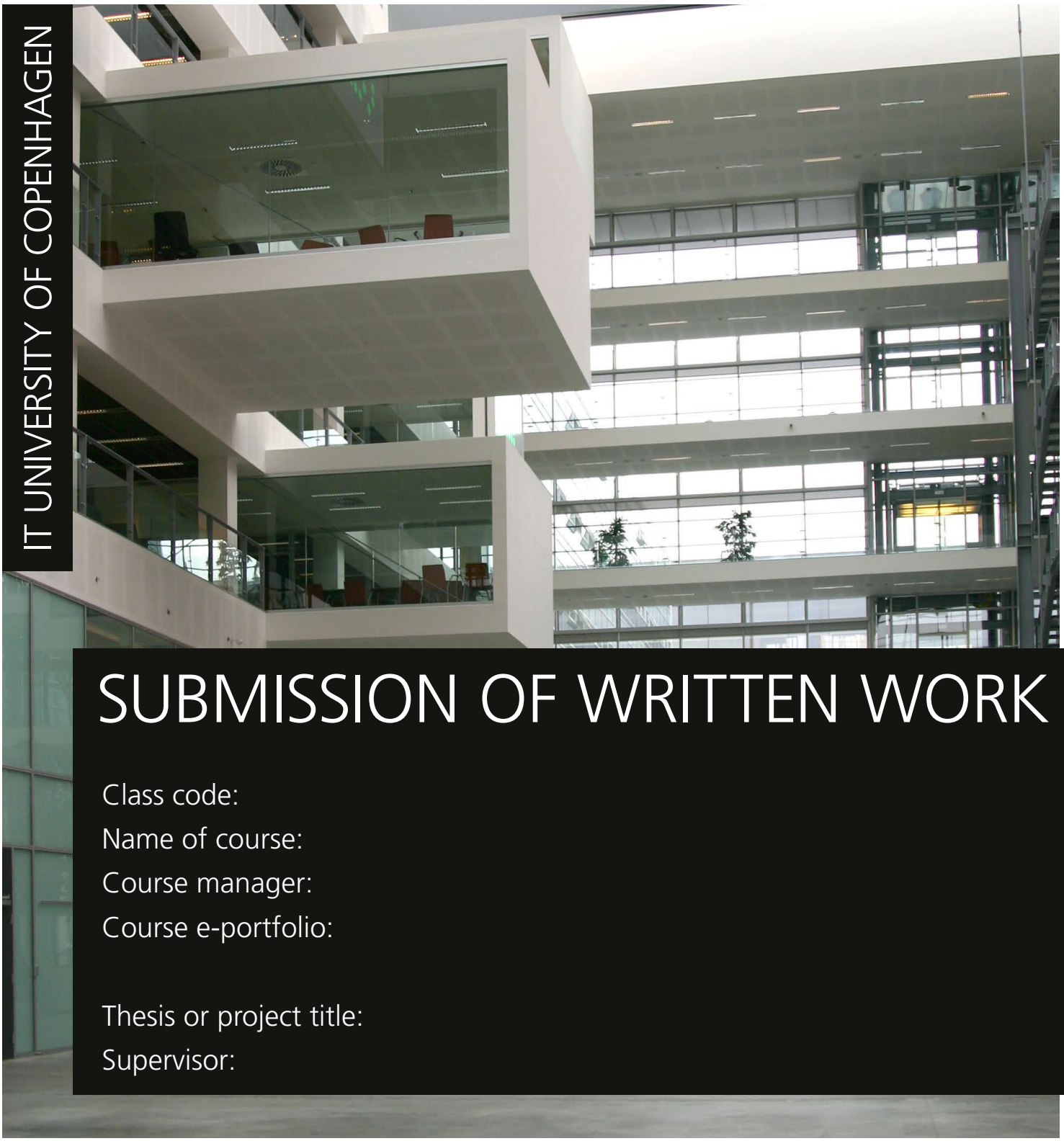
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course:

Course manager:

Course e-portfolio:

Thesis or project title:

Supervisor:

Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1. _____	_____	_____@itu.dk
2. _____	_____	_____@itu.dk
3. _____	_____	_____@itu.dk
4. _____	_____	_____@itu.dk
5. _____	_____	_____@itu.dk
6. _____	_____	_____@itu.dk
7. _____	_____	_____@itu.dk
8. _____	_____	_____@itu.dk

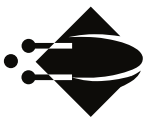
Multilingual Hate Speech Detection

Detecting the Types and Targets of Offensive Language in English and Danish Social Media Data

Gudbjartur Sigurbergsson - gusi@itu.dk

Advisor: Leon Derczynski

Submitted: June 2019



IT University
of Copenhagen

Abstract

The presence of offensive language on social media platforms and the implications this poses is becoming a major concern in modern society. Given the enormous amount of content created every day, automatic methods are required to detect and deal with this type of content. Until now, most of the research has focused on solving the problem for the English language, while the problem is obviously multilingual in nature.

In our work, we develop a Danish dataset for the task of offensive language detection. It contains user generated comments from various social media platforms, and to our knowledge, it is the first of its kind. Our dataset is annotated to capture the types and target of offensive language using the annotation schema presented in [41]. We, furthermore, publish a Danish hate speech lexicon, also the first of its kind. Finally, we develop effective automatic methods for the detection of offensive language using machine learning and natural language processing techniques. Our methods are designed to work well for both the English and the Danish language, and capture the types and targets of offensive language, making them effective in the detection of different types of offensive language such as hate speech and cyberbullying events.

Acknowledgements

First and foremost I would like to thank my supervisor, Leon Derczynski, for being the best mentor a master's student could ask for. He has been a great source of inspiration and has constantly brought new and interesting ideas to the table. Without his help this thesis would not have been possible.

Special thanks to my dear friends Styrmir Svavarsson, Herdís Arngrímsdóttir and Oddur Kristjánsson, and my sister Birna Særós Sigurbergsdóttir, for the exceptional feedback, source of motivation and countless invaluable conversations.

Last but not least, I would like to thank my parents for always believing in me and the constant support throughout the years.

Signature

Guðbjartur Sigurbergsson June 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Offensive language in user-generated content on online platforms and its implications has been getting a lot of attention over the last couple of years. This interest is sparked by the fact that many of the online social media platforms have come under scrutiny on how this type of content should be detected and dealt with. It is, however, far from trivial to deal with this type of language directly due to the gigantic amount of user-generated content created every day. For this reason, automatic methods are required, using natural language processing (NLP) and machine learning techniques.

Types of offensive language. Offensive language varies a lot, ranging from simple profanity to much more severe types of language. One of the more troublesome types of language is hate speech and the presence of hate speech on social media platforms has been shown to be in correlation with hate crimes in real life settings [25]. It can be quite hard to distinguish between generally offensive language and hate speech as few universal definitions exist [10]. There does, however, seem to be

a general consensus that hate speech can be defined as language that targets a group with the intent to be harmful or to cause social chaos. This targeting is usually done on the basis of some characteristics such as race, color, ethnicity, gender, sexual orientation, nationality or religion [36]. In section 2.2, hate speech is defined in more detail. Offensive language, on the other hand, is a more general category containing any type of profanity or insult. Hate speech can, therefore, be classified as a subset of offensive language. In [41], effective guidelines are proposed on how to classify offensive language as well as the type and the target of offensive language. These guidelines capture the characteristics of generally offensive language, hate speech and other types of targeted offensive language such as cyberbullying.

1.1 Motivation

Given the fact that the research on offensive language detection has to a large extent been focused on the English language, we set out to explore the design of models that can successfully be used for both English and Danish. To accomplish this, an appropriate dataset must be constructed, annotated with the guidelines described in [41]. We, furthermore, set out to analyze the linguistic features that prove hard to detect by analyzing the patterns that prove hard to detect.

1.2 Problem Definition

We approach the task at hand as a supervised classification problem. Formally, classification problems of this kind can be described by the following statement: Given a set of user-generated samples,

$S = \{s_1, s_2, s_3, \ldots s_n\}$, labeled with labels $L = \{l_1, l_2, l_3, \ldots, l_k\}$, find a function F, that maps a sample, s_i, to the appropriate label, l_j.

1.3 Contribution

We construct a Danish dataset containing user-generated comments from *Reddit* and *Facebook*, annotated using the schema introduced in [41]. We develop four automatic classification systems, each designed to work for both the English and the Danish language. In the detection of offensive language in English, the best performing system achieves a macro averaged F1-score of 0.74, and the best performing system for Danish achieves a macro averaged F1-score of 0.70. In the detection of whether or not an offensive post is targeted, the best performing system for English achieves a macro averaged F1-score of 0.62, while the best performing system for Danish achieves a macro averaged F1-score of 0.73. Finally, in the detection of the target type in a targeted offensive post, the best performing system for English achieves a macro averaged F1-score of 0.56, and the best performing system for Danish achieves a macro averaged F1-score of 0.63.

Chapter 2

Related Work

In recent years there has been a lot of interest in the natural language processing community concerning the vast amount of offensive and aggressive language on online social media platforms and the implications and issues this poses. Substantial effort has been put into the research and development of automatic systems designed to classify different types of offensive and harmful language. In this chapter, we discuss some of the challenges that arise when developing automatic systems for harmful language detection in online communications and explore a few of the different types of tasks that have been considered in the literature. Finally, a short overview of some of the more prominent set of features and models that have been used in previous work are presented.

2.1 Challenges

Automatic detection and classification of offensive language in user-generated content is far from trivial due to many factors such as the

high variety and noisiness of the data. In [27] some of the common challenges that arise when developing automatic classification systems for these types of tasks are discussed. First of all, [27] note the common practice in online communities to obfuscate offensive words and phrases to prevent manual and/or automatic intervention. This makes it impossible for simple keyword spotting algorithms to be effective since obfuscations such as *ni99er*, *whoopiuglyniggerratgolberg* and *JOOZ* are common on these social media platforms, making the effectiveness of simply looking for certain keywords very limited. Secondly, even if a simplistic blacklist approach would be effective, it is impossible to keep track of all racial, sexist and generally offensive terms as these tend to evolve quickly over time and this happens exponentially fast in online communities. Any blacklist approach would, therefore, require constant manual efforts to keep up-to-date with trends in the language for each community as they develop over time. It is, furthermore, pointed out that language that might be considered harmful and offensive in some communities may be considered fully acceptable in other communities, which further increases the need for manual efforts [27]. Thirdly, the offensiveness of a conversation often crosses sentence boundaries so the full context is required in order to correctly classify the offensiveness. A thorough understanding of the context is also required to be able to accurately identify phenomenons such as sarcasm, which is listed in [27] as one of the major challenges when it comes to these kinds of tasks.

2.2 Different Types of Tasks

Many different types of sub-tasks have been considered in the literature on offensive and harmful language detection, ranging from the detection

of general offensive language to more refined tasks such as hate speech detection [10], and cyberbullying detection [38].

A key aspect in the research of automatic classification methods for language of any kind is having substantial amount of high quality data, that reflects the goal of the task at hand and contains a decent amount of samples belonging to each of the classes being considered. Since most researchers approach this problem as a supervised classification task the data needs to be annotated according to a well defined annotation schema that clearly reflects the problem statement. The quality of the data is of vital importance, since low quality data is unlikely to provide meaningful results. In recent years, a number of tasks have been considered and a great amount of data has been gathered, processed, and published in the process. In the following paragraphs a few different types of these tasks are listed, as well as the datasets used in these efforts.

Cyberbullying. Cyberbullying, is a form of online harassment. It is commonly defined as targeted insults or threats against an individual [41]. In [38], three factors are mentioned as indicators of cyberbullying: the intention to cause harm, repetitiveness, and an imbalance of power. This type of online harassment most commonly occurs between children and teenagers, and cyberbullying acts are prohibited by law in several countries, as well as many of the US states [14].

In [37] the focus is on classifying cyberbullying events in Dutch. They define cyberbullying as textual content that is published online by an individual and is aggressive or hurtful against a victim. The dataset is created by collecting data from *Ask.fm* (a social networking site where users can post questions and answers) and the final

dataset consists of 85,485 Dutch posts, annotated by two annotators. The annotation-schema used consists of two steps. In the first step, a three-point harmfulness score is assigned to each post as well as a category denoting the authors role (i.e. harasser, victim, or bystander). In the second step a more refined categorization is applied, by annotating the posts using the the following labels: *Threat/Blackmail*, *Insult*, *Curse/Exclusion*, *Defamation*, *Sexual Talk*, *Defense*, and *Encouragement to the harasser*. The refined classes in step two and the sparsity of the data make the classification task difficult as each class contains few samples, which explains the low F1-score of 0.55 in their best performing model.

Hate Speech. As discussed in Chapter 1, hate speech is generally defined as language that is targeted towards a group, with the intend to be harmful or cause social chaos. This targeting is usually based on characteristics such as race, color, ethnicity, gender, sexual orientation, nationality or religion [36]. Hate speech is prohibited by law in many countries, although the definitions may vary. In article 20 of the *International Covenant on Civil and Political Rights (ICCPR)* it is stated that *"Any advocacy of national, racial or religious hatred that constitutes incitement to discrimination, hostility or violence shall be prohibited by law"* [18]. In Denmark, hate speech is prohibited by law, and is formally defined as public statements where a group is threatened, insulted, or degraded on the basis of characteristics such as nationality, ethnicity, religion, or sexual orientation [3]. Hate speech is also generally prohibited by law in the European Union, where it is defined as public incitement to violence or hatred directed against a group defined on the basis of characteristics such as race, religion, and national or ethnic origin [1]. Hate speech is, however, not prohibited by law in the United States. This is due to the

fact that hate speech is protected by the freedom of speech act in the *First Amendment of the U.S. Constitution* [7].

In [10] the focus is on classifying hate speech by clearly distinguishing between general offensive language and hate speech. They define hate speech as *"language that is used to express hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group"* [10]. As the authors point out, the high use of profanity on social media makes it vitally important to be able to effectively distinguish between generally offensive language and the more severe hate speech. The dataset is constructed by gathering data from Twitter, using a hate speech lexicon to query the data. The crowd sourced annotation platform *CrowdFlower* [1] is used for the annotation work, and each tweet is coded by three or more annotators. The majority decision from these annotators is then used to assign labels. The final dataset in [10] consists of 24,802 annotated tweets, with only 5% of them belonging to the hate speech class. While their model achieves an impressive overall F1-score of 0.90, the recall for the hate speech category is only 0.61, which is most likely a direct result of the high imbalance in their dataset.

Contradicting definitions. When navigating the literature on the topic of offensive speech detection, it becomes clear that one of the key challenges in doing meaningful research on the topic are the differences in both the annotation-schemas and the definitions used, since it makes it difficult to effectively compare results to existing work, as pointed out by several authors ([27], [36], [39], [41]). These issues become clear when comparing the work of [38], where racist and sexist remarks are classified as a subset of *insults*, to the work of [28], where similar remarks

[1] www.figure-eight.com

are split into two categories; *hate speech* and *derogatory language*. Another clear example of conflicting definitions becomes visible when comparing [40], where *hate speech* is considered without any consideration of overlaps with the more general type of offensive language, to [10] where a clear distinction is made between the two, by classifying posts as either *Hate speech*, *Offensive* or *Neither*. This lack of consensus and the potential overlap led [39] to propose annotation guidelines for tasks within the offensive and harmful language category. They introduce a typology that simply asks two questions: *Is the language directed towards a specific individual or enitity or is it directed towards a generalized group?* and *Is the abusive content explicit or implicit?*. In [42] they, however, argue that these proposed guidelines do not effectively capture both the type and target of the offensive language. They propose a new three-level annotation model which is designed to capture characteristics from previous work on the topic, and can therefore easily be extended to work with existing datasets from previous work. This three-level annotation model is the one used in our work and is discussed in more detail in Chapter 3.

2.3 Features

One of the most important factors to consider when it comes to automatic classification tasks of any kind is what features should be used. Various features have been explored in the literature on offensive and abusive language detection and in the following paragraphs we discuss some of them. In all the examples discussed here the tasks were approached as supervised classification tasks.

Top-level features. As discussed in [36], the most obvious information comes from top-level features such as bag-of-words, uni-grams and more complex n-grams, and the literature certainly supports this. In their work on cyberbullying detection, [37] make use of word n-grams, character n-grams, and bag-of-words. The word n-grams are used as binary features, indicating the presence of a word n-gram in a post, and the character n-grams are used to provide some abstraction from the word level. They report the uni-gram bag-of-word features as the most predictive, followed by character tri-gram bag-of-words. In [28], where the focus is on abusive language detection, they make use of character n-grams, as they are able to model the obfuscation of offensive words effectively, as well as token n-grams. In their results, the character n-grams are the most predictive features, underlying the need for the modeling of un-normalized text (such as obfuscated terms). As pointed out in [10], these simple top-level feature approaches are good but not without their limitations, since they often have high recall but lead to high rate of false positives. This is due to the fact that the presence of certain terms can easily lead to misclassification when using these types of features. Many words, however, do not clearly indicate which category the text sample belongs to, e.g. the word *gay* can be used in both neutral and offensive contexts.

Linguistic Features Various authors have explored refined linguistic features in addition to some of the top-level features discussed earlier. [28] make use of a number of linguistic features, including the length of samples, average word lengths, number of periods and question marks, number of capitalized letters, number of URLs, number of polite words, number of unknown words (by using an English dictionary),

and number of insults and hate speech words. Although these features have not proven to provide much value on their own, they have been shown to be a good addition to the overall feature space [28].

Word Generalization. The top-level features discussed so far often yield decent performance in general language classification tasks. This performance is, however, often limited when it comes to more refined tasks such as hate speech detection since most often the goal is to classify small samples of text. This becomes problematic since the top-level features require the predictive words to occur in both the training set and the test sets, as discussed in [36]. For this reason, some sort of word generalization is required. Recently, the literature has been moving towards a more distributed word representation and word embeddings (created using neural networks) have become increasingly popular. In [28], they explore three types of embedding-derived features. First, they explore pre-trained embeddings derived from a large corpus of news samples. Secondly, they make use of *word2vec* [23] to generate word embeddings using their own corpus of text samples. Both the pre-trained and word2vec models represent each word as a 200 dimensional distributed real number vector. Lastly, they develop 100 dimensional *comment2vec* model, based on the work of [20]. Their results show that the comment2vec and the word2vec models provide the most predictive features [28]. In [6] they experiment with pre-trained *GloVe* embeddings [30], learned *FastText* embeddings [24], and randomly initialized learned embeddings. Interestingly, the randomly initialized embeddings slightly outperform the others [6].

Sentiment Analysis. A somewhat obvious source of information to determine the offensiveness of a sample comes from its sementic score, and this has been used widely in the literature in addition to other features. In [37], sentiment scores are used to represent the number of positive, negative, and neutral words, averaged over the sample length. Although the sentiment scores perform poorly on their own, the authors report an increase in accuracy when combined with the other features. In [10], a sentiment lexicon designed for social media is used to assign sentiment scores to each tweet. The authors of [10], however, point out that this can lead to some misclassification, since their classifier relies too heavily on these scores and makes decisions solely based on whether or not a post contains terms with negative sentiment scores.

2.4 Models

A large variety of different models have been explored in the literature on offensive, abusive and hateful language detection. Most of these efforts approach the task as a supervised classification problem and the methods range from simple lexical and statistical approaches to linear machine learning models and deep neural networks. Most of the best performing system in the literature use either machine learning algorithms or deep neural networks, and in the following paragraphs some of the more prominent ones are discussed.

Linear Machine Learning Models. In their work on hate speech detection, [10] explore a variety of machine learning algorithms introduced in prior work such as *Logistic Regression*, *Naive Bayes*, *Decision Trees*, *Random Forest*, and *Linear Support Vector Machines (SVMs)*. Their final

model uses a *Logistic Regression* with *L2 Regularization*, resulting in an overall precision of 0.91, recall of 0.90 and F1-score of 0.90. In our work we base one of our classifiers (the Logistic Regression classifier, section 5.2.4) on the architecture proposed in [10].

Deep Learning Models. In their work on hate speech detection, the authors of [6] explore various different neural network architectures including *Convolutional Neural Networks (CNNs)* and *Long Short Term Memory Networks (LSTMs)*. Their best model uses a *LSTM*, combined with *Gradient Boosted Decision Trees*, and randomly initialized word embeddings (similar to what is discussed in section 5.1.8), resulting in a F1-score of 0.93.

2.5 OffensEval

The authors of [41] created and managed a shared task on the topic of offensive language detection named *OffensEval*[2]. The task consists of three sub-tasks: A, B, and C. In sub-task A, the goal is to classify a user-generated post as either offensive (OFF) or not offensive (NOT). In sub-task B, the goal is to classify offensive posts as either untargeted (UNT) or targeted insults/threats (TIN). In sub-task C, the goal is to determine if a targeted offensive post is targeted towards an individual (IND), a group (GRP) or some other entity (OTH). Sub-task C, therefore, captures the characteristics of both cyberbullying (in the IND category) and hate speech (in the GRP category), as they are defined in section 2.2. The sub-tasks and the task structure is described in more detail in Chapter 3.

[2]www.competitions.codalab.org/competitions/20011

The participating teams in OffensEval use a variety of deep learning models including Convolutional Neural Netoworks (CNNs), Recurrent Neural Networks (RNNs), Bi-Directional Long Short Term Memory Networks (BiLSTMs), as well as some state-of-the art deep learning models such as ELMO [31], and BERT [11]. The results and the models used are presented by the task authors in [42]. Based on these results, it seems that deep learning models provide notable benefits over the more simple and traditional models, as they generally rely on a smaller set of features. In fact, state-of-the-art models such as BERT [11] have been shown to work over a variety of language tasks, such as question answering and language inference, without substantial task-specific architectural modifications [11]. The next few paragraphs describe the best performing systems in OffensEval as they are presented in [42].

Sub-task A. In the detection of offensive language in sub-task A (section 3.1.1), a system using the deep learning model BERT [11] was the most successful, resulting in a macro averaged F1-score of 0.829.

Sub-task B. In the detection of whether an offensive post is targeted or not, in sub-task B (section 3.1.2), the best performing system used a rule based approach, resulting in a macro averaged F1-score of 0.755. The second and third teams used ensembles of deep learning models (including BERT [11]) and achieved macro averaged F1-scores of 0.739 and 0.719.

Sub-task C. In the detection of the type of target in sub-task C (section 3.1.3), the top performing team uses the deep learning model BERT [11]. They use pre-trained *GloVe* [30] word-embeddings to represent the

words of a sample in a dense vector space, while maintaining semantic similarity (similar to what is discussed in 5.1.8). Their system achieved a macro averaged F1-score of 0.660.

In table 2.1 the results from *OffensEval* are presented as they appear in [42]. These results also include the models (CNN, BiLSTM, and SVM) and baselines (section 5.2.3) introduced in [41].

Sub-task A		Sub-task B		Sub-task C	
Team Ranks	F1 Range	Team Ranks	F1 Range	Team Ranks	F1 Range
1	0.829	1	0.755	1	0.660
2	0.815	2	0.739	2	0.628
3	0.814	3	0.719	3	0.626
4	0.808	4	0.716	4	0.621
5	0.807	5	0.708	5	0.613
6	0.806	6	0.706	6	0.613
7	0.804	7	0.700	7	0.591
8	0.803	8	0.695	8	0.588
9	0.802	9	0.692	9	0.587
CNN	**0.800**	**CNN**	**0.690**	10	0.586
10	0.798	10	0.687	11-14	.571-.580
11-12	.793-.794	11-14	.680-.682	15-18	.560-.569
13-23	.783-.789	15-24	.660-.671	19-23	.547-.557
24-27	.772-.779	**BiLSTM**	**0.660**	24-29	.523-.535
28-31	.765-.768	25-29	.640-.655	30-33	.511-.515
32-40	.750-.759	**SVM**	**0.640**	34-40	.500-.509
BiLSTM	**0.750**	30-38	.600-.638	41-47	.480-.490
41-45	.750-.749	39-49	.553-.595	**CNN**	**0.470**
46-57	.730-.739	50-62	.500-.546	**BiLSTM**	**0.470**
58-63	.721-.729	**All TIN**	**0.470**	**SVM**	**0.450**
64-71	.713-.719	63-74	.418-.486	46-60	.401-.476
72-74	.704-.709	75	0.270	61-65	.249-.340
SVM	**0.690**	76	0.121	**All IND**	**0.210**
75-89	.619-.699	**All UNT**	**0.100**	**All GRP**	**0.180**
90-96	.500-.590			**All OTH**	**0.090**
97-103	.422-.492				
All NOT	**0.420**				
All OFF	**0.220**				
104	0.171				

Table 2.1: Macro averaged F1-scores for the competing teams in OffensEval for sub-task as they are presented in [42]. The baselines and models from [41] are also included in bold.

Chapter 3

Task Framing

As mentioned in the introduction, our work is to a large extent based on the definitions and structure introduced in [41]. In this chapter we give a comprehensive overview of the structure of the task and describe the dataset provided in [41].

3.1 Classification Structure

The goal of our task is to identify offensive content in social media data using automatic methods. The offensive content is broken into three sub-tasks to be able to effectively identify both the type and the target of the offensive posts. These three sub-tasks are chosen with the objective of being able to capture different types of offensive language, such as hate speech and cyberbullying (section 2.2). This annotation structure can, therefore, be used in a wide range of offensive language detection tasks. These three sub-tasks are described in detail in the following sections.

3.1.1 Sub-task A - Offensive language identification

In sub-task A the goal is to classify posts as either offensive or not offensive. Offensive posts include insults and threats as well as any form of untargeted profanity [42]. Each sample is annotated with one of the following labels:

- Not Offensive (NOT). In English this could be a post such as *#TheNunMovie was just as scary as I thought it would be. Clearly the critics don't think she is terrifyingly creepy. I like how it ties in with #TheConjuring series*. In Danish this could be a post such as *Kim Larsen var god, men hans død blev alt for hyped*.
- Offensive (OFF). In English this could be a post such as *USER is a #pervert himself!*. In Danish this could be a post such as *Kalle er faggot...*

3.1.2 Sub-task B - Automatic categorization of offensive language types

In sub-task B the goal is to classify the type of offensive language by determining if the offensive language is targeted or not. Targeted offensive language contains insults and threats to an individual, group, or others [42]. Untargeted posts contain general profanity while not clearly targeting anyone [42]. Only posts labeled as offensive (OFF) in sub-task A are considered in this task. Each sample is annotated with one of the following labels:

- Targeted Insult (TIN). In English this could be a post such as *@USER Please ban this cheating scum*. In Danish this could be a post such as *Hun skal da selv have 99 år, den smatso*.

- Untargeted (UNT). In English this could be a post such as *2 weeks of resp done and I still don't know shit my ass still on vacation mode*. In Danish this could be a post such as *Dumme svin...*

3.1.3 Sub-task C - Offensive language target identification

In sub-task C the goal is to classify the target of the offensive language. Only posts labeled as targeted insults (TIN) in sub-task B are considered in this task [42]. Each sample is annotated with one of the following labels:

- Individual (IND): Posts targeting an individual. A named person, or an unnamed person that is clearly part of the conversation. Insults and threats against an individual in this context are often referred to as cyberbullying [42]. In English this could be a post such as *@USER Is a FRAUD Female @USER group paid for and organized by @USER*. In Danish this could be a post such as *USER du er sku da syg i hoved*. These examples further demonstrate that this category captures the characteristics of cyberbullying, as it is defined in section 2.2.

- Group (GRP): Posts targeting a group of people based on ethnicity, gender or sexual orientation, political affiliation, religious belief, or other characteristics. These types of insults and/or threats against groups in this context are often defined as hate speech [42]. In English this could be a post such as *#Antifa are mentally unstable cowards, pretending to be relevant*. In Danish this could be a post such as *Åh nej! Svensk lorteret!* These examples clearly show that this category captures the characteristics of hate speech as it is defined in section 2.2.

- Other (OTH): The target of the offensive language does not fit the criteria of either of the previous two categories. [42]. In English this could be a post such as *And these entertainment agencies just gonna have to be an ass about it.*. In Danish this could be a post such as *Netto er jo et tempel over lort*.

3.2 The OLID Dataset

The authors of [42] publish a large dataset (titled *OLID*) alongside the shared task. It contains English tweets, gathered using the Twitter API to search for tweets containing certain keywords [41]. The data was annotated using the schema described in section 3.1 by using the crowdsourcing platform Figure Eight[1]. The resulting dataset contains 14,100 annotated tweets in total, and is split into a train and a test set of sizes 13,240 and 860. The distribution of labels in the *OLID* dataset is presented in table 3.1, using the same format as presented in [41].

A	B	C	Train	Test	Total
OFF	TIN	IND	2,407	100	2,507
OFF	TIN	OTH	395	35	430
OFF	TIN	GRP	1,074	78	1,152
OFF	UNT		524	27	551
NOT			8,840	620	9,460
All			13,240	860	14,100

Table 3.1: The distribution of labels in the *OLID* dataset for both the train and test set [42].

[1] www.figure-eight.com

Chapter 4

Danish Dataset

The main goal of our work is to develop an offensive and hate speech detection system for the Danish language. We construct a Danish dataset suitable for the tasks and to our knowledge no such dataset exists. Our resulting dataset consists of 3600 user-generated posts from various social media platforms, annotated with the schema presented in [41]. This chapter details the construction of this Danish dataset, as well as the annotation procedure. We also discuss the construction of a Danish hate speech lexicon. We, furthermore, discuss some of the privacy concerns and the necessary steps that need to be taken when developing a dataset from user-generated content. Finally, the resulting dataset is presented as well as the distribution of samples between the different classes.

4.1 Collection

One of the main concerns when it comes to collecting data for the task of offensive language detection is to find high quality sources of user-generated content that represent each class in the annotation-schema

to some extent. In our exploration phase we considered various social media platforms such as *Twitter*[1], *Facebook*[2], and *Reddit*[3].

Twitter. Twitter has been used extensively as a source of user-generated content in various language classification tasks and it was the first source considered in our initial data collection phase. The platform provides excellent interface for developers making it easy to gather substantial amount of data with limited efforts. It quickly became clear, however, after some initial inspection of the data collected that this would not be a suitable source of data for our task. This is due to the fact that *Twitter* has limited usage in Denmark, resulting in low quality data with most of the classes of interest unrepresented.

Facebook. We next considered *Facebook*, and the public page for the Danish media company *Ekstra Bladet*[4]. We looked at user-generated comments on articles posted by Ekstra Bladet, and initial analysis of these comments showed great promise as they vary a lot. The user behaviour on the page and the language used ranges from neutral language to very aggressive, where some users pour out sexist, racist and generally hateful language. We did, however, face some obstacles when collecting data from Facebook, due to the fact that Facebook recently made the decision to shut down all access to public pages through their developer interface[5]. This makes computational data collection approaches impossible for public pages and user-generated comments.

[1] www.twitter.com
[2] www.facebook.com
[3] www.reddit.com
[4] www.facebook.com/ekstrabladet
[5] www.newsroom.fb.com/news/2018/04/restricting-data-access

Given this obstacle, we turned to manual collection of randomly selected user-generated comments from Ekstra Bladet's public page, and these efforts resulted in a dataset of 800 comments of high quality.

Reddit. Given that language classification tasks in general require substantial amounts of data, our exploration for suitable sources continued and our search next led us to *Reddit*. We wrote a simple script[6] using the developer interface and used it to collect the top 500 posts from the Danish sub-reddits *r/DANMAG*[7] and *r/Denmark*[8], as well as the user comments contained within each post. These efforts resulted in over 13 thousand user-generated comments.

4.2 Danish Hate Speech Lexicon

In efforts to maximize the number of user-generated comments from Reddit belonging to the classes of interest in our final dataset we published a survey on Reddit[9], asking Danish speaking users to suggest offensive, sexist, and racist terms. As mentioned in Chapter 2, the language and user behaviour varies a lot between platforms, so the goal with posting the survey on Reddit was to obtain platform specific terms. These efforts resulted in a list of 113 offensive and hateful terms which were then used to query the user comments from Reddit. The remainder of comments in the corpus (i.e. the comments that did not contain any of the hateful terms from the lexicon) were shuffled and a subset

[6] www.github.com/gsig123/reddit_scraper
[7] www.reddit.com/r/DANMAG/
[8] www.reddit.com/r/denmark/
[9] www.reddit.com/r/Denmark/comments/9ozhdc/hateracistsexistetc terms in danish/

of this corpus was then used to fill the remainder of the final dataset. The resulting dataset contains 3600 user-generated comments, 800 from *Ekstra Bladet on Facebook*, 1400 from *r/DANMAG* and 1400 from *r/Denmark*. A detailed breakdown of the final dataset is presented in section 4.5. The full *Danish Hate Speech lexicon* from our survey can be found in Appendix A.

4.3 Privacy Concerns

In light of the *General Data Protection Regulations in Europe (GDPR)*[10] and the increased concern for online privacy, we applied some necessary pre-processing steps on our dataset to ensure the privacy of the authors of the comments that were used. First of all, personally identifying content (such as the names of individuals, not including celebrity names) was removed. This was handled by replacing each name of an individual (i.e. author or subject) with *@USER*, as presented in both [41] and [10]. Secondly, all comments containing any sensitive information were removed. We classify sensitive information as any information that can be used to uniquely identify someone by the following characteristics; racial or ethnic origin, political opinions, religious or philosophical beliefs, trade union membership, genetic data, and bio-metric data.

4.4 Annotation Procedure

We base our annotation procedure on the guidelines and schemas presented in [41], discussed in detail in section 3.1. As a warm-up procedure, the first 100 posts were annotated by two annotators (the author

[10] www.eugdpr.org

and the supervisor) and the results compared. This was used as an opportunity to refine the mutual understanding of the task at hand and to discuss the mismatches in these annotations for each sub-task. We used a *Jaccard index* [15] to assess the similarity of our annotations. In sub-task A the Jaccard index of these initial 100 posts was 41.9%, 39.1% for sub-task B , and 42.8% for sub-task C. After some analysis of these results and the posts that we disagreed on it became obvious that to a large extent the disagreement was mainly caused by two reasons:

1. Guesswork of the context where the post itself was too vague to make a decisive decision on whether it was offensive or not without more context. An example of this is a post such as *Skal de hjælpes hjem, næ nej de skal sendes hjem*, where one might conclude, given the current political climate, that this is an offensive post targeted at immigrants. The context is, however, lacking so we cannot make a decisive decision. This post should, therefore, be labeled as non-offensive, since the post does not contain any profanity or a clearly stated group.

2. Failure to label posts containing some kind of profanity as offensive (typically when the posts themselves were not aggressive, harmful, or hateful). An example could be a post like *@USER sgu da ikke hans skyld at hun ikke han finde ud af at koge fucking pasta*, where the post itself is rather mild, but the presence of *fucking* makes this an offensive post according to our definitions.

In light of these findings our internal guidelines were refined so that no post should be labeled as offensive by interpreting any context that is not directly visible in the post itself (tackling disagreement case 1)

and that any post containing any form of profanity should automatically be labeled as offensive (handling disagreement case 2). These stricter guidelines made the annotation procedure considerably easier while ensuring consistency. The remainder of the annotation task was performed by the author, resulting in 3600 annotated samples.

4.5 Final Dataset

In table 4.1 the distribution of samples by sources in our final dataset is presented. Although a useful tool, using the hate speech lexicon, discussed in section 4.2, as a filter only resulted in 232 comments. As mentioned in section 4.2, the remaining comments from Reddit were then randomly sampled from the remaining corpus.

Data Source	Number of Comments	% of Total
Facebook - Ekstra Bladet	800	22.2
Reddit - r/Denmark w offensive term	200	5.6
Reddit - r/Denmark w/o offensive term	1,200	33.3
Reddit - r/DANMAG w offensive term	32	0.9
Reddit - r/DANMAG	1,368	38.0

Table 4.1: The distribution of samples by sources in our final dataset. "w offensive terms" represents that the samples were retrieved using the Danish hate speech lexicon (section 4.2) as a filter.

The fully annotated dataset was split into a train and test set, while maintaining the distribution of labels from the original dataset. The training set contains 80% of the samples, and the test set contains 20%. Table 4.2 presents the distribution of samples by label for both the train and test set. It is clear that the dataset is highly skewed, with around 88% of the posts labeled as not offensive (NOT). This is, however, generally the case when it comes to user-generated content on online plat-

forms, and any automatic detection system needs be able to handle the problem of imbalanced data in order to be truly effective.

Sub-Task A	Sub-Task B	Sub-Task C	Train	Test	Total
OFF	TIN	IND	77	18	95
OFF	TIN	OTH	30	6	36
OFF	TIN	GRP	98	23	121
OFF	UNT		147	42	189
NOT			2,527	632	3,159
ALL			2,879	721	3,600

Table 4.2: The distribution of labels in the annotated Danish dataset for both the train and test set.

Chapter 5

Methods

Our goal is to create an automatic offensive language classification system designed to work well for both English and Danish. To accomplish this, we consider a wide variety of features and experiment with a number of model architectures. All of our development work is performed using *Python 3* and a variety of language modeling and machine learning libraries such as *NLTK* [22], *Scikit Learn* [22], *Tensorflow* [5], and *Keras* [9]. In this chapter we discuss the development of of our systems by describing in detail the features used and our final model architectures.

5.1 Features

As with any classification task the set of features used in the prediction models are of vital importance and this is especially true for language classification tasks. It is far from trivial to represent natural language in a computational friendly way while maintaining important syntactic and contextual information. In this section we describe in detail the features used in our models and the motivation behind these decisions.

5.1.1 Tokenization

The first step in the feature extraction process is the tokenization of all samples in the vocabulary in order to simplify the inputs while maintaining the essential information. The tokenization step involves splitting each sample into a list of tokens (i.e. words and symbols) and removing non-essential tokens from the list. These non-essential tokens are tokens that do not attribute any notable value to the meaning of the text samples, and are therefore non-essential in our classification task. First, all digits, emoticons, and URLs are removed from the sentences using regular expressions. Secondly, Twitter hashtags (e.g. #hashtag) and mentions (e.g. @mention) are replaced with *HASHTAGHERE* and *MENTIONHERE*, inspired by the work of [10]. Next, so called *stop-words* and punctuation symbols are removed using the natural language processing library *NLTK* [22]. Stop-words are words that are too frequent in natural language to be good indicators for any sort of classification task (e.g. *is, if, but, ...*). Finally, all remaining tokens are lower-cased. The final output from this tokenization step is a simplified list of tokens for each sample in the corpus.

5.1.2 N-grams

So called *n-grams*[1] are used within a few of our feature extraction methods. N-grams represent a contiguous sequence of n items in a sample, and allow us to derive meaning from contiguous sequences instead of only considering a single item at a time. N-grams can be used with various types of items in a text sequence such as characters and words. N-grams can be of various sizes and derive their name from the size

[1] https://en.wikipedia.org/wiki/N-gram

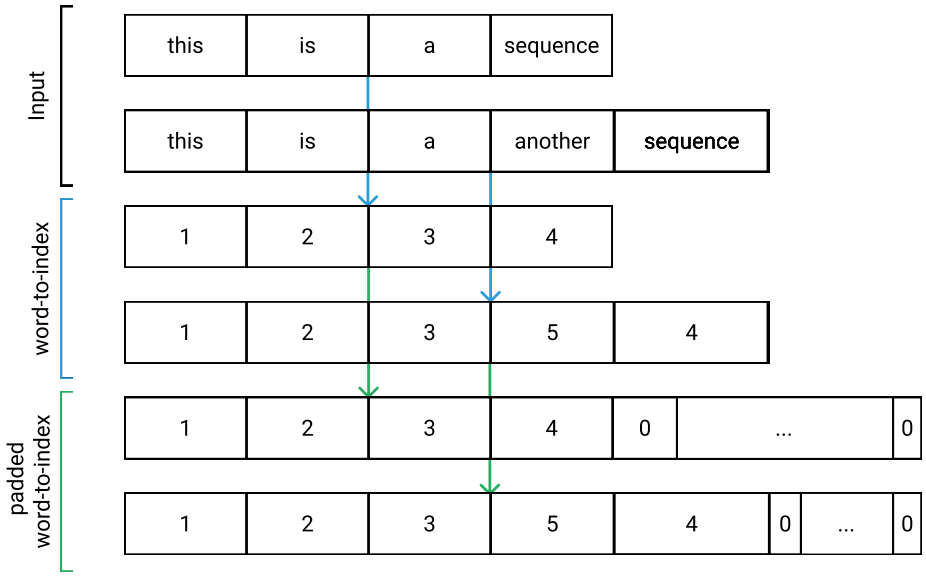
(e.g. uni-grams for 1-grams, and bi-grams for 2-grams). Given the sentence: "I like n-grams a lot" the following uni-, bi-, and tri-word-grams can be derived: "I", "like", "n-grams", "a", "lot", "I like", "like n-grams", "n-grams a", "a lot", "I like n-grams", "like n-grams a", "n-grams a lot". Word n-grams can often be useful in language classification tasks, as the meaning of a single word is most often controlled by the surrounding words.

5.1.3 Word-to-Index

In order to prepare our tokenized sequences from section 5.1.1 and make them ready to use with our deep learning models we need to transform them into sequences of real numbers of a fixed length. We create a *word-to-index* dictionary that maps all tokens in our vocabulary to a unique integer index. This word-to-index dictionary is then used to transform all token sequences into sequences of integers. These sequences are of various lengths as the number of tokens vary from sample to sample but our deep learning models require these sequences to be of a fixed length. In order to accomplish this we use the *Pad Sequences*[2] module from Keras [9] to pad the sequences to a fixed length of 100. The sequence length was determined during the initial experimental phase and according to those experiments this sequence length provided a good balance between relatively low training time and high accuracy. The sequences that contain fewer than 100 tokens are padded (using a unique padding integer) to the required length (by appending the padding integer to the sequence), while the sequences that are longer are cut down. Figure 5.1 visualizes the word-to-index and padding process.

[2] www.keras.io/preprocessing/sequence/#pad_sequences

Figure 5.1: A visualization of the word-to-index and padding process. In this example, the integer 0 is used as a padding index.



5.1.4 Sentiment Scores

As mentioned in Chapter 2, sentiment scores are a common addition to the feature space of classification systems dealing with offensive and hateful speech. Sentiment scores give the polarity of a word or a text sample (e.g. negative, neutral or positive). They are independent models, trained specifically for predicting the sentiment of samples. In our work we experiment with sentiment scores and some of our models rely on them as a dimension in their feature space. To compute these sentiment score features our systems make use of two Python libraries: *VADER* [17] and *AFINN* [26]. VADER is a sentiment analyzer specifically tuned to work well with social media data in English. Our mod-

els use the *compound* attribute, which gives a normalized sum of sentiment scores over all words in the sample. The compound attribute ranges from -1 (extremely negative) to $+1$ (extremely positive). Since VADER provides limited support for other languages than English, we use AFINN for our Danish models. AFINN is a wordlist-based approach, with full support for Danish. It gives an integer sentiment score for each text sample based on the number of positive or negative terms. Positive scores indicate a positive sample, and negative scores indicate a negative sample.

5.1.5 Linguistic Features

As well as some of the top-level features mentioned so far, some of our methods also make use of additional low-level linguistic features inspired by the work of [10]. These linguistic features are the following; syllable count, character count, word count, average syllable count, number of unique terms, and the number of twitter objects (i.e. hashtags, mentions, re-tweets and URLs). We also make use of *Flesch-Kincaid Grade Level* and *Flesch Reading Ease scores*[3]. Flesch Reading Ease is a way to measure how easy a sentence is to read; higher scores indicate that the sentence is easy to read (equation 5.1). The Flesch-Kincaid Grade Level is a metric assessing the level of reading ability required to easily understand a sample of text (equation 5.2).

$$207.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right) \tag{5.1}$$

[3] www.en.wikipedia.org/wiki/Flesch-Kincaid_readability_tests

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59 \tag{5.2}$$

5.1.6 Part-of-Speech Tagging

Some of our methods make use of *Part-of-Speech (POS) Tagging*[4]. POS-tagging is the process of assigning every word in a sample a part of speech token (i.e. noun, verb, adjective, etc.) and these POS-tags are a common addition to the feature space of language modeling algorithms. These POS-tags provide some generalization over the token level features, and sequences of POS-tags are often good indicators of the type and function of a text sample. In our methods, we start by transforming each tokenized sample into a sequence of POS-tags using the *NLTK* [22] library. These sequences of POS-tags are then transformed into a matrix where each row contains the frequency of uni-, bi-, and tri-grams of the POS-tags in the corresponding sample. Unfortunately, the NLTK POS-tagger does not support the Danish language. These features are, therefore, only used in systems for the English language.

5.1.7 TF-IDF

TF-IDF stands for *term frequency-inverse document frequency*[5] and the TF-IDF score is used to evaluate how important a token is to a sample in the overall corpus. The TF-IDF weight consists of two parts, the term frequency (TF) and the inverse document frequency (IDF). The term frequency calculates the number of times a token appears in a sample,

[4] www.en.wikipedia.org/wiki/Part-of-speech_tagging
[5] www.tfidf.com

divided by the number of tokens in the sample (equation 5.3). The inverse document frequency measures how important a token is in the overall corpus, by computing the logarithm of the number of samples in the corpus divided by the number of samples containing the token (equation 5.4). The inverse document frequency, therefore, scales up the score for rare tokens that might still be important for the context, while scaling down the score for very frequent tokens that might provide less meaning (such as stop-words, section 5.1.1). The final TF-IDF score is the product of the two parts (equation 5.5). In some of our methods, we make use of TF-IDF scores by constructing a matrix where each row contains the compounded TF-IDF scores for uni-, bi-, and tri-grams contained in the corresponding sample. We use the *TfidfVectorizer* package from *Scikit Learn* [29], with the *max_features* parameter set to 100.

$$\text{TF} = \frac{\text{Number of times the token appears in the sample}}{\text{Total number of tokens in the sample}} \tag{5.3}$$

$$\text{IDF} = log_e \left(\frac{\text{Total number of samples}}{\text{Number of samples with the token in it}} \right) \tag{5.4}$$

$$\text{TF-IDF} = \text{TF} \cdot \text{IDF} \tag{5.5}$$

5.1.8 Word Embeddings

As discussed in Chapter 2, many authors have experimented with and used *word embeddings* as features in various offensive language detection tasks. Word embeddings are distributed real number vector representations of words, designed to maintain semantic and contextual

similarity (i.e. similar words have similar word embedding vectors). As mentioned in [32], one of the key advantages of word embeddings is that they enable generalization of words that do not appear in labeled training data by embedding lexical features from a large corpus into a relatively low dimensional Euclidian space. Word embedding models are generally created using neural networks that are trained on a large unlabeled corpus. In our work we make use of two types of word embeddings; pre-trained *FastText embeddings* [13] and *randomly initialized learned embeddings*. In our initial experimental phase we also considered *Word2Vec* [23] trained on our own corpus, pre-trained *GloVe* [30] vectors and FastText trained on our own corpus. These initial experiments indicated lower performance from these additional methods. The increased effectiveness of pre-trained FastText embeddings over GloVe and Word2Vec can likely be explained by the fact that FastText is trained on character n-grams, while both GloVe and Word2Vec are trained on word n-grams. Given the fact that we are working with social media data, it is likely that a lot of the words in our corpus are rare, making the character n-gram approach more effective. The following sections describe the pre-processing steps necessary to make use of the pre-trained FastText embeddings and the randomly initialized learned embeddings.

Pre-trained Embeddings. The pre-trained FastText [24] embeddings are trained on data from the *Common Crawl*[6] project and Wikipedia, in 157 languages (including English and Danish). The vast amount of data used in the creation of these embeddings is promising since they should

[6] www.commoncrawl.org

prove effective in providing generalization for a wide variety of language modeling tasks. The FastText pre-trained embeddings are provided in a .txt file, containing over 2 million tokens for each language, along with their embeddings. FastText also provides trained models that can be used to predict word embeddings for *out-of-vocabulary (OOV)* words (i.e. tokens that were not found in the pre-trained FastText .txt file). This is a major advantage since a lot of the challenges that arise when using pre-trained word embeddings come from the fact that it is common that some words in the corpus are not found in the pre-trained corpus. This often happens because of misspellings and obfuscation of words, which is common in raw social media data. In our work, we make use of these FastText pre-trained embeddings by performing the following steps:

1. Tokenize the full corpus (for each language).
2. Look up the word embeddings for each token in the pre-trained vocabulary. If it exists, we assign the token the existing word embedding, otherwise we save the token as an OOV token.
3. Use the pre-trained FastText OOV model to predict word embeddings for the OOV tokens, assigning the resulting embedding vectors to the OOV tokens.
4. Construct a word-to-index dictionary containing a token to integer mapping for all tokens in our corpus.
5. Use the word-to-index dictionary to construct an embedding matrix, where row i contains the token embedding for the token that maps to index i in the word-to-index dictionary.

The resulting embedding matrix is then used to create the *embedding layer* in the models that make use of pre-trained embeddings (*Fast-BiLSTM* and *AUX-Fast-BiLSTM*).

Randomly Initialized Learned Embeddings. Some of our models make use of randomly initialized embeddings, that are updated during training. In this case, the embedding matrix for the *embedding layer* is initialized using a random uniform distribution.

5.2 Models

We introduce a variety of models in our work to compare different approaches to the task at hand. First of all, we introduce naive baselines that simply classify each sample as one of the categories of interest (based on [41]). Next, we introduce a logistic regression model based on the work of [10], using the same set of features as introduced there. Finally, we introduce three deep learning models: Learned-BiLSTM, Fast-BiLSTM, and AUX-Fast-BiLSTM. The logistic regression model is built using Scikit Learn [29] and the deep learning models are built using Keras [9]. The following sections describe these model architectures in detail, the algorithms they are based on, and the features they make use of.

5.2.1 Logistic Regression

One of our model architecture uses a *Logistic Regression* as the classification algorithm. Logistic Regression is a widely used statistical method for modeling dependent variables[7]. It is an extension to linear regres-

[7] www.en.wikipedia.org/wiki/Logistic_regression

sion, where the target variables are categorical[8]. Logistic regression predicts the probability of events by using a *logit* function. This logit function is usually a Sigmoid function (equation 5.7), mapping continues variables to discrete values. A logistic regression (equation 5.8) is computed by applying the Sigmoid function to the linear regression (equation 5.6). Here, y is the dependent variable, $X_1, \ldots, X_n$ are the explanatory variables, and $\beta_0, \ldots, \beta_n$ are the constants we are trying to estimate[9]. In logistic regression, the estimation is then typically done by using *Maximum Likelihood Estimation*[10].

$$y = \beta 0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n \tag{5.6}$$

$$p = \frac{1}{1 + e^{-y}} \tag{5.7}$$

$$p = \frac{1}{1 + e^{-(\beta 0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n)}} \tag{5.8}$$

5.2.2 Long Short Term Memory Networks

Our neural network models make use of *Recurrent Neural Networks (RNNs)* [12] as a key element in their architecture. RNNs are designed to capture information in sequences, which makes them a good fit for the task at hand. RNNs compute a fixed size vector representation of a sequence, by reading in n vectors $x_1, \ldots x_n$, and producing an output vector h_n that depends on the entire sequence [33]. This vector, h_n, can

[8] www.datacamp.com/community/tutorials/understanding-logistic-regression-python

[9] www.datacamp.com/community/tutorials/understanding-logistic-regression-python

[10] www.en.wikipedia.org/wiki/Maximum_likelihood_estimation

then be used in the consecutive layers of the neural network. The RNN layer is trained alongside the whole network, so that the hidden representation of the sequences and the information necessary for the task at hand is captured during the training phase [33]. *Long short term memory networks (LSTMs*) [16] are an extension of RNNs, designed to prevent vanishing gradients, which is a common problem when using RNNs with longer sequences [33]. This is accomplished, in simple terms, by adding layers to determine which information from the previous items in a sequence should be kept (*the input gate layer*) and which information should be thrown away (*the forget gate layer*) [11]. Other methods such as *Hidden Markov Models (HMM)* [34] are commonly used to model sequences. The main benefit of using LSTMs in language modeling tasks comes from the fact that LSTMs have the ability to forget information (using the forget gate layer) while maintaining long-range dependencies.

Bi-directional RNNs (BiRNNs) are another extension of RNNs, where each input sequence is read twice, from left to right and right to left [33]. This bi-directional approach is based on the idea that the elements in a sequence are based on both the previous and future elements. This is usually the case with natural language, making this a promising approach for the task at hand. Our models make use of *Bi-directional LSTMs (BiLSTMs*), where the output vectors from the forward and backward pass are concatenated together. This method is used as an attempt to capture complex hidden patterns in the input sequences.

[11] www.colah.github.io/posts/2015-08-Understanding-LSTMs/

5.2.3 Baselines

Following the work of [41], we create simple baseline prediction models that simply classify all samples as the class containing the largest amount of samples. This allows us to investigate the properties and distribution of the samples in the datasets, and to evaluate how well our classifiers are performing. The baseline models are the following:

- Sub-Task A: All NOT for both languages.
- Sub-Task B: All TIN for both languages.
- Sub-Task C: All IND for English and All GRP for Danish.

5.2.4 Logistic Regression Classifier

We base one of our model on the works of [10], where the objective is to distinguish between neutral, offensive and hateful language.

As mentioned in section 2.2, their model achieves an impressive F1-score of 0.90, which sparked the interest to experiment with the architecture presented in [10] as a possible solution to the problem at hand. The logistic regression classification model uses a wide variety of features. First, all samples are tokenized using the procedure introduced in section 5.1.1. Next, uni-grams, bi-grams, and tri-grams are created, weighted by their TF-IDF score (as described in section 5.1.7). To capture information about the syntactic structure, a POS-tag uni-, bi-, and tri-gram matrix is created using the method discussed in section 5.1.6. Given the limited support for Danish, the POS-tags are only included in the classifier for the English language (as discussed in 5.1.6). Sentiment scores (section 5.1.4) are also calculated for each sample, as well

as linguistic counters for the number of syllables, number of characters, number of tokens, average number of syllables, Flesch Reading Ease and Flesch-Kincaid Grade Level scores, number of re-tweets, number of Twitter mentions, number of Twitter hashtags, and number of URLs (as described in section 5.1.5). All of the features are then concatenated together and used as inputs to the classifier. To reduce the dimensionality of the feature space, the *SelectFromModel* module from Scikit Learn [29] is used to select the best set of features.

The logistic regression is implemented using the *LogisticRegression* module from the Scikit Learn library [29]. The class weights are set to *balanced* and the optimization algorithm used is *Limited-memory BFGS (lbfgs)* [21]. The penalty is set to *L2*, and the regularization strength is set to 0.01.

5.2.5 Hyper-Parameters in Deep Learning Models

In order to get as close as possible to the optimal set of hyper-parameters for our deep learning models we perform hyper-parameter tuning. *Grid Search* is one of the most popular method of parameter optimization. It selects the optimal set of parameters from a provided set. Other methods, such as *Random Search* [8], might offer speed and computational efficiency but this comes at the cost of accuracy. In our work we perform *Grid Search Cross Validation* using the corresponding module from Scikit Learn [29] to determine the optimal dropout amount, the batch size, the optimizer and the learning rate.

In the search for the optimial optimizer we consider Adam [19] and Stochastic Gradient Descent (SGD) [35]. Stochastic Gradient Descent is an optimization algorithm for unconstrained optimization problems.

In its approximation of the true gradient it considers a single training sample at a time [4]. Adam is a more memory efficient stochastic optimization method. It computes individual adaptive learning rates for different parameters, while Stochastic Gradient Descent has the same type of effect on all parameters [19].

From the Grid Search Cross Validation experiments we conclude that the best set of hyper-parameters for all of our models are the following: batch size of 128, Adam [19] as the optimization algorithm with a learning rate of 0.001, and a dropout rate of 0.2 between all layers. To tackle the imbalance of the samples in our dataset we use class weights in all of our deep learning models. Each class is given a weight equal to the inverse of the number of samples it contains. We, furthermore, use grid search cross validation to determine the number of nodes for each layer in our models by trying out a large variety of combinations, using the parameters found in the previous step and training for 20 epochs. The optimal number of nodes according to these experiments are discussed in the following subsection.

5.2.6 Learned-BiLSTM Classifier

The *Learned-BiLSTM model* is a deep neural network model built with the Keras Sequential API[12]. It consists of four parts; a randomly initialized embedding layer (as described in section 5.1.8), a bi-directional long short memory (BiLSTM) layer, a fully connected hidden layer, and a fully connected output layer. The model takes padded word-to-index sequences of length 100 as input (section 5.1.3), where each sequence represents a sample from the dataset. The embedding layer consists of

[12]www.keras.io/getting-started/sequential-model-guide/

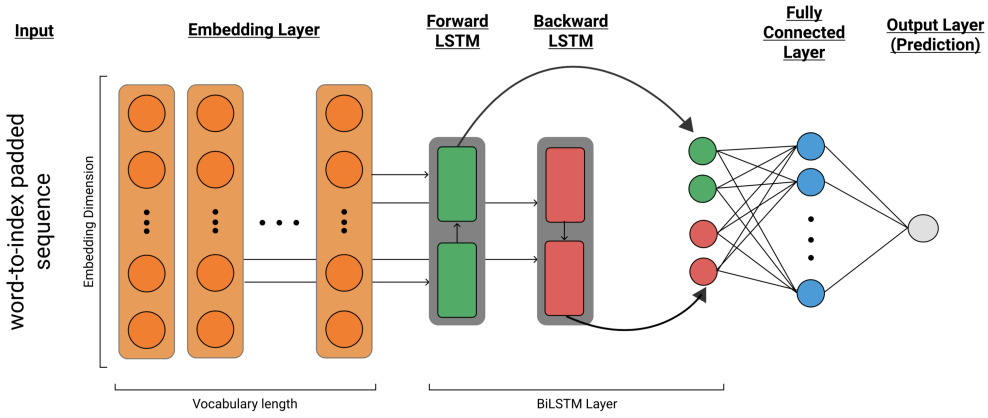
300 dimensional vectors, where each vector represents a unique token in the tokenized corpus. The integers from the input sequences are used to select which vectors from the embedding layer should be used for the current sample. The selected vectors are then used as input to the BiLSTM layer. The BiLSTM layer consists of two parts; a forward and a backward LSTM, each of size 20. Each sequence of word embeddings from the embedding layer is run through both the forward and backward LSTMs and the resulting vectors are then concatenated together, creating a vector of size 40. This vector is then used as input to the fully connected hidden layer, which contains 16 hidden units. This hidden layer is then fully connected to the output layer, which represents the output of the network (i.e. the predictions of the model). The output consists of a single node for sub-tasks A and B (binary NOT/OFF, UNT/TIN) and 3 nodes in sub-task C (IND/GRP/OTH). The activation function used in the LSTM layers is *tanh* and *ReLU* is used in the hidden layer. For sub-tasks A and B, the activation function for the output layer is *Sigmoid*, and *Softmax* is used for sub-task C. The loss is calculated using the *Binary Crossentropy* loss function. During training, the embedding vectors in the embedding layer are updated, which is why our model is named Learned-BiLSTM. In figure 5.2, the architecture of the model is presented.

5.2.7 Fast-BiLSTM Classifier

The *Fast-BiLSTM* model is built using the same layers and the same set of hyper-parameters as the *Learned-BiLSTM* model. The only difference is that instead of initializing the embedding layer randomly and learning the embeddings during training, the embedding layer is initialized with

the FastText embeddings (as discussed in section 5.1.8). These embeddings stay fixed and are not updated during the training of the model. Figure 5.2 presents a high level overview of the model architecture.

Figure 5.2: The high level architecture of both the Learned- and Fast-BiLSTM models. The only difference between the two is that the vectors in the embedding layer are updated during training for the Learned-BiLSTM model, while they stay fixed in the Fast-BiLSTM model. The output layer consists of one node in the case of sub-task A (NOT/OFF) and B (UNT/TIN), and 3 nodes in the case of sub-task C (IND, GRP, OTH).



5.2.8 AUX-Fast-BiLSTM Classifier

The Fast-BiLSTM model is fairly simple, since the only features it accepts are the padded word-to-index sequences. In order to experiment with a wider combination of features, we extend the Fast-BiLSTM model to *AUX-Fast-BiLSTM*, which accepts auxiliary features in the form of a feature matrix, where each row contains the additional features for the corresponding sample.

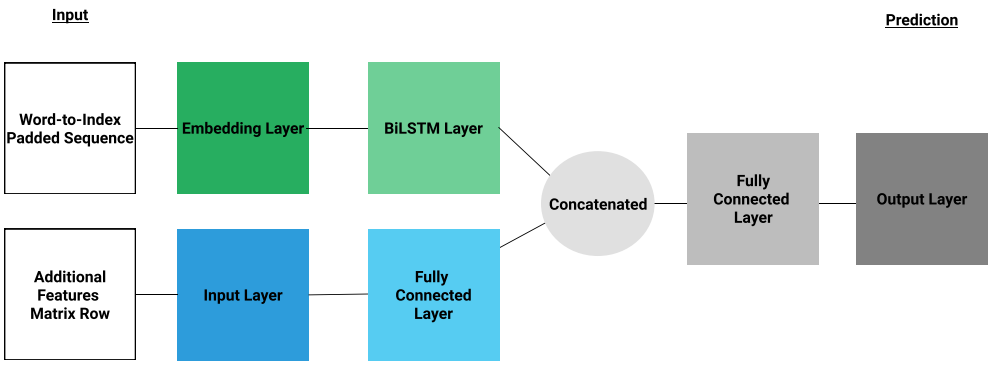
These additional features are created by tokenizing each sample (section 5.1.1), and computing the following; sentiment scores (section 5.1.4), uni-, bi-, and tri-grams weighted by their TF-IDF scores (section 5.1.7),

bi-, uni-, and tri-gram POS-tags (section 5.1.6), counters for the number of characters, number of syllables, number of words, number of Twitter hashtags, number of URLs, number of Twitter mentions, and number of Twitter re-tweets, as well as the Flesch reading ease and Flesch-Kincaid reading grade level scores (section 5.1.5). Note that these are the same features as used in the *Logistic Regression* classifier (section 5.2.4).

The model is built using the Keras Functional API[13] and consists of two parts. First of all, we have the previously described embedding layer and BiLSTM layer for the padded word-to-index features. The second part, which accepts the additional feature matrix rows as input, consists of an input layer of size equal to the number of features and a fully connected hidden layer with 16 hidden units. The output from both the fully connected layer and the BiLSTM layer is then concatenated and used as input to an additional fully connected hidden layer, also containing 16 units. Finally, we have the output layer, with the same number of units as in the other deep learning models. The activation function used in the LSTM layers is *tanh*, *ReLU* is used in both of the hidden layers and *Sigmoid* is used in the output layer in the case of sub-task A and B, and *Softmax* in the case of sub-task C. The loss is then calculated using the *Binary Crossentropy* loss function, as it is in the other deep learning models. A high-level overview of the architecture described in this section is presented in figure 5.3.

[13] www.keras.io/getting-started/functional-api-guide/

Figure 5.3: A high-level overview of the AUX-Fast-BiLSTM model architecture.



Chapter 6

Experimental Setup

We train all models from Chapter 5 over all the sub-tasks introduced in Chapter 3. Since the goal is to develop systems that can be used for both English and Danish we train our models using datasets from each language. The models are then tested using established test sets for each sub-task discussed in Chapters 3 and 4. To evaluate the performance of our models we use established quality metrics such as *Recall*, *Precision*, and the *F1-score*. We, furthermore, conduct error analysis of the predictions made by the best performing models in our evaluation phase (section 7.3). In this chapter, we discuss the datasets used in training for each of the sub-tasks and each language (section 6.1), and the training and testing procedure (section 6.2). Finally, we present the quality metrics used in the evaluation of our models (section 6.3.1).

6.1 Datasets

We conduct independent experiments for each language. The following sections describe in detail the datasets used in these experiments for

each sub-task and each language.

6.1.1 English

- **Sub-task A - Offensive language identification**. In our experiments for English and sub-task A, we make use of the full *OLID* training dataset (discussed in Chapter 3) as well as the dataset introduced in [10] (which we refer to as *HSAOFL*). The HSAOFL dataset is originally annotated with different guidelines, annotating posts as either offensive, hate speech, or neither. Given that for sub-task A, the offensive and hate speech labels belong to the same category within the annotation schema used in our work, we transform all samples labeled with either of these to OFF. Similarly, the neither category is mapped to NOT. This makes it trivial to use the HSAOFL dataset with our models without any modifications. Table 6.1 presents the distribution of labels in the modified HSAOFL dataset. In our experiments for sub-task A, all models are trained using both the OLID dataset on its own as well as the OLID and HSAOFL datasets combined. The OLID dataset contains 13,240 samples (table 3.1) and the HSAOFL dataset contains 24,783 samples (table 6.1). The test set used in this sub-task contains 860 samples (table 3.1).

- **Sub-task B - Automatic categorization of offensive language types**. In sub-task B, we only make use of the OLID training set. We only include samples that are labeled as OFF in sub-task A, since the goal of this task is only to determine if the offensive samples are targeted or not. The training set for this sub-task contains 4,400 samples and the test set contains 240 samples (table 3.1).

- **Sub-task C - Offensive language target identification**. In sub-task C, we also only make use of the OLID training set. Similarly, we only include samples that are labeled as TIN in sub-task B, since the goal of this task is only to determine the type of the targeted samples. The training set for this sub-task contains 3,876 samples and the test set contains 213 samples (table 3.1).

A	Train
OFF	20,620
NOT	4,163
All	**24,783**

Table 6.1: The distribution of samples by labels in the modified HSAOFL dataset.

6.1.2 Danish

- **Sub-task A - Offensive language identification**. In sub-task A, we use the full Danish training dataset in the training of all models. The training set for this sub-task contains 2,879 samples and the test set contains 721 samples (table 4.2).

- **Sub-task B - Automatic categorization of offensive language types**. In sub-task B we use a subset of the Danish training dataset, only including samples labeled as OFF in sub-task A (for the same reasons as mentioned earlier). The training set for this sub-task contains 352 samples and the test set contains 89 samples (table 4.2).

- **Sub-task C - Offensive language target identification**. In sub-task C, we use a subset of the Danish training dataset, only including

samples labeled as TIN in sub-task B (again for the same reasons as mentioned earlier). The training set for this sub-task contains 205 samples and the test set contains 47 samples (table 4.2).

6.2 Model Training and Testing

The Logistic Regression classifier (section 5.2.4) is trained for 100 iterations and the deep learning classifiers (Learned-BiLSTM, Fast-BiLSTM and AUX-Fast-BiLSTM) are trained for 100 epochs. In the deep learning classifiers we train each for 10 epochs at a time, with a batch size of 128. The epoch that provided the best result is then reported for each classifier in Chapter 7.

6.3 Evaluation

In the evaluation of our models we make use of pre-defined test sets for each sub-task within each language (Chapters 3 and 4). We use our models to predict the category of the samples in these sets and calculate some quality metrics from these predictions. The quality metrics used are Recall, Precision, F1-score, and macro averaged F1-score. These are the same metrics as were used in [42] and will therefore provide an easy way to directly compare our results. The following section describes these metrics in detail.

6.3.1 Quality Metrics

When working on classification tasks in general, the results will contain a mix of true positives (T_p), true negatives (T_n), false positives (F_p)

and false negatives (F_n). To try to evaluate the overall quality of the solutions, the first step is usually to look at the *Precision* and *Recall*. Precision (p) is defined as the number of true positives over the sum of true positives and false positives [2] (equation 6.1). This is the percentage of the classified items that are relevant. *Recall* (r), on the other hand, is defined as the number of true positives over the sum of true positives and false negatives [2] (equation 6.2). This is the percentage of total relevant results correctly classified.

$$p = \frac{T_p}{T_p + F_p} \tag{6.1}$$

$$r = \frac{T_p}{T_p + F_n} \tag{6.2}$$

Recall and precision are good metrics to look at, but they normally come at the cost of one another. If all samples are classified as a single class, that class will have perfect recall but low precision (and vice versa). For this reason a more balanced metric is required. F1-score is defined as the harmonic mean between precision and recall (equation 6.3), and gives an idea of how close the two are.

$$\text{F1-score} = \frac{2 \cdot p \cdot r}{p + r} \tag{6.3}$$

Since our task is of a multi-class nature, and the samples in our datasets are highly imbalanced, we use a macro averaged F1-score as the main evaluation metric for our classifiers This is the main metric used in [42] (table 2.1) and will, therefore, give us an easy way to compare our classifiers to the ones presented there. The macro averaged F1-score

evaluates each class independently and then calculates the unweighted average of the F1-scores [42]. Macro averaged F1-score provides some benefits over simpler metrics such as accuracy. The main benefit becomes clear when the data is highly skewed. Classifying all items as the most well represented class would result in good accuracy but low macro averaged F1-scores.

Chapter 7

Results and Analysis

This chapter details the results from the experiments described in Chapter 6. For each sub-task (A, B, and C, section 3.1) we present the results for all methods in each language, evaluated by their macro averaged F1-score. For the best performing system in each sub-task we also explore the *Recall*, *Precision*, and *F1-Score* for each category.

7.1 Experiments

7.1.1 Sub-task A - Offensive language identification

English. In table 7.1, the results from our experiments for sub-task A and the English language are presented. The best performing system is the Fast-BiLSTM model (section 5.2.7), trained for 100 epochs, using the OLID dataset. The model achieves a macro averaged F1-score of 0.735. Comparing these results to the results from the shared *OffensEval* task (table 2.1) puts our results in the same range as the BiLSTM based methods introduced there.

A somewhat surprising result is that the additional training data from the HSAOFL dataset [10] provides limited benefits and does not consistently improve the results of our models. For the models that solely make use of word embeddings (Learned-BiLSTM and Fast-BiLSTM) the results are actually worse with the additional training data. This is likely due to the fact that in these models we limit the sequence length of the embeddings based features to 100. Given that the datasets are vastly different and were collected at different times with different objectives, this likely causes the feature sequences to vastly differ, making it harder to detect notable patterns in the training data. On the other hand, in the models that make use of a range of additional features (Logistic Regression and AUX-Fast-BiLSTM), the additional features are created using the full sequence length of samples. In this case, the additional training data proves helpful, resulting in improved macro F1-scores.

Model	**Train Set**	**Macro F1**
All NOT	-	0.419
Logistic Regression	OLID	0.724
Learned-BiLSTM (10 Epochs)	OLID	0.707
Fast-BiLSTM (100 Epochs)	**OLID**	**0.735**
AUX-Fast-BiLSTM (10 Epochs)	OLID	0.692
Logistic Regression	OLID + HSAOFL	0.728
Learned-BiLSTM (10 Epochs)	OLID + HSAOFL	0.704
Fast-BiLSTM (100 Epochs)	OLID + HSAOFL	0.688
AUX-Fast-BiLSTM (20 Epochs)	OLID + HSAOFL	0.712

Table 7.1: Results from our experiments for sub-task A and English.

Danish. In table 7.2 the results from our experiments for sub-task A and the Danish language are presented. The best performing system is the Logistic Regression model (section 5.2.4), obtaining a macro averaged

F1-score of 0.699. This is the second best performing model for English while surprisingly the best performing model for English (Fast-BiLSTM) performs worst for Danish.

The low scores for Danish compared to English can likely be explained by the low amount of data in the Danish dataset. The Danish training set contains 2,879 samples (table 4.2) while the English training set contains 13,240 samples (table 3.1). Futhermore, it is worth stating that in the English dataset around 33% of the samples are labeled offensive while in the Danish set this rate is only at around 12%. The effect that this under represented class has on the Danish classification task can be seen in more detail in table 7.3, where the recall, precision, and F1-scores are presented by category for the best performing systems in both English and Danish. The differences in both recall and precision by category for the Danish language are far higher than for English, further indicating that the imbalance in the Danish set has a significant effect on the results. The models that make use of additional features (AUX-Fast-BiLSTM and Logistic Regression) perform better than the other two in Danish, indicating that a larger feature space can prove helpful for low resource languages and under represented classes.

Model	**Train Set**	**Macro F1**
All NOT	-	0.467
Logistic Regression	**DA**	**0.699**
Learned-BiLSTM (10 Epochs)	DA	0.658
Fast-BiLSTM (100 Epochs)	DA	0.630
AUX-Fast-BiLSTM (50 Epochs)	DA	0.675

Table 7.2: Results from our experiments for sub-task A and Danish.

Model	R NOT	R OFF	P NOT	P OFF	F1 NOT	F1 OFF
Fast BiLSTM EN	0.835	0.646	0.859	0.603	0.847	0.624
Logistic Regression DA	0.913	0.506	0.929	0.450	0.921	0.476

Table 7.3: Recall (R), precision (P), and F1 score by class for our best performing models in sub-task A. Baselines also included to get an idea of the class distribution.

7.1.2 Sub-task B - Automatic categorization of offensive language types

English. In table 7.4 the results are presented for sub-task B and the English language. The Learned-BiLSTM model (section 5.2.6) trained for 60 epochs performs the best, obtaining a macro F1-score of 0.619.

As seen in table 7.3, the recall and precision scores are significantly lower for the UNT category than the TIN category. One reason for these low scores is the fact that the training data used in this task is highly skewed, with only around 14% of the posts labeled as UNT. We can see that the pre-trained embedding model, Fast-BiLSTM, performs the worst, with a macro averaged F1-score of 0.567. This clearly indicates that this approach is not a good fit for detecting subtle differences in offensive samples in highly skewed data, while the more complex feature models seem to perform better.

Danish. Table 7.5 presents the results for sub-task B and the Danish language. The best performing system is the AUX-Fast-BiLSTM model (section 5.2.8) trained for 100 epochs, which obtains an impressive macro F1-score of 0.729. This supports our theory that simple models that only rely on pre-trained word embeddings are not the optimal approach for this task.

As seen in table 7.6, the limiting factor seems to be the recall for the UNT category, while the precision scores are in good sync. As

mentioned in Chapter 2, the best performing system for sub-task B in OffensEval was a rule based system, suggesting the fact that more refined features, such as manually constructed lexicons, can further improve the performance of models for this task. The better performance of our models for Danish compared to English can most likely be explained by the fact that the training set used for Danish is more balanced, with around 42% of the posts labeled as UNT.

Model	Train Set	Macro F1
All TIN	-	0.470
Logistic Regression	OLID	0.593
Learned-BiLSTM (60 Epochs)	**OLID**	**0.619**
Fast-BiLSTM (10 Epochs)	OLID	0.567
AUX-Fast-BiLSTM (50 Epochs)	OLID	0.595

Table 7.4: Results from our experiments for sub-task B and English.

Model	Train Set	Macro F1
All TIN	-	0.346
Logistic Regression	DA	0.594
Learned-BiLSTM (40 Epochs)	DA	0.643
Fast-BiLSTM (100 Epochs)	DA	0.681
AUX-Fast-BiLSTM (100 Epochs)	**DA**	**0.729**

Table 7.5: Results from our experiments for sub-task B and Danish.

Model	R UNT	R TIN	P UNT	P TIN	F1 UNT	F1 TIN
Learned BiLSTM EN	0.370	0.892	0.303	0.918	0.333	0.905
AUX-Fast-BiLSTM DA	0.690	0.766	0.725	0.735	0.707	0.750

Table 7.6: Recall (R), precision (P), and F1 score by class for our best performing models in sub-task B. Baselines also included to get an idea of the class distribution.

7.1.3 Sub-task C - Offensive language target identification

English. The results for sub-task C and the English language are presented in table 7.7. The best performing system is the Learned-BiLSTM model (section 5.2.6) trained for 10 epochs, obtaining a macro averaged F1-score of 0.557. This is an improvement over the models introduced in [41], where the BiLSTM based model achieves a macro F1-score of 0.470 (table 2.1).

The main limitations of our model seems to be in the classification of OTH samples, as seen in table 7.9, where both the recall and precision are significantly lower for that category. This can, again, be explained by the imbalance in the training data (table 3.1), where only around 10% of the samples are labeled as OTH. It is, however, interesting to see that this imbalance does not effect the GRP category as much, which only constitutes about 28% of the training samples. A probable cause for the differences in these, is the fact that the definitions of the OTH category are vague, capturing all samples that do not belong to the previous two.

Danish. Table 7.8 presents the results for sub-task C and the Danish language. The best performing system is the same as in English, the Learned-BiLSTM model (section 5.2.6), trained for 100 epochs, obtaining a macro averaged F1-score of 0.629. Given that this is the same model as the one that performed the best for English, this further indicates that task specific embeddings are helpful for more refined classification tasks.

It is interesting to see that both of the models using the additional set of features (Logistic Regression and AUX-Fast-BiLSTM) perform the worst. This indicates that these additional features are not beneficial for

this more refined sub-task in Danish. It is, however, worth mentioning that the amount of samples used in training for this sub-task is very low, consisting of only 205 samples (table 4.2). The imbalance in the dataset does not seem to have as much effect in Danish as it does in English, as can be seen in table 7.9. Only about 14% of the samples are labeled as OTH in the training set (table 4.2), but both the recall and precision scores are much closer than they are for English.

Model	Train Set	Macro F1
All IND	-	0.213
Logistic Regression	OLID	0.458
Learned-BiLSTM (10 Epochs)	**OLID**	**0.557**
Fast-BiLSTM (50 Epochs)	OLID	0.516
AUX-Fast-BiLSTM (40 Epochs)	OLID	0.536

Table 7.7: Results from our experiments for sub-task C and English.

Model	Train Set	Macro F1
All GRP	-	0.219
Logistic Regression	DA	0.438
Learned-BiLSTM (100 Epochs)	**DA**	**0.629**
Fast-BiLSTM (60 epochs)	DA	0.579
AUX-Fast-BiLSTM (100 Epochs)	DA	0.401

Table 7.8: Results from our experiments for sub-task C and Danish.

Model	R IND	R GRP	R OTH	P IND	P GRP	P OTH	F1 IND	F1 GRP	F1 OTH
Learned-BiLSTM EN	0.670	0.667	0.343	0.770	0.634	0.273	0.717	0.650	0.304
Learned-BiLSTM DA	0.556	0.696	0.667	0.667	0.640	0.571	0.606	0.667	0.615

Table 7.9: Recall (R), precision (P), and F1 score by class for our best performing models in sub-task C. Baselines also included to get an idea of the class distribution.

7.2 Analysis of Deep Learning Classifiers

We perform some analysis of our deep learning classifiers by training the best performing systems and withholding 10% of the training samples to use as a validation set. The models are trained on the remaining training set, following the same procedure as discussed in Chapter 6, while the validation set is not used in the training of the models. After each epoch the training and validation loss is calculated using the *Binary Crossentropy* loss function. The goal of this analysis is to determine if the best performing deep learning models are over-fitting. In order to determine this, we plot the training and validation loss curves over the training epochs. If the curve of the validation loss starts to diverge from the training loss curve it indicates over-fitting. Other than the change in the training set sizes, the model parameters are the same as discussed in section 5.2.

7.2.1 Sub-task A - Offensive language identification

In sub-task A the best performing model for the English language, according to our experiments in section 7.1.1, is the Fast-BiLSTM classifier (section 5.2.7). In figure 7.1 the train and validation loss are plotted over the training epochs. The validation loss curve diverges from the training loss curve at around 40 epochs indicating that the model is indeed over-fitting to some extent.

For the Danish language the best performing system, according to our experiments in section 7.1.1, is the Logistic Regression classifier (section 5.2.4). Given the fact that we cannot analyze the Logistic Regression classifier in the same manner as the deep learning models it will not be included in this section.

Figure 7.1: The train and validation loss curve for the Fast-BiLSTM classifier for sub-task A and the English language.

7.2.2 Sub-task B - Automatic categorization of offensive language types

For sub-task B and the English language, the best performing system, according to our experiments in section 7.1.2, is the Learned-BiLSTM classifier (section 5.2.6). In figure 7.2 (a), the validation and training loss curves are plotted. We can see that the model over-fits aggressively after just 10 epochs, and the training loss approaches 0 as the number of epochs increases. A possible reason for this aggressive over-fitting is the fact that in the Learned-BiLSTM classifier, all parameters of the model are updated during training, including the word-embeddings.

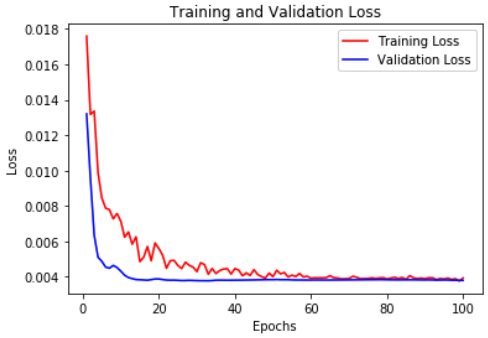
For the Danish language the best performing system, according to our experiments in section 7.1.2, is the AUX-Fast-BiLSTM classifier (section 5.2.8). In figure 7.2 (b) the training and validation loss curves are plotted. There is not a clear indication of over-fitting, and interestingly, the validation loss remains constantly lower than the training loss. A possible explanation for this is the fact that the validation set is quite small, only containing 35 samples.

The AUX-Fast-BiLSTM classifier should be more unlikely to over-fit than the Learned-BiLSTM classifier, since the embedding layer is initial-

ized with pre-trained FastText [24] embeddings (section 5.1.8) and these word-embeddings are not updated during training.

Figure 7.2: The train and validation loss for the best performing models in sub-task B for each language.

(a) Learned-BiLSTM - English



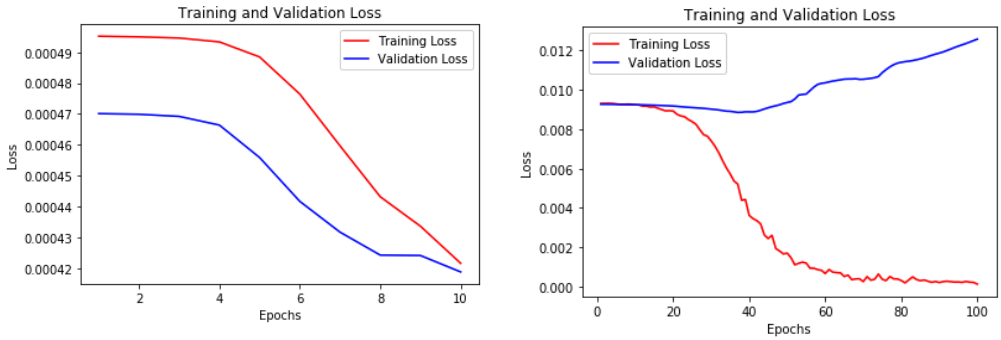
(b) AUX-Fast-BiLSTM - Danish

7.2.3 Sub-task C - Offensive language target identification

In sub-task C the best performing system for both English and Danish, according to our experiments in section 7.1.3, is the Learned-BiLSTM classifier (section 5.2.6). Note, however, that for English, the classifier is only trained for 10 epochs, while the classifier for Danish is trained for 100 epochs. In figure 7.3 the training and validation loss curves are plotted for each language. After 10 epochs for the English language, the classifier does not show any signs of over-fitting.

We can, however, see that the classifier for the Danish language behaves similarly to the Learned-BiLSTM classifier for English and sub-task B (section 7.2.2), where the validation loss diverges after only 20 epochs. The Learned-BiLSTM is, therefore, quite troublesome when it comes to over-fitting. A possible solution might be to decrease the size of the embedding layer, downgrading the number of parameters that can be tuned during training.

Figure 7.3: The train and validation loss for the best performing models in sub-task C for each language.



(a) Learned-BiLSTM - English (b) Learned-BiLSTM - Danish

7.3 Error Analysis

We perform analysis of the misclassified samples in the evaluation of our best performing models. To accomplish this, we compute the TF-IDF scores for a range of n-grams (section 5.1.7). We then take the top scoring n-grams in each category and try to discover any patterns that might exist. We also perform some manual analysis of these misclassified samples. The goal of this process is to try to get a clear idea of the areas our classifiers are lacking in. The following sections describe this process for each of the sub-tasks.

7.3.1 Sub-task A - Offensive language identification

English. The classifier seems to struggle with identifying obfuscated offensive terms. This includes words that are concatenated together, such as *barrrysoetorobullshit*. The classifier also seems to associate *she* with offensiveness, and samples containing *she* are misclassified as offensive in several samples while *he* is more rarely associated with offensive language.

We discover several examples where our classifier classifies posts containing profanity as offensive that are labeled as non-offensive in the test set. Posts such as *Are you fucking serious?* and *Fuck I cried in this scene* are labeled non-offensive in the test set, but according to the definitions introduced in the annotation guidelines (Chapter 3) these posts should indeed be classified as offensive, which raises some concerns about the quality of the annotations in the test data.

Danish. For Danish, the best classifier seems to be inclined to classify longer sequences as offensive. The mean character length of the samples misclassified as offensive is 204.7, while the mean character length of the samples misclassified as not offensive is 107.9. This can be explained by the fact that sub-task A categorizes every post containing any form of profanity as offensive, so more words increase the likelihood of some of them being profanity.

The classifier also seems to suffer from the same limitations as the classifier for English when it comes to obfuscated words, misclassifying samples such as *Hahhaaha lær det biiiiiaaaatch* as non-offensive. It also seems to associate the occurrence of the word *svensken* with offensive language, and quite a few samples containing that word are misclassified as offensive. This can be explained by the fact that offensive language towards Swedes is common in the training data, resulting in this association. From this, we can conclude that the classifier relies too much on the presence of individual keywords, and is lacking in interpreting the context of these keywords.

7.3.2 Sub-task B - Automatic categorization of offensive language types

English. In sub-task B the obfuscation problem seems to prevail. Our classifier misses clear indicators of targeted insults such as *WalkAwayFromAllDemocrats*. It also seems to rely too highly on the presence of profanity, misclassifying samples containing terms such as *bitch, fuck, shit, etc.* as targeted insults.

The issue of the data quality is also concerning in this sub-task, as we discover samples containing clear targeted insults such as *HillaryForPrison* being labeled as untargeted in the test set.

Danish. Our Danish classifier also seems to be missing obfuscated words such as *kidsarefuckingstupid* in the classification of targeted insults. It relies to some extent to heavily on the presence of profanity such as *pikfjæs, lorte* and *fucking*, and misclassifies untargeted posts containing these keywords as targeted insults.

7.3.3 Sub-task C - Offensive language target identification

For both English and Danish, misclassification based on obfuscated terms as discussed in the previous sections seems to be an issue for sub-task C as well. Given the fact that our classifiers mostly rely on word-level information, the problem of obfuscated terms could be tackled by introducing character level features such as character level n-grams.

Chapter 8

Conclusion

Offensive language on online social media platforms is becoming a major liability for most of the major platforms. One of the more severe type of offensive language is hate speech, and the presence of hate speech has been shown to be in correlation with hate crimes in real life settings [25]. Due to the vast amount of user-generated content on online platforms, automatic methods are required to detect this kind of harmful content. Until now, most of the research on the topic has focused on solving the problem for English. Due to this fact, we explore and develop automatic detection systems to tackle the problem of offensive speech for both the English and the Danish language.

8.1 Contribution

Danish dataset. We constructed and annotated a Danish dataset for the task of offensive language detection (Chapter 4), shaped by the structure introduced in Chapter 3. To our knowledge, this is the first Danish dataset developed for the task at hand. The dataset consists of 3600

user-generated comments from the social media platforms Facebook and Reddit.

Danish hate speech lexicon. We constructed a Danish hate speech lexicon (section 4.2), and to our knowledge, this lexicon is the first of its kind. The lexicon was constructed from survey data on the topic posted on Reddit, targeted at native Danish speakers in the community. The lexicon consists of 113 offensive and hateful terms in Danish, and is presented in Appendix A.

Automatic detection systems. We successfully developed automatic systems for offensive language identification, the categorization of offensive language types and offensive language target identification following the task structure discussed in Chapter 3. We train our systems to work with both the English and Danish language, and to our knowledge, this is the first time a system of this kind has been developed for Danish. In the task of offensive language identification (sub-task A, section 3.1) our best performing system for the English language is the Fast-BiLSTM classifier (section 5.2.7) which achieves a macro averaged F1-score of 0.735. For Danish, the best performing system is the Logistic Regression classifier (section 5.2.4) which achieves a macro averaged F1-score of 0.699. In the categorization of offensive language types (sub-task B, section 3.1) the best performing system for the English language achieves a macro averaged F1-score of 0.619 using the Learned-BiLSTM classifier (section 5.2.6). For Danish, the best performing system uses the AUX-Fast-BiLSTM classifier (section 5.2.8), achieving a macro averaged F1-score of 0.729. Finally, in the offensive language target identification (sub-task C, section 3.1) the best performing system

is the Learned-BiLSTM classifier (section 5.2.6), which achieves a macro averaged F1-score of 0.557 for English and 0.629 for Danish.

8.2 Future Work

During the time of our work we have been in contact with the authors of [41] and [42]. Given the fact that our dataset is shaped after the guidelines proposed by [41], we will contribute our Danish dataset and our Danish hate speech lexicon (Chapter 4) to a shared task on the topic, *OffensEval 2020*, managed by the authors of [42]. The task will be multilingual in nature, where the goal is to develop offensive language detection systems following the structure introduced in Chapter 3 for multiple languages (including English and Danish).

Appendix A

Danish Hate Speech Lexicon from Reddit Survey

- bekvem
- bessefar
- bondejokke
- bondeknold
- bonderøv
- bondsk
- bæskubber
- bøssekarl
- establishment
- fisefornem
- fjeldabe
- fjæs
- floskelmager
- flæbe
- fløde bolle
- frøæder
- gnom
- hadsprædikant
- hedenskab
- hjemmefødning
- kopist
- kraftidiot

- krigsliderlig
- kvasiintellektuel
- kvindagtig
- lebbe
- lort
- ludder
- middelalderlig
- møgunge
- nigger
- offergøre
- offergørelse
- papmor
- partout
- perker
- pigebarn
- pigefnidder
- plapre
- plasticmor
- røvhul
- skaffedyr
- skrælling
- slipsedyr
- snerpe
- snotdum
- snotunge
- spastiker
- stikker
- støjbergsk
- svans
- symbolpolitik
- torsk
- tude
- tyskertøs
- vatpik
- Amatører
- bidesild
- bløddyr
- bollefjæs
- fedtefyre
- hundehoveder
- fnatmider

- fæhoveder
- grødbønder
- hængerøve
- ignoranter
- jammerkommoder
- karklud
- klamhuggere
- klodsmajor
- lusepustere
- narrehatte
- pattebørn
- pjalt
- pjok
- pudseklud
- skidespræller
- skvadderhoveder
- skvat
- skvatpissere
- slapsvanse
- snotklatte
- elendige socialdemokrater
- Sindssyge
- kvindemenneske
- svabrefjams
- Hestepære
- kolort
- kolibriafføring
- myggefjols
- kældernisse
- buskerusker
- hårtygger
- våben
- ledningsfletter
- højreradikal
- højreekstremist
- fremmedfjendsk
- nynazist
- kartoffel
- ny bruger
- kvindehader
- hvid

- privilegeret
- heteronormativ
- undertrykker
- krænker
- kristen
- muslimer
- multikultur
- nazist
- sort

Bibliography

[1] Eu council framework decision 2008/913/jha. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=LEGISSUM%3Al33178. Accessed: 2019-05-29.

[2] Scikit learn - metrics. http://scikit-learn.org. Accessed: 2019-04-19.

[3] Straffeloven § 266 b. https://danskelove.dk/straffeloven/266b. Accessed: 2019-05-29.

[4] Stochastic Gradient Descent. https://scikit-learn.org/stable/modules/sgd.html. Accessed: 2019-05-29.

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[6] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760. International World Wide Web Conferences Steering Committee, 2017.

[7] James Banks. Regulating hate speech online. *International Review of Law, Computers & Technology*, 24(3):233–239, 2010.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] François Chollet et al. Keras. https://keras.io, 2015.

[10] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. Automated hate speech detection and the problem of offensive language. In *Eleventh International AAAI Conference on Web and Social Media*, 2017.

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

[12] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[13] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[14] Trudy M Gregorie. Cyberstalking: Dangers on the information superhighway. *National Center for Victims of crime*, 2001.

[15] Lieve Hamers et al. Similarity measures in scientometric research: The jaccard index versus salton's cosine formula. *Information Processing and Management*, 25(3):315–18, 1989.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.

[18] Sarah Joseph and Melissa Castan. *The international covenant on civil and political rights: cases, materials, and commentary*. Oxford University Press, 2013.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[21] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[22] Edward Loper and Steven Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[25] Karsten Müller and Carlo Schwarz. Fanning the flames of hate: Social media and hate crime. *Available at SSRN 3082972*, 2018.

[26] Finn Årup Nielsen. A new anew: Evaluation of a word list for sentiment analysis in microblogs. *arXiv preprint arXiv:1103.2903*, 2011.

[27] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153. International World Wide Web Conferences Steering Committee, 2016.

[28] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web*, pages 145–153. International World Wide Web Conferences Steering Committee, 2016.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[30] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[31] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[32] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns. *arXiv preprint arXiv:1707.06961*, 2017.

[33] Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529*, 2016.

[34] Sean R Eddy. Hidden markov models. *Current Opinion in Structural Biology*, 6:361–365, 06 1996.

[35] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[36] Anna Schmidt and Michael Wiegand. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10, 2017.

[37] Cynthia Van Hee, Els Lefever, Ben Verhoeven, Julie Mennes, Bart Desmet, Guy De Pauw, Walter Daelemans, and Véronique Hoste. Detection and fine-grained classification of cyberbullying events. In *Proceedings of the international conference recent advances in natural language processing*, pages 672–680, 2015.

[38] Cynthia Van Hee, Ben Verhoeven, Els Lefever, Guy De Pauw, Véronique Hoste, and Walter Daelemans. Guidelines for the fine-grained analysis of cyberbullying. 2015.

[39] Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. Understanding abuse: A typology of abusive language detection subtasks. *arXiv preprint arXiv:1705.09899*, 2017.

[40] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL student research workshop*, pages 88–93, 2016.

[41] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Predicting the type and target of offensive posts in social media. *arXiv preprint arXiv:1902.09666*, 2019.

[42] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). *arXiv preprint arXiv:1903.08983*, 2019.