated with addressing class imbalance so they have been left to the next section where this topic is addressed.

It should be pointed out at this stage that the analysis of the parameters in this section is very much based on observation and manual analysis of the performance results returned. In the next section, a more scientific grid based approach will be used in order to fine tune the parameter values.

5.4 Scikit-learn - Class Imbalance

In this section, similar to NLTK, the class imbalance issue is tackled and will use the same under sampling of the majority class, over sampling of the minority class and hybrid sampling techniques that were initially explored in Section 5.2.

However, before addressing the class imbalance, two other very useful features of Scikit-Learn, the pipeline and grid based searches, are utilised to fine turn the classifiers from the previous section.

5.4.1 Pipeline and Grid Search

The Scikit-Learn `Pipeline` class, when used in conjunction with the `GridSearchCV` class, can be used to assemble several transformation and estimator steps that can be executed together in sequence while setting different parameters for each transformer or estimator. In this tentative examining of these features the transformer that will be examined is the TF-IDF transformer and the estimators, or classifier, will be the Multinomial Naive Bayes. Later, when looking at cost based estimation, the Linear Support Vector classifier will also be examined.

When using a grid search, an initial range of values for a set of parameters are defined that are then exhaustively examined such that all possible combinations of parameter values are tested. Where parameters are numeric the initial range of values are typically well spread out and then fine tuned over a number of searches until the optimal value is found. When applied to the Multinomial Naive Bayes estimator with the TF-IDF transformer, the initial parameter ranges applied were as shown in Listing 5.14. The data is then fitted and the best score achieved that can be obtained, as well as the parameter values that yielded the best score, are returned. Some knowledge of the transformers and estimator used in the grid search is assumed as blindly setting arbitrary values for random parameters could give unexpected results that on the surface appear good but on closer examination do not yield the desired results. When used with the Multinomial Naive Bayes estimator the default scoring value is the overall accuracy of the model. The scoring value could be set to score on recall or precision, but this could return a perfect recall score where all samples are given the same class. This would not be the desired result.

Listing 5.14: Using grid search to fine tune parameter values

```
# Create a pipeline with a transformer and a estimator
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf',   MultinomialNB()),
])

# Define the parameter ranges
parameters = {
    'tfidf__stop_words': [None, 'english'],
    'tfidf__ngram_range': [(1, 1),
                           (1, 2),
                           (1, 3),
                           (2, 2),
                           (2, 3),
                           (3, 3)],
    'tfidf__norm': ['l1', 'l2'],
    'clf__alpha': [10, 1.0, 0.1, 0.01],
}
```

The results returned from the first running of the grid search object were:

```
Done in 163.616s
Best score: 0.873
Best Parameters:
    clf__alpha:         0.1
    tfidf__ngram_range: (1, 3)
    tfidf__norm:        'l2'
    tfidf__stop_words:  None
```

Although further fine tuning of the `alpha` parameter was performed, no further gains were made. When these parameter values were fed back into the Scikit-Learn Python script as it existed at the end of the previous section it can be seen, as shown in Figure 5.14 and Table 5.6, that whilst the overall accuracy of the model has indeed improved, this improvement was mostly achieved by a significant improvement in the precision of the positive class at the cost of a small reduction in the positive class recall. It could be argued, depending on the final production use of the model, that this reduction in positive class recall is an acceptable sacrifice given that the maximum model accuracy has been achieved. Balancing this trade off between gains in one area over losses in another will, in the final model, be determined by whether it is more important to correctly predict the most amount of bullying samples as bullying, a high recall value, while accepting that a significant proportion of not bullying sample will incorrectly be classified as bullying or is it better to maximise the precision of each class.

In Figure 5.14 and Table 5.6 `run_01` represents the previous manually tuned parameters and `run_02` the results obtained using the parameters from the grid search.

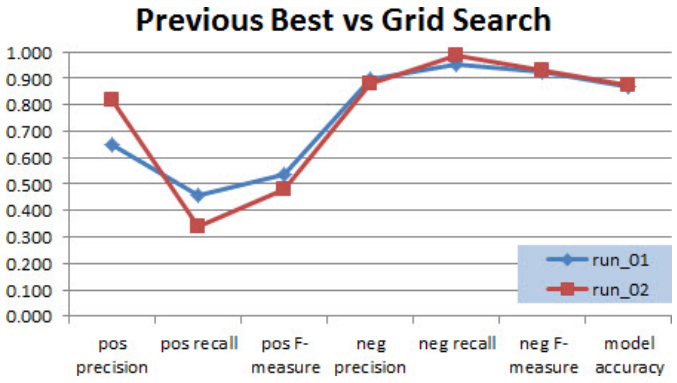

FIGURE 5.14: Graph showing a comparison of the results from the best manually tuned parameters against the best grid search parameters for a Multinomial Naive Bayes classifier

From this point on, as class imbalance options are investigated, the pipeline and grid search classes will continue to be used. The approach is to run the pipeline grid search on the various datasets and then use the parameters returned to get the complete set of performance measures from the model.

A sample Python grid search script is given in Appendix A.7.

5.4.2 Class Sampling

In this section, the sampling logic previously used with the NLTK models are now applied to the pipeline grid search using Scikit-Learn Multinomial Naive Bayes and Linear Support Vector classifiers with a TF-IDF transformer. Datasets with ratios of 3:1,

TABLE 5.6: Table showing a comparison of the results from best manually tuned parameters against the best grid search parameters for a Multinomial Naive Bayes classifier

Performance	**run_01**	**run_02**
pos precision	0.648	0.816
pos recall	0.456	0.342
pos F-measure	0.534	0.480
neg precision	0.898	0.882
neg recall	0.950	0.984
neg F-measure	0.922	0.930
model accuracy	0.868	0.875
run time	2.370	3.519
Confusion Martix	**run_01**	**run_02**
pos samples	344	345
neg samples	1565	1564
Pred Pos True Pos	157	117
Pred Pos True Neg	77	21
Pred Neg True Neg	1488	1543
Pred Neg True Pos	187	228

2:1 and 1:1, are generated for under sampling of the majority class and over sampling of the minority class and for the hybrid approach the negative class and positive class are sampled to ratios of 70:30, 60:40 and 50:50 respectively. The results from these model executions are shown in Figure 5.15.

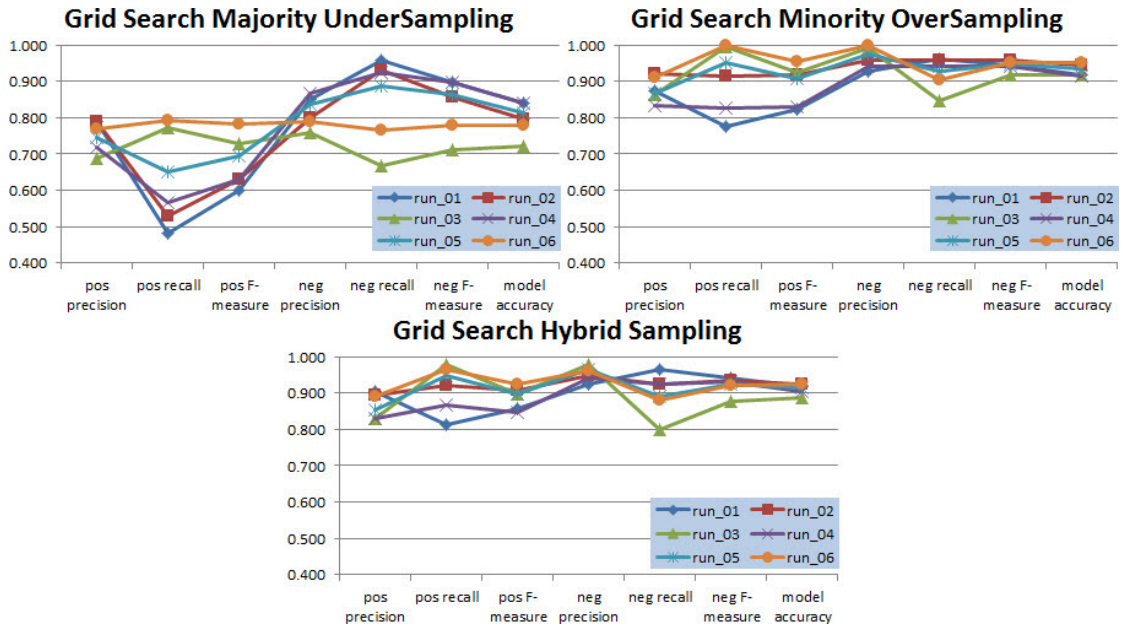

FIGURE 5.15: Graphs showing the performance results for grid searches using under, over and hybrid sampling with Multinomial Naive Bayes and Linear Support Vector classifiers

In Figure 5.15 `run_01, 02 & 03` are the results from the Multinomial Naive Bayes classifier and reflect ratios `3:1, 2:1 1:1 or 70:30, 60:40 & 50:50` respectively while `run_04, 05 & 06` are the results from the Linear Support Vector classifier and are representative of the same sample ratios. With just a cursory look at these charts, it is quite clear that over sampling of the minority class and hybrid sampling both significantly out perform under sampling of the majority class. Because of these results, under sampling will not be analysed further. The actual performance values achieved for over sampling and hybrid sampling are given in Table 5.7.

TABLE 5.7: Table comparing the grid search performance of over sampling and hybrid sampling using Multinomial Naive Bayes and Linear Support Vector classifiers

Over Sampling						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
pos precision	0.874	0.920	0.864	0.832	0.868	0.912
pos recall	0.776	0.914	0.994	0.826	0.952	1.000
pos F-measure	0.822	0.916	0.926	0.830	0.908	0.954
neg precision	0.928	0.958	0.992	0.942	0.974	1.000
neg recall	0.962	0.958	0.846	0.942	0.928	0.904
neg F-measure	0.944	0.958	0.916	0.942	0.948	0.950
model accuracy	0.916	0.945	0.919	0.915	0.935	0.952
run time	4.056	4.765	6.871	5.274	6.312	4.017
Hybrid Sampling						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
pos precision	0.904	0.894	0.830	0.830	0.854	0.890
pos recall	0.812	0.922	0.978	0.866	0.948	0.964
pos F-measure	0.856	0.908	0.898	0.848	0.898	0.926
neg precision	0.924	0.948	0.978	0.940	0.964	0.962
neg recall	0.964	0.926	0.798	0.924	0.892	0.880
neg F-measure	0.942	0.936	0.878	0.930	0.926	0.920
model accuracy	0.917	0.925	0.889	0.904	0.915	0.924
run time	3.784	3.521	3.579	3.964	2.467	2.374
Percentage Difference						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
pos precision	3.43%	-2.83%	-3.94%	-0.24%	-1.61%	-2.41%
pos recall	4.64%	0.88%	-1.61%	4.84%	-0.42%	-3.60%
pos F-measure	4.14%	-0.87%	-3.02%	2.17%	-1.10%	-2.94%
neg precision	-0.43%	-1.04%	-1.41%	-0.21%	-1.03%	-3.80%
neg recall	0.21%	-3.34%	-5.67%	-1.91%	-3.88%	-2.65%
neg F-measure	-0.21%	-2.30%	-4.15%	-1.27%	-2.32%	-3.16%
model accuracy	0.14%	-2.14%	-3.28%	-1.15%	-2.08%	-2.93%
run time	-6.71%	-26.11%	-47.91%	-24.84%	-60.92%	-40.90%

When analysing each models performance results, the nature of the research in hand, and its objectives, are the steering force giving guidance to determine which model

best meets the criteria. The main objective of this research is to identify bullying questions such that they could potentially be flagged either before posting or before being read. The perfect solution would be one where all questions identified as bullying, the positive class, and all not bullying questions, the negative class, are correctly identified all of the time. Failing this, because of the potential harm that could be caused by a vulnerable teen reading a bullying question, the most important criterion is that as many bullying questions as possible are identified. This implies that maximising positive class recall is the number one criteria when classifying samples. However, simply over predicting samples as bullying would not be an acceptable solution. Consider an analogy to spam detection in emails. If ham emails are continuously quarantined because they are incorrectly labelled as spam, confidence in the filter would quickly be lost leading to it being un-installed. So the second criterion, when evaluating these models, is to also maximise negative class recall.

It is worth remembering the performance measurements achieved are the average results from five-fold cross validation where 80% of the samples were used to train the model and 20% used for testing. Examining first the over sampling models, the top three positive recall values are from the linear support vector model with ratio of 1:1, the Naive Bayes model with ratio 1:1 and linear support vector model with ratio 2:1. Although positive recall for the first two models are 100% and 99.4% respectively, the negative recall for the linear support vector 2:1 ratio model is actually the highest. Given the known risk of over fitting when over sampling is used, and the possible inability of the model to generalise to the entire population, the most likely best candidate model from the over sampling experiments would be `run_05`. This is the linear support vector classifier using a negative to positive sample ratio of 2:1 and a n-gram seting of (1, 3). The results from the hybrid experiments were very similar with both classifiers using 50:50 sample ratios having the best positive recall. Once again `run_05` with the middle sample ratio, in this case 60:40 negative to positive, and the Linear SV classifier with a (1, 2) n-gram parameter had probably the best overall balanced performance. Comparing the top two models in each sampling experiment directly there is very little difference in the positive recall values with over sampling having the edge by just over 0.5% though the advantage was wider at 2.5% on negative recall. However, given that hybrid sampling was nearly 61% faster, as a smaller dataset was used, it would be difficult to choose between the two without performing some more detailed testing.

Also of note, and important to highlight, was the different parameter values returned from the grid search for each model execution, shown in Table 5.8. It is no surprise that the most popular n-gram range scheme, especially with over sampling and hybrid sampling, is the (1, 3) scheme which includes uni-grams, bi-grams and tri-grams. The inclusion of all n-grams, coupled with the fact that all but one model also included stop

words, must have aided greatly in the identification of each class. L2 was also the most popular normalisation parameter setting with five out of every six models using this setting.

TABLE 5.8: Table showing the parameter values returned for each sampling technique by a grid search

Under Sampling Parameters						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
tfidf__ngram_range:	(1, 2)	(1, 2)	(1, 1)	(1, 3)	(1, 1)	(1, 2)
tfidf__norm	l2	l2	l1	l2	l2	l2
tfidf__stop_words	None	None	None	None	None	None
clf__alpha	0.1	0.2	0.2			
Over Sampling Parameters						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
tfidf__ngram_range	(1, 3)	(1, 3)	(1, 3)	(1, 3)	(1, 3)	(1, 2)
tfidf__norm	l2	l2	l1	l2	l2	l2
tfidf__stop_words	None	None	None	None	None	None
clf__alpha	0.1	0.3	0.08			
Hybrid Sampling Parameters						
	run_01	**run_02**	**run_03**	**run_04**	**run_05**	**run_06**
tfidf__ngram_range	(1, 3)	(1, 3)	(1, 3)	(1, 3)	(1, 2)	(1, 3)
tfidf__norm	l2	l2	l1	l2	l2	l2
tfidf__stop_words	None	None	None	None	None	english
clf__alpha	0.2	0.4	0.0075			

A sample Python script showing how over sampling was implemented is given in Appendix A.8.

5.5 Scikit-learn - Cost Sensitive Learning

Having looked at under, over and hybrid sampling methods attention now turns towards a cost sensitive learning approach. Cost sensitive learning is considered an algorithmic solution to class imbalance where there is a cost involved in predicting either a false positive or false negative, with no cost for predicting a true positive or true negative. False positives and false negatives can also be treated, costed, differently as well. Consider a cancer diagnoses, for example. It is better to predict a false positive and have the patient undergo further testing to discover they are free from cancer rather than predict a false negative where the patient receives no further tests or treatment which could have grave implications.

Unfortunately, Scikit-Learn does not have a true cost sensitive implementation analogous to the MetaCost algorithm implemented in Weka though some classifiers do offer `class_weight` and `C` parameters, which are optimized implementations of over sampling and under sampling [25].

1. **C**: float, optional (default=1.0)
   Parameter that tells the learner how much to avoid misclassifying each training example. Larger values of `C` will cause a smaller margin hyperplane to be chosen while a small value will increase the size of the hyperplane thus increasing the risk of misclassification.

2. **class_weight**: {dict, 'auto'}, optional
   Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The 'auto' mode uses the values of y to automatically adjust weights inversely proportional to class frequencies.

It was hoped to evaluate the C Support Vector Classification learner in addition to the Linear Support Vector classifier already examined. However, the results returned were either disappointing or the learner proved impossible to fine tune either returning all positive or all negative predictions when training.

5.6 Choosing the Best Classifiers

Having completed all NLTK and Scikit-Learn modelling, the next task was to identify which models performed the best to be further evaluated using simulation and which models, even though their performance measures are good, are to be discarded. As previously discussed, the criteria to determine the best model was that it should first maximise recall for the positive bullying class and secondly that it must also maximise recall for the negative not bullying class. As discovered in the literature review in Section 3.3, Kubat and Matwin [29] referred to this measure as the geometric mean of the accuracies measured separately on each class or as $g = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$, from now on referred to as the *g-performance*. This measure is essentially the square root of the positive class recall multiplied by the negative class recall. The top classifiers will be further evaluated in the next section by way of a simulation of live data.

5.6.1 G-Performance

The measure used to determine which models were the best was g-performance. The value for each model was calculated and a heat map generated from the results is shown

in Figure B.1. It can clearly be seen that there are six clusters of results that are very similar and that all these clusters represent either over sampling of the minority class or a hybrid sampling approach.

NLTK Initial	0.7498	0.7509	0.6821	0.5740	0.7378	0.5163	0.2348
NLTK Under Sampling 1:1	0.6742	0.6563	0.6805	0.5889	0.7006	0.5413	0.2378
NLTK Under Sampling 2:1	0.7344	0.7301	0.6836	0.5787	0.7381	0.5302	0.2361
NLTK Under Sampling 3:1	0.7488	0.7537	0.6882	0.5775	0.7406	0.5197	0.2335
NLTK Over Sampling 1:1	0.7502	0.7289	0.8681	0.9225	0.7903	0.9390	0.9920
NLTK Over Sampling 2:1	0.7927	0.7870	0.8687	0.9288	0.8234	0.9448	0.9918
NLTK Over Sampling 3:1	0.7894	0.7884	0.8178	0.8385	0.8040	0.8600	0.8524
NLTK Hybrid Sampling 50:50	0.7616	0.7426	0.8547	0.9199	0.8007	0.9369	0.9918
NLTK Hybrid Sampling 60:40	0.7960	0.7851	0.8649	0.9256	0.8350	0.9418	0.9916
NLTK Hydrib Sampling 70:30	0.8145	0.8124	0.8421	0.8739	0.8338	0.8938	0.9083
Scikit NB Initial	0.3000	0.3066	0.2793	0.2933	0.3924	0.2191	0.0775
Scikit SV Initial	0.6731	0.6936	0.5531	0.4498	0.6646	0.4138	0.2278
Scikit SV Initial TF-IDF	0.4195	0.3376	0.3317	0.6809	0.6837	0.6689	0.6582
Scikit Grid Under Sampling	0.6795	0.7021	0.7190	0.7240	0.7597	0.7789	
Scikit Grid Over Sampling	0.8640	0.9357	0.9170	0.8821	0.9399	0.9508	
Scikit Grid Hybrid Sampling	0.8847	0.9240	0.8834	0.8945	0.9196	0.9210	

FIGURE 5.16: Heat map showing the g-performance measurement for all models

5.6.2 Best Models

Though the NLTK models developed using tri-grams have the best g-performance values, it was felt that it would be remiss to restrict further testing only to this model type and not to also test other model types as well. Testing will be performed in three phases. First, the top NLTK over sampling models will be analysed, followed by the top NLTK hybrid sampling model, before finally the top SciKit-Learn models are investigated.

5.6.3 Initial Models

In total 8 NLTK over sampling models were identified for further analysis. These models, listed in best g-performance order, are shown in Table 5.9. The first column provides a reference name for each model that will be used later for identification.

Considering Table 5.9 it is seen that training data with stop words removed produced better models than those developed using training data that included stop words. It was also seen that tri-grams have returned better results than bi-grams. However, there is not a clear sampling ratio that performs better. A Naive Bayes learner was used in all models.

Table 5.9: Table listing the top performing NLTK over sampling models listed in order of their g-performance.

Ref:	Model Type Ratio	Sampling Words	Stop	N-Grams	G-Performance
1	NLTK Over Sampling	1:1	Removed	Tri-grams	0.9920
2	NLTK Over Sampling	2:1	Removed	Tri-grams	0.9918
3	NLTK Over Sampling	2:1	Removed	Bi-grams	0.9448
4	NLTK Over Sampling	1:1	Removed	Bi-grams	0.9390
5	NLTK Over Sampling	2:1	Included	Tri-grams	0.9288
6	NLTK Over Sampling	1:1	Included	Tri-grams	0.9225
7	NLTK Over Sampling	2:1	Included	Bi-grams	0.8687
8	NLTK Over Sampling	1:1	Included	Bi-grams	0.8681

In addition to the NLTK over sampling models, 8 NLTK hybrid sampling models were identified for further analysis. These models, listed in best g-performance order, are shown in Table 5.10.

Table 5.10: Table listing the top performing NLTK hybrid sampling models listed in order of their g-performance.

Ref:	Model Type Ratio	Sampling Words	Stop	N-Grams	G-Performance
9	NLTK Hybrid Sampling	50:50	Removed	Tri-grams	0.9918
10	NLTK Hybrid Sampling	60:40	Removed	Tri-grams	0.9916
11	NLTK Hybrid Sampling	60:40	Removed	Bi-grams	0.9418
12	NLTK Hybrid Sampling	50:50	Removed	Bi-grams	0.9369
13	NLTK Hybrid Sampling	60:40	Included	Tri-grams	0.9256
14	NLTK Hybrid Sampling	50:50	Included	Tri-grams	0.9199
15	NLTK Hybrid Sampling	60:40	Included	Bi-grams	0.8649
16	NLTK Hybrid Sampling	50:50	Included	Bi-grams	0.8547

Considering Table 5.10 it is seen, once again, that training data with stop words removed produced better models than those developed using training data that included stop words. It was also seen that tri-grams have returned better results than bi-grams but there is not a clear sampling ratio that performs better. A Naive Bayes learner was used in all models.

The final 8 models to consider are all Scikit-Learn models and are shown in Table 5.11

From a quick analysis of Table 5.11 the following observations can be made. It would appear that classifiers developed using the support vector learner performed slightly better than the Naive Bayes learner. It is also clear that the best model performances achieved, during training at least, was when stop words were included in the dataset and when uni-grams, bi-grams and tri-grams were utilised. There is no sampling ratio that is an obvious best performer.

TABLE 5.11: Table listing the top performing Scikit-Learn models listed in order of their g-performance.

Ref:	Model Type Ratio	Samp Words	Stop	N-Grams	G-Perf
17	Support Vector Over Sampling	1:1	Included	1, 2	0.9508
18	Support Vector Over Sampling	2:1	Included	1, 2, 3	0.9399
19	Naive Bayes Over Sampling	2:1	Included	1, 2, 3	0.9357
20	Naive Bayes Hybrid Samplingng	60:40	Included	1, 2, 3	0.9240
21	Support Vector Hybrid Sampling	50:50	Removed	1, 2, 3	0.9210
22	Support Vector Hybrid Sampling	60:40	Included	1, 2	0.9196
23	Naive Bayes Over Sampling	1:1	Included	1, 2, 3	0.9170
24	Support Vector Hybrid Sampling	70:30	Included	1, 2, 3	0.8945

5.7 Applying the Best Classifiers

A novel approach is now introduced to determine which of the top models perform best in a simulation of a real life scenario.

5.7.1 Scenario Overview

A classifier will be used to batch process a corpus of previously unseen questions, called `dataset_02`, to determine whether or not they are bullying in nature. The predictions, bullying or not bullying, from each batch of questions processed, are then fed back into the training dataset and a new model is generated. The next batch of questions is then classified. This process continues until all samples have been processed.

A third test dataset, `dataset_03`, never used at any point in the development of any of the classifiers, will be used to gauge the performance of each of the models before and after the batch processing, in order to determine which model produced the best classifier.

The steps to achieve this can be summarised as follows:

- Prepare the two datasets put aside for this testing in Chapter 4 by generating NLTK or Scikit-Learn corpora as required.
- Classify both datasets using the three top models and record the performance results.
- Simulate the growth in each of the models using `dataset_02`.

- Re-evaluate `dataset_03` again using each of the final models and see if the models performance has increased or decreased. Also compare the complete set of before and after results from `dataset_02`.

5.7.2 Data Preparation

At the end of Chapter 4 `dataset_02` and `dataset_03` had not been preprocessed or stored in the correct corpora format for NLTK or Scikit-Learn. In a real life scenario, the processing required to do this would all have to happen as the data is received. However, in this simulation, all preparatory work was performed upfront. Each dataset was first preprocessed and cleansed using the same Python script as described in Section 4.5. Next, each dataset was saved in the basic NLTK corpus format and renamed as `sim_01` and `hb_01`. The NLTK models to be tested required the dataset in bi-gram and tri-gram format, and with and without stop words removed. This produced eight further NLTK corpora called `sim_02`, `sim_03`, `sim_12`, `sim_13`, `hb_02`, `hb_03`, `hb_12` and `hb_13`. Both of the Scikit-Learn models used the NLTK `sim_01` and `hb_01` corpora but with all the samples presented in a single comma separated file format as before.

5.7.3 Initial Analysis

It would not be feasible, at this stage, to fully evaluate all 24 models identified for further analysis. In order to reduce the number, an initial simple analysis was performed whereby all unseen samples in the simulation and hold-back datasets were classified, and the percentage of bullying questions predicted was calculated. In Table 5.12 the 24 models being evaluated are listed. First, on the left-hand side, they are listed by g-performance and then, on the right-hand side, in the percentage of bullying questions each model found. When manually classifying the training data, the percentage of bullying questions identified was 15.17%. This is also shown in the table. To proceed to the next phase, it was necessary to identify the models with the best potential to generalise to the population.

There are several conclusions that can be made by reviewing the percentage of the unseen samples that were predicted as bullying. Looking first at the top 5 performing models, models 16, 15, 8, 7 and 23, each has classified over 20% of the questions as bullying. As the manually classified samples only had 15.17% cyberbullying content, it is highly probable that these classifiers are predicting a large number of false positives. It would not be expected that such a classifier would predict a higher percentage of a minority class in this manner. A cursory examination of the questions classified as bullying confirmed this to be the case so these models were not considered further. In

TABLE 5.12: Table showing the top 24 models sorted by g-performance, left-hand side, and percentage of unseen data predicted as bullying, right-hand side

Sort By G-Performance			**Sort By Predicted Bullying**		
Ref:	**Bullying**	**G-Perf**	**Ref:**	**Bullying**	**G-Perf**
1	2.34%	0.9920	16	29.69%	0.8547
9	2.50%	0.9918	15	27.94%	0.8649
2	2.32%	0.9918	8	25.30%	0.8681
10	2.36%	0.9916	7	24.84%	0.8687
17	15.51%	0.9508	23	23.44%	0.9170
3	12.74%	0.9448	21	17.87%	0.9210
11	13.43%	0.9418	14	17.80%	0.9199
18	13.49%	0.9399	13	16.82%	0.9256
4	13.56%	0.9390	22	16.65%	0.9196
12	14.98%	0.9369	6	16.03%	0.9225
19	10.04%	0.9357	17	15.51%	0.9508
5	15.32%	0.9288	5	15.32%	0.9288
				15.17%	
13	16.82%	0.9256	12	14.98%	0.9369
20	14.30%	0.9240	20	14.30%	0.9240
6	16.03%	0.9225	24	13.58%	0.8945
21	17.87%	0.9210	4	13.56%	0.9390
14	17.80%	0.9199	18	13.49%	0.9399
22	16.65%	0.9196	11	13.43%	0.9418
23	23.44%	0.9170	3	12.74%	0.9448
24	13.58%	0.8945	19	10.04%	0.9357
7	24.84%	0.8687	9	2.50%	0.9918
8	25.30%	0.8681	10	2.36%	0.9916
15	27.94%	0.8649	1	2.34%	0.9920
16	29.69%	0.8547	2	2.32%	0.9918

general, when classifying unseen data in this manner, a prediction rate similar to, or lower than, the training data is what would be expected. For this reason, and the need to reduce the number of models to be considered further, the next 5 models, 21, 14, 13, 22 and 6, were also ruled out of consideration.

At the bottom of the table it can be seen that four models performed really poorly on the unseen data. Models 9, 10, 1 and 2, all only managed to identify approximately 2.5% of the unseen data as cyberbullying. This number appears to be unrealistically low and it was noticed that these, in fact, were the top four models when ordered by g-performance value. Further examination revealed that these were all NLTK models utilising tri-grams and the highest ratios of over sampling and hybrid sampling. This led to the conclusion that these models had been over fitted to the training data. A detailed examination was not feasible in the time available. However, the distribution of the tri-gram tokens seen in Section 4.4, where the overwhelming majority of the tokens only appeared once in

the dataset, does support this conclusion that the models were over fitted. These four models were immediately rejected.

Of the remaining ten models the five with the lowest cyberbullying percentages, models 4, 18, 11, 3, 19 were also discarded leaving five models to be further examined. These models are highlighted in Table 5.12 and are:

- Scikit-Learn Support Vector with 1:1 over sampling, stop words included and uni-grams and bi-grams
- NLTK 1:1 Naive Bayes over sampling, stop words removed and tri-grams
- NLTK 50:50 Naive Bayes hybrid sampling, stop words removed and bi-grams
- Scikit-Learn Naive Bayes with 60:40 hybrid sampling, stop words included and uni-grams, bi-grams and tri-grams
- Scikit-Learn Support Vector with 70:30 hybrid sampling, stop words included and uni-grams, bi-grams and tri-grams

It was reassuring to see the variety of model types, sampling types and even stop word and n-gram differences across the five models selected. The next step is to perform the simulation of the evolution of each of these models as unseen samples are classified and then fed back into the model.

One final observation, from the analysis of Table 5.12, is that the top four models, sorted by g-performance, were the worst four when sorted by percentage of unseen samples classified as bullying. It was also noted that the worst four models, again when sorted by g-performance, yielded the models that gave the highest percentage of bullying samples in the unseen data. Expanding these observations further, seven out of the top ten by g-performance were in the bottom 10 by percentage predicted, and nine of the bottom ten by g-performance appeared in the top ten sorted by bullying percentage predicted. It was noted earlier that NTLK with tri-grams and stop words removed did not generalise well to the unseen data. Further examination showed that the top 4 models, when sorted by bullying percentage predicted, were all NLTK models using bi-grams and with stop words included. This may just be a coincidence, given the selection of the models and how they were sorted, but it could well be that a combination of there two types of models could prove very accurate if developed.

5.7.4 Batch Processing of Samples

Each of the five models identified were then further examined using the simulation dataset as follows:

- The simulation dataset of 87,205 samples was divided into twenty distinct datasets.
- Then, for each of these datasets:
  - The initial manually classified training dataset was loaded
  - The training dataset was sampled as required by the model, either hybrid sampling or over sampling
  - The model under test was recreated initially using the original training data for the first iteration. Subsequent iterations use the original training data and all simulation samples classified in the previous iterations
  - The simulation dataset portion is classified using this model
  - The classified simulation samples were appended to the training dataset for the next iteration
  - Process repeated until there were no further simulation datasets
- Once all the simulation samples are included in the final model, the hold back dataset was classified and the results were written to a file.

This simple process was repeated for all models. In addition to writing the hold back classification results to file, each of the results for the classification of the simulation samples were also written to file. The results of this batch processing and classification of the hold back data was then analysed.

The goal of this exercise was to determine if the model, without any further manual intervention, would continue to correctly identify cyberbullying. There are three possible outcomes to be considered:

- The model starts to under-predict cyberbullying
  In this case the percentage of cyberbullying samples seen in the hold-back will be significantly less than before the simulation.
- The predictions from the model remain constant
  The percentage of hold-back cyberbullying samples predicted after the simulation is similar to what was seen before when only the manually classified samples were used to train the model.

- The model starts to over-predict cyberbullying
  Here the number of hold-back samples predicted as cyberbullying, after the simulation dataset has been processed, will greatly exceed the original number predicted.

Ideally only a small variance, when compared to the number originally predicted by each model, and to the percentage seen in the manually classified training data, will be seen.

A Python script, giving a sample implementation of this simulation, is given in Appendix A.9.

5.7.5 Analysis of Batch Processing Results

Once the simulation had been run for each model the results were analysed. It was immediately obvious that both of the NLTK models appear to have over-predicted samples as cyberbullying. The NLTK Naive Bayes model using 1:1 over sampling, stop words removed and tri-grams predicted 37.2% of the hold-back samples as bullying. The NLTK Naive Bayes model using 50:50 hybrid sampling, stop words removed and bi-grams predicted that 40.1% of the hold-back samples were bullying. These numbers are representative of a potential significant increase in the number of false positive predictions made by the model. Taking a closer look at the number of samples predicted as bullying and not bullying, for each of the NLTK Naive Bayes models we get the figures in Table 5.13. The column labelled `Model 02` represent the over sampling model, with `Model 03` representing the hybrid sampling model.

TABLE 5.13: Table showing the number of bullying, not bullying, changed and unchanged sample predictions for each of the NLTK models

NLTK Model Analysis

Description	**Model 02**	**Model 03**
Total Samples	9464	9331
Before simulation		
Bullying	1474	1425
Not Bullying	7990	7906
After simulation		
Bullying	3519	3792
Not Bullying	5945	5539
Unchanged		
Bullying	1401	1364
Not Bullying	5872	5478
Changed		
Bullying to Not Bullying	73	61
Not Bullying to Bullying	2118	2428

Only 73 samples in `Model 02`, and 61 samples in `Model 03`, changed classification from bullying to not bullying. However, 2118 and 2428 samples respectively changed from not bullying to bullying. This confirms that samples reclassified as bullying, after the simulation was run, caused the increase in the percentage of bullying samples. Unless the cause of this large increase in bullying predictions can be fully explained, the use of these NLTK models, in the unsupervised scenario described here, would not be recommended. Further investigation is beyond the scope of this research but would be prime material for future work.

In contrast to the extremely high bullying percentages seen from the NLTK model, the Scikit-Learn models returned values more in-line with the expected value. The first model, Scikit-Learn support vector with 1:1 sampling returned a bullying percentage of 19.25%. Approximately 4% more bullying questions were predicted in the hold-back samples after the simulation was run, and this value does appear to be on the high side. The second scikit-learn model, Naive Bayes learner with 60:40 hybrid sampling, returned 15.78% as the percentage of questions classified as bullying. This value, considered in isolation from all further analysis, appears to be the most realistic result achieved. So if the best model was to be chosen at this point, based solely on the percentage of questions classified as bullying, this model would be the one. The final scikit-learn model, support vector with 70:30 hybrid sampling identified 8.03% of the hold back samples as bullying. This value appears on the low side. All three models were further examined similar to the NLTK models. The results of this analysis is shown in Table 5.14.

Table 5.14: Table showing the number of bullying, not bullying, changed and unchanged sample predictions for each of the Scikit-Learn models

Scikit-Learn Model Analysis

Description	**Model 01**	**Model 04**	**Model 05**
Total Samples	11193	11193	11193
Before simulation			
Bullying	1750	1614	1594
Not Bullying	9443	9579	9599
After simulation			
Bullying	2155	1766	899
Not Bullying	9038	9427	10294
Unchanged			
Bullying	1586	1251	732
Not Bullying	8874	9064	9432
Changed			
Bullying to Not Bullying	164	363	862
Not Bullying to Bullying	569	515	167

With a 44% reduction in the number of questions classified as bullying, between the first run and the second run after the simulation completed, the third scikit-learn model,

`model 05` was discarded and not considered further. To continue the analysis of `model 01` and `model 04`, the percentage of change of each measure was calculated from the data in Table 5.14. Figure 5.17 shows the results of these calculations.

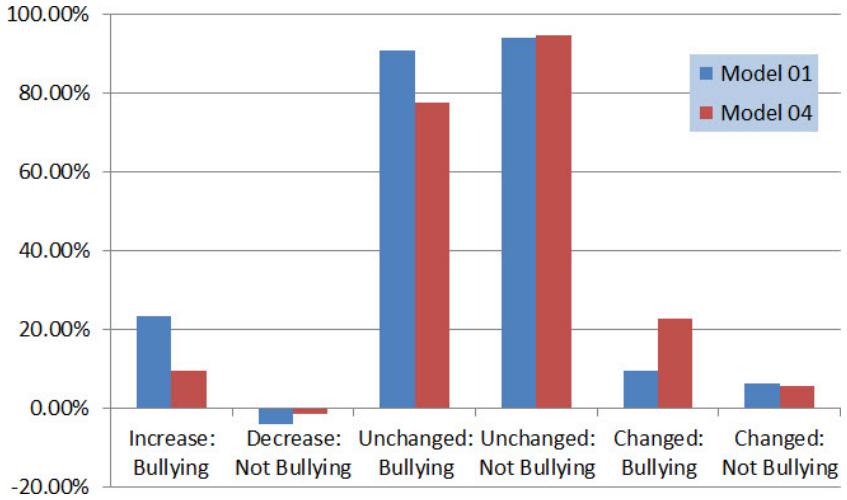

FIGURE 5.17: Chart showing the percentage change in multiple measures between the two classifications performed on on the hold back data

Figure 5.17 can be broken into two parts for further examination. The first analysis considers the `Increase: Bullying` and `Decrease: Not Bullying` columns. These two columns are representative of the change in the number of bullying classifications, and the number of not bullying classifications. The number of samples classified as bullying using both models increased. However, the number of bullying classifications made by `model 01` increased by over 23%. This represents 150% growth in excess of `model 04`. Though the percentage decrease in the number of not bullying samples appears less significant, this is caused by an imbalance in the classes. The decrease in the number of not bullying classifications in both classes is reflective of the increase in numbers discussed.

The analysis of columns 3 to 6 in 5.17, shows how stable the classifications made by each model are, or to look at it another way, by how much the classifications of each model changed between the two runs. Column 3, `Unchanged: Bullying`, and column 5, `Changed: Bullying`, show the number of samples predicted as bullying by `model 01` in both executions. A value of 90.63% unchanged bullying classifications, and 9.37% changed, shows that this model is very consistent in the bullying predictions that it makes. On the other hand, `model 04` shows a significantly lower value of 77.51%. This means that 22.49% of the bullying predictions `model 04` made in the first run, changed to not bullying after the simulation exercise. Columns 4 and 6, `Unchanged: Not Bullying` and `Changed: Not Bullying` are similar percentages for the samples predicted as not bullying. There is not much to choose from between the two models, with unchanged values of 93.97% and 94.26% respectively for `model 01` and `model 04`.

To further analyse the performance of the two models, questions that changed classification were examined. Each question that changed classification was manually classified, in total over 1,600 samples, and then compared to the prediction made by the models. Figure 5.18 shows the results of this analysis.

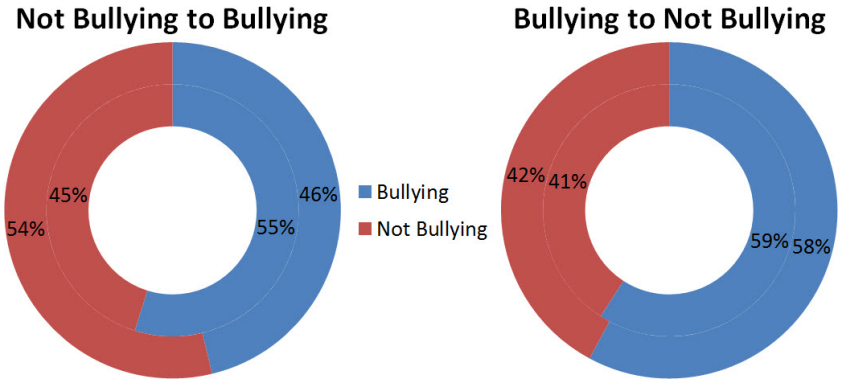

FIGURE 5.18: Graphs showing the percentage of questions that changed classification from bullying to not bullying, and from not bullying to bullying, for the final two models

The doughnut on the left-hand side represents the samples that changed classification from not bullying to bullying. The donut on the right is the samples that changed from bullying to not bullying. The inside ring of each doughnut is `model 01`, the outside ring is `model 04`. Considering the samples that changed from not bullying to bullying first. It is seen that, although neither model excelled, `model 01` correctly reclassified 55% of questions to bullying. At 59% and 58% neither model was very accurate when reclassifying bullying questions as not bullying. It should, however, be noted that `model 01` only reclassified 164 bullying questions, or 1.46% of all samples, which is a very low percentage. Of these 164 samples there were 67, or 41%, correctly reclassified.

The final analysis performed on each model was a manual classification of bullying and not bullying samples that did not change classification between the two model executions. The results of this reclassification are shown in Figure 5.19. Once again it is seen that `model 01`, on the inside of each doughnut, has outperformed `model 04` by correctly predicting 92% of bullying questions, and 87% of not bullying questions. This model, when originally developed is Section 5.4, had a positive class recall value of 1.000, 100%, and a negative class recall of 0.904, 90.4%. So, although the models performance on unseen data was not as good when compared to the values achieved during training, they are still more than satisfactory.

Of the five models analysed, the model that appears to have performed the best is the scikit-learn support vector model using 1:1 over sampling, stop words included and uni-grams and bi-grams. This model was also considered the best performer of the five model before the automatic evolution of the model using the simulation dataset. This is

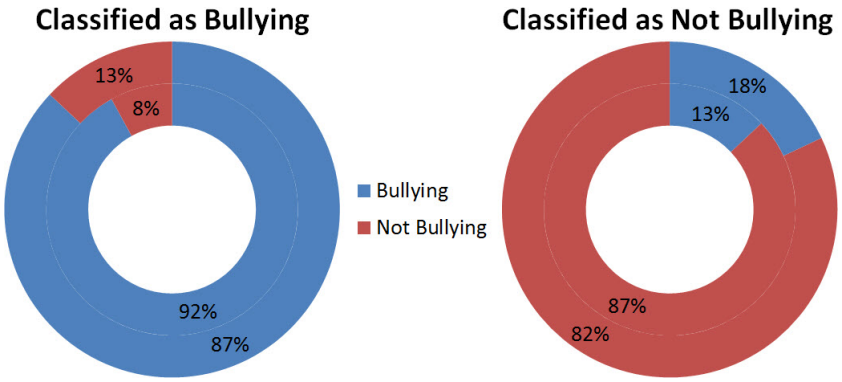

FIGURE 5.19: Graphs showing the percentage of bullying questions and not bullying question that did not change classification and where the class was correctly predicted

far from a definitive declaration of the best model. There is significant potential here for future work as discussed in Chapter 6.

5.8 Summary and Conclusion

This chapter focused on describing how Python, NLTK and Scikit-Learn were used in the development of a number of models that could be used to classify samples as either bullying or not bullying. The modelling processes used were broken into several sections where each section built on the ideas introduced, and results achieved, in the previous sections cumulating in Section 5.7 where the five best models developed were analysed in detail.

In 5.1 the modelling techniques and required corpus structure were introduced. The datasets generated in Chapter 4 were read in and the features, or tokens, present in each of the corpora were extracted before being used to train a Naive Bayes classifier. Although the performance of these first classifiers were not great, especially on the positive bullying class, the necessary first steps to develop a better model had been successfully navigated. It was also examined whether the n-gram generation built into the NLTK performed any better than the pre-generated datasets. It was found that there was no performance improvement in the model and that generation took longer.

In Section 5.2 the issue of class imbalance was addressed. The negative majority class, in this case not bullying samples, significantly outnumbered the positive bullying class. To address this under sampling of the majority class, over sampling of the minority class and hybrid sampling approaches were tested using the same model and datasets developed in the previous section. Although some of the approaches gave very promising results, it was highlighted that over sampling too much could lead to over fitting and under sampling can lead to data loss. In this section there was also an investigation into

balancing the classes using the most frequently occurring features but this path did not yield results worth pursuing.

In Section 5.3 the Scikit-Learn Python package was introduced, and an initial suite of models were developed. Rather than using feature based learning, introduced in the development of the NLTK models, the Scikit-Learn models instead used a Term Frequency, Inverse Document Frequency (TF-IDF) approach. In addition to using TF-IDF, only one dataset was used as it was found that the n-gram generation included with Scikit-Learn gave better results then the manually generated n-gram datasets. A Naive Bayes learner was again used, but a Linear Support Vector Classifier was introduced. Although there was no significant performance improvement using Scikit-Learn there was a noted decrease in execution time for model generation and samples classification.

In Section 5.4, in addition to tackling class imbalance using the same sampling techniques described in Section 5.2, the Pipeline and Grid Search packages were introduced that allowed an automated approach to determining the best parameter values for the TF-IDF, Naive Bayes and Linear SV operators. Similar promising results to those achieved using sampling and the NLTK were achieved but again the faster total run time of the Scikit-Learn models were highlighted.

Cost sensitive Learning using Scikit-Learn was attempted in Section 5.5 but the results were either disappointing or the models proved difficult to train either predicting all samples as positive or negative.

The criteria to be used to determine the best models was described in Section 5.6. The g-performance value of every model developed, $g = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$, was calculated and a heat map of the results was used to identify the top 24 models. In these 24 models there were eight NLTK over sampling models, eight NLTK hybrid sampling models and eight assorted scikit-learn models.

Section 5.7 began by providing a description of the scenario to be used to evaluate the models identified in Section 5.6. The simulation and hold back datasets were introduced, and preprocessing required discussed. Each of the 24 models then classified the hold back dataset and the five best candidates were identified. In the top models there were three scikit-learn models and two NLTK models. These top models were further analysed by simulating the evolution of each as new, unseen samples, were classified and then fed back into the training data for the next iteration. A control dataset was labelled as the hold back dataset and the second dataset, the simulation dataset, was used to evolve the models. The NLTK models were quickly discarded as both were found to be over predicting samples as bullying. Following further analysis of the Scikit-Learn models it

was determined that the Linear Support Vector classifier using 1:1 over sampling, stop words included, uni-grams and bi-grams was deemed the best solution developed.

Chapter 6

Conclusions

The application of text mining tools, techniques and processes to the automatic detection of cyberbullying text is still a relatively new research area [34] [45]. This dissertation was undertaken in the hope that the findings of this research would add to the subject body of knowledge, and foster further research interest in the area. In this chapter, the objectives of the thesis are reviewed, and the findings of the research are discussed. Consideration is given to the conclusions that can be drawn from the results achieved before making suggestions for possible future work.

In Chapter 3 it was seen that children and teenagers are using the internet and social media applications to bully each other and that this bullying is known as cyberbullying [31] [32]. Cyberbullying was defined as an electronic act, such as images, text or videos, posted to a social media site, that are aggressive in nature against a victim who is typically unable to defend themselves. These postings are made with the sole intent of hurting or embarrassing the victim [38] [40] [41]. These posts can be sexual or racial in nature, or a direct attack against a persons intelligence or physical appearance [43].

The effects of cyberbullying on its victims can sometimes have tragic consequences. The daily torment and abuse suffered by the victims of cyberbullying can have a dire psychological and emotional toll. The negative affects of posts that contain cyberbullying, of a personal or sensitive nature, can be internalised by children and young adults leading to significant and emotional psychological suffering [41]. Other effects of cyberbullying can include loneliness, anxiety, low self worth, depression, intra-personal problems, school absence, violent and aggressive behaviour and even suicidal thoughts [34] [45].

When training a classifier to identify documents of different types, a ‘‘bag of words’’ approach is commonly used, sometimes supplemented with other techniques such as examining the role of each party in a bullying incident, or by analysis of the social

network graphs of the bully and of the victim. When solely focusing on the documents, or posts, to be classified, the detection of on-line harassment or bullying can consider the content, sentiment and contextual features of the post including such features as N-Grams, foul or curse words and TF-IDF word vectors [54]. Preprocessing steps can also be applied to the content of the post including stemming, stop word removal and the removal of repeated characters that may have been added by the user that made the post for additional emphasis [41]. Many classifiers were seen to be used including support vector machines, Naive Bayes and decision trees.

Some common obstacles to the development of classifiers to identify cyberbullying text include anonymity, the lack of suitable datasets, the classification of data and class imbalance. Social media sites that permit the anonymous posting of content are prone to cyberbullying [32] [64]. When users can post anonymously there is a perception that they are safe and free to bully without fear of discovery [54]. It was also suggested that anonymous cyberbullying posts can create a heightened sense of fear in the victim as they are confronted with the unknown, with anonymity giving the bully a feeling of power and control over their victim [38]. Anonymous posting also prevents a pattern of abuse being discovered as each anonymous comment on a social media site has to be considered in complete isolation without the possibility of building up a profile of a bully and their social network.

Nearly every major paper reviewed expressed frustration at the lack of a standard labelled dataset that could be used in the research of cyberbullying detection. MySpace, Kongragate and Slashdot datasets provided by Fundación Barcelona Media for the CAW 2.0 Workshop [55] was used [38] [54] as well as YouTube video comments [40] [41], Twitter tweets [34] [45] and data scraped from the www.formspring.me website [32] [64]. In most cases, the data was manually classified using a simple binary label by the authors, or annotators known to the authors [39] [40] [41] [54], but the Mechanical Turk service offered by Amazon was also used on occasion [32] [64].

6.1 Summary of research activities

The primary objective of this thesis was to develop a model, using the Python programming language, that could be used to determine if samples from an unseen dataset should be classified as bullying in nature or classified as not bullying. To meet the objective of this dissertation the following activities were undertaken:

- **Construct a new cyberbullying dataset**
  A dataset of over 110,000 samples was created from data scraped from the Ask.fm

website. Each sample was in the form of a question of up to 300 characters. A number of preprocessing steps were performed on the samples including converting all characters to lower case, removing non ASCII characters, URLs, punctuation, digits and repeating characters and also replacing some common abbreviations with the complete word.

- **Classify the new cyberbullying dataset**
  In order to train a classifier a sample of the complete dataset was manually classified. The criteria used to classify a question as bullying or not was defined resulting in a classified dataset of approximately 9,500 samples of which 1,644 were classified as bullying and 7,899 were classified as not bullying.

- **Develop multiple classifiers**
  Multiple different classifiers were developed using standard text mining techniques such as n-grams, stop word removal, feature selection and TF-IDF word vectors. These models were developed using Python, the Natural Language Toolkit and the Scikit-Learn suite of libraries.

- **Address class imbalance**
  Several different techniques were used to address the class imbalance between the positive bullying class and negative not bullying class. Under sampling of the majority negative class, over sampling of the minority positive class and hybrid sampling which used a combination of under and over sampling were all tested. Some cost based testing was also attempted.

- **Identify the top classifiers**
  All of the models developed were analysed to identify the top classifiers. The models were evaluated against their g-performance, the square root of the positive class recall multiplied by the negative class recall.

- **Evaluate top classifiers to determine the best**
  Once the top models had been identified, each in turn was further evaluated with previously unseen samples in order to determine which classifier generalises best. This testing, in an attempt to simulate a real life scenario, used an iterative process with previously unseen samples being classified and then appended to the models training data. The model was then regenerated, including these newly classified samples, before additional unseen samples are classified. This process of classify, append and regenerate was repeated multiple times. The results from this testing were then examined in order to determine which classifier could be considered the best.

6.2 Discussion

The research question was to determine whether it would be possible to develop a supervised classifier, using Python and standard text mining techniques, that could be used to predict whether an unseen sample should be considered as bullying or not bullying.

The first significant artefact produced as a result of this research is the Ask.fm dataset. The lack of a recognised dataset for use in the research of automated cyberbullying detection is a noted barrier to work in this area. Also, the unsuitability or the unavailability of the datasets used in similar research projects led to the sourcing and construction of a dataset specifically for use in this dissertation from the Ask.fm social networking site. Ask.fm had been successfully used in a similar project [11], not to mention that it has recently been in the news headlines as been associated with cyberbullying activity. Over 100,000 sample records, as described in Chapter 4, were scraped from the Ask.fm site and then a sample set of approximately 9,500 were manually classified. Before classification of the training dataset could begin, the criteria used to identify bullying content was documented and explained. If others wished to build on the work in this thesis, or contest its findings, then the rational behind the decisions made are transparent and understandable. It is very important not to underestimate the consideration given to defining the cyberbullying criteria, the classification effort or the potential future uses of this dataset.

In reviewing and discussing the modelling process, it is worth breaking this large body of work down into smaller parts that can more easily be examined.

The use of Python and the NLTK and Scikit-Learn packages is a distinguishing feature of this research. Python was chosen for use in this project because of it cross platform support, its ease of use and the ability to quickly see results. Python also has many community supported tools such as NLTK and Scikit-Learn. Python easily met and exceeded all expectations by providing more potential data research opportunities than possible to explore in the limited time available. As was shown, the models developed using both NLTK and Scikit-Learn were extremely feature rich with huge potential for further more detailed research. Especially noteworthy was the speed at which the models were generated and once the model had been created how quickly it could provide a predicted class for an unseen sample. Including some data processing, n-gram generation and file IO activity, the final models using the simulation data, approximately 100,000 samples, were generated in under 5.5 seconds. The classification of the hold back data, including reading the data from file, took only an average of approximately 0.38 seconds for 11,193 samples. This equates to 3.4×10^{-5} seconds for each sample. These are

timings from a standard laptop and could be greatly improved upon by the use of faster hardware and also through an optimisation analysis of the python coding to identify bottle necks. Another exciting aspect of the use of Python is the possibility of deploying any potential cyberbullying filtering solutions on mobile devices using Android, iOS or the windows Mobile operating systems. This is explored further in future work.

With 84% of all sample questions being asked anonymously the use of some of supplemental techniques discovered in the literature review, such as gender analysis, social network evaluation and role determination, were not possible in this research. Other approaches, such as a foul word dictionary, were also discarded in favour or a purely statistical bag of words approach. Word features, TF-IDF word vectors, n-gram combinations and including and removing stop words were all utilised. In line with other research, the data was cleansed before using it for model generation and sample prediction.

As is common in other related research papers in this field, it was found that there was a positive class imbalance that affected the performance of the learner algorithms utilised. The use of over sampling of the positive minority class, under sampling of the majority negative class and a hybrid sampling approach is not a new concept. It was shown that over sampling can lead to over fitting to the training data resulting in a model that did not generalise to the population. However, over sampling and hybrid sampling were chosen as the most effective sampling techniques ahead of under sampling.

In the literature, there are many suggestions of how best evaluate the performance of a classifier model like those generated in this dissertation. The method chosen was to maximise the recall of both the positive and negative class by using the g-performance measurement. Although a bag of words approach has been used multiple times in previous text mining research, even papers directly related to the cyberbullying issue, none of these, to the knowledge of the author, have attempted an evaluation of the built models as described in Chapter 5 Section 5.7. The availability, and use, of a large dataset of unseen samples that could be used to simulate the evolution of a model and subsequent analysis of the performance of the newly evolved model is novel.

6.3 Future Work

The work performed in this research effort has only scratched the surface of the possibilities offered by the use of Python in the automated detection of cyberbullying content.

When the classifier models were generated, only Naive Bayes and Linear Support Vector learner algorithms were used. There are many other types of supervised and unsupervised learners that were not investigated including the use of ensemble methods. It should also

be possible to develop multiple classifiers that each individually specialise on identifying very specific bullying content attributes and then use a majority vote to classify each sample. The confidence of the prediction might also be utilised.

There is great scope for further investigation into methods to handle the class imbalance problem. For example, identifying a true cost based classifier that is available in Python or, if one don't exist, then the development of a classifier similar to the Weka MetaCost learner. Another avenue of investigation is the use of Synthetic Minority Over-sampling Technique (SMOTE), or the adaptation of this technique for use in a sparse TF-IDF word vector. A weighted approach to hybrid sampling, where the predictive power of each training sample is taken into account, is another possible research opportunity. Rather than randomly choosing samples, the positive and negative samples that are most predictive could be chosen ahead of other more generic samples. Though not directly related to class imbalance the inclusion of a cost for over or under sampling data at the model evaluation stage could prove both novel and enlightening. Further investigation into formalising such an approach is warranted.

Part of speech tagging (POS), is the process of identifying the types of each word in a corpus. For example, to tag all words in the corpus as being a noun, verb, article, or adjective, for example, based on its definition and relationship or adjacency with others in the sentence or paragraph. POS was not used in this research project, but an investigation to determine if its inclusion would positively impact the performance and stability of the models is warranted.

The models developed in this paper were simple binary classifiers where the sample was either bullying or not. It would be interesting to examine if the different types of bullying, for example sexual, racial, threatening, stalking, grooming, could be used to develop a multiclass classifier, maybe utilising Latent Dirichlet allocation for topic identification

The majority of children and teens can now access social media networking sites using mobile devices. Given that Python is available on most of these devices, it would be a challenging software engineering project to develop a tool that integrated the work in this paper with common social networking applications to provide instant filtering, and maybe censorship, to allow parents prevent the delivery of harmful content.

Towards the end of this project, a potential option to allow the inclusion of uni-grams, bi-grams and tri-grams in the same NLTK model was discovered. It would be interesting to repeat this research using n-grams and NLTK in this manner to see if any improvement in the model performance can be achieved. Alternatively investigate if manually generating these n-grams up-front, during initial data exploration, is a viable option.

Finally, this research only used the questions from the Ask.fm dataset. With each question there is an associated answer. Could an analysis of the answers assist in the identification of bullying content. For example, is bullying met with bullying? It would also be revealing to perform sentiment analysis of the answers to determine from the responses given if a user was maybe at risk from self-harm or even having suicidal thoughts.

Appendix B describes a novel approach that could be used to identify the best models without having to evaluate all models individually. The ideas presented, though interesting, need more work in order to determine a more robust and predictable solution. It would, however, be very desirable to develop such a formula for use in any similar future research.

6.4 Conclusion

To conclude, the research question was:

> The research question of this thesis is whether standard data mining techniques, for example n-grams, stop word removal, feature selection, term frequency inverse document frequency word vectors, can be used to develop a classifier in Python which can be used to predict whether or not unseen samples are bullying in nature.

From the literature review, modelling and analysis activities performed it was shown that it is possible to develop a cyberbullying classifier using Python and that this dissertation contributes an extensive investigation into the area and is a valuable addition to the subject matter body of knowledge.

Appendix A

Selected Python Scripts

Selected Python scripts that are important to this research effort are shown in full here.

A.1 Parse Raw HTML Files from Ask.fm

```
# BeautifulSoup needed to parse downloaded html files
from bs4 import BeautifulSoup

# glob and os are used to get the list of html files
import glob
import os

# MySQLdb is needed to write the data to the database
import MySQLdb as mdb
import sys

# Variables for the anonymous question id, user id and asker id
q_id = 1
u_id = 1
a_id = 1

q_id_str = ''
u_id_str = ''
a_id_str = ''

# Distionary to hold anonymous ask ids
ask_ids = {'no_ask_id': '9999'}

# Set the folder where the html files are
os.chdir("R:\\Raw")

# Get the list of files
for file in glob.glob("*.htm"):

  # Reset variables for the anonymous question id
```

```
31    q_id = 1
32
33    q_id_str = ''
34    u_id_str = ''
35    a_id_str = ''
36
37    # Parse the html file using the html parser
38    soup = BeautifulSoup(open(file), 'html.parser')
39
40    # Connect to the database for each html file
41    try:
42      con = mdb.connect(
43        'localhost',
44        'root',
45        'mysqlroot',
46        'thesis',
47        use_unicode=True,
48        charset="utf8")
49
50      cur = con.cursor()
51
52      # Find each occurrence of the question box and process
53      questions = soup.findAll("div", {"class": "questionBox"})
54
55      for a_question in soup.findAll("div",
56                      {"class": "questionBox"}):
57
58        # Generate the anonymous user id and question id
59        u_id_str = str(u_id).zfill(3)
60        q_id_str = u_id_str + str(q_id).zfill(5)
61
62        # These elements / attributes will always exist
63        question_id = a_question["id"].split("_")[2].strip()
64        user_id = a_question.find(
65          attrs="reportFlagBox").a["href"].split("/")[3].strip()
66        question = a_question.find(
67          attrs="question").span.span.get_text().replace(
68          "\n",
69          "").strip()
70        answer = a_question.find(
71          attrs="answer").get_text().replace(
72          "\n",
73          "").strip()
74        ask_time = a_question.find(
75            attrs="time").a.get_text().strip()
76
77        # These elements / attributes may or may not exist
78        try:
79          if a_question.find(
80            attrs="author nowrap").a["href"].split("/")[3]:
81            ask_id = a_question.find(
82              attrs="author nowrap").a["href"].split(
83                  "/")[3].strip()
84        except (AttributeError, KeyError):
85          ask_id = "no_ask_id"
86
87        try:
```

```
88          if a_question.find(
89            attrs="likeList people-like-block").a.get_text(
90                ).split(" ")[0]:
91            ask_likes = a_question.find(
92              attrs="likeList people-like-block").a.get_text(
93                  ).split(" ")[0].strip()
94        except (AttributeError, KeyError):
95          ask_likes = ""
96
97        # Anonymise the id of the asker
98        if ask_ids.has_key(ask_id):
99          a_id_str = ask_ids[ask_id]
100       else:
101         a_id_str = str(a_id).zfill(3)
102         a_id = a_id + 1
103         ask_ids.update({ask_id: a_id_str})
104
105       cur.execute(
106         "INSERT INTO thesis.raw_data (question_id     \
107                                     , q_id,user_id    \
108                                     , u_id,ask_id     \
109                                     , question        \
110                                     , answer          \
111                                     , ask_time        \
112                                     , ask_likes       \
113                                     , a_id)           \
114         VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",
115         (question_id,
116          q_id_str,
117          user_id,
118          u_id_str,
119          ask_id,
120          question,
121          answer,
122          ask_time,
123          ask_likes,
124          a_id_str))
125
126       # Increase question number
127       q_id = q_id + 1
128
129     con.commit()
130
131 # Handle database errors
132 except mdb.Error as e:
133
134     print "Error %d: %s" % (e.args[0], e.args[1])
135     sys.exit(1)
136
137 # Close the database connection
138 finally:
139     if con:
140       con.close()
141
142 u_id = u_id + 1
```

A.2 Clean data

```
import re
import string
import unicodedata
import stopwatch
from replacers import RegexpReplacer
from replacers import RegexpReplacerAge
from replacers import RepeatReplacer

# Import DBConn to connect to database
from thesis_db import DBConn

# Create a connection to the thesis database
my_conn = DBConn()
conn = my_conn.create_conn()
cur = conn.cursor()

# First of all handle the bullying cases
cur.execute("select distinct question from sample_data_02 \
             where  class = 'not_bullying'")

for row in cur:

    # Convert question to lower case
    question = row['question']
    question_lower = question.lower()

    # Convert question to ascii
    question_ascii = unicodedata.normalize(
        'NFKD',
        question_lower).encode(
        'ascii',
        'ignore')

    # Remove URL
    question_url = re.sub(
        r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*',
        '',
        question_ascii)

    # Remove any remaining punctuation
    question_punct = question_url.translate(
        string.maketrans(
            "",
            ""),
        string.punctuation)

    # Replace ages 11 - 25 with text values
    age_replacer = RegexpReplacerAge()
    question_age = age_replacer.replace(question_punct)

    # Remove any remaining digits
    question_no_digit = question_age.translate(
        None, string.digits)
```

```
54
55      # Remove repeating characters
56      rep_replacer = RepeatReplacer()
57      question_rep_list = []
58
59      for word in question_no_digit.split():
60          question_rep_list.append(
61              rep_replacer.replace(word))
62
63      question_no_repeat = ' '.join(question_rep_list)
64
65      # Fix abbreviations to display whole words
66      replacer = RegexpReplacer()
67      question_replace = replacer.replace(question_no_repeat)
68
69      # Write to file
70      filename = 'R:/Data/Corpora/01/not_bullying' + str(
71          count_total).zfill(5) + '.txt'
72      file = open(filename, 'w')
73      file.write(question_replace)
74      file.close()
75
76  conn.close()
```

A.3 Replacer helper class

```
1   import re
2   import csv
3   import yaml
4   import enchant
5   from nltk.corpus import wordnet
6   from nltk.metrics import edit_distance
7
8   ######################################################
9   ## Replacing Words Matching Regular Expressions ##
10  ######################################################
11
12  replacement_patterns = [
13      (r'(\su\s)', ' you '),
14      (r'(\su$)', ' you'),
15      (r'(^u\s)', 'you '),
16      (r'(\sur\s)', ' your '),
17      (r'(\sur$)', ' your'),
18      (r'(^ur\s)', 'your '),
19      (r'(\sr\s)', ' are '),
20      (r'(\sr$)', ' are'),
21      (r'(^r\s)', 'are '),
22      (r'(\sn\s)', ' and '),
23      (r'(\sn$)', ' and'),
24      (r'(^n\s)', 'and '),
```

```
    (r'(\sk\s)', ' ok '),
    (r'(\sk$)', ' ok'),
    (r'(^k\s)', 'ok '),
    (r'(\sy\s)', ' why '),
    (r'(\sy$)', ' why'),
    (r'(^y\s)', 'why '),
    (r'(\sw\s)', ' with '),
    (r'(\sw$)', ' with'),
    (r'(^w\s)', 'with '),
]

digit_replacement_patterns = [
    (r'(\s11\s)', ' eleven '),
    (r'(\s11$)', ' eleven'),
    (r'(^11\s)', 'eleven '),
    (r'(\s12\s)', ' twelve '),
    (r'(\s12$)', ' twelve'),
    (r'(^12\s)', 'twelve '),
    (r'(\s13\s)', ' thirteen '),
    (r'(\s13$)', ' thirteen'),
    (r'(^13\s)', 'thirteen '),
    (r'(\s14\s)', ' fourteen '),
    (r'(\s14$)', ' fourteen'),
    (r'(^14\s)', 'fourteen '),
    (r'(\s15\s)', ' fifteen '),
    (r'(\s15$)', ' fifteen'),
    (r'(^15\s)', 'fifteen '),
    (r'(\s16\s)', ' sixteen '),
    (r'(\s16$)', ' sixteen'),
    (r'(^16\s)', 'sixteen '),
    (r'(\s17\s)', ' seventeen '),
    (r'(\s17$)', ' seventeen'),
    (r'(^17\s)', 'seventeen '),
    (r'(\s18\s)', ' eighteen '),
    (r'(\s18$)', ' eighteen'),
    (r'(^18\s)', 'eighteen '),
    (r'(\s19\s)', ' nineteen '),
    (r'(\s19$)', ' nineteen'),
    (r'(^19\s)', 'nineteen '),
    (r'(\s20\s)', ' twenty '),
    (r'(\s20$)', ' twenty'),
    (r'(^20\s)', 'twenty '),
    (r'(\s21\s)', ' twenty one '),
    (r'(\s21$)', ' twenty one'),
    (r'(^21\s)', 'twenty one '),
    (r'(\s22\s)', ' twenty two '),
    (r'(\s22$)', ' twenty two'),
    (r'(^22\s)', 'twenty two '),
    (r'(\s23\s)', ' twenty three '),
    (r'(\s23$)', ' twenty three'),
    (r'(^23\s)', 'twenty three '),
    (r'(\s24\s)', ' twenty four '),
    (r'(\s24$)', ' twenty four'),
    (r'(^24\s)', 'twenty four '),
    (r'(\s25\s)', ' twenty five '),
    (r'(\s25$)', ' twenty five'),
    (r'(^25\s)', 'twenty five '),
```

```
]


class RegexpReplacer(object):

    def __init__(self, patterns=replacement_patterns):
        self.patterns = [(re.compile(regex), repl)
                         for (regex, repl) in patterns]

    def replace(self, text):
        s = text

        for (pattern, repl) in self.patterns:
            (s, count) = re.subn(pattern, repl, s)

        return s


class RegexpReplacerAge(object):

    def __init__(self, patterns=digit_replacement_patterns):
        self.patterns = [(re.compile(regex), repl)
                         for (regex, repl) in patterns]

    def replace(self, text):
        s = text

        for (pattern, repl) in self.patterns:
            (s, count) = re.subn(pattern, repl, s)

        return s


class RepeatReplacer(object):

    def __init__(self):
        self.repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)')
        self.repl = r'\1\2\3'

    def replace(self, word):
        if wordnet.synsets(word):
            return word

        repl_word = self.repeat_regexp.sub(self.repl, word)

        if repl_word != word:
            return self.replace(repl_word)
        else:
            return repl_word
```

A.4 Simple NLTK Naive Bayes Classifier

```
import collections
import nltk.metrics
import os
import random

from nltk.corpus.reader import CategorizedPlaintextCorpusReader
from nltk.classify import NaiveBayesClassifier

import stopwatch

# Time the script execution
time = stopwatch.Timer()

def word_feats(words):
    return dict([(word, True) for word in words])

# Set the working folder to the nltk_data corpora folder
os.chdir ('c:/Tools/nltk_data/corpora')

reader = CategorizedPlaintextCorpusReader('./cyberbullying_01',
                                          r'.*\.txt',
                                          cat_pattern=r'(\w+)/*')

# The questions containing bullying are the positive tests
pos_ids = reader.fileids('bullying')

# The questions not containing bullying are the negative tests
neg_ids = reader.fileids('not_bullying')

pos_feat = [(word_feats(reader.words(fileids=[f])),
             'bullying')
            for f in pos_ids]

neg_feat = [(word_feats(reader.words(fileids=[f])),
             'not_bullying')
            for f in neg_ids]

# Manually implement cross validation
# Not provided out of the box by NLTK
# Performing 10 fold cross validation

#Initialise variables to hold x-validation performance totals
pos_pre  = 0
pos_rec  = 0
pos_F    = 0
neg_pre  = 0
neg_rec  = 0
neg_F    = 0
accuracy = 0

# Variable to hold the number of folds
fold_size = 10

# Calculate the size of each fold based on the total
```

```
55 #   number of samples
56 pos_fold_size = int(len(pos_feat) / fold_size)
57 neg_fold_size = int(len(neg_feat) / fold_size)
58
59 # Iterate through each fold
60 for fold in range(fold_size):
61
62     # The start and stop index for the pos samples
63     pos_start = fold * pos_fold_size
64     pos_stop  = (fold + 1) * pos_fold_size
65
66     # The start and stop index for the neg samples
67     neg_start = fold * neg_fold_size
68     neg_stop  = (fold + 1) * neg_fold_size
69
70     # Make sure the the stop index says with-in the
71     #   bound of the size of the sample
72     if pos_stop > len(pos_feat):
73         pos_stop = len(pos_feat)
74
75     if neg_stop > len(neg_feat):
76         neg_stop = len(neg_feat)
77
78     # Create the test and train datasets
79     test  = pos_feat[pos_start:pos_stop] + \
80             neg_feat[neg_start:neg_stop]
81     train = ( (pos_feat[:pos_start] + \
82                pos_feat[pos_stop:])
83               + (neg_feat[:neg_start] + \
84                  neg_feat[neg_stop:]) )
85
86     # Create the classifier using the train dataset
87     classifier = NaiveBayesClassifier.train(train)
88
89     # The NTLK metrics package uses sets to
90     # calculate performace metrics
91     # Create empty reference and test sets
92     refsets  = collections.defaultdict(set)
93     testsets = collections.defaultdict(set)
94
95     #Populate the metrics sets with values
96     for i, (feats, label) in enumerate(test):
97         refsets[label].add(i)
98         observed = classifier.classify(feats)
99         testsets[observed].add(i)
100
101    # Calculate precision. recall, f-measure
102    #   and accuracy
103    pos_pre  += nltk.metrics.precision(refsets['bullying'],
104                                       testsets['bullying'])
105    pos_rec  += nltk.metrics.recall(refsets['bullying'],
106                                    testsets['bullying'])
107    pos_F    += nltk.metrics.f_measure(refsets['bullying'],
108                                       testsets['bullying'])
109    neg_pre  += nltk.metrics.precision(refsets['not_bullying'],
110                                       testsets['not_bullying'])
111    neg_rec  += nltk.metrics.recall(refsets['not_bullying'],
```

```
                                        testsets['not_bullying'])
    neg_F     += nltk.metrics.f_measure(refsets['not_bullying'],
                                        testsets['not_bullying'])
    accuracy  += nltk.classify.accuracy(classifier, test)

    # Show the top 5 most informative features
    classifier.show_most_informative_features(5)

time.stop()

# Print out the overall performace measures
print 'pos samples:\t',    len(pos_ids)
print 'pos precision:\t',  pos_pre / fold_size
print 'pos recall:\t',     pos_rec / fold_size
print 'pos F-measure:\t',  pos_F / fold_size
print 'pos samples:\t',    len(neg_ids)
print 'neg precision:\t',  neg_pre / fold_size
print 'neg recall:\t',     neg_rec / fold_size
print 'neg F-measure:\t',  neg_F / fold_size
print 'model accuracy:\t', accuracy / fold_size
print 'Total Time:\t',     time.elapsed
```

A.5 NLTK Naive Bayes Classifier with Feature Selection

```
import collections
import itertools
import nltk.metrics
import os
import random

from nltk.corpus.reader import CategorizedPlaintextCorpusReader
from nltk.classify import NaiveBayesClassifier

import stopwatch

# Time the script execution
time = stopwatch.Timer()

def pos_word_feats(words):
    return dict([(word, True) for word in words
                 if word in top_pw])

def neg_word_feats(words):
    return dict([(word, True) for word in words
                 if word in top_nw])

# Set the working folder to the nltk_data corpora folder
os.chdir ('c:/Tools/nltk_data/corpora')

reader = CategorizedPlaintextCorpusReader('./cyberbullying_13',
```

```
                                                r'.*\.txt',
                                                cat_pattern=r'(\w+)/*')

# The questions containing bullying are the positive tests
pos_ids = reader.fileids('bullying')

# The questions not containing bullying are the negative tests
neg_ids = reader.fileids('not_bullying')

pos_words = []
neg_words = []

for fileid in pos_ids:
    pos_words += [word for word in (reader.words(fileid))]

for fileid in neg_ids:
    neg_words += [word for word in (reader.words(fileid))]

pw_dist =  nltk.FreqDist(pos_words)
nw_dist =  nltk.FreqDist(neg_words)

top_pw = pw_dist.keys()[:int(len(pw_dist.keys())*.10)]
top_nw = nw_dist.keys()[:int(len(nw_dist.keys())*.10)]

pos_feat = [(pos_word_feats(reader.words(fileids=[f])),
             'bullying')
            for f in pos_ids]

neg_feat = [(neg_word_feats(reader.words(fileids=[f])),
             'not_bullying')
            for f in neg_ids]

...
```

From this point on the script is identical to Section A.4

A.6 Simple Scikit-Learn Classifier

```

import csv
import numpy as np
import os
import random
import stopwatch

# from operator import itemgetter

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import cross_validation
from sklearn.svm import LinearSVC
```

```
13 from sklearn import metrics
14 from sklearn.metrics import classification_report
15
16 # Time the script execution
17 time = stopwatch.Timer()
18
19 # Change folder to where the csv files are located
20 os.chdir('R:\\Corpora')
21
22 # Read in the csv file and create a cvs reader
23 cvs_file   = open('cyberbullying_13.csv')
24 csv_reader = csv.reader(cvs_file)
25
26 # Create empty lists for the labels and the data
27 target = []
28 data = []
29
30 # Read the labels and data values into lists
31 for line in csv_reader:
32     target.append(int(line[0]))
33     data.append(line[1])
34
35 # Close file
36 cvs_file.close()
37
38 # Loop 5 times to get an average representation
39 for count in range(5):
40
41     X_train, X_test, y_train, y_test = \
42               cross_validation.train_test_split(
43                   data, target, test_size=0.2,
44                   random_state=(random.randrange(1,100)))
45
46     tfidf = TfidfVectorizer()
47
48     X_train = tfidf.fit_transform(X_train)
49     X_test  = tfidf.transform(X_test)
50
51     # model = MultinomialNB().fit(X_train, y_train)
52     model = LinearSVC().fit(X_train, y_train)
53
54     y_pred = model.predict(X_test)
55
56     print 'Model Accuracy: ' \
57           + str(metrics.accuracy_score(y_test, y_pred))
58
59     print 'Classification Report:'
60     print classification_report(y_test, y_pred)
61
62     print 'Confusion Matrix:'
63     print metrics.confusion_matrix(y_test, y_pred)
64
65 time.stop()
66 print 'Total Time:\t',     time.elapsed
```

A.7 Scikit-Learn Grid Search Example

```
import csv
import numpy as np
import os
import random
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import cross_validation
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.grid_search import GridSearchCV
from sklearn.pipeline import Pipeline
from time import time

# Change folder to where the csv files are located
os.chdir('R:\\Corpora')

# Read in the csv file and create a cvs reader
cvs_file   = open('cyberbullying_01.csv')
csv_reader = csv.reader(cvs_file)

# Create empty lists for the labels and the data
target_pos = []
data_pos = []
target_neg = []
data_neg = []
target = []
data = []

# Read the labels and data values into lists
for line in csv_reader:
    target_class = int(line[0])
    if target_class == 0:
        target_pos.append(target_class)
        data_pos.append(line[1])
    else:
        target_neg.append(target_class)
        data_neg.append(line[1])

# Close file
cvs_file.close()

# Shuffle the neg data
random.shuffle(data_neg)
random.shuffle(data_pos)

print len(data_pos)
print len(data_neg)

# Ratio 3:1 required
multi = (len(data_neg)/1) / len(data_pos)
modul = (len(data_neg)/1) % len(data_pos)
```

```
54
55 data_pos_final = []
56 target_pos_final = []
57
58 for n in range(multi):
59     data_pos_final = data_pos_final + data_pos
60     target_pos_final = target_pos_final + target_pos
61
62 data_pos_final = data_pos_final + data_pos[:modul]
63 target_pos_final = target_pos_final + target_pos[:modul]
64
65 target = target_pos_final + target_neg
66 data = data_pos_final + data_neg
67
68 print len(data_pos_final)
69 print len(target)
70 print len(data)
71
72 # Create a pipeline with a transformer and a estimator
73 pipeline = Pipeline([
74     ('tfidf', TfidfVectorizer()),
75     ('clf',   LinearSVC()),
76 ])
77
78 # Define the parameter ranges
79 parameters = {
80     'tfidf__stop_words': [None, 'english'],
81     'tfidf__ngram_range': [(1, 1),
82                            (1, 2),
83                            (1, 3),
84                            (2, 2),
85                            (2, 3),
86                            (3, 3)],
87     'tfidf__norm': ['l1', 'l2'],
88 }
89
90 # Create the grid search object
91 grid_search = GridSearchCV(pipeline,
92                            parameters,
93                            cv=5)
94
95 # Fit the data and time
96 t0 = time()
97 grid_search.fit(data, target)
98 print("done in %0.3fs" % (time() - t0))
99
100 # Print the best score obtained
101 print("Best score: %0.3f" % grid_search.best_score_)
102
103 # Print the parameter values that gave the best score
104 best_parameters = grid_search.best_estimator_.get_params()
105 for param_name in sorted(parameters.keys()):
106     print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

A.8 Scikit-Learn Over Sampling Classifier

```
import csv
import numpy as np
import os
import random
import stopwatch

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import cross_validation
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.metrics import classification_report

# Time the script execution
time = stopwatch.Timer()

# Change folder to where the csv files are located
os.chdir('R:\\Corpora')

# Read in the csv file and create a cvs reader
cvs_file   = open('cyberbullying_01.csv')
csv_reader = csv.reader(cvs_file)

# Create empty lists for the labels and the data
target_pos = []
data_pos = []
target_neg = []
data_neg = []
target = []
data = []

# Read the labels and data values into lists
for line in csv_reader:
    target_class = int(line[0])
    if target_class == 0:
        target_pos.append(target_class)
        data_pos.append(line[1])
    else:
        target_neg.append(target_class)
        data_neg.append(line[1])

# Close file
cvs_file.close()

# Shuffle the neg data
random.shuffle(data_neg)
random.shuffle(data_pos)

print len(data_pos)
print len(data_neg)

# Ratio 1:1 required
multi = (len(data_neg)/1) / len(data_pos)
```

```
54 modul = (len(data_neg)/1) % len(data_pos)
55
56 data_pos_final = []
57 target_pos_final = []
58
59 for n in range(multi):
60     data_pos_final = data_pos_final + data_pos
61     target_pos_final = target_pos_final + target_pos
62
63 data_pos_final = data_pos_final + data_pos[:modul]
64 target_pos_final = target_pos_final + target_pos[:modul]
65
66 target = target_pos_final + target_neg
67 data = data_pos_final + data_neg
68
69 len(data_pos)
70 print len(target)
71 print len(data)
72
73 # Loop 5 times to get an average representation
74 for count in range(5):
75     # Close file
76     cvs_file.close()
77
78     X_train, X_test, y_train, y_test = \
79               cross_validation.train_test_split(
80                   data, target, test_size=0.2,
81                   random_state=(random.randrange(1,100)))
82
83     tfidf = TfidfVectorizer(
84           stop_words=None
85         , ngram_range=(1, 2)
86         , norm='l2'
87     )
88
89     X_train = tfidf.fit_transform(X_train)
90     X_test  = tfidf.transform(X_test)
91
92     model = LinearSVC().fit(X_train, y_train)
93
94     y_pred = model.predict(X_test)
95
96     print 'Model Accuracy: ' \
97           + str(metrics.accuracy_score(y_test, y_pred))
98
99     print 'Classification Report:'
100    print classification_report(y_test, y_pred)
101
102    print 'Confusion Matrix:'
103    print metrics.confusion_matrix(y_test, y_pred)
104
105 time.stop()
106 print 'Total Time:\t',     time.elapsed
```

A.9 Scikit-Learn Simulation Sample Script

```
import csv
import numpy as np
import os
import random
import stopwatch

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import cross_validation
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.metrics import classification_report
import stopwatch

# Time the script execution
from time import time

# Change folder to where the csv files are located
os.chdir('R:\\Corpora')

# Read in the simulation samples
sim_file   = open('sim_01.csv')
sim_reader = csv.reader(sim_file)

sim_samples = []

for new_line in sim_reader:
    sim_samples.append(new_line[0])

sim_file.close()

# Shuffle the simulation data
random.shuffle(sim_samples)

# The number of simulation samples
sim_len = len(sim_samples)
sim_size = int(sim_len / 20)

# Loop 20 time
for loop in range(20):

    t0 = time()

    # Reset empty lists for the labels and the data
    target_pos = []
    data_pos   = []
    target_neg = []
    data_neg   = []
    target     = []
    data       = []

    # Read in the csv file and create a cvs reader
    cvs_file   = open('cb_01.csv')
```

```
54      csv_reader = csv.reader(cvs_file)
55
56      # Read the labels and data values into lists
57      for line in csv_reader:
58          target_class = int(line[0])
59          if target_class == 0:
60              target_pos.append(target_class)
61              data_pos.append(line[1])
62          else:
63              target_neg.append(target_class)
64              data_neg.append(line[1])
65
66      # Close files
67      cvs_file.close()
68
69      # Shuffle the neg data
70      random.shuffle(data_neg)
71      random.shuffle(data_pos)
72
73      # Ratio 2:1 required
74      multi = (len(data_neg)/1) / len(data_pos)
75      modul = (len(data_neg)/1) % len(data_pos)
76
77      data_pos_final = []
78      target_pos_final = []
79
80      for n in range(multi):
81          data_pos_final = data_pos_final + data_pos
82          target_pos_final = target_pos_final + target_pos
83
84      data_pos_final = data_pos_final + data_pos[:modul]
85      target_pos_final = target_pos_final + target_pos[:modul]
86
87      y_train = target_pos_final + target_neg
88      X_train = data_pos_final + data_neg
89
90      tfidf = TfidfVectorizer(
91            stop_words=None
92          , ngram_range=(1, 2)
93          , norm='l2'
94      )
95
96      # Train Model
97      X_train = tfidf.fit_transform(X_train)
98      model = LinearSVC().fit(X_train, y_train)
99
100     if (loop == 19):
101         X_test  = sim_samples[sim_size * loop:sim_len]
102     else:
103         X_test = sim_samples[sim_size * loop:sim_size * (loop+1)]
104
105     X_test2 = tfidf.transform(X_test)
106     y_pred  = model.predict(X_test2)
107
108     print (time() - t0)
109
110     # Append classified samples to training data
```

```
    out_file_name_01 = 'cb_01.csv'
    out_file_01 = open(out_file_name_01, 'a')

    for n in range(len(X_test)):
        out_file_01.write('"')
        out_file_01.write(str(y_pred[n]))
        out_file_01.write('","')
        out_file_01.write(str(X_test[n]))
        out_file_01.write('"\n')
    out_file_01.close()

    # Write predictions to csv file for analysis
    out_file_name_02 = 'sim_02.csv'
    out_file_02 = open(out_file_name_02, 'a')

    for n in range(len(X_test)):
        out_file_02.write('"')
        out_file_02.write(str(y_pred[n]))
        out_file_02.write('","')
        out_file_02.write(str(X_test[n]))
        out_file_02.write('"\n')
    out_file_02.close()

# Classify hold back samples
t0 = time()

hb_file   = open('hb_01.csv')
hb_reader = csv.reader(hb_file)
hb_samples = []
for new_line in hb_reader:
    hb_samples.append(new_line[0])
hb_file.close()

hb_test = tfidf.transform(hb_samples)
hb_pred = model.predict(hb_test)

print("done in %0.3fs" % (time() - t0))

# Write hold back prediction to file for analysis
out_hb_name = 'hb_02.csv'
out_hb = open(out_hb_name, 'w')

for n in range(len(hb_samples)):
    out_hb.write('"')
    out_hb.write(str(hb_pred[n]))
    out_hb.write('","')
    out_hb.write(str(hb_samples[n]))
    out_hb.write('"\n')

out_hb.close()
```

Appendix B

Novel Method to Choose the Best Classifiers

It was mentioned in future work, in Chapter 6, that it would be beneficial if it was possible to identify the top n models without having to manually evaluate each separately. The ideas presented here are not necessarily a complete solution. They do, however, represent a possible future line of investigation. Were the ideas suggested here a proven methodology, the model evaluation sections of 5 would have read something like this.

B.1 Choosing the Best Classifiers

Having completed all NLTK and Scikit-Learn modelling, the next task was to identify which models performed the best to be further evaluated using simulation and which models, even though their performance measures are good, are to be discarded. As previously discussed, the criteria to determine the best model was that it should first maximise recall for the positive bullying class and secondly that it must also maximise recall for the negative not bullying class. As discovered in the literature review in Section 3.3, Kubat and Matwin [29] referred to this measure as the geometric mean of the accuracies measured separately on each class or as $g = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$, from now on referred to as the *g-performance*. This measure is essentially the square root of the positive class recall multiplied by the negative class recall. Also to be taken into consideration is the nature of any manipulation performed on the data, for example, over sampling or under sampling, and also the run time. With all these factors taken into consideration the top classifiers will be further evaluated in the next section by way of a simulation of live data.

It was decided that the three best models across all evaluation criteria would go forward for further examination.

B.1.1 G-Performance

The first measure evaluated to determine which models were the best was g-performance. The value for each model was calculated and a heat map generated from the results is shown in Figure B.1. It can clearly be seen that there are six clusters of results that are very similar and that all these clusters represent either over sampling of the minority class or a hybrid sampling approach.

NLTK Initial	0.7498	0.7509	0.6821	0.5740	0.7378	0.5163	0.2348
NLTK Under Sampling 1:1	0.6742	0.6563	0.6805	0.5889	0.7006	0.5413	0.2378
NLTK Under Sampling 2:1	0.7344	0.7301	0.6836	0.5787	0.7381	0.5302	0.2361
NLTK Under Sampling 3:1	0.7488	0.7537	0.6882	0.5775	0.7406	0.5197	0.2335
NLTK Over Sampling 1:1	0.7502	0.7289	0.8681	0.9225	0.7903	0.9390	0.9920
NLTK Over Sampling 2:1	0.7927	0.7870	0.8687	0.9288	0.8234	0.9448	0.9918
NLTK Over Sampling 3:1	0.7894	0.7884	0.8178	0.8385	0.8040	0.8600	0.8524
NLTK Hybrid Sampling 50:50	0.7616	0.7426	0.8547	0.9199	0.8007	0.9369	0.9918
NLTK Hybrid Sampling 60:40	0.7960	0.7851	0.8649	0.9256	0.8350	0.9418	0.9916
NLTK Hydrib Sampling 70:30	0.8145	0.8124	0.8421	0.8739	0.8338	0.8938	0.9083
Scikit NB Initial	0.3000	0.3066	0.2793	0.2933	0.3924	0.2191	0.0775
Scikit SV Initial	0.6731	0.6936	0.5531	0.4498	0.6646	0.4138	0.2278
Scikit SV Initial TF-IDF	0.4195	0.3376	0.3317	0.6809	0.6837	0.6689	0.6582
Scikit Grid Under Sampling	0.6795	0.7021	0.7190	0.7240	0.7597	0.7789	
Scikit Grid Over Sampling	0.8640	0.9357	0.9170	0.8821	0.9399	0.9508	
Scikit Grid Hybrid Sampling	0.8847	0.9240	0.8834	0.8945	0.9196	0.9210	

FIGURE B.1: Heat map showing the g-performance measurement for all models

B.1.2 Execution Time

The second evaluation criterion was the execution run time of each model. The longest run time was 27.1118 seconds whilst the fastest was 0.727 seconds. Clearly, in their raw format, these numbers are out of proportion with the g-performance measures. To resolve this all times were normalised between 0 and 1. The normalised times were then weighted to reflect, that whilst important, run time is not a deciding factor. A weighting of 0.2 was applied to reflect that the g-performance measurement was 5 times more important than the run time measure. Through examination it was found that below 5, run time was too influential, whilst above 10, it made little impact. Figure B.2 shows the resulting heat map with the fastest and slowest times identified.

It should be noted that low runtime was considered as positive implying that 0.2 is representative of the fastest time and 0.0 was the slowest time recorded. What is very clear from the heat map is that Scikit-Learn significantly outperforms the NLTK.

NLTK Initial	0.1181	0.1248	0.0623	0.0384	0.1345	0.0789	0.1200
NLTK Under Sampling 1:1	0.1650	0.1680	0.1345	0.1348	0.1715	0.1619	0.1713
NLTK Under Sampling 2:1	0.1534	0.1556	0.1202	0.1031	0.1565	0.1340	0.1503
NLTK Under Sampling 3:1	0.1395	0.1404	0.0904	0.0748	0.1498	0.1264	0.1424
NLTK Over Sampling 1:1	0.0726	0.0814	0.0169	0.0000	0.0980	0.0661	0.0944
NLTK Over Sampling 2:1	0.1004	0.1079	0.0444	0.0212	0.1189	0.0872	0.1105
NLTK Over Sampling 3:1	0.1127	0.1194	0.0554	0.0327	0.1282	0.0948	0.1168
NLTK Hybrid Sampling 50:50	0.1219	0.1266	0.0818	0.0672	0.1370	0.1159	0.1348
NLTK Hybrid Sampling 60:40	0.1213	0.1264	0.0755	0.0573	0.1363	0.1106	0.1299
NLTK Hydrib Sampling 70:30	0.1150	0.1233	0.0627	0.0448	0.1332	0.1027	0.1215
Scikit NB Initial	0.1961	0.1957	0.1939	0.1917	0.1991	0.1960	0.1976
Scikit SV Initial	0.1959	0.1961	0.1918	0.1916	0.1972	0.1962	0.1981
Scikit SV Initial TF-IDF	0.1958	0.1916	0.1875	0.1958	0.1908	0.1871	0.1875
Scikit Grid Under Sampling	0.1916	0.1953	0.1875	0.1831	0.2000	0.1959	
Scikit Grid Over Sampling	0.1748	0.1694	0.1534	0.1655	0.1577	0.1751	
Scikit Grid Hybrid Sampling	0.1768	0.1788	0.1784	0.1755	0.1868	0.1875	

FIGURE B.2: Heat map showing the normalised execution run time for all models

It is unfortunate that the execution times used here include all data reading, data writing, data manipulation and model training times. It would have been ideal if the time taken to predict each sample was available for each model. The reason the prediction time is important, is that generating the model, and all that entails, could be done on a high specification back-end server. If the work described here was deployed on a smart phone, for example, the application would probably only have to predict single samples at a time, using a previously generated model deployed using Pickle. From work in Section 5.7, classification times for a single sample were in the region of 0.0013 seconds for an NLTK model, and 0.000034 seconds for a scikit-learn model. Both of these times are so fast as to suggest that the end user would not notice the difference, making this measurement redundant. However, these times were achieved on a high-end laptop with significantly more processing power than the average smart phone. If this measure was to be included in any novel model evaluation method, the values used would have to be based on actual times from the targeted platform.

B.1.3 Data Manipulation

The final evaluation criterion to be considered was the manipulation performed on the data. It was important to acknowledge the fact that the sampling applied directly affects the performance of the model and some counter balance, or penalty, should be

applied. Having made the decision that the models where data manipulation was used would be penalised it was decided that the maximum penalty would be 0.1 or that the g-performance was 10 times more important than any data manipulation penalty. Using 0.1 as the maximum value, the different class ratios were penalised as shown in Table B.1. The resulting heat map of penalties for all models run is shown in Figure B.3

TABLE B.1: Table showing the penalties applied to the various ratios when sampling of data was used

Class Ratio	**Penalty**
3:1	0.05
2:1	0.075
1:1	0.1
70:30	0.05
60:40	0.075
50:50	0.1
None	0

NLTK Initial	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NLTK Under Sampling 1:1	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
NLTK Under Sampling 2:1	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750
NLTK Under Sampling 3:1	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500
NLTK Over Sampling 1:1	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
NLTK Over Sampling 2:1	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750
NLTK Over Sampling 3:1	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500
NLTK Hybrid Sampling 50:50	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
NLTK Hybrid Sampling 60:40	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750	0.0750
NLTK Hydrib Sampling 70:30	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500	0.0500
Scikit NB Initial	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Scikit SV Initial	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Scikit SV Initial TF-IDF	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Scikit Grid Under Sampling	0.0500	0.0750	0.1000	0.0500	0.0750	0.1000	
Scikit Grid Over Sampling	0.0500	0.0750	0.1000	0.0500	0.0750	0.1000	
Scikit Grid Hybrid Sampling	0.0500	0.0750	0.1000	0.0500	0.0750	0.1000	

FIGURE B.3: Heat map showing the penalties applied to each model where sampling of data was involved

B.1.4 Best Models

The best models were then calculated as *Best Model = Max(G-Performance + Run Time - Ratio Penalty)*. So models with high g-performance but with low run time weighting and low data manipulation penalties will be chosen. The final heat map, identifying the three best models is shown in Figure B.4. The three models are:

1. NLTK Naive Bayes Hybrid Sampling with 60:40 ratio using tri-grams excluding stop words

2. Scikit-Learn Support Vector Hybrid Sampling with 60:40 ratio using uni-grams and bi-grams including stop words

3. Scikit-Learn Naive Bayes Over Sampling with 2:1 ratio using uni-grams, bi-grams and tri-grams including stop words

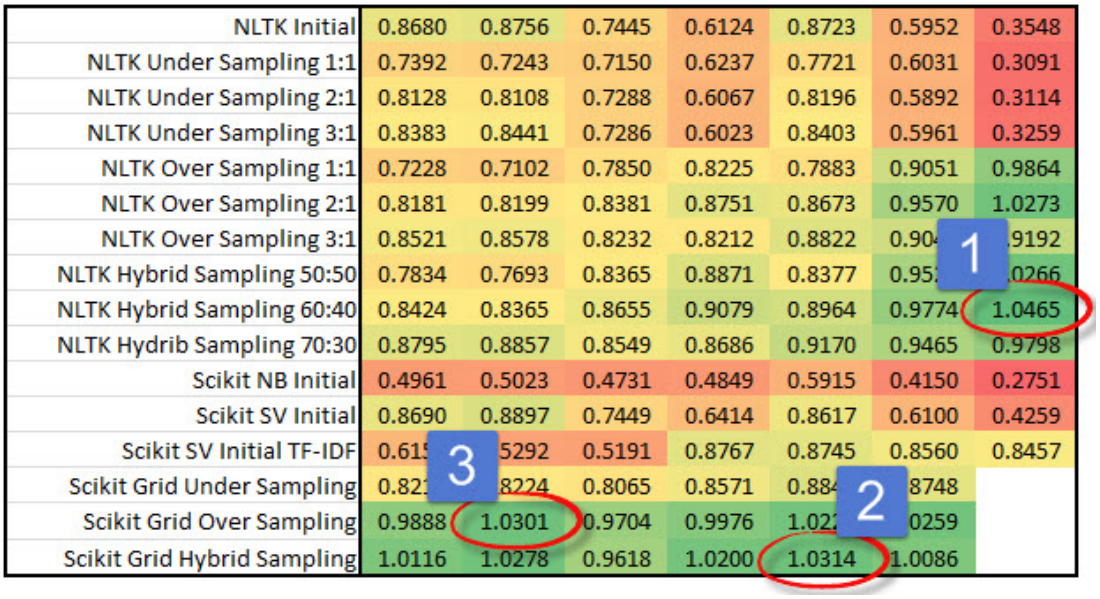

FIGURE B.4: Heat map the final results and identifying the best models

It is clear that under sampling never approached the performance levels of over sampling and hybrid sampling and that the middle sampling ratios always appeared to give more balanced results then the other ratios.

These three models will now be further examined in the next section.

B.2 Applying the Best Classifiers

Now that the best three classifiers have been determined the final analysis task is to determine which of the three perform best in a simulation of a real life scenario.

B.2.1 Scenario Overview

A classifier is to be developed that will batch process a corpus of previously unseen questions, called `dataset_02`, to determine whether or not they are bullying in nature.

The predictions, bullying or not bullying, from each batch of questions processed are then fed back into the training dataset and a new classifier model is generated. The next batch of questions is then classified. This process continues until all sample have been processed.

A third test dataset, `dataset_03`, never used at any point in the development of any of the classifiers will be used to gauge the performance of each of the models before and after the batch processing in order to determine which model produced the best classifier.

The steps to achieve this can be summarised as follows:

- Prepare the two datasets put aside for this testing in Chapter 4 by generating NLTK or Scikit-Learn corpora as required.
- Classify both datasets using the three top models and record the performance results.
- Simulate the growth in each of the models using `dataset_02`.
- Re-evaluate `dataset_03` again using each of the final models and see if the models performance has increased or decreased. Also compare the complete set of before and after results from `dataset_02`.

B.2.2 Data Preparation

At the end of Chapter 4 `dataset_02` and `dataset_03` had not been preprocessed or stored in the correct corpora format for NLTK or Scikit-Learn. In a real life scenario, the processing required to do this would all have to happen as the data is received. However, in this simulation all preparatory work was performed upfront. Each dataset was first preprocessed and cleansed using the same Python script as described in Section 4.5. Next, each datasets was saved in the basic NLTK corpus format and renamed as `simulation_01` and `hold_back_01`. The NLTK model that was to be tested required the dataset in tri-gram format with stop words removed producing two further NLTK corpora called `simulation_13` and `hold_back_13`. Both of the Scikit-Learn models used the NLTK `simulation_01` and `hold_back_01` corpora but with all the samples presented in a single comma separated file format as before.

B.2.3 Initial Analysis of Models

Once each dataset had been prepared they were submitted to their respective model for analysis. The results from the two Scikit-Learn models will be discussed shortly but first

the performance of the NLTK model is analysed. It was immediately noticed that the number of examples presented to the model was significantly fewer than expected. Also, the performance of the model was very poor in comparison to the performance achieved by the training model.

Originally there were 87,205 and 11,193 samples respectively in the simulation and hold back datasets. However, after processing was applied to these datasets to remove stop words and then convert them to tri-grams, it resulted in only 53,804 and 6,976 samples submitted to the NLTK model. A reduction in sample numbers of approximately 38%. On analysis, it was clear that the cause of this was the conversion to tri-grams. When a sample question had less than three tokens a tri-gram could not be generated resulting in an empty sting. In a production environment the loss of 38% of all samples because of a processing step would not be an acceptable solution.

The other issue with the NLTK model was its predictive performance on the previously unseen data in the simulation and hold back datasets. After manual classification of the training dataset, approximately 17% of the samples were identified as bullying. After training, the NLTK classifier could nearly identify 100% of all positive, bullying, and negative, not bullying, samples. Although a classifier is never expected to perform as well on unseen data only 2.5% of the hold back samples and 2.4% of the simulation samples were identified as bullying. Although further more detailed analysis would be required to confirm this, a quick review of the tokens in the samples that were identified as bullying suggested that the NLTK model had been over fitted to the training data and did not generalise well to unseen data.

Based on these finding the NLTK model was dismissed as a viable solution. No further testing using this model was performed.

Because the Scikit-Learn TF-IDF vectorise class also included uni-gram tokens when generating bi-grams of a sample and both uni-grams and bi-grams when generating tri-grams these models did not suffer the same loss of samples that the NLTK model did. The Scikit-Learn hybrid sampling model classified 1,878 of the hold back samples and 14,172 of the simulation samples, equivalent to 16.8% and 16.3% respectively, as bullying. These numbers are very consistent with the percentage of bullying samples seen in the training dataset. The over sampling model did not appear to have performed as well identifying 1,128, 10.1% and 8,752, 10.0% samples in hold back and simulation datasets. Although not appearing to have performed as well as the hybrid sampling model, it did still appear to have performed consistently across the two datasets.

B.2.4 Batch Processing of Samples

Each Scikit-Learn models was then further examined using the simulation dataset as follows:

- The simulation dataset of 87,205 samples was divided into twenty distinct datasets.
- Then for each of these datasets:
  - The training dataset was loaded
  - The training dataset was sampled as required by the model, either hybrid sampling or over sampling
  - A classifier model was created using the sampled training data
  - The simulation dataset was classified using the model
  - The classified simulation samples were appended to the training dataset
  - Process repeated until there were no further simulation datasets
- Once all the simulation samples are included in the final model, the hold back dataset was classified and the results were written to a file.

This simple process was repeated for hybrid sampling and over sampling. In addition to writing the hold back classification results to file, each of the results for the classification of the simulation samples were also written to file. The results of this batch processing and classification of the hold back data was then analysed.

B.2.5 Analysis of Batch Processing Results

The first thing to note was that the increase in processing time as the size of the training set grew appeared linear. This linearity would make it possible to predict the time required for larger datasets. It was also observed that there was a significant difference between the two different types of sampling used with over sampling taking more time. This difference is probably down to the fact that uni-grams, bi-grams and tri-grams are used with over sampling opposed compared to just uni-grams and bi-grams with hybrid sampling. The processing time results can be seen in Figure B.5 starting with the original training dataset of 9,543 samples for the first iteration all the way to 96,748 samples for the last.

The hybrid sampling model was the first to be examined. The initial model developed using only the training dataset predicted 1,878 of the hold back dataset samples as

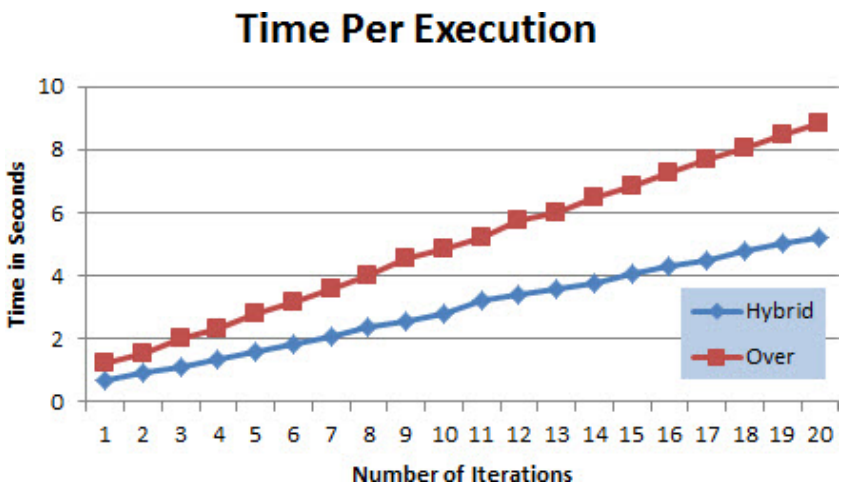

FIGURE B.5: Chart plotting the different processing time when batch processing using hybrid and over sampling

bullying. The model at the completion of the batch processing experiment predicted 2,449 bullying samples. This represents a 30.4% increase in the number of samples classified as bullying. On closer examination it was seen that of the 1,878 original bullying predictions only 1,685 were again classified as bullying, 193 bullying samples were reclassified as not bullying and 764 not bullying samples were now classified as bullying.

The over sampling model was then analysed. The initial model developed using only the training dataset predicted 1,128 of the hold back dataset samples as bullying. The model at the completion of the batch processing experiment predicted 1,218 bullying samples. This represents a 8.0% increase in the number of samples classified as bullying, a much smaller increase than seen with the hybrid sampling classifier. On closer examination it was seen that of the 1,128 original bullying predictions 898 were again classified as bullying, 230 bullying samples were reclassified as not bullying and 320 not bullying samples were now classified as bullying.

As a percentage, just under 90% of the samples using hybrid sampling were predicted as bullying before and after the batch processing but only 80% of the over sampling predictions remained the same. Looking at the number of new bullying classifications, as a percentage of the original number, the hybrid sampling was again the better performer at an additional 40.6% new bullying classifications compared to 28.4%. Finally, only 10.3% of samples initially classified as bullying using hybrid sampling were reclassified as not bullying compared to 20.4%. These would all suggest that the hybrid sampling model performed better by maintaining the highest percentage of predicted bullying samples, by adding a higher percentage of new bullying samples and also by reclassifying fewer bullying samples as not bullying.

It would be impossible, time wise, to re-examine all 11,193 samples in the hold back dataset from both sampling methods so the samples that changed classification were all reviewed and manually classified to determine if any patterns existed in the way samples were reclassified.

Hybrid sampling was analysed first. In total 193 samples changed from bullying to not bullying and 764 changed from not bullying to bullying. Each of the 957 samples, which represents 8.5% of the total amount, was manually classified as either bullying or not bullying using the criteria defined in Section 4.3. The percentage of each class that was either correctly or incorrectly reclassified was calculated. The results are shown in Figure B.6.

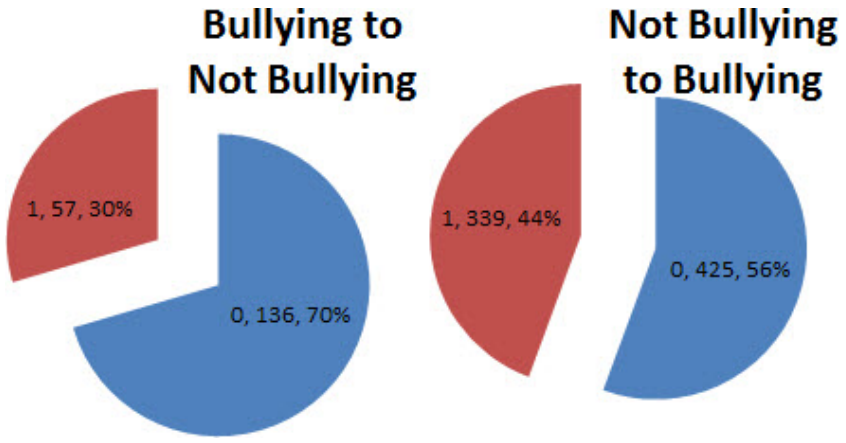

FIGURE B.6: Analysis of hybrid sampling prediction performance on the hold back dataset after batch processing of simulation data

Of the 193 samples that were reclassified from bullying to not bullying, shown on the left hand side of Figure B.6, it was determined that 136 or 70% of the samples were incorrectly reclassified and were, in fact, originally correctly classified as bullying. Of the 764 samples that were reclassified as bullying it was seen that 425, 56%, were correctly reclassified as bullying, but it was also shown that 339 or 44% of the samples should not have been reclassified.

When the analysis of the over sampling was complete, a similar picture was seen as shown in Figure B.7.

On examination of the numbers shown in Figures B.6 and B.7 it would appear, when looking past the initial earlier analysis which suggested that hybrid sampling was the better performing model, that neither model appears very stable with a high percentage of incorrect predictions. However, given that an acceptable number of false positives, samples incorrectly predicted as bullying, with a high positive class recall value is the primary criteria to which a model must perform, the hybrid sampling method is clearly the better model.

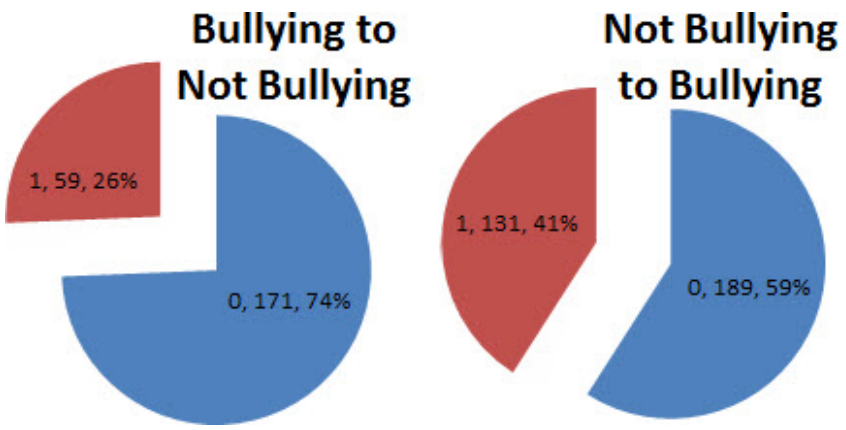

FIGURE B.7: Analysis of over sampling prediction performance on the hold back dataset after batch processing of simulation data

To complete the analysis of the Scikit-Learn hybrid sampling model, 100 of the 1686 samples predicted as bullying that did not change classification after the batch processing of the additional samples were manually classified. Of the 100 samples, 87% were correctly classified as bullying. The original model had an positive recall value of 94.9% so although this recall value is not as good, the overall performance of the Scikit-Learn hybrid sampling model is very satisfactory.

Bibliography

[1] stopbullying.gov. What is bullying? a new uniform definition for research, 2014-02-10. URL http://www.stopbullying.gov/blog/2014/02/10/what-bullying-new-uniform-definition-research.

[2] stopbullying.gov. What is cyberbullying, 2012-03-07. URL http://www.stopbullying.gov/cyberbullying/what-is-it/index.html.

[3] Fiachra Ó Cionnaith. Third suicide in weeks linked to cyberbullying, 2012. URL http://www.irishexaminer.com/ireland/third-suicide-in-weeks-linked-to-cyberbullying-212271.html.

[4] Ralph Riegel. Cyber-bullies claimed lives of five teens, 25 January 2013 10:30. URL http://www.herald.ie/news/cyberbullies-claimed-lives-of-five-teens-29043544.html.

[5] Laura Smith-Spark. Hanna smith suicide fuels calls for action on ask.fm cyberbullying, 2013-08-09. URL http://www.cnn.com/2013/08/07/world/europe/uk-social-media-bullying/index.html.

[6] Pew Research Center. Teens fact sheet. URL http://www.pewinternet.org/fact-sheets/teens-fact-sheet/.

[7] Heidi Seybert. Internet use in households and by individuals in 2012, 2012-12-13. URL http://epp.eurostat.ec.europa.eu/cache/ITY_OFFPUB/KS-SF-12-050/EN/KS-SF-12-050-EN.PDF.

[8] Giovanna Mascheroni and Kjartan Ólafsson. Net children go mobile: Risks and opportunities., 2014-05. URL http://www.netchildrengomobile.eu/reports/.

[9] Ditch the Label. The annual bullying survey 2014. 2014.

[10] Amanda Lenhart, Mary Madden, Aaron Smith, Kristen Purcell, Kathryn Zickuhr, and Lee Rainie. Teens, kindness and cruelty on social network sites, 2011-11-09. URL http://www.pewinternet.org/2011/11/09/teens-kindness-and-cruelty-on-social-network-sites/.

[11] D Colton and M. Hofmann. Automated detection of cyberbullying. *The IT&T 13th International Conference on Information Technology and Telecommunication*, pages 21--28, 2014.

[12] Webwise. Ask.fm: A guide for parents and teachers. `http://www.webwise.ie/AskfmGuide.shtm`, 2013. Visited November 29, 2013.

[13] Ask.fm. Safety and security essentials. `http://ask.fm/about/policy/safety-security-essentials`, 2014. Visited July 10, 2014.

[14] Ask.fm. Privacy policy. `http://ask.fm/about/policy/privacy-policy`, 2014. Visited July 10, 2014.

[15] Ask.fm. Terms of service. `http://ask.fm/about/policy/terms-of-service`, 2014. Visited July 10, 2014.

[16] Ask.fm. Abuse policy. `http://ask.fm/about/safety/abuse-policy`, 2014. Visited July 10, 2014.

[17] Ask.fm. Dos and don'ts. `http://ask.fm/about/safety/dos-and-donts`, 2014. Visited July 10, 2014.

[18] Ask.fm. Faqs for parents. `http://ask.fm/about/safety/faqs-for-parents`, 2014. Visited July 10, 2014.

[19] Ask.fm. User guidance. `http://ask.fm/about/safety/user-guidance`, 2014. Visited July 10, 2014.

[20] Ask.fm. Letter from safety expert. `http://ask.fm/about/safety/letter-from-safety-expert`, 2014. Visited July 10, 2014.

[21] Quinta Group. Python at google. URL `http://quintagroup.com/cms/python/google`.

[22] The official home of the python programming language. URL `https://www.python.org/`.

[23] Natural language toolkit. URL `http://www.nltk.org/`.

[24] George A. Miller. WordNet: A lexical database for english. 38(11):39–41, 1995-11. ISSN 0001-0782. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825--2830, 2011.

[26] Allen Downey. *Think Python*. Green Tea Press, 2012. ISBN 144933072X. URL http://greenteapress.com/thinkpython/.

[27] Jacob Perkins. *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing, 2010. ISBN 1849513600, 9781849513609.

[28] Raúl Garreta and Guillermo Moncecchi. *Learning scikit-learn : Machine Learning in Python*. Packt Publishing, 2013. ISBN 1783281936.

[29] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, page 179–186. Morgan Kaufmann, 1997.

[30] Vinita Nahar, Xue Li, and Chaoyi Pang. A step towards combating cyberbullying: Automated detection. pages 54--73, 2013-05-23.

[31] M. Dadvar, F. M. G. de Jong, R. J. F. Ordelman, and R. B. Trieschnigg. Improved cyberbullying detection using gender information, 2012-02-23. URL http://eprints.eemcs.utwente.nl/21608/.

[32] April Kontostathis, Kelly Reynolds, Andy Garron, and Lynne Edwards. Detecting cyberbullying: Query terms and techniques. In *Proceedings of the 5th Annual ACM Web Science Conference*, WebSci '13, page 195–204. ACM, 2013. ISBN 978-1-4503-1889-1. doi: 10.1145/2464464.2464499. URL http://doi.acm.org/10.1145/2464464.2464499.

[33] U.S. Department of Health and Human Services. What is bullying, bullying definition. http://www.stopbullying.gov/what-is-bullying/definition/index.html, 2013. Visited October 16, 2013.

[34] Jun-Ming Xu, Xiaojin Zhu, and Amy Bellmore. Fast learning for sentiment analysis on bullying. In *Proceedings of the First International Workshop on Issues of Sentiment Discovery and Opinion Mining*, WISDOM '12, page 10:1–10:6. ACM, 2012. ISBN 978-1-4503-1543-2. doi: 10.1145/2346676.2346686. URL http://doi.acm.org/10.1145/2346676.2346686.

[35] John Archer and Sarah M Coyne. An integrated review of indirect, relational, and social aggression. 9(3):212--230, 2005. ISSN 1088-8683. doi: 10.1207/s15327957pspr0903_2.

[36] Todd D. Little, Christopher C. Henrich, Stephanie M. Jones, and Patricia H. Hawley. Disentangling the "whys" from the "whats" of aggressive behaviour. 27(2):122--133, 2003-03-01. ISSN 0165-0254, 1464-0651. doi: 10.1080/01650250244000128. URL http://jbd.sagepub.com/content/27/2/122.

[37] Karen Nylund, Amy Bellmore, Adrienne Nishina, and Sandra Graham. Subtypes, severity, and structural stability of peer victimization: what does latent class analysis say? 78(6):1706--1722, 2007-12. ISSN 0009-3920. doi: 10.1111/j.1467-8624.2007.01097.x.

[38] Vinita Nahar, Xue Li, and Chaoyi Pang. An effective approach for cyberbullying detection. 3(5):238--247, 2013-05. URL `http://www.jcisme.org/paperInfo.aspx?PaperID=13552`.

[39] Maral Dadvar, Roeland Ordelman, Franciska de Jong, and Dolf Trieschnigg. Towards user modelling in the combat against cyberbullying. In Gosse Bouma, Ashwin Ittoo, Elisabeth Métais, and Hans Wortmann, editors, *Natural Language Processing and Information Systems*, number 7337 in Lecture Notes in Computer Science, pages 277--283. Springer Berlin Heidelberg, 2012-01-01. ISBN 978-3-642-31177-2, 978-3-642-31178-9. URL `http://link.springer.com/chapter/10.1007/978-3-642-31178-9_34`.

[40] Maral Dadvar, Dolf Trieschnigg, Roeland Ordelman, and Franciska de Jong. Improving cyberbullying detection with user context. In Pavel Serdyukov, Pavel Braslavski, Sergei O. Kuznetsov, Jaap Kamps, Stefan Rüger, Eugene Agichtein, Ilya Segalovich, and Emine Yilmaz, editors, *Advances in Information Retrieval*, number 7814 in Lecture Notes in Computer Science, pages 693--696. Springer Berlin Heidelberg, 2013-01-01. ISBN 978-3-642-36972-8, 978-3-642-36973-5. URL `http://link.springer.com/chapter/10.1007/978-3-642-36973-5_62`.

[41] Karthik Dinakar, Roi Reichart, and Henry Lieberman. Modeling the detection of textual cyberbullying. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011-06-07. URL `http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/3841`. The scourge of cyberbullying has assumed alarming proportions with an ever-increasing number of adolescents admitting to having dealt with it either as a victim or as a bystander. Anonymity and the lack of meaningful supervision in the electronic medium are two factors that have exacerbated this social menace. Comments or posts involving sensitive topics that are personal to an individual are more likely to be internalized by a victim, often resulting in tragic outcomes. We decompose the overall detection problem into detection of sensitive topics, lending itself into text classification sub-problems. We experiment with a corpus of 4500 YouTube comments, applying a range of binary and multiclass classifiers. We find that binary classifiers for individual labels outperform multiclass classifiers. Our findings show that the detection of textual cyberbullying can be tackled by building individual topic-sensitive classifiers.

[42] M. Rybnicek, R. Poisel, and S. Tjoa. Facebook watchdog: A research agenda for detecting online grooming and bullying activities. In *2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2854--2859, 2013-10. doi: 10.1109/SMC.2013.487.

[43] Karthik Dinakar, Birago Jones, Catherine Havasi, Henry Lieberman, and Rosalind Picard. Common sense reasoning for detection, prevention, and mitigation of cyberbullying. 2(3):18:1–18:30, 2012-09. ISSN 2160-6455. doi: 10.1145/2362394.2362400. URL `http://doi.acm.org/10.1145/2362394.2362400`.

[44] Ying Chen, Yilu Zhou, Sencun Zhu, and Heng Xu. Detecting offensive language in social media to protect adolescent online safety. pages 71--80. IEEE, 2012-09. ISBN 978-1-4673-5638-1, 978-0-7695-4848-7. doi: 10.1109/SocialCom-PASSAT.2012.55. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6406271`.

[45] Jun-Ming Xu, Kwang-Sung Jun, Xiaojin Zhu, and Amy Bellmore. Learning from bullying traces in social media. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, page 656–666. Association for Computational Linguistics, 2012. ISBN 978-1-937284-20-6. URL `http://dl.acm.org/citation.cfm?id=2382029.2382139`.

[46] Nancy Willard. Cyberbullying and cyberthreats: Effectively managing internet use risks in schools, 2006. URL `http://www.occhd.org/system/files/1041/original/Cyberbullying_and_Cyberthreats.pdf`.

[47] Nancy E Willard. *Cyberbullying and cyberthreats: responding to the challenge of online social aggression, threats, and distress*. Research Press, 2007. ISBN 0878225374 9780878225378.

[48] Ptaszynski, Michal, Pawel Dybala, Tatsuaki Matsuba, Fumito Masui, Rafal Rzepka, and Kenji Araki. Machine learning and effect analysis against cyber-bullying. 2010. URL `http://www.academia.edu/2660807/In_the_Service_of_Online_Order_Tackling_Cyber-Bullying_with_Machine_Learning_and_Affect_Analysis`.

[49] Ministry of Education, Culture, Sports, Science and Technology. *'Netto jou no ijime' ni kansuru taiou manyuaru jirei shuu (gakkou, kyouin muke) ["Bullying on the net" Manual for handling and the collection of cases (directed to school teachers)] (in Japanese)*. Ministry of Education, Culture, Sports, Science and Technology, 2008.

[50] Brian O'Neill and Thuy Dinh. Cyberbullying among 9-16 year olds in ireland. 2013-02-04. URL `http://arrow.dit.ie/cserrep/31`.

[51] S. Aggarwal, M. Burmester, P. Henry, L. Kermes, and J. Mulholland. Anti-cyberstalking: the predator and prey alert (PAPA) system. In *First International Workshop on Systematic Approaches to Digital Forensic Engineering, 2005*, pages 195--205, 2005-11. doi: 10.1109/SADFE.2005.2.

[52] P. Elzinga, K.E. Wolff, and J. Poelmans. Analyzing chat conversations of pedophiles with temporal relational semantic systems. In *Intelligence and Security Informatics Conference (EISIC), 2012 European*, pages 242--249, 2012-08. doi: 10.1109/EISIC.2012.12.

[53] Zellig Harris. Distributional structure. *Word*, 10(23):146--162, 1954.

[54] Dawei Yin, Brian D. Davison, Zhenzhen Xue, Liangjie Hong, April Kontostathis, and Lynne Edwards. Detection of harassment on web 2.0. 2009-01-01.

[55] Fundación Barcelona Media (FBM). *CAW 2.0 Training Datasets*. 2009. URL `http://caw2.barcelonamedia.org/?page_id=98`.

[56] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[57] Julie Lovins. Development of a stemming algorithm. 11, 1968. URL `http://stinet.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0735504`.

[58] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011. ISBN 1107015359, 9781107015357.

[59] Andrew Ortony, Gerald L. Clore, and Mark A. Foss. The referential structure of the affective lexicon. 11(3):341–364, 1987. ISSN 1551-6709. doi: 10.1207/s15516709cog1103_4. URL `http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1103_4/abstract`.

[60] April Kontostathis, Lynne Edwards, and Amanda Leatherman. ChatCoder: Toward the tracking and categorization of internet predators. In *PROC. TEXT MINING WORKSHOP 2009 HELD IN CONJUNCTION WITH THE NINTH SIAM INTERNATIONAL CONFERENCE ON DATA MINING (SDM 2009). SPARKS, NV. MAY 2009.*, 2009.

[61] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. 11(1):10–18, 2009-11. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL `http://doi.acm.org/10.1145/1656274.1656278`.

[62] Perverted-justice.com - the largest and best anti-predator organization online. URL `http://www.perverted-justice.com/index.php`.

[63] Jason Bengel, Susan Gauch, Eera Mittur, and Rajan Vijayaraghavan. ChatTrack: Chat room topic detection using classification. In *In 2nd Symposium on Intelligence and Security Informatics (in review*, page 266–277, 2004.

[64] K. Reynolds, A. Kontostathis, and L. Edwards. Using machine learning to detect cyberbullying. In *2011 10th International Conference on Machine Learning and Applications and Workshops (ICMLA)*, volume 2, pages 241--244, 2011-12. doi: 10.1109/ICMLA.2011.152.

[65] Zhi Xu and Sencun Zhu. Filtering offensive language in online communities using grammatical relations. In *Proceedings of The Seventh Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS'10)*, 2010.

[66] Casey Whitelaw, Navendu Garg, and Shlomo Argamon. Using appraisal groups for sentiment analysis. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, page 625–631. ACM, 2005. ISBN 1-59593-140-6. doi: 10.1145/1099554.1099714. URL `http://doi.acm.org/10.1145/1099554.1099714`.

[67] Burr Settles. Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, page 1467–1478. Association for Computational Linguistics, 2011. ISBN 978-1-937284-11-4. URL `http://dl.acm.org/citation.cfm?id=2145432.2145588`.

[68] T. L. Griffiths and M. Steyvers. Finding scientific topics. 101:5228--5235, 2004.

[69] Shlomo Argamon, Moshe Koppel, Jonathan Fine, and Anat Rachel Shimoni. Gender, genre, and writing style in formal written texts. 23:321–346, 2003.

[70] June F. Chisholm. Cyberspace violence against girls and adolescent females. 1087 (1):74–89, 2006. ISSN 1749-6632. doi: 10.1196/annals.1385.022. URL `http://onlinelibrary.wiley.com/doi/10.1196/annals.1385.022/abstract`.

[71] Vinita Nahar, Sayan Unankard, Xue Li, and Chaoyi Pang. Sentiment analysis for effective detection of cyber bullying. In Quan Z. Sheng, Guoren Wang, Christian S. Jensen, and Guandong Xu, editors, *Web Technologies and Applications*,

number 7235 in Lecture Notes in Computer Science, pages 767--774. Springer Berlin Heidelberg, 2012-01-01. ISBN 978-3-642-29252-1, 978-3-642-29253-8. URL http://link.springer.com/chapter/10.1007/978-3-642-29253-8_75.

[72] Daegon Cho, Soodong Kim, and A. Acquisti. Empirical analysis of online anonymity and user behaviors: the impact of real name policy. In *2012 45th Hawaii International Conference on System Science (HICSS)*, pages 3041--3050, 2012-01. doi: 10.1109/HICSS.2012.241.

[73] Choe Sang-hun. South korean court overturns online name verification law. 2012-08-23. ISSN 0362-4331. URL http://www.nytimes.com/2012/08/24/world/asia/south-korean-court-overturns-online-name-verification-law.html.

[74] Gary Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? In Robert Stahlbock, Sven F. Crone, and Stefan Lessmann, editors, *DMIN*, pages 35--41. CSREA Press, 2007-12-20. ISBN 1-60132-031-0. URL http://dblp.uni-trier.de/db/conf/dmin/dmin2007.html#WeissMZ07.

[75] RULEQUEST RESEARCH. Data mining tools see5 and c5.0, 2013-03.

[76] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0.

[77] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. 16(1):321–357, 2002-06. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622407.1622416.

[78] Claire Cardie. Improving minority class prediction using case-specific feature weights. In *Proceedings of the Fourteenth International Conference on Machine Learning*, page 57–65. Morgan Kaufmann, 1997.

[79] Philip K. Chan and Salvatore J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, page 164–168. AAAI Press, 1998.

[80] V. García, J. S. Sánchez, R. A. Mollineda, R. Alejo, and J. M. Sotoca. The class imbalance problem in pattern classification and learning. 2007.

[81] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009. URL http://www.nltk.org/book/.