A Critical Evaluation of Hate Speech Classifiers

Vinícius Miranda
College of Computational Sciences, Minerva Schools at KGI
Data Sciences and Artificial Intelligence

May 20, 2020

Abstract

In recent years, the spread of hate speech has attracted the attention of academia, governments, and industry alike in response to its dire consequences, online and otherwise. For example, the United Nations Human Rights Council (2018) implicated social media platforms in contributing to the genocide in Myanmar by allowing the systematic dissemination of hatred against Rohingya Muslims. The sheer scale of the generation of online content motivates the necessity of automatic moderation mechanisms, such as the classification and removal of hate speech. The first achievement of this research project is to provide a critical evaluation of the state of the art of the hate speech classification literature by replicating and analysis of three prominent papers. The second contribution is to provide the blueprint of a generalizable training strategy for a hate speech classifier that incorporates transfer learning strategies in Natural Language Processing (NLP).

Keywords: Hate speech, natural language processing, transfer learning, neural networks, deep learning, classification.

Contents

1 Introduction

The spread of hate speech online has united a rare coalition of governments, international organizations, and private companies under the same goal of preventing its rise. In May 2016, Facebook, Microsoft, Twitter, YouTube, and the European Commission came together under a landmark code of conduct focused on combating hate speech (European Commission, 2019). The consequences of such speech can be incredibly dire. The United Nations Human Rights Council (2018) has implicated Facebook on the dissemination of hatred against the Rohingya Muslims in Myanmar. This genocide illustrates the relevance and urgency of action. The moderation of content by human agents, however, cannot effectively counter the issue when Facebook alone boasts of 1.52 billion daily active users (Facebook, n.d.). Hence, there is a growing consensus that machine-learning applications are necessary to tackle this task at scale, with the Centre on Regulation in Europe "highlighting that AI may trigger important efficiency gains in this area [of hate speech content moderation]" (Finck, 2019).

The objective of this Capstone project is to contribute to the research on automated hate speech detection by providing a critical appraisal of seminal research and achieving state-of-the-art performance with a new classifier architecture. Specifically, the main contributions are as follows. (1) An analysis of the replication attempt of three papers associated with substantial breakthroughs in hate speech classification. (2) The employment of **data augmentation** to counter the **class imbalance** present in most publicly available hate speech datasets.[1] (3) To the best of my knowledge, this is the first application of **transfer learning** to training a hate speech classifier. The project will produce both a research paper and accompanying publicly available Python code to ensure the reproducibility of the results.

2 Literature Review

This section provides an overview of recent developments in the classification of hate speech. Several authors have recently presented comprehensive reviews of the literature. Fortuna (2017), for example, summarized several approaches to hate speech classification while providing a new annotated dataset in Portuguese. Hemker (2018), similarly, focused on a more technical and in-depth overview of classification strategies and implemented a new classifier architecture based on combining different types of neural networks. Schmidt and Wiegard (2017) focused on a more specific review of **Natural Language Processing (NLP)** approaches to this classification task. Due to constant innovations in the field of NLP, even a single year can render these reviews incomplete. The most current example is the utilization of transfer learning techniques, which started gaining traction only in 2018 (e.g., Howard & Ruder, 2018).

Other papers in the field have a relatively narrow scope and attempt to improve the **performance** of classifiers. Badjatiya et al. (2017), for instance, were the first to apply **deep learning**

[1] Class imbalance is a particularly pressing issue in this line of research since most publicly available hate speech datasets are comparatively small. For example, all the Twitter datasets investigated by Gröndahl et al. (2018) contain only a few hundreds or thousands of hate speech data points. This number is too low for most algorithms, especially considering the complexity of hate speech.

approaches to hate speech classification, significantly improving the state-of-the-art performance at the time. Since then, the application of deep learning methods to the task gained traction, leading to mixed results. Biere (2018), for instance, implemented a simple version of a **neural network** classifier with poor results, showing that these models need to be implemented with care. Risch and Krestel (2018), in contrast, applied a creative **ensemble** of deep learning and **classical approaches** to an aggressiveness classification task leading to an impressive performance.

Whereas the issue of performance has propelled the field forward, many authors have investigated other dimensions of automated classification, including robustness to attacks, interpretability, the number of classes to classify, and the value of context. For instance, Gröndahl et al. (2018) examined ways to fool classifiers by simulating the ways users might try to evade automatic detection by changing their message in a way that preserved the content while changing its format. Wang (2018) applied standard computer vision methods to analyze the inner workings of neural networks, uncovering several shortcomings that could be addressed by future research. Davidson, Warmsley, Macy, and Weber (2017) were the first to annotate a dataset with a non-binary class breakdown. In other words, they classified tweets as hate speech, offensive speech, or neither, theorizing that discriminating between hate and offensive speech might help the classifier to abstract the difference between these two classes. Finally, Gao and Huang (2017) showed that context information (e.g., username and the title of the news article under which the user posted a comment) could further improve classification performance.

Since the field of natural language processing has evolved at an incredible pace in recent years, there is an opportunity to apply recent breakthroughs to the task of classifying hate speech. This Capstone project aims to provide a critical appraisal of the literature while providing a novel implementation of a deep learning model powered by transfer learning. In the sections that follow, I provide a full description of the classification task and its challenges, including background on the definition of hate speech, an account of data quality concerns, and a consideration of recent methodological attempts to tackle the problem of classification.

2.1 What is Hate Speech?

A universal definition of hate speech does not exist. Whereas several legislative bodies across the world have encoded protections against hate speech into law, the concept of what constitutes hate speech may vary substantially. Furthermore, researchers, technology companies, and international organizations have also devised their definitions. Despite these variations, the European Union has been influential in legislating and prosecuting hate speech for several years, defining it as

> *"[the] public incitement to violence or hatred directed against a group of persons or a member of such a group defined on the basis of race, colour, descent, religion or belief, or national or ethnic origin."* (Framework Decision 2008/913/JHA, 2008)

Fortuna (2017) created a comparative analysis of several hate speech definitions and found the requirement that the speech should have specific targets to be a common thread among all definitions surveyed. The condition is reflected in the EU conceptualization since it requires speech to be directed against a group or member thereof. Other conceptualizations of hate speech vary and may be concerned with the explicit incitation of violence, the intention to attack or diminish, or may provide attempts at humor with special status, where otherwise hateful speech is considered

	Explicit	*Implicit*
Directed	"Go kill yourself", "You're a sad little f*ck" (Van Hee et al., 2015a), "@User shut yo beaner ass up sp*c and hop your f*ggot ass back across the border little n*gga" (Davidson et al., 2017), "Youre one of the ugliest b*tches Ive ever fucking seen" (Kontostathis et al., 2013).	"Hey Brendan, you look gorgeous today. What beauty salon did you visit?" (Dinakar et al., 2012), "(((@User))) and what is your job? Writing cuck articles and slurping Google balls? #Dumbgoogles" (Hine et al., 2017), "you're intelligence is so breathtaking!!!!!!" (Dinakar et al., 2011)
Generalized	"I am surprised they reported on this crap who cares about another dead n*gger?", "300 missiles are cool! Love to see um launched into Tel Aviv! Kill all the g*ys there!" (Nobata et al., 2016), "So an 11 year old n*gger girl killed herself over my tweets? ^_^ thats another n*gger off the streets!!" (Kwok and Wang, 2013).	"Totally fed up with the way this country has turned into a haven for terrorists. Send them all back home." (Burnap and Williams, 2015), "most of them come north and are good at just mowing lawns" (Dinakar et al., 2011), "Gas the skypes" (Magu et al., 2017)

Figure 1: A simple typology of hate speech in two different dimensions.
Source: Waseem et al. (2017).

benign due to the use of irony or sarcasm (Fortuna, 2017). These variations create a grey area of offensive speech that may be characterized as hateful by some definitions and not others. When considering the task of automating hate speech classification, we should note that the definition of what constitutes hate speech is a property of the dataset that is utilized for training the model. The architecture of our model does not influence what is considered hate speech or not and aims solely to abstract a definition from the data on which it is trained.

2.2 Hate Speech Typology

Hate speech can be broken down into different types according to its features. Waseem, Davidson, Warmsley, and Weber (2018) provides a thoughtful discussion of some of these features and how they can lead a typology of hate speech, such as the one shown in Figure 1.

Whereas explicit hate speech makes use of curse words, the implicit variation is more nuanced and may avoid expletives in favor of sarcasm and ambiguous terms. Implicit hate speech is thus harder to detect since models do not have a signal as strong as a hateful keyword to facilitate the detection. Furthermore, hate speech may vary along an axis of directness. Direct hate speech is aimed at a single individual who is explicitly referred to or mentioned, whereas generalized hate speech targets a group of people with a shared characteristic.

Most datasets label the data only as hate speech or not, hence not taking advantage of any nuance within these classes (e.g., Wulczyn, Thain, & Dixon, 2017; Waseem & Hovy, 2017). Davidson et al. (2017) were the first to demonstrate the benefit of finer labeling by annotating a dataset with a third class that distinguished between hate speech and merely offensive speech, which led to improved results. Hemker (2018) went further and broke down the hate speech class of the Davidson et al. (2017) dataset into directed (i.e., targeting a person) or generalized (e.g., a group) hate speech. Contrary to expectations, however, the breakdown worsened the performance of the classifier. One possible explanation is that the problem of small data becomes even more accentuated after the breakdown since the subcategories naturally contain even less data than the full hate speech class, which tends to be underrepresented in hate speech datasets.

Table 1: Class breakdown in four datasets. Sourced from Gröndahl et al. (2018).

Dataset	Domain	Classes (size)	Source
W	Wikipedia	Personal attacks (13590) Ordinary (102274)	Wulczyn, Thain, and Dixon (2017)
T1	Twitter	Hate speech (1430) Offensive (19190) Ordinary (4163)	Davidson et al. (2017)
T2	Twitter	Racist ∪ Sexist (5013) Ordinary (10796)	Waseem and Hovy (2016)
T3	Twitter	Hateful / Racist (414) Ordinary (2021)	Zhang, Robinson, and Tepper (2018)

2.3 Data

The hate speech classification literature is built upon several datasets labeled by researchers in recent years. Unfortunately, there is no single dataset that serves as a benchmark for this field, which renders it difficult to judge the relative merits of different classification strategies or compare the outcomes of different papers. Most of the research focuses on hate speech in English, with sparse studies focusing on other languages such as Dutch, Italian, German, and Portuguese (Tulkens, Hilte, Lodewyckx, Verhoeven & Daelemans, 2016; Ross et al., 2017; Vigna, Cimino, Dell'Orletta, Petrocchi, & Tesconi, 2017; Fortuna, 2017).

Besides the greater availability of English-language datasets, data processing methods in English are also more readily available and robust, such as the pre-trained word embeddings discussed in the *Methodology* section. Hence, this paper will also focus exclusively on hate speech in English. Gröndahl et al. (2018) compiled four different datasets for their analysis, shown in Table 1. The most widely referenced dataset seems to be the one created by Davidson et al. (2017), used by Hemker (2018), Biere (2018) and several others.

2.3.1 Inter-Rater Agreement

One inevitable problem with data on hate speech is the subjectivity of categorization. When two people assign different labels to a datum, there is inter-rater disagreement. The level of inter-rater disagreement is an essential measure of data quality since it underscores how much uncertainty there is in the labels. Furthermore, a high level of disagreement indicates subjectivity in the classification task, which in turn means that it is harder for an algorithm to abstract a pattern from the data. In the T1 dataset, each datum was labeled independently by at least three people, and the category with most votes was chosen as the final label. For example, if two evaluators considered a given message offensive and another considered it hate speech, the message would receive the label of offensive. 87.4% of the data points categorized as hate speech in this dataset generated disagreement among evaluators, underscoring the high level of subjectivity in the annotation. The dataset provides a breakdown of how each rater voted. However, to the best of my knowledge, that information has not been used in model training.

2.3.2 Labeler Bias

The definition of hate speech utilized by labelers and their level of disagreement are relevant latent properties of a hate speech dataset. A final property is the collection of biases from labelers themselves. For example, if a labeler has an inclination to classify racist slurs as hate speech but sexist comments as merely offensive, such inclination will also manifest in the data. In truth, this bias has been noted by both Davidson et al. (2017) and Waseem and Hovy (2016). In seeking to classify the data correctly, the model will also reproduce the same biases. This limitation underscores the importance of data quality and robust labeling procedures since they have repercussions on the fairness of final models irrespective of the model's architecture and training process.

2.4 Class Imbalance and Data Augmentation

A common issue with hate speech datasets is that they are imbalanced, meaning that only a small minority of the data is labeled as hateful. This imbalance derives from the fact that although hate speech is a systematic problem, it still constitutes a small minority of all speech on the internet. The problem with imbalanced datasets is that they can cause machine learning algorithms to overfit on the majority class since they can attain high scores on traditional measures of performance (e.g., accuracy) solely by doing well on the majority class. On the S1 dataset, for example, only 5.7% of the data is hate speech. It is common, then, for classifiers to attain an accuracy higher than 90% on the whole dataset while having a very low **recall** on the hate speech data. Biere (2018), for example, achieved an overall accuracy of 91% while incorrectly classifying 80% of the hate speech class.

Hemker (2018) and Risch and Krestel (2018) tackled the problem of class imbalance by augmenting their data. Data augmentation is a process through which the representation of the data is transformed in a way that the algorithm processes it as a different input, whereas its real-world interpretation is maintained. In textual data, an example of augmentation is the replacement of a word by its synonym since we change one of the words of the message without changing its meaning. Risch and Krestel (2018) attempted to pursue this strategy by applying a cyclical translation scheme in which they translated a sentence from English to another language and then back to English. Although there is no guarantee that the augmented sentence's meaning is indeed maintained, the authors observed marginally better results. The scheme is, however, too simplistic due to the highly nuanced nature of hate speech. Words can have a particular connotation that does not transfer to other words with similar denotational meaning.

This shortcoming motivated Hemker's (2018) implementation of a more sophisticated procedure called threshold augmentation, which was far more successful. The method takes advantage of word embeddings (see *Methodology*) that guarantee that words with similar embeddings are employed in similar contexts, securing alignment in both their denotational and connotational meanings.

2.5 Models

2.5.1 Statistical Methods

The hate speech classification literature has traditionally employed statistical methods such as logistic regression, support vector machines, and tree-based models such as random forests and XGBoost.[2] Davidson et al. (2017), for example, tested several models to find that a logistic regression classifier provided the best results. Waseem and Dovy (2016) also reported good results on a different dataset with a logistic regression classifier. More recently, researchers such as Badjatiya et al. (2017) have implemented classifiers that use these methods in conjunction with deep learning approaches, a strategy that has significantly improved results. Nowadays, traditional statistical methods are rarely featured in state-of-the-art research, which has focused on deep learning approaches instead, due to their better performance.

2.5.2 Deep Learning

Badjatiya et al. (2017) claim to have been the first to apply deep learning models to the task of hate speech classification. Several other researchers have since followed suit and have actively pushed the state-of-the-art performance with innovative designs of neural networks. Such designs are varied and include hybrids of different network types and, sometimes, their combination with classic statistical models. The most common network types are convolutional neural networks (CNNs), recurrent neural networks (RNNs), and MLPs (multilayer perceptrons). CNNs are most often utilized in computer vision applications but can work well as feature extractors across different application domains. RNNs are known for their ability to model sequential data and are thus particularly well-suited for natural language processing. Finally, MLPs represent a simpler architecture that may be appended to the end of a model as a general-purpose classifier. All these architectures are explained in greater technical detail in the *Methodology* section.

Most recently, Hemker (2018) reported an unprecedented 84% recall on the hate speech class of the T1 dataset. The result is possibly driven by the introduction of new data augmentation techniques, comprehensive hyperparameter tuning of the deep learning models, and an innovative classifier architecture. The author employed an effective combination of the three architectures discussed above. One principal goal of this project is to check the reproducibility of the results obtained by Hemker (2018).

2.6 Generalizability and Robustness

Gröndahl et al. (2018) were the first to explore how classifiers fared when classifying data from datasets other than the one with which they were trained. Their goal was to test whether the performance of classifiers generalized to other datasets and how vulnerable they were to adversarial input. Input is said to be adversarial when the data is purposefully manipulated with the intent of leading the algorithm to classify it incorrectly (i.e., to classify hate speech as non-hateful). The authors showed that models do not generalize well, most likely due to the reasons associated with data quality discussed above, such as diverging definitions of hate speech across datasets and high

[2] XGBoost is a powerful tree-based machine learning algorithm.

levels of inter-rater disagreement. Furthermore, they found that some models and preprocessing strategies are more vulnerable to simple attacks than others. For example, word-based models (i.e., those which model text as a sequence of words instead of as a sequence of characters) completely collapse if one removes the white spaces between words.

2.7 Interpretability

One prominent shortcoming of deep learning approaches is that it is tough to understand *how* classification occurs. The field of computer vision has devised creative ways of interpreting how neural networks process input, and Wang (2018) aimed to translate these techniques to hate speech classification. The author performed **occlusion tests** to analyze the sensitivity of a neural network classifier to different regions of the input text. In these tests, we take a single word of the input sentence at a time, and we replace it with a token that represents an unknown or out-of-vocabulary word. Then, we focus on the probability of the class that the classifier initially predicts for the full sentence.[3] We analyze the probabilities of this class for all the spin-off sentences we generated, which gives us a measure of the importance of each word to the full-sentence prediction. Figure 2 presents a summary of four undesirable features of the algorithm uncovered by these tests.

For the sake of an example, let us focus on the issue of localization. Take a five-word hate speech sentence that is misclassified as offensive speech. We would generate five spin-off sentences where the unknown token replaced each word at a time. Since the classifier predicts that the sentence is offensive, we focus on the probabilities assigned to the offensive class. If the probability when the first word is replaced is close to one, we interpret that the classification of the sentence as offensive remains mostly the same even in the absence of the first word. Conversely, if the probability when the last word is occluded is close to zero, we reason that the sentence would no longer be classified as offensive *in virtue of* the absence of the last word. Hence, this word is critical to the original classification of the full sentence.

Concerning localization, we say that there is a problem of overlocalization if the classifier is sensitive to only one or two words even though, upon looking at the sentence, we can see that other words should also be regarded as similarly relevant. Overlocalization is a problem in data where the hateful content is spread over the entire message, leading to misclassification. Efforts to interpret deep learning methods can provide valuable insight into the shortcomings of current state-of-the-art algorithms and give direction to future research efforts that aim to improve classifiers.

2.8 Summary of Current Research

This section presented a non-technical overview of the current directions of the research and some of the principal challenges of developing a hate speech classifier. The underlying limitation present

[3]Here, it is essential to distinguish between the predicted *probabilities* and predicted *class* given by a neural network model. These models output a vector of probabilities for each class in the classification problem, which we can then use to find the class assigned with the largest probability. For example, the vector (0.1, 0.2, 0.7) shows the predicted *probabilities* for each class. With it, we infer that the third class is the predicted class since it has the highest probability.

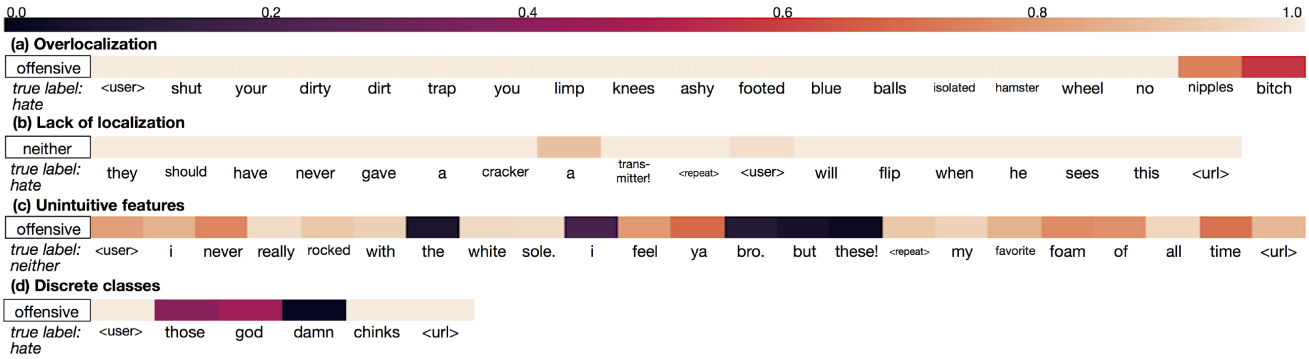


Figure 2: Wang (2018) performed occlusion tests to show some undesirable features of a deep learning classifier. The darker the tone, the more sensitive the classification was to the absence of that word. For (a), several words conceptually contribute to the offensiveness of the sentence, but the classifier is sensitive only to the last two words.

across the literature is the poor quality of the publicly available datasets, which are built around varying definitions of hate speech, have high levels of inter-rater disagreement, and may contain a high degree of labeler bias. These issues compound the challenge of designing an effective classifier. A final challenge of the available datasets is their degree of imbalance, which incentivizes models to overfit the classes of non-hateful speech, leading to a low recall of hate speech data.

Despite these difficulties, significant developments in the field of natural language processing have been made in recent years with the development of deep learning architectures, both as standalone classifiers and in conjunction with more traditional statistical methods. Deep learning models can work effectively as feature extractors and have the ability to model the sequential aspect of text data, furthering their ability to abstract intricate patterns from the data. These developments have led to the consistent updating of the state of the art across language modeling tasks, hate speech classification included.

Researchers have also pushed the field forward by investigating and mitigating the weaknesses of previous models. Data augmentation techniques, for example, have become more sophisticated and can significantly help to reduce data imbalance. Visualization techniques, on the other hand, have aided the interpretation of deep learning models, which is crucial to allow the identification of biases and shortcomings in complex models. Finally, the evaluation of hate speech classifiers has been expanded to include not only performance metrics but also their ability to withstand adversarial attacks, which is paramount to their application in real-world contexts.

We thus observe two different approaches to innovation within the field of hate speech classification. Part of the field has focused on translating the breakthroughs in the broader field of NLP to hate speech, devising new architectures, and implementing new methodologies with the primary goal of pushing the state-of-the-art performance. The second approach has focused less on performance and more on a more holistic characterization of these classifiers, which may focus on assessing their degree of robustness to adversarial attacks or pursue a qualitative understanding of the bias that these models may exhibit. Both strategies are essential to the goal of designing models that help us reduce the prevalence of hate speech on the internet.

3 Methodology

3.1 Data Preprocessing

Preparing raw textual data to be interpreted by an algorithm is a multi-step process involving several a priori decisions. Consider, for example, the following questions. Should we model a sentence as a sequence of words or as a sequence of characters? What is the most productive way of mapping whatever basic unit we choose (called *grams* in machine learning parlance) into numbers that can be interpreted by an algorithm? Decisions such as these have dramatic repercussions in the effectiveness and robustness of a classifier. Furthermore, we are also interested in extracting as much information as we can from the text. Schmidt and Wiegand (2017) survey several strategies in natural-language processing that have been used in hate speech classification, especially earlier in the literature (also shown in Gröndahl et al., 2018). I will now briefly explain the most common strategies.

3.1.1 Noise Removal

Traditionally, researchers heavily preprocessed text data before feeding it to a machine learning algorithm. The objective was to creatively generate features from the raw text and help algorithms find patterns in the data. Aspects of the data that are not discriminative—not helpful in achieving our goal—are referred to as noise. In hate speech classification, we are interested in finding words, patterns, and the like that are predictive of whether the sentence we are analyzing is hate speech or not. The frequency of the word "the" in a given sentence are not particularly helpful since hate speech and ordinary speech are likely to have very similar occurrences of the word "the" due to its high frequency and role in the English language. Words such as these are called stop words, "which help build ideas but do not carry any significance themselves" (Rajaraman & Ullman, 2011, p. 8). This problem is particularly relevant in word-based models that consider words as independent units since, otherwise, we may argue that "the" may affect or intensify the meaning of the following word. For such models, it is a common practice to remove stop words from the source text before processing it further. When the classifier does account for relationships among words, such as in more recent deep learning models, it is considered a best practice not to remove stop words.

There are several other strategies in NLP to remove noise. Stemming, for example, is a process through which we remove the characters of a word that are not part of its stem. Stemming will turn different words that play a similar semantic function into the same token, which may be helpful. Another strategy is to lower-case the entire dataset, since we may reasonably think that the case of a word is not relevant - however, upper-case letters or words can help us identify entities or special emphases. Moreover, depending on the source of our text, it may be that idiosyncratic features are helpful to the classification task. For example, internet discourse usually contains URLs as users present links to external websites. However, it is not obvious how we could leverage the information from a URL to many NLP tasks, so they are often removed or replaced by a URL token (i.e., a <URL>placeholder). Finally, note that preprocessing is often more art than science, and researchers have different preferences and strategies on how to do it, which also depends on the model being employed. Classifiers based on neural networks tend to perform better when the data is not heavily preprocessed.

3.1.2 Surface Features

Once the text is cleaned, the next step is to start extracting or mining features from the data. Surface features are those that are a constituent part of the text and do not require additional modeling. A sentence's words are thus surface features, whereas its classification as showcasing positive or negative sentiment is not, since sentiment analysis requires additional modeling. A standard set of surface features is the **bag of words (BOW)**, where we create a vocabulary of all words the dataset contains. If this vocabulary is ordered, then for a given sentence, we can represent data points as vectors whose entries contain the frequency with which each word of the vocabulary appears in that sentence. This strategy is one of the simplest ways to encode a sentence in a way that can be processed by an algorithm.

One of the principal shortcomings of the BOW approach is that it ignores the word order within a document. See, for example, how the sentences "the movie was not good, it was bad actually" and "the movie was not bad, it was good actually" have the opposite meaning but the same BOW unigram representation. The creation of **n-grams** is a way to preserve that information. The n-gram is an n-item-long sequence of tokens, which can usually be words, syllables, or characters. We can then model a document as an ordered list or bag of n-grams. For example, a word-level bigram representation of "I currently live in London" is "<I, currently>, <currently, live>, <live, in>, <in, London>," whereas a character-level trigram representation would be "<I, ,c>, <, c, u>, ..., <d,o,n>". The set of all n-grams present in the data can be coded into an ordered vocabulary just as in the BOW method, which allows us to represent a sentence as the frequency of occurrence of its n-grams. However, the problem with this strategy is that even for small datasets, we should have thousands and maybe millions of n-grams occurring in the data. Hence, a method of feature selection is usually employed to select the n-grams that are most predictive for the task at hand.

Finally, surface features may include less apparent characteristics of the text, such as the frequency of occurrence of punctuation, URLs, hashtags, the average number of syllables, the length of the sentence, among others. The extent to which these features are predictive or helpful will be highly dependent on the task (Schmidt & Wiegand, 2017).

3.1.3 Part-of-Speech Tagging

One issue with the simple approaches described above is that they ignore that the same word can have different lexical categories that substantially change their meaning. For example, the word *break* takes a different meaning when it functions as a verb than when it works as a noun (Hemker, 2018). One way to disambiguate the meaning of *break* is to classify its **lexical category** or **part-of-speech (POS)**. Hence, in POS tagging, we aim to label each word of a sentence with its corresponding POS tag (Bird, Klein, & Loper, 2009, p. 179). For example, the word-level unigram representation of the sentence "You will break the break" becomes

('you', 'PRP'), ('will', 'MD'), ('break', 'VB'), ('the', 'DT'), ('break', 'NN'),

where 'VB' corresponds to 'verb base form', 'NN' to a noun, and the other tags to other lexical

categories.[4] The POS tagger can thus disambiguate the meaning of *break*. Although enriching tokens with their POS tags has an intuitive appeal, Schmidt and Wiegand (2017) reported that the tags are not necessarily helpful in improving classifier performance. The cause of this result may be because distinguishing tokens by their POS tag increases the number of unique grams in the dataset and further amplifies the problem of data sparsity.

3.1.4 TF-IDF

The term frequency-inverse document frequency (TF-IDF) is a more sophisticated way of handling words that are either too frequent (i.e., stop words) or too rare in our dataset. Instead of representing a data point through the frequency of its constituent words, we compose a measure that accounts for both the relevance of the term within the document (i.e., a single data point) and within the corpus (i.e., the dataset) as a whole. Note that there are variations on how to compute the TF-IDF and the one I demonstrate below is taken from Rajaraman and Ullman (2011, p. 8). The term frequency of term i in document j is given by

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{1}$$

where f_{ij} is the frequency or count of the term i in document j, whereas the denominator is the highest frequency of any term within the same document, thus, the TF measure is responsible for providing a within-document measure of the importance of the term i. On the other hand, the inverse document frequency of the term i is defined as

$$IDF_i = \log_2 \left(\frac{N}{n_i} \right) \tag{2}$$

where N is the number of documents within the corpus, and n_i is the number of documents within which the term i appears. Notice that the rarer the term i is within the corpus, the larger will be its IDF. Finally, the TF-IDF for the term i in document j is simply the product of its corresponding TF and IDF. The TF-IDF will be highest for terms that are rare in the corpus but frequent in the document. It will, conversely, be lowest for terms that are either infrequent within a given document or very frequent within the corpus.

3.2 Word Embeddings

Let us revisit one of the critical issues associated with the BOW methods. There are usually thousands of words or n-grams occurring in a corpus. If we encode a document as a vector of frequencies or TF-IDFs of its terms, we will inevitably end up with a representation that is mostly composed of zeros since most of the words in the corpus will not occur in the document. For example, if we have a corpus with ten thousand words and wish to encode the sentence "I hate you," we will create a 10,000D vector with only three entries equal to one. This encoding is hugely wasteful both in terms of memory and processing resources. How can we render these representations more effective?

[4] Tags generated with the *pos_tag* function from the package NLTK v3.4.5 in Python 3.7.4 (Bird et al., 2009).

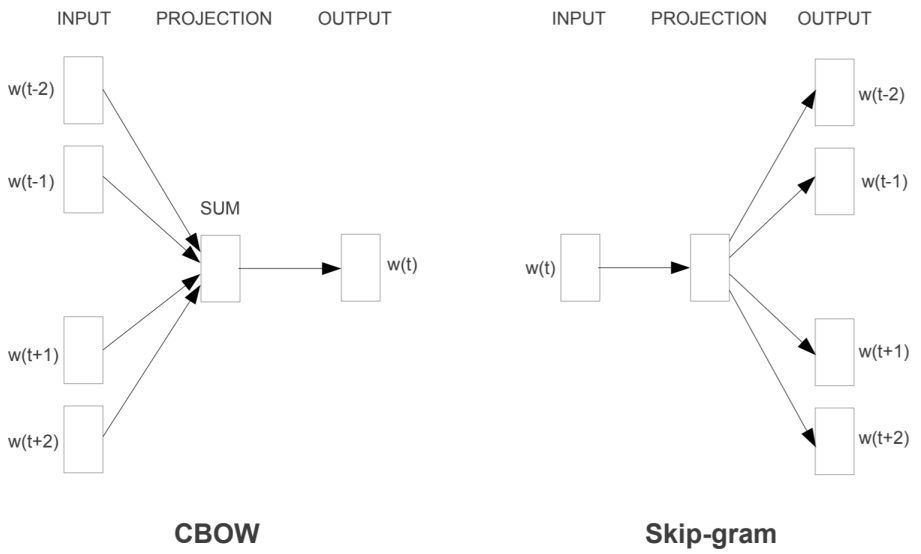


Figure 3: Word2Vec model architectures. Sourced from Mikolov et al. (2013a).

A **word embedding** is, at a basic level, a one-to-one mapping between words in our vocabulary and points within a vector space of a fixed dimension. Say we have a word embedding with 50 dimensions. Then, for each word in our corpus, the embedding will return a 50D vector corresponding to that word. Furthermore, the representation of a sentence would no longer be a vocabulary-sized vector of frequencies. Instead, we can replace the words by their indices in the ordered vocabulary and pass a list of ordered indices to the word embedding. It would return us a list of 50D vectors, each corresponding to a word in the input sequence.

In doing so, our representation of a data point has changed from a 10,000D sparse vector to a list of 50D word embeddings (alternatively, a matrix where the rows are words in the document). However, besides reducing the dimensionality of the input data and resolving the issue of sparsity, what is the benefit of these vector representations? How do we guarantee that the vector representations are meaningful?

3.2.1 Word2Vec

The answer lies in conceptualizing word embeddings as a specialized model that learns to create useful vector representations of words. For example, we want words with a similar meaning to map to points in the vector space that are close to each other. Conversely, we wish that words with opposite meanings map to points far apart. Mikolov, Chen, Corrado, and Dean (2013a) introduced their seminal **word2vec** model as a way to progressively learn these semantic and syntactic representations by considering the contexts in which the words in the corpus appears. There are two original implementations of word2vec, continuous bag-of-words (CBOW) and skip-gram, whose architectures are shown in Figure 3. I provide details on the skip-gram model since it generally performs better than the CBOW implementation.

In the skip-gram architecture, we want to train a model that receives a term from the corpus on which it is trained and predicts the neighboring terms within a given context window of size

c. In Figure 3, the model is represented with a window of size 2. Thus, for the ordered sequence $w_1, w_2, \ldots, w_T$ of words in the corpus, the objective of the word2vec model is to maximize the probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{3}$$

as shown in Mikolov, Sutskever, Chen, Corrado, and Dean (2013b). Put differently, we aim to maximize the average log conditional probability of the context words w_{t+j} given a center word w_t. The conditional probability is computed with a softmax function

$$p(w_O|w_I) = \frac{\exp(v'^T_{w_O} v_{w_I})}{\sum_{w=1}^{W}\exp(v'^T_w v_{w_I})} \tag{4}$$

where W is the set of words in the corpus, v is the input vector representation of w, and v' its output vector representation (Mikolov et al., 2013b). As a reminder, the softmax is simply a function that squishes word *scores*, which may be valued as any real number, into word probabilities between zero and one that collectively sums to one.

How does the word2vec model maximize the average log conditional probability for the words in the corpus? It does so via a simple two-layer neural network. Although we have not considered the theory behind networks yet, we can understand the word2vec architecture as follows. The network implements a $W \times D$ lookup table, where W is the number of words in the corpus, and D is the dimensionality of the embedding we want to create. Hence, to retrieve the D-dimensional embedding for the ith word in the corpus, we retrieve the ith row of this lookup table. This table is the first layer of the neural network.

The second layer is somewhat similar; this time being a matrix with dimensions $D \times W$, that when multiplied by an embedding will return us a W-dimensional vector that represents the predicted scores for every word in the corpus. When we apply the softmax function to this vector, we retrieve the associated probability of every word in our corpus that is present in the context window of the input word. Since we know which words co-occurred near our input word, we can tweak the value in the $W \times D$ lookup table and $D \times W$ matrix to produce a more accurate estimate in the future, which is to say, we *train* this neural network. The details on how training occurs are detailed in the deep learning section below. For now, note that after the network is trained, the $W \times D$ lookup table represents our desired word embedding. This embedding can be extremely powerful. Mikolov et al. (2013b) report that, for example, that the sum of vector representation for "Russia" and "river" is close to "Volga River," whereas the sum of "Germany" and "capital" is close to Berlin, demonstrating the non-trivial semantic relationships described by operations within this vector space. These examples are illustrative of the vector space's general ability to account for relatively complex semantic relationships.

Finally, note that I have omitted several implementation details for the sake of succinctness. For example, stop words should still be a concern since their representations will either be unnecessary if we remove these words when preprocessing the data or of little relevance to the actual classification task. The word2vec model deals with this problem by *subsampling* words that frequently occur in the corpus. Another major shortcoming of the architecture discussed above is the high dimensionality of the output vector W, since the number of words in large corpora can reach the order of millions. This issue prevents us from effectively training the embedding. Mikolov

et al. (2013b) address this problem by *negative sampling*. The insight of the technique is that whereas our output vector of word probabilities has a dimensionality of W, there is only one correct context word to predict. Hence, there are $W - 1$ incorrect words whose embeddings we will tweak only ever so slightly. We can instead train the embedding of the correct context word and a few randomly sampled incorrect words **only**, rendering the training process much more efficient.

3.2.2 Pre-trained Embeddings

Since the introduction of word2vec and the popularization of word embeddings, researchers have devised new ways of training them. Pennington, Socher, and Manning (2014), for example, created a different algorithm for learning vector representations of words called GloVe. Regardless of the training procedure, word embeddings can be easily made available for use by others. After all, a pre-trained embedding is nothing more than a lookup table that maps the words it has seen during training to vectors. Researchers can load these embeddings and test which one yields the best result for their particular task, which is a function of the training algorithm, the embedding dimensionality, the size of the corpus on which the embedding was trained, and the origins of the corpus (e.g., Wikipedia, Google News, Twitter).

3.3 Deep Learning

3.3.1 Feedforward Neural Networks

The machine learning model at the core of deep learning is the neural network. We have already seen how the word2vec model is an example of a two-layer neural network. More generally, a neural network is simply a composition of several functions (Goodfellow, Bengio, & Courville, 2016, p. 164), where each function corresponds to a layer in the network. In word2vec, the lookup table is the first layer and the matrix that follows is the second layer. In this formal definition, we must also consider the application of the softmax function as a dedicated third layer. More commonly, however, the application of a function to the output of a layer will be referred to as that layer's **activation function**.

The fundamental component of a neural network is the neuron, loosely modeled after its biological counterpart. A neuron receives and performs a linear transformation on the signals from the previous network layer. The output of this operation is then passed to an activation function, which receives its name due to its role in determining whether the neuron is active by altering the signal that is passed to the next layer (CS231n, n.d.). A visualization of this process is shown in Figure 4.

The output of a neuron is thus given by

$$n_{out} = f\left(\sum_i w_i x_i + b\right) = f(x^T w + b) \tag{5}$$

where w is the vector of weights, b the bias, and f the activation function of the neuron. When we train the network, we are gradually changing the values of w and b so that our network error decreases. Since a layer will be composed of multiple neurons, each of which has its vector of weights, a network layer can be thought of as a matrix of weights. The output of the layer is the product of the input vector and this matrix passed through the activation function.

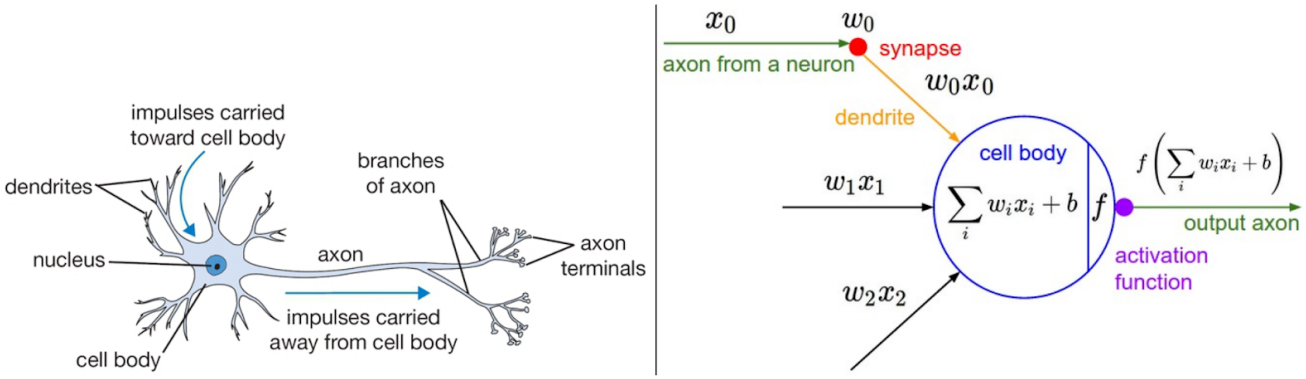


Figure 4: Schematic visualization of the similarities between biological and machine neurons. From CS231 (n.d.).

How does the network learn? We saw that in the word2vec model, the network outputs a vector of probabilities for all words in the vocabulary, but there are only a few correct context words. We want the model to output a probability of one to the correct words and of zero to everything else. We can design a **loss function** whose role is to be a measure of the distance between the model's prediction and the truth. In the simple case of a binary classification task, a standard loss function is the binary cross-entropy, given by

$$\mathcal{L}_{BC} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{6}$$

where $\hat{y}$ is the predicted probability and y the true probability, i.e., zero for the incorrect predictions and one for the correct ones (Hemker, 2018). Note that since $\hat{y}$ is always between zero and one, the log will always be negative, which in turn means the loss will always be positive. In a multiclass classification task, we can generalize the formula above to the cross-entropy loss

$$\mathcal{L}_{CC} = \sum_{c=1}^{M} \mathcal{L}_{BC}(c) \tag{7}$$

In other words, we sum the binary cross-entropy loss for all M classes (Hemker, 2018). Equipped with a measure of the error of the network, we are ready to minimize it. Note that the loss is a function of our predictions, which are, in turn, a function of the weights in every layer of the network. Hence, we can compute the gradient of the loss for the weights of any layer in the network. The negative of the gradient gives the direction of the steepest descent of the loss as a function of those weights. The negative gradient thus determines the direction in which we should tweak these weights to minimize the loss. **Gradient descent** is thus the process of successive tweaking the network's weights to minimize its loss function.

3.3.2 Convolutional Neural Networks (CNNs)

So far, the neural layers we have considered are either functions (e.g., the softmax function) or the equivalent of a matrix of weights that computes a linear combination of the inputs in each its neurons. This type of layer is **fully-connected** since each input is connected to every neuron of

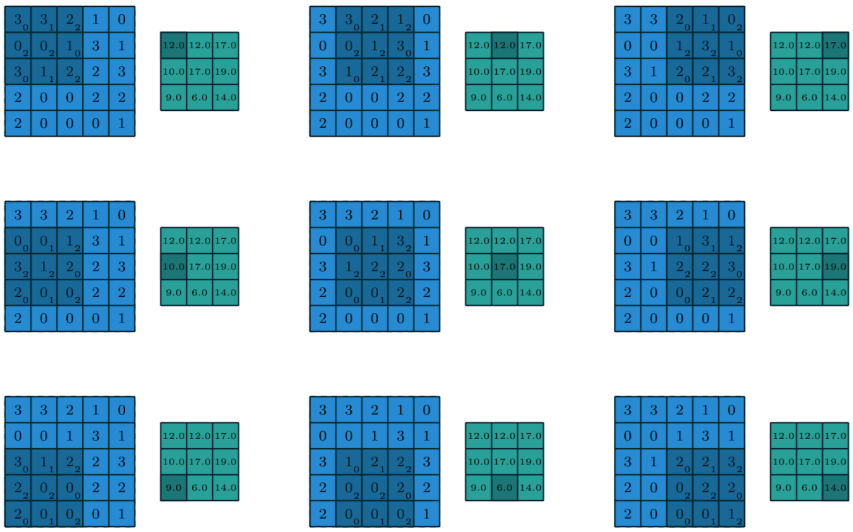


Figure 5: The convolution operation. From Dumoulin and Visin (2016).

the layer. Recall that we have conceptualized this layer as the product between the input vector and the weight matrix of the layer. As such, fully-connected layers require that their input be one-dimensional.[5] For higher-dimensional data, such as grayscale (2D) or colored (3D) images, the data must be converted or flattened to a 1D representation before it can be passed to a fully-connected layer. An critical shortcoming associated with flattening data is that the spatial information contained in the original representation is lost.

Convolutional neural networks contain at least one convolutional layer that has the property of preserving the spatial information in high-dimensional input. The function represented by a convolutional layer is not a linear combination but rather a series of convolutions. The convolution can be visualized as an operation that slides a grid-like object through the data. This object is referred to as **kernel**, which contains a set of weights and operates on a localized region of the data at a time. Figure 5 shows an example where we slide a 3x3 kernel through a 5x5 matrix to obtain a 3x3 output. The subscripts represent the weights of the kernel. The sum of the product between the kernel weights and the matrix elements within a region gives the output of the convolution.

In image processing, the weights of the kernel determine whether the output will sharpen, blur, highlight the edges of the images (e.g., Figure 6). More generally, kernels take account of spatial information to highlight specific features of the input according to their weights. In a fully-connected layer, the network learns the weights of the matrix of its constituent neurons. In a convolutional layer, the network learns the weights of its kernels as they become increasingly specialized in singling out predictive features in the data.

A convolutional layer, however, usually includes two additional operations. The first one is

[5]The input should be 1D with respect to the representation of a single data point. Inputs will be two-dimensional when we are passing a batch of data through the network at once.

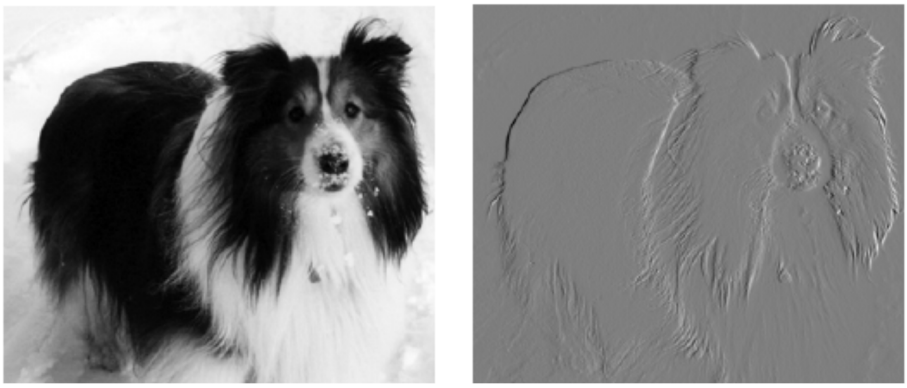


Figure 6: The image on the right shows the result of passing a vertical edge detection filter on the image on the left. From Goodfellow et al. (2016, p. 334).

the non-linearization of the convolution output by passing it through an activation function. The second operation is called pooling, which is responsible for reducing the dimensionality of an input object by taking the average or maximum value of its subregions. As such, pooling is similar to a convolution, but instead of passing a kernel that linearly combines the values of the subregion, it merely takes their average or finds the maximum value instead (Dumoulin & Visin, 2016). Figure 7 provides an example. The goal of pooling is to reduce the dimensionality of the data and to render the network more invariant to changes in the input within these small pooled subregions (Goodfellow et al. 2016, p. 336).

We have so far focused our discussion of convolutional layers on the domain of image processing in two dimensions. Notwithstanding, a convolutional layer can work with one-, three-, and higher-dimensional data since both the convolution and pooling operations are not restricted to two dimensions. In natural language processing, data has a single dimension represented by the order of the words. Hence, convolution and pooling are also one-dimensional.

3.3.3 Recurrent Neural Networks (RNNs)

We have seen how convolutional neural networks aim to mitigate the loss of spatial information in feedforward neural networks. Similarly, recurrent neural networks are designed to account for sequential patterns in the data. In time-series data, for example, the information that immediately precedes a data point is highly predictive of its value. In real-world texts, words are not distributed at random, but rather are heavily predetermined by the context formed by their respective preceding words. Feedforward networks are incapable of accounting for these dependencies since they treat inputs as independent of each other. Recurrent neural networks are those that establish a connection between the processing of different inputs, such as the one shown in Figure 8 (Goodfellow et al., 2016).

Several complex architectures have been built on the basic idea of allowing connections between internal layers of a neural network. These architectures attempt to mitigate significant shortcomings of the simple RNN model. One key issue with the network shown in Figure 8 is that it struggles to learn long-term dependencies, such as the ones present in natural language data.

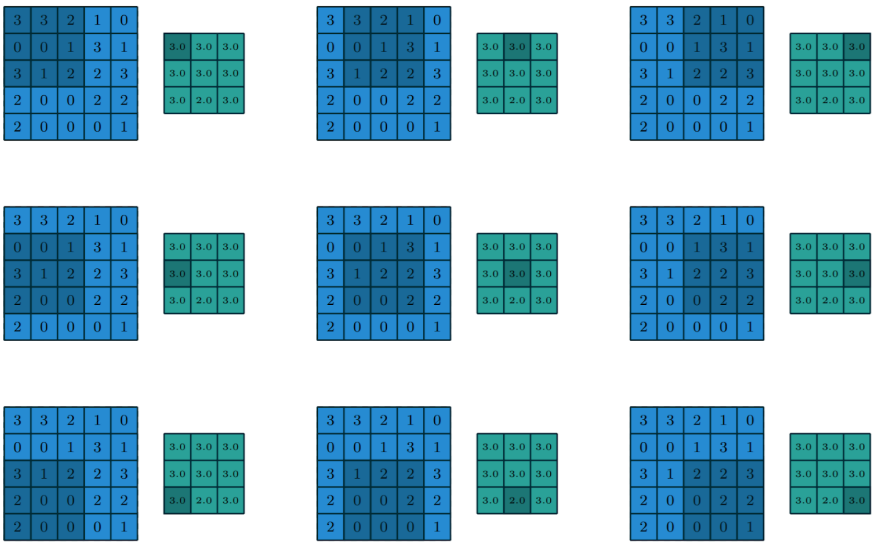


Figure 7: A demonstration of the max pooling operation. From Dumoulin and Visin (2016).

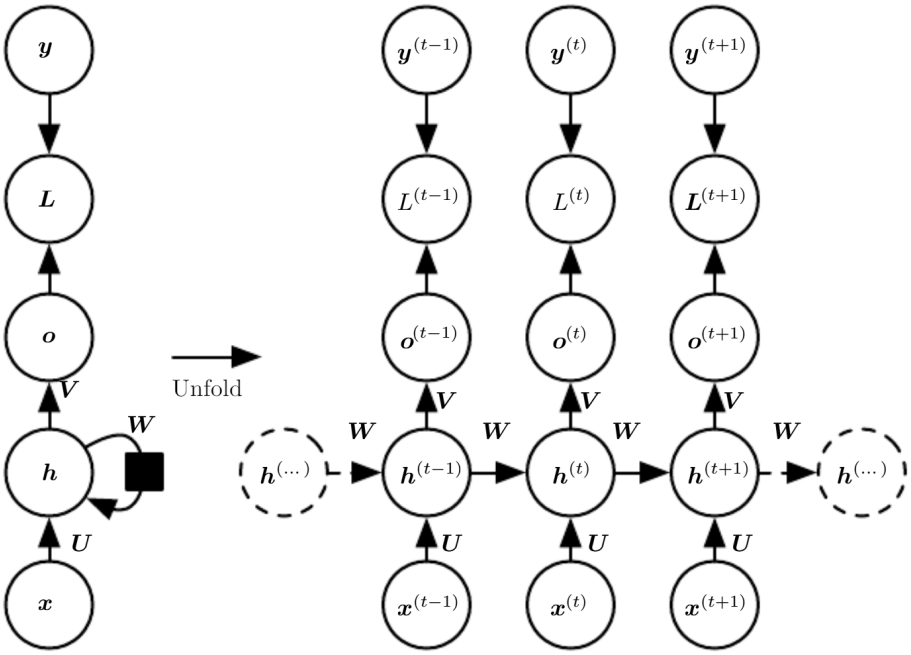


Figure 8: An RNN where the output of an internal layer at time step t, $h(t)$, is passed as an input to the same internal layer at the next step. $x(t)$ is the network input, $o(t)$ the output, $L(t)$ the loss, and $y(t)$ the true label of $x(t)$. U, W, and V are weight matrices.
From Goodfellow et al. (2016, p. 373).

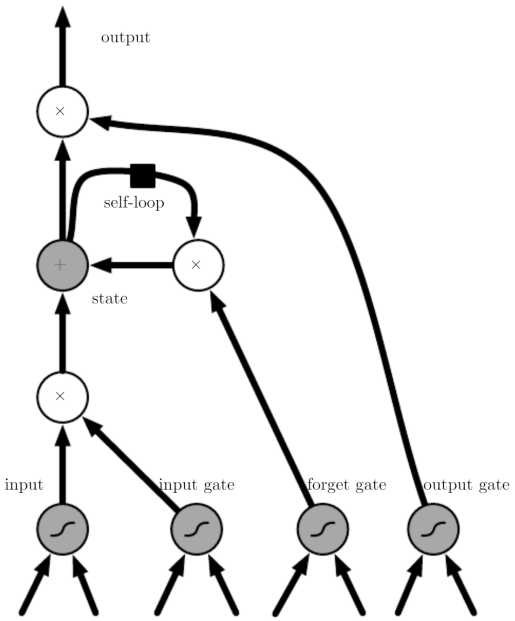


Figure 9: The internal design of an LSTM layer, demonstrating how the data flows through its internal gates. From Goodfellow et al. (2016, p. 405).

The underlying reason is that the computations through multiple time steps heavily dilute the relationship between distant data points.

Long Short-Term Memory (LSTM) networks are the most famous architecture to focus on mitigating this problem. An LSTM layer employs a complex combination of three gates that are designed to enable the network to learn both short- and long-term dependencies (Figure 9). The exact operations performed within the cell are not particularly relevant, but we should know that the layer receives three streams of data, one from the input and two from the last state of the computation. The LSTM layer has thus two recurrent connections, one focused on modeling short-term dependencies and the other long-term ones. LSTMs have shown incredible performance across a range of applications and especially in NLP (Goodfellow et al., 2016, p. 404).

3.4 Transfer Learning

Transfer learning is the technique of employing a pre-trained network (or part thereof) in a new task. For example, consider the scenario in which we designed a convolutional network to classify millions of images into 1000 different classes (e.g., the ImageNet dataset). During the process of training, the convolutional layers inside of our network presumably learned numerous image features to perform well in the classification task, ranging from simple patterns such as lines and circles to more sophisticated features such as the outlines of objects and animals. The insight of transfer learning is that this "knowledge" the network acquired is useful not only to the original classification task but, in varying degrees, to any image classification task. If we now wish to train a network to classify pictures as either containing a dog or a cat, we can reuse the convolutional layers of our original network to this new task.

The underlying assumption is that the features that the network has learned to recognize will remain useful for the new task. This assumption can be stronger or weaker depending on how similar the new classification domain is to the original one. For example, if our original network was trained to recognize different dog breeds, then it will probably be more useful in classifying cat breeds rather than flower species. The similarity in the internal structure of the new and original domains dictate which layers of the pre-trained network we will preserve. The more different the domains are the less information for the pre-trained network we keep. Since later layers of CNNs are increasingly specialized, we tend to recycle earlier layers and reset later ones to random values. Of course, regardless of how many layers we transfer, we still train the network under the new classification task.

Transfer learning in computer vision is intuitive as images of real-world objects share many features that can be useful across classification tasks. Can transfer learning also be useful in NLP? The answer is yes, but it took the field much longer to recognize the potential of transfer learning in language modeling tasks. At least part of the reason for the delay is that the intuition behind the technique is elusive—it is less clear what exactly the network learns in the chief task that is useful in the new one. Furthermore, NLP network architectures have become increasingly varied and specialized. Whereas in computer vision, we can map the effectiveness of transfer learning to the features picked by convolutional layers, in NLP, the knowledge learned via the primary task is much more architecture-dependent.

In this project, the focus is on implementing transfer learning via the method described by Howard and Ruder (2018). To the extent that computer vision models learn the features of real-world objects that are useful across image classification tasks, ULMFiT aims to learn language features helpful across text classification tasks. It does so via a three-stage process that distinguishes between two different networks—the language model (LM) and the classifier. The language model takes text as input and predicts the next word in the text. The classifier, conversely, takes text as input and predicts its class (e.g., hate speech or not). The focus of the former model is to gain general knowledge about how the language works before attempting to understand the specific classification task. The three steps of ULMFiT are thus (Howard & Ruder, 2018):

1. *LM pre-training.* We select a vast corpus to give the model information about the language. For example, WikiText-103 is a corpus of 103 million words composed of good and featured articles from Wikipedia (Merity, Xiong, Bradbury, & Socher, 2016). We train the network on WikiText-103 so that it learns general-purpose knowledge of the English language.

2. *LM fine-tuning.* Now, we follow the same procedure but instead using the corpus of the classification text. In other words, instead of predicting the next word of Wikipedia entries, the network predicts the next word of tweets in our hate speech dataset. The objective is to learn features that are more specific and relevant to the target domain (e.g., Twitter).

3. *Classifier fine-tuning.* Finally, we append a classification layer that is appropriate for the new task to the end of our network. For example, the last layer may predict whether the input is hate speech, offensive, or neither. This step is necessary since we no longer need a network that predicts word probabilities but instead one that classifies the input. We train the classification layer while benefiting from the features that earlier layers have learned.

Note that this procedure can be used with different network architectures. The authors used an LSTM variant called AWD-LSTM, which features several dropout features that enhance its performance (Merity, Keskar, & Socher, 2017). I employ the same architecture.

3.5 Data Augmentation

The goal of data augmentation for text is to generate semantically equivalent sentences using different words. The purpose is to both remediate the class imbalance in the data and help the model generalize better to new input. We have previously discussed how a cyclical translation scheme (e.g., English > French > English) is unlikely to preserve the meaning of the original sentence. Hemker (2018) presents a technique called threshold-based augmentation that imposes additional safeguards on the structural equivalence between original and augmented text. The key insight of the augmentation scheme is the use of word embeddings. When an embedding is trained, it learns word vector representations by attempting to predict the neighboring terms around a context window for any given word. As a result, words that appear in similar contexts are assigned to the same region of the vector space. Then, we can assume that if two words are "close enough" in this vector space, they are possibly interchangeable. Notice that contrary to synonym replacement techniques, we are not interested in preserving the exact semantic meaning, but rather in generating an alternative sentence that conveys the same idea (and hence whose label would remain the same). For example, the sentence "he is worthless" and "they are trash" contain no synonyms but convey very similar meanings. The way the embedding would recognize this equivalence is by noticing that words such as "trash" and "worthless" can appear in the same contexts.

To proceed, we must formalize the idea of "closeness" in a vector space. This proximity can be conveyed by the cosine similarity of two vectors, computed as

$$\cos\theta = \frac{u \cdot v}{|u||v|} \tag{8}$$

for two arbitrary vectors u and v. Vectors that appear in largely similar contexts will have a cosine similarity close to one, whereas the similarity of vectors showing in contrasting contexts will be close to minus one. We can thus use the cosine similarity to search for the words most similar to any word we are interested in augmenting.

Hemker (2018) imposes two requirements for the augmentation of a word to succeed. The first is that the cosine similarity between the candidate word must exceed some prespecified threshold. For example, we may say that we are only willing to exchange two words if their cosine similarity is higher than 0.70. The higher the threshold, the more conservative the augmentation process will be. Moreover, we also require that the candidate word should have the same part-of-speech as the original one. This condition aims to prevent the generation of ungrammatical sentences since two words may appear in similar contexts but have different lexical categories. These two strategies significantly ameliorate the concerns of cyclical translation schemes. However, the scheme is not perfect and does not guarantee that all augmented samples will preserve the meaning of the original input.

The author reports that a cosine similarity threshold of 0.75 led to the best performance upon training the classifier. Unfortunately, no other details on the methodology are given. It is not clear

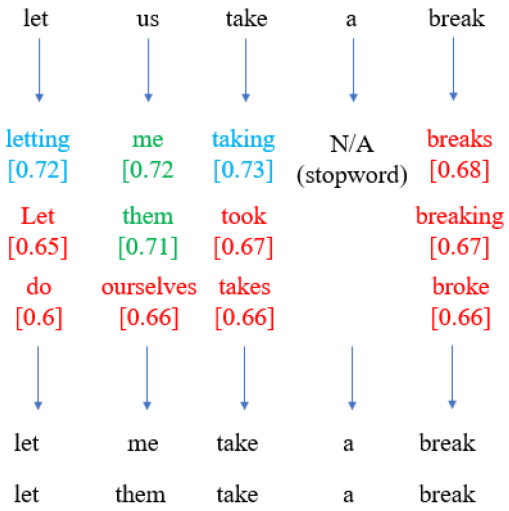


Figure 10: A demonstration of the data augmentation technique with a threshold of 0.7. The candidate words in blue are rejected since they have a different POS tag.

whether only words with specific POS tags were augmented, how many augmented samples were generated per input, and whether only the hate speech class or the entire dataset were expanded. Without these instructions, strict replication is impossible. My goal will be to employ this general strategy sensibly and assess whether the results Hemker (2018) reports are plausible.

3.6 Reporting Metrics

A brief comment on the choice of reporting metrics is now necessary. As a reminder, precision and recall are possibly the most common metrics in the field of machine learning. The former computes the ratio between true positives and predicted positives (i.e., how many tweets predicted to be hate speech are hate speech). In contrast, the latter calculates the ratio between true positives and actual positives (i.e., the fraction of hate speech tweets we correctly identified as hate speech). Since each metric indicates a relevant aspect of performance, we can combine them into a derivative metric called F1 score, computed as the harmonic mean of precision and recall, i.e.,

$$F1 = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{9}$$

Now, each class of a classification problem has its F1 score, since precision and recall are defined for a specific class. Hence, we may compute the F1 scores of the hate speech, offensive, and neither classes. Adding another layer of complication, we may choose to aggregate these three scores into one, providing a single measure of performance. The simple mean of class-specific F1 scores is called macro-F1, and their weighted mean is called weighted-F1 (i.e., the weights are the number of samples of each class in the dataset). It is also possible to compute the micro-F1 score, defined by the formula above, but computed over global metrics, ignoring classes altogether.[6] It is

[6]For example, global precision equals the ratio of all true positives and the number of predictions. Note that

important to note that both weighted- and micro-F1 are particularly bad choices for imbalanced datasets where we care about the minority class since they favor the majority class. Finally, note that all of these metrics are bounded between zero and one.

The choice of metric is vital since it may hide or expose notable flaws in an algorithm. As a simple example, in the T1 dataset, a classifier that completely misses the hate speech class may achieve a micro- or weighted-F1 score above 0.90, since the hate class represents less than 10% of the dataset. Such a choice is arguably misleading, and indefensible choices on reporting metrics have partially plagued the literature on hate speech classification. Since we are primarily interested in identifying hate speech, I argue **hate recall** and **hate F1 score** are essential metrics to report. We should cast a doubtful eye on papers that report aggregated F1 scores; these metrics may hide severe flaws in performance.

3.7 Implementation

For the sake of clarity, let us briefly summarize the models and strategies implemented in this Capstone project. The main goal is to analyze the effectiveness of transfer learning via ULMFiT in hate speech classification for the Davidson et al. (2017) dataset. We will take advantage of fastai's (2019) API to quickly implement an AWD-LSTM classifier trained via ULMFiT and other optimization techniques such as discriminative learning, gradual unfreezing, and bidirectional models. The data preprocessing is also handled under the hood by the library. These instructions are implemented in notebook 4 of the repository associated with this project.

Next, we want to analyze whether data augmentation can further enhance the performance of the classifier. Hence, we augment the whole dataset following a threshold data augmentation procedure using a 300D GloVe pre-trained embedding, a threshold value of 0.75, and replacing nouns only, following Hemker's (2018) alleged optimal hyperparameters. Once we have the augmented dataset, we follow the same training regime specified in the paragraph above. After we split the original dataset into training and validation sets, we add the augmented tweets corresponding to the hate speech samples within the training set. We must be careful to augment the training set only so that the model is not exposed to augmented versions of tweets in the validation set, which would inflate performance since the two samples are very similar. Moreover, to guarantee a better estimate of the out-of-sample error, I perform a 5-fold cross-validation upon the augmented ULMFiT procedure.

4 Results

4.1 Davidson et al. (2017)

I performed a full replication of Davidson et al. (2017) based on the code the authors made publicly available. The authors utilize several text mining techniques to the data, through which they extract lexical and syntactic features, such as word grams and the Penn Part-of-Speech (POS)

since a false positive for one class is a false negative for another, global precision equals global recall and, as a result, also equals the micro F1 score.

Table 2: Replication of Davidson et al. (2017). Green shade indicates successful replication.

Metric	Original	Replication
Micro F1	0.90	0.90
Hate F1	0.51	0.50
Hate Recall	0.61	0.61

tag grams, accompanied by other surface-level features, such as hashtag counts and reading ease scores. I replicate the final classifier the authors reported, composed of a logistic regression model with L1 regularization for dimensionality reduction and another logistic regression model with L2 regularization for the classification task itself. The results presented in Figure 11 are obtained by performing 5-fold cross-validation on the model.

The recall of the algorithm on the hate speech class is 60%, meaning that the classifier correctly classifies 60% of the tweets that are labeled as hate speech. The recall is much better for the other two groups, with a 91% on the *offensive* and 94% on the *neither* classes. These results are mostly identical to the ones presented at Davidson et al. (2017). The asymmetry in how effective the model is in categorizing the hate and non-hate classes comes back to the imbalance of the dataset. Since a minority of the data is hate speech, the model can minimize its loss very effectively during training by being conservative on what it considers hate speech. Hence, it learns that by pure virtue of the base rate, its prediction is more likely to be correct if it predicts that a data point is offensive rather than hate speech. Nevertheless, we should note that this algorithm, while simple, performs well above chance. Whereas a naive classifier would find only 33% of the data to be hate speech (given that there are three classes), the logistic regression model does much better. The authors also note in an ex-post qualitative analysis of the misclassified samples that some of them are mislabeled. They provide an example of the tweet "*When you realize how curiosity is a b*tch #CuriosityKilledMe*," whose true class is hate speech even though its meaning is benign. The mislabeling is probably caused by the inattention of the human raters, who may have skimmed some samples and thus assigned incorrect labels. Such errors are hard to correct and demonstrate the inherent difficulties in labeling data.

4.2 Badjatiya et al. (2017)

Similarly, I conducted a replication of the main results discussed by Badjatiya et al. (2017). The authors used the dataset from Waseem and Hovy (2016), which classifies tweets as racist, sexist, or neither. They perform experiments with both deep learning and classical approaches. Furthermore, they also test some combinations of approaches, where a first model is trained as a feature extractor, and a second model performs the classification.

In their implementation of a Bag of Words model, they tokenize a tweet as a bag of words, retrieve a pre-trained embedding for each word (if available), and use the average embedding of the entire tweet as features. These features are then passed to a second model, such as an XGBoost or support vector classifier. They also implement a TF-IDF model that extracts uni-, bi-, and trigrams from the tweet data that are passed as features to another model. Their implementation of deep learning models is composed of three different architectures, which include a convolutional

Table 3: Replication of Badjatiya et al. (2017). W-P., W-R., and W-F1 stand for weighted precision, recall, and F1 score, respectively.

	Method	Original*			Replication**		
		W-P.	**W-R.**	**W-F1**	**W-P.**	**W-R.**	**W-F1**
Part A: Baselines	TF-IDF + Balanced SVM	0.823	0.826	0.823	0.816	0.816	0.816
	TF-IDF + GBDT	0.826	0.817	0.801	0.819	0.807	0.813
	BoWV + Balanced SVM	0.758	0.692	0.705	0.791	0.788	0.789
	BoWV + GBDT	0.765	0.774	0.759	0.800	0.802	0.801
Part B: DNNs Only	CNN + Random Embedding	0.816	0.816	0.816	0.813	0.816	0.814
	CNN + GloVe	0.836	0.837	0.835	0.839	0.840	0.839
	LSTM + Random Embedding	0.802	0.805	0.803	0.805	0.804	0.804
	LSTM + GloVe	0.834	0.835	0.836	0.807	0.809	0.808
Part C: DNNs + GBDT***	CNN + GloVe + GBDT	0.869	0.869	0.864	0.864	0.864	0.864
	CNN + Random Embedding + GBDT	0.913	0.913	0.912	0.864	0.864	0.864
	LSTM + GloVe + GBDT	0.865	0.864	0.859	0.849	0.848	0.848
	LSTM + Random Embedding + GBDT	0.934	0.934	0.934	0.930	0.930	0.930

*I do not present non-weighted and hate-specific statistics since Badjatiya et al. (2017) trained their models on Waseem and Hovy's (2016) dataset, which contains sexist and racist classes instead of a single hate class. All other replication and models presented in this paper are trained on the T1 dataset.
**Green shade indicates a successful replication, with a difference between original and replicated results smaller than 0.020. Read shade indicates irreproducible results with differences larger than 0.02.
***All results for part C are invalid due to an incorrect training procedure that exposes the validation set during training. Hence, performance is inflated. The problem is discussed in the *Analysis* section.

layer connected to a dense classification layer, an LSTM layer connected to a dense layer, and a fastText model. The authors mistakenly conceive or inappropriately implement fastText as an embedding layer connected to a dense layer. In order not to propagate the confusion further, and since this architecture is not the top performer among the authors' set of models, I do not replicate the alleged fastText model.

4.3 Hemker (2018)

Finally, I conducted a partial replication of the results reported by Hemker (2018). I focus on reproducing the performance of the best performing architecture on non-augmented data. The model is a combination of a 300D pretrained GloVe embedding layer, a convolutional layer, and an LSTM layer trained on the Davidson et al. (2017) dataset. The hyperparameters reported by the author are shown in Figure 11.

I should note that the code base for this paper is not available. For my best attempt of interpreting the descriptions given by the author, I show a summary of the replication and the confusion matrix of the model below.

Parameter	Value
Optimiser	RMSprop
Learning Rate	0.01
Dropout	0
Momentum	0.7
Batch Size	80
Kernel Size	25
Strides	9
Convolutional Filters	256

Figure 11: Hyperparameters for the top-performing model reported by Hemker (2018).

Table 4: Replication of top-performing non-augmented classifier from Hemker (2018).

Metric	Original	Replication
Micro F1	0.956	0.880
Hate F1	—	0.226
Hate Recall	—	0.17

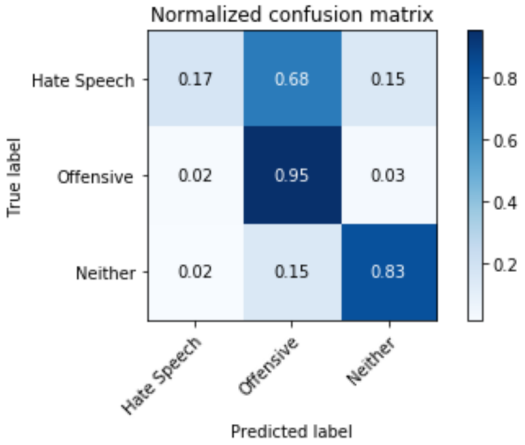


Figure 12: Confusion matrix of the replicated classifier of Hemker (2018).

Table 5: Performance of three original training strategies and relevant replications.

#	Training Strategy	Hate Recall	Hate F1
(1)	Davidson et al. (2017)	**0.60**	**0.50**
(2)	Hemker (2018)	0.17	0.23
(3)	ULMFiT	0.30	0.38
(4)	(3) + Data Aug.	0.45	0.41
(5)	(4) + 5-fold CV	0.46	0.44

4.4 Original Work

Here, I provide the results of my attempts of using transfer learning and data augmentation to enhance the performance of a deep learning classifier. There are three sets of results. First, I implement transfer learning via ULMFiT with the AWS-LSTM architecture described above on the T1 dataset. Then, I generate up to five augmented samples for each tweet labeled as hate speech and perform the same training procedure. Finally, I perform 5-fold cross-validation to increase the stability of the measures of classifier performance. The results are summarized in Table 5, which also includes the replication of Davidson et al. (2017) and Hemker (2018) for comparison purposes since their classifiers are trained on the same dataset.

5 Analysis

5.1 Replications

I break down the analysis of the replications of Davidson et al. (2017), Badjatiya et al. (2017), and Hemker (2018) according to overarching themes that include lack of clarity concerning software requirements, insufficient information to allow for replication, and methodological mistakes.

5.1.1 Software Requirements

There are two types of complications related to reproducibility that apply to all the papers surveyed. The first is the lack of clarity over software requirements. Davidson et al. (2017), for example, include no information regarding which packages and the corresponding versions they used to attain their results. Furthermore, they explicitly state in their GitHub repository that the code base is not maintained and ask researchers not to post issues or questions regarding compatibility. The active discouragement of replication attempts is very unfortunate. Badjatiya et al. (2017) fare slightly better in this regard since they provide package names (without versions) and some instructions on how to use the codebase. The lack of version details is, however, a severe issue since packages such as Keras and Tensorflow are actively evolving and may produce different results (or break altogether) if the incorrect version is used.

Finally, Hemker (2018) has not made a code base available for replication. Therefore, success in reproducing the author's results is improbable. When inquired about sharing the code base privately, the author's research team informed me that they had submitted a revised version of the paper to a conference and were awaiting a response before rendering the code available (G.

Rizos, personal communication, Jun 5, 2019). Even though the team succeeded in their goal of being published (Rizos, Hemker, & Schuller, 2019), the associated code repository remains empty.

5.1.2 Insufficient Instructions

The second issue of reproducibility is the insufficiency of information, which arises when either the codebase or the implementation details in the original paper are missing, contrived, or underspecified. This problem is similarly rampant. Davidson et al. (2017), for instance, never explicitly address whether they build character or word-level grams. In the paper, we can infer that they build word grams since the authors also mention the construction of POS tag grams (and only words can have lexical categories). However, the tokenizer in their repository breaks sentences down to characters instead of words.[7] The inconsistency is probably a mistake made by the authors when publicizing their code. However, I discuss it here since the lack of clarity of the original paper makes it non-obvious.

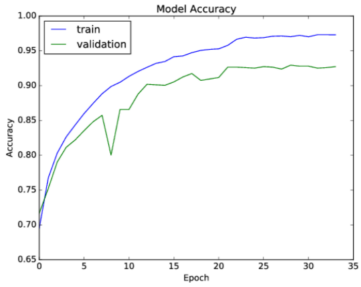
Despite this problem, it is possible to replicate Davidson et al. (2017) since the authors make available a single notebook that, after a few corrections, is capable of reproducing their stated results. In the case of Badjatiya et al. (2017), the codebase also allows one to infer the hyperparameters. However, several results are irreproducible due to conceptual and methodological mistakes (more on this issue below). The problem is taken to its extreme with Hemker (2018), where there is no code base available to supplement the methodological gaps unaddressed by the paper. Hence, the results are irreproducible in principle due to the lack of information.

5.1.3 Methodological Mistakes

Perhaps more concerning are the methodological mistakes that went unnoticed in the review processes behind the acceptance of Badjatiya et al. (2017) and Hemker (2018). Although the code base of Davidson et al. (2017) contains a few problems, it seems that these were introduced when making the repository available since once these are remediated, it is possible to obtain the results published by the authors. The same is not the case for the two other papers.

In Badjatiya et al. (2017), there are three serious errors. The first is the claim of implementation of a fastText model (Bojanowski., Grave, Joulin, & Mikolov, 2017). fastText is a library created by the Facebook AI Research team that learns word representations that make use of subword information (e.g., char n-grams). The code made available by Badjatiya et al. (2017) makes no use of the fastText library or subword information, making the claim of a fastText model spurious. The second issue affects the CNN and LSTM models that make use of a GloVe embedding. The authors implement a cross-validation scheme but do not reset non-embedding layers across folds. Hence, the model is exposed to the validation set of all folds but the first one, inflating its performance. The authors acknowledged the mistake on Github and claimed it was introduced when preparing the repository, a claim supported by the fact that when the error is redressed, we can obtain the results from the paper. The problem has yet to be corrected in the online repository.

[7] See this commit where I fix the issue.



		Predicted		
		0	1	2
Actual	0	0.84	0.11	0.04
	1	0.01	0.98	0.01
	2	0.02	0.01	0.97
		Accuracy		0.96

Figure 13: Figures 4.3b (left) and 4.2c (right) from Hemker (2018). The graph provides evidence that the author is reporting performance on the training data.

Finally, in the last combination of deep learning models and XGBoost, the authors again incorrectly implement a cross-validation procedure since they do not guarantee that the two models use the same training set. As a result, part of the validation set is exposed to the deep learning model during training, inflating the performance of the combined model. This error, however, seems not to be limited to the online repository. The performance of these combined models is much lower than reported once the cross-validation procedure is corrected, providing evidence that the published results are irreproducible and mistaken. Part C of *Table 3* shows the performance obtained when replicating Badjatiya et al.'s (2017) codebase as is, but it should be disregarded due to this grievous methodological mistake.

With Hemker (2018), the problems are equally severe but less well-defined. Since we cannot rely on a codebase to dispel confusions, we must investigate and speculate on the cause of the irreproducibility of the author's results. The first thing we note is that some of the results reported are incredibly high. The GloVe+CNN+GRU architecture trained on threshold-augmented data, for example, allegedly attains a suspicious 93.4% recall on the hate speech class.

One possible explanation for this and other similarly high levels of performance is that the metrics reported were measured on the training set instead of the validation set. We find evidence of this when we interpret Figures 4.2 and 4.3 of the original paper, which report metrics on different network topologies. Figure 4.2(c) shows an accuracy of 96% for the GloVe + CNN + GRU architecture, whereas 4.3(c) plots how the accuracy on both training and validation sets changes over time for the same architecture. However, looking closely, we can see that the validation set's accuracy plateaus well below 95%—it is the training set's accuracy that rises to 96% (see Figure 14). Reporting performance on training data is an instance of glaring malpractice and sheds doubt on the whole paper, especially given the absence of a codebase for investigation. Other suspicious results reported in the paper are that learning rates ranging from 0.001 to 0.01 lead to broadly similar training trajectories (Figure A.1) and that a network with a dropout of 1 (the author never specifies how dropout is implemented) learns just as effectively as a network with dropout of 0 (Figure A.3).

In practice, we see that the learning rate of 0.01 is too high for the network to learn. The classifier skips the minima of the loss function without converging. I lower the learning rate to 0.001 in an attempt to make training viable, which is the only change made to the hyperparameters shown in Figure 12. With the lower learning rate, the network does converge, but it mostly misses the hate speech class while attaining very high levels of accuracy on the offensive and neither

classes. It is unclear why that is the case, but a hypothesis would be that it has to do with the relatively high value of the stride hyperparameter. The stride determines the size of the jump of the kernel within the convolutional layer. For a sequence length of 100, kernel size of 25, and the padding value of 3,[8] the convolution output would have dimensionality of 10.[9] Especially when we consider that several tweets are much shorter than 100 words, much of the signal is likely lost due to the size of the kernel and its stride. This reasoning also brings us to suspect the author's implementation once again. Figure A.7 (Hemker, 2018), which seeks to demonstrate the effect of stride size on the classifier, shows similar learning trajectories for values ranging from 1 to 20, a dubious result.

5.2 Original Work

The combination of transfer learning and the AWS-LSTM architecture succeeds in identifying 30% of the hate speech samples in the original dataset, a level of performance that is further enhanced to 46% with data augmentation of the hate speech class.[10] We can see that the data augmentation procedure helps increase performance, but even the augmented model falls short of the performance benchmarks set by Davidson et al. (2017). The most likely explanation for the impressive performance set by this benchmark is that the authors employ an intensive data mining process supported by an array of heuristic and pre-trained methods that are particularly relevant to the English language. For example, they employ predicted sentiments, reading ease scores, and POS tag grams as additional features for the final classifier.

Although these techniques can increase the performance, they come at a cost to generality since the underlying algorithms may only work for English-language input. Conversely, deep learning approaches tend to be extremely generalizable since the network learns directly from the tokens within the text. The main barrier to language transfer that plagues both deep learning and classical approaches is tokenization. Some languages may rely extensively on affixes to construct meaning, which would present a challenge to a traditional tokenizer that breaks down input based on the use of spaces. Other languages, such as Mandarin, do not use spaces altogether. The task of creating an algorithm that can dynamically learn a language's morphology to create an appropriate tokenizer is an active area of research with many recent breakthroughs. In general and except for the soon-to-be-overcome exception of tokenizers, deep learning approaches are much more generalizable than heuristic-based ones and are transferred to other languages much more readily.

Data augmentation successfully raises hate speech recall by about 50% when compared to the baseline ULMFiT implementation. Although that's a positive result, the F1 score of the hate class improves only slightly. The discrepancy entails that although we are doing slightly better in identifying hate speech overall, part of the increase in the recall is due to a more lax definition of

[8] I added the padding value despite Hemker's (2018) omission of the hyperparameter since it is likely that the author made use of it in his Keras-based implementation. The padding guarantees that the convolution will not skip the words at the border of the sequence.

[9] (batch size, number of filters, 10), to be precise.

[10] Since the architecture and training strategy of (2) and (3) are precisely the same, I focus the analysis on the latter, cross-validated results since they are methodologically equivalent but more credible.

hate speech by the classifier that leads to a loss in precision. In other words, the classifier predicts hate speech more often, and although it finds more of the hateful tweets than it did previously, it also incorrectly classifies the non-hateful ones more often. Hence, although Hemker's (2018) threshold data augmentation effectively reduces data imbalance and, as a result, removes some of the bias favoring the majority class, it does not necessarily help the model learn the structure of hate speech more effectively. This limitation is reasonable since the augmentation procedure is based on word embeddings that are also part of the primary model, potentially rendering the differences between original and augmented samples less useful to the classifier.

Finally, I note that whereas the top-performing deep learning classifier falls significantly behind Davidson et al.'s (2017) model with respect to recall, their hate F1 score is much more similar. Hopefully, it is a matter of time before novel architectures can both preserve full generalizability and supersede heuristic-based approaches.

6 Conclusion

The two goals of this Capstone project were to provide a critical evaluation of the hate speech classification literature and implement a reproducible classifier powered by transfer learning. In attempting to replicate Davidson et al. (2017), Badjatiya et al. (2017), and Hemker (2018), it is found that some of the results on which the field stands are irreproducible. The causes vary but often include the lack of specification over software requirements, insufficient methodological instructions, or simply incorrect implementations. Much as the broader literature on deep learning, the field of hate speech classification should pay much greater heed to the reproducibility of the results.

As a positive response to this trend, it is the complementary goal of this paper to both introduce new techniques of transfer learning and data augmentation in NLP to this classification task while safeguarding the easy reproducibility of chief results. The accompanying repository contains a fully functional codebase, specification over software requirements, and instructions on how to replicate the performance achieved both by replicated and original classifiers. In doing so, I hope to inspire future researchers never to lose sight of those who will one day seek to learn, leverage, and build upon what we have accomplished.

7 Glossary

I provide a glossary to render this proposal more accessible to a non-technical audience. The first time a concept defined in the glossary appears in the main text, it will be **bolded**.

Accuracy: The proportion of correct predictions by total predictions.

Adversarial input: Input manipulated in an attempt to misrepresent its real class to fool the algorithm. For example, a hate speech message with some whitespaces excluded is still hate speech (since a human reader can readily interpret it), but an algorithm might fail to classify it as such due to the whitespace removal.

Class imbalance: Datasets in which the class proportions differ substantially are imbalanced. For instance, a dataset may contain only 5% of its data as hate speech.

Classical methods: Methods that do not make use of neural networks, such as logistic regression, support vector machines, and random forests.

Classifier performance: Performance is a general term for the scores of a classifier along standard metrics such as accuracy, precision, and recall.

Data augmentation: The process of transforming the mathematical representation of the input without changing its meaning vis-à-vis the task at hand. For example, in computer vision applications it is common to rotate or flip images since these transformations change their vector representation but not their content. A picture which includes a dog still contains a dog after being flipped.

Deep learning: An umbrella term for machine learning methods which make use of deep neural networks and associated methods.

Ensemble methods: Methods which are composed of individual models whose output is aggregated. For example, a classifier might have both classical and neural network component which are combined to provide the final output.

Generalizability: The ability of a classifier to maintain its performance when classifying data coming from a dataset different than the one with which it was trained. Such data may also be synthetic or simulated data, produced with the aims of testing specific features of the classifier (e.g., robustness to adversarial input).

Grams: An ordered sequence of atomic components from the input text. For example, the sentence "I am here" has three-word unigrams ("I", "am", "here"), three-word bigrams ("I am", "am here"), and nine character unigrams (corresponding to each character including whitespaces). N-Grams are usually characters or words and n can be any number, usually from one to five.

Hyperparameters: A parameter which is set by the researcher before the training process, as opposed to one that can be changed dynamically during training.

Lexical features: They are features connected to the form of the message, such as its constituent characters and words.

Multi-class classification: A classification task in which the data may map to more than two categories. For example, a message can be hate, offensive, or ordinary speech.

Natural Language Processing (NLP): A field of computer science dedicated to the representation, understanding, and reproduction of natural human language.

Neural networks: These are algorithms which resemble the structure of the brain, with atomic components whose connections allow the network to learn non-trivial relationships between features of the data.

Occlusion tests: Tests in which part of the input is deleted before being passed to the algorithm to assess the sensitivity of the prediction to the removed section. For example, we might replace "I am here" for "I am <unknown>", where <unknown> represents unrecognized words, to assess the sensitivity of the prediction to the word "here."

Penn Part-of-Speech (POS) tag: A representation of a message which replaces each word or collections thereof by their syntactic role in the phrase. "I am here" translates to "<pronoun> <predicate>."

Recall: The proportion of the right classes found, i.e., correctly classified. For example, the recall of the hate speech class will be given by the ratio between the number of hate speech data appropriately classified and the total number of hate speech data.

Syntactic features: They are features related to the syntax of the data. For example, Penn Part-of-Speech (POS) tags (and its grams) are syntactic features.

8 Bibliography

References

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017, April). Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion* (pp. 759-760). International World Wide Web Conferences Steering Committee.

Biere, S. (2018). Hate Speech Detection Using Natural Language Processing Techniques (Master's thesis). Retrieved from `https://bit.ly/2UQzLwY`

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* O'Reilly Media, Inc.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics, 5*, 135-146.

CS231n. (n.d.). Neural Networks Part 1: Setting up the Architecture. Retrieved from `http://cs231n.github.io/neural-networks-1/`

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017, May). Automated hate speech detection and the problem of offensive language. In *Eleventh International AAAI Conference on Web and Social Media*.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285.*

Dunlosky, J., Rawson, K. A., Marsh, E. J., Nathan, M. J., & Willingham, D. T. (2013). Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest, 14*(1), 4-58.

European Commission (January 15, 2019). Countering illegal hate speech online #NoPlace4Hate. Retrieved from `https://bit.ly/2HiIhkM`

Facebook (n.d.). Company Info. *Facebook Newsroom.* Retrieved from `https://bit.ly/1uZUCLf`

fastai (2019). A Code-First Introduction to NLP course [GitHub repository]. Retrieved from `https://github.com/fastai/course-nlp`

Finck, M. (February 1, 2019). Artificial Intelligence Tools and Online Hate Speech. Center on Regulation in Europe. Retrieved from `https://bit.ly/2FcHvEv`

Fortuna, P. C. T. (2017). Automatic detection of hate speech in text: an overview of the topic and dataset annotation with hierarchical classes (Master's thesis). Retrieved from `https://bit.ly/2MW5tWC`

Framework Decision 2008/913/JHA on combating certain forms and expressions of racism and xenophobia by means of criminal law (2008). *Official Journal L328 of 6.12.2008.* Retrieved from `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=LEGISSUM:l33178`

Gao, L., & Huang, R. (2017). Detecting online hate speech using context aware models. *arXiv preprint arXiv:1710.07395.*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT press.

Gröndahl, T., Pajola, L., Juuti, M., Conti, M., & Asokan, N. (2018, October). All You Need is "Love": Evading Hate Speech Detection. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security* (pp. 2-12). ACM.

Hemker, K. (2018). Data Augmentation and Deep Learning for Hate Speech Detection (Master's thesis). Retrieved from `https://bit.ly/2SnjylP`

Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146.*

Human Rights Council (September 12, 2018). Report of the independent international fact-finding mission on Myanmar. *A/HRC/39/64.* Retrieved from `https://bit.ly/2BQmCiK`

Lerner, J. S., Li, Y., Valdesolo, P., & Kassam, K. S. (2015). Emotion and decision making. *Annual review of psychology, 66*, 799-823.

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843.*

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182.*

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

Rajaraman, A., & Ullman, J. D. (2011). *Mining of massive datasets.* Cambridge University Press.

Risch, J., & Krestel, R. (2018). Aggression identification using deep learning and data augmentation. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)* (pp. 150-158).

Rizos, G., Hemker, K., & Schuller, B. (2019). Augment to Prevent: Short-Text Data Augmentation in Deep Learning for Hate-Speech Classification. In *CIKM '19: Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (pp. 991-1000). Repository available at `https://github.com/glam-imperial/TextAug`

Ross, B., Rist, M., Carbonell, G., Cabrera, B., Kurowsky, N., & Wojatzki, M. (2017). Measuring the reliability of hate speech annotations: The case of the european refugee crisis. *preprint arXiv:1701.08118.*

Schmidt, A., & Wiegand, M. (2017). A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media* (pp. 1-10).

Tulkens, S., Hilte, L., Lodewyckx, E., Verhoeven, B., & Daelemans, W. (2016). A dictionary-based approach to racism detection in dutch social media. *preprint arXiv:1608.08738.*

Vigna, F., Cimino, A., Dell'Orletta, F., Petrocchi, M., & Tesconi, M. (2017). Hate me, hate me not: Hate speech detection on facebook. In *Proceedings of the First Italian Conference on Cybersecurity*, 86–95.

Wang, C. (2018). Interpreting Neural Network Hate Speech Classifiers. In *Proceedings of the Second Workshop on Abusive Language Online* (pp. 86-92). Retrieved from `https://bit.ly/2Hi0OcI`

Waseem, Z., Davidson, T., Warmsley, D., & Weber, I. (2017). Understanding abuse: a typology of abusive language detection subtasks. *arXiv preprint arXiv:1705.09899.*

Waseem, Z., & Hovy, D. (2016). Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop,* pp. 88–93.

Wulczyn, E., Thain, N., & Dixon, L. (2017). Ex Machina: Personal Attacks Seen at Scale. In *Proceedings of the 26th International Conference on World Wide Web,* pp. 1391–1399.

9 Appendix A — LO & HC Applications

9.1 Capstone LOs

#qualitydeliverables. The salient points that attest to the project's quality are as follows. With regards to stylistic conventions, the written report adheres to the APA guidelines, it is formatted with LaTeX showing a high degree of professionalism, its tone is well suited to the level of published research, and it contains no typos. Furthermore, the treatment of its content is both well organized and exhaustive, being modeled after a Master's thesis in the field of Computer Science. The methodology section, for example, equips an educated but non-technical reader with the knowledge necessary to grasp the challenges and innovations involved in this project.

However, to the extent that the written paper is on par with published research in the field, it is the codebase that pushes the standard of quality and reproducibility much beyond current practice. The GitHub repository provides information about the coding environment, along with associated packages and their respective versions, to reproduce each part of this Capstone project. It also presents a copy of the unmaintained repositories of Davidson et al. (2017) and Badjatiya et al. (2017) with all outstanding problems fixed and a summary of replication failures and successes

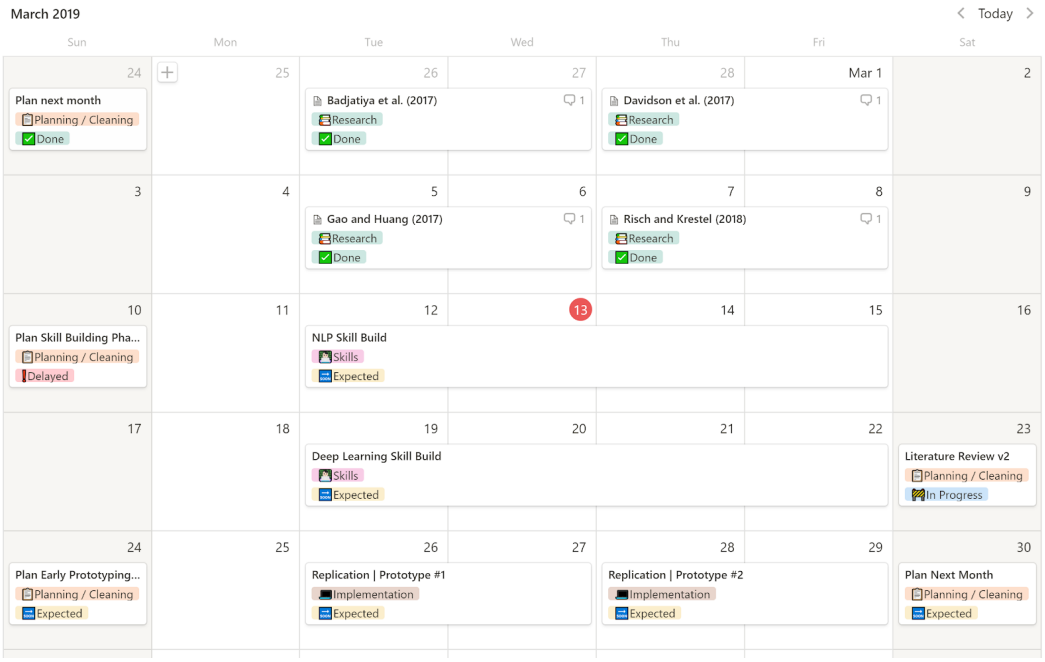


Figure 14: Planning architecture during the research phase.

to aid future researchers. Finally, I display all results in Jupyter notebooks that show the process I followed step by step. In summary, this Capstone project adheres to the highest standards of reproducibility, much beyond current practice in the field.

#planningarchitecture. The architecture of planning methods has evolved along the course of this project. At a broad level, we can discern research and implementation phases. The critical challenge of the research phase is to prevent information overload due to the exposure to and consumption of a wide array of research papers. The primary strategy was to establish a very well-organized *Notion*[11] page, previously described in Milestones 7 & 9 in the following manner:

> *I organize the management of this project around monthly sprints. A sprint symbolizes a self-contained stage of project development which encompasses a series of outcomes. I develop each sprint in detail in the last Sunday of the preceding month, when specific tasks and deadlines are assigned. The current one (March), which focuses on skill building and replications, is shown in Figure 4 [A.1].* (dos Santos, V. M. S., 2019).

The implementation phase involved bridging skill gaps, writing, and coding. During the summer, I planned and organized two Capstone Hackathons, a full day to work on Capstone with other Minervans to enhance productivity and accountability (see #accountability) and completed Udacity's Deep Learning Nanodegree (see #research). The planning architecture changed in Senior year since the main challenge was no longer managing uncertainty and information overload but instead maintaining consistency and productivity. The structure was, in a way, more straightforward, as I capitalized on the increasing expectations of our assignments and my established

[11] See here.

commitments as a driving force of progress. On a week-to-week level, I scheduled two dedicated time slots to work on Capstone and stuck to them religiously, which is evidenced by the consistent level of output I demonstrated throughout, in both writing and coding. Barred the one significant change of scope in Fall 2019, I delivered on the timelines I proposed, such as the one in *Capstone project update & Plans [Fall 2019]* (see *Individualized deliverable [Fall 2019]*) and the one in *Completed work & Plans [Fall 2019]* (see *Progress & Plans [Spring 2020]*).

The consistency of progress, the sensitivity to project phases and requirements (e.g., reading vs. writing), the creativity and effectiveness in the establishment of structures, both for information management (i.e., emphNotion page) and accountability (i.e., Hackathons) justify a high level of mastery of this learning outcome.

#feedback. I have demonstrated diligence in appraising and integrating both granular feedback on clarity and stylistic choices and strategic feedback concerning the most fruitful directions of this Capstone project. About clarity and style, I consistently addressed all feedback provided in a given assignment by the time the next one was due. For example, the literature review section on *Interpretability* went through three iterations until reaching its current stage, thanks to the helpful feedback provided by Prof. Wilkins.

On the level of overall strategy and direction, I have shown competency in both seeking and responding to feedback. I proactively reached out to Oren to have him as a peer reviewer for the *Peer feedback (Fall 2019)* assignment since I knew he was skilled in deep learning. That proved very fruitful since his feedback motivated a change in the scope of this project. I shifted away from an in-depth comparative analysis in favor of the application of transfer learning. I displayed a high degree of conscientiousness in the appraisal of that feedback and in responding effectively to the repercussions that such a change of scope would naturally cause (e.g., taking a course to learn about transfer learning, see #research). In summary, I have made extensive use of feedback throughout this project and have benefited from it extensively, due to my proactiveness in seeking it, my disposition to critically consider it, and my flexibility and diligence in responding to it.

#research. When I settled on this research topic back in Fall 2018, I had only completed two computational sciences courses at Minerva (CS110 and CS112). Furthermore, to this day, I have not taken any machine learning or deep learning courses at Minerva (such as CS156 or the deep learning tutorial). Hence, I have relied on my initiative, learning, and research efforts to consolidate my knowledge in this field and bridge skill gaps. This level of independence and self-determination is, by itself, an indicator of research competence since it implies the ability to digest and aggregate novel information, troubleshoot roadblocks effectively, identify and address technical gaps, and devise ways to safeguard consistency and motivation.

More practically, I have also demonstrated a high degree of research effectiveness and throughput. When necessary, I make use of primary sources, such as when considering the competing definitions of hate speech employed by government, companies, and academia. In most other cases, I survey a wide range of published research from both academic journals and conference proceedings to appraise the current trends and directions of the field of hate speech classification, culminating in the well-substantiated literature review I produced.

Concerning technical skills, I have actively pursued the knowledge necessary to produce a CS project of high methodological quality, and that is current with recent innovations. I have

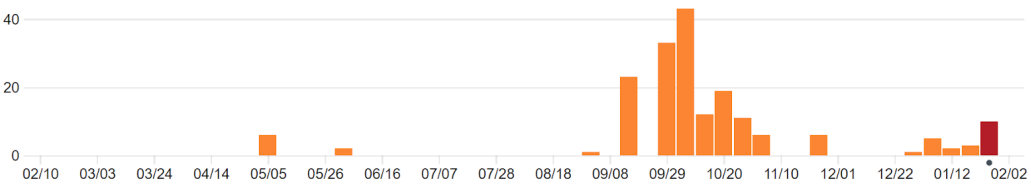


Figure 15: Commit frequency during the development of this Capstone project.

proactively completed Udacity's Deep Learning Nanodegree (with an estimated 4-month part-time commitment)[12] and fast.ai's NLP course (fastai, 2019), which allowed me to make adequate progress since I was equipped with the knowledge necessary to tackle my goals.

#connect. I exercised this learning outcome on three fronts. First, during my summer internship at Facebook, I connected with relevant data scientists and engineers working on hate speech classification to understand better the state of the art inside of the company. Specifically, I attended two "NLP Lunches"—events where someone would make a presentation about their work or research related to hate speech NLP during lunchtime—, met one-on-one with a software engineer working on hate speech, and followed their updates on the intranet. This level of exposure helped me to understand the divide between academia and industry better and partially informed my decision to focus on performance rather than more qualitative analyses.

Moreover, in performing the three replications, I also contacted authors in an attempt to elucidate their implementation or request data that was otherwise not publicly available. Finally, in the peer feedback assignment, I specifically sought out Oren as a reviewer, since I was aware of his experience working with deep learning and thought he would provide valuable insight. This strategy was very fruitful, and his feedback helped to point out transfer learning as a recent breakthrough worth investigating.

#metrics. My primary metric was whether I had dedicated at least 4 hours to Capstone-related work every week. I knew that I would inevitably work more in some weeks, but establishing a minimum level of dedication was important in guaranteeing I had momentum throughout. Depending on the tasks of a given week or project phase, I also developed subsidiary metrics. For example, during code-heavy weeks, I could use GitHub commits as a way to track my progress, and thus attempted to submit a commit as often as possible. My performance could easily be tracked by the platform's dashboard and provided a compelling psychological incentive towards consistency. The entire project amounted to 193 commits, whose frequency throughout the past year is shown in Figure 15.

We see that most of the codebase was developed in September and October when I was working on the replications. I should mention that January is under-counted here. Since I was working on the cloud due to GPU requirements, and since I did not connect that machine to GitHub, I was committing code less frequently than if I were working locally.

For writing, I established a tracking system where I could quickly check the percentage-wise completion of different sections of my Capstone and prioritize writing work accordingly. Figure

[12] See the Verified Certificate of Completion.

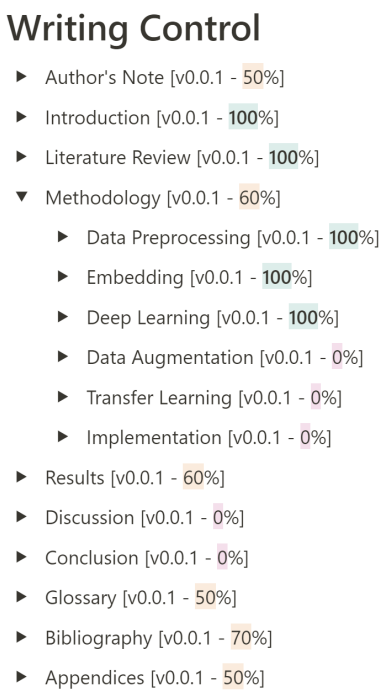


Figure 16: Writing progress tracking system.

16 shows the state of this writing control system in December 2019.

#accountability. I want to consider the amalgamation of the considerations above as my case for accountability. I have made consistent progress throughout the history of this project, never stalling, but in each assignment delivering substantial work and insight. That is due to my initiative in creating both planning and execution structures that safeguard the project as a constant priority in the sea of other obligations. Furthermore, I effectively took account of unknowns and skill gaps by proactively communicating with external experts and by allocating time to additional coursework that would equip me with the skills to complete the project. I responded effectively to constructive feedback, both big and small, by iterating both text and code multiple times.

Finally, I also worked with others to enhance both my motivation and progress. I did so most effectively during the summer when I organized study sessions for those interested in making progress during the school break. I called these "Capstone Hackathons" and proactively organized them by inviting other students on Facebook, coordinating a group chat, and creating a structure focused on providing motivation and support. **Appendix B** contains an example of the message I shared at the start of the group chat, setting the group up for the hackathon to happen on the coming Saturday. I organized two of these events during the summer, one on June 30 and another on July 21.

9.2 Project-Specific LOs

#CS156neuralnetworks. A large part of this project is dedicated to explaining, implementing, and training neural networks. Concerning their elucidation, I provide a wide-ranging methodolog-

ical treatment that discusses both the motivations and technical underpinnings of feedforward, convolutional, and recurrent neural networks. I also implement very recent technical innovations in the field, such as the AWS-LSTM architecture (Merity et al. 2017), NLP transfer learning via ULMFiT (Howard & Ruder, 2018), and data augmentation (Hemker, 2018). Some of these techniques were not available a mere two years ago, so the techniques I implement are very close to the current state of the art.

Let us consider the technical achievements of this project in greater depth. I demonstrate skill in correctly implementing neural networks both from scratch and by using the API of third-party libraries. To replicate Bajdatiya et al. (2017), I debugged and implemented their architectures in Keras with Tensorflow. To replicate Hemker (2018), conversely, I implement the networks in PyTorch from scratch while also designing test utilities to guarantee correctness. Finally, to produce my primary results, I leverage fast.ai's library and APIs, at times designing clever mechanisms to suit my requirements, such as implementing a function to split train and test sets that augments samples in the training set only (e.g., notebook #5).

Furthermore, the training regime I implement for my original work is also in line with very recent innovations. This regime includes data augmentation for text, discriminative learning rates, gradual unfreezing, and training bidirectional models. Moreover, although fast.ai's library streamlines many of these features, I implement Hemker's (2018) data augmentation successfully almost from scratch, requiring a complex interplay of data cleaning, manipulation of word embeddings, POS tag validation, and careful splitting for train and test sets.

#CS156overfitting. The most salient attempts to prevent overfitting are the following. First, in the replication of Hemker (2018), I implement the network's training regime from scratch (see *train_classifier()* in *classifier_utils.py*). An essential feature of this implementation is that it keeps track of the lowest validation loss during training. When the network reduces the validation loss, we save the model weights to disk. Hence, if the loss starts to increase due to overfitting, we can quickly recover the best performing model without rerunning the training procedure. On a similar note, when using fast.ai's library, I monitored performance closely and trained the model only for as long as validation loss was smaller or similar to the training loss (e.g., notebook #5).

The use of AWS-LSTM and its motivation have a higher degree of sophistication. Merity et al. (2017) describe at length the array of regularization (i.e., dropout preventing) strategies that are employed in this network architecture. They amalgamate several then-state-of-the-art techniques in their LSTM variant. These include DropConnect, a dropout mask on the recurrent connections between hidden layers, embedding dropout, and backpropagation through time (BPTT) sequences of varying length (Merity et al., 2017). I demonstrated competence and creativity in employing recent innovations to prevent overfitting in the original classifiers I implemented.

#VMnlp. This Capstone represents a mini-exposition of theoretical underpinnings and empirical applications of the last decade in NLP and, as such, represents a remarkable achievement. With the replication of Davidson et al. (2017) and the accompanying discussion of classical data preprocessing steps, I provide an overview of techniques that were and often remain very relevant to non-deep learning applications of NLP. I bridge this historical foundation with more recent innovations, discussing and leveraging pre-trained word embeddings, architectures employed by earlier researchers, and an original application of transfer learning to hate speech classification. I thus demonstrate both an attunement to the cutting edge for deep learning research for NLP and

an appreciation of the foundations of this research field. Furthermore, I showcase and complement these achievements with several notebooks I made available in this project's GitHub repository, which provide walkthroughs of increasing complexity over the content covered in this Capstone. Hopefully, this material will both aid further researchers in reproducing my results and serve as pedagogical resources for those seeking to learn more about NLP.

#CS156modelmetrics. I follow the conventions of published research on NLP for my choices of metrics to report. They include a confusion matrix of model predictions, recall of the class of primary interest (i.e., hate speech), total accuracy, and the class-weighted F1 score, which all provide a different perspective on the effectiveness of a classifier. For the replications, I focus on comparing how well I was able to approximate the performance given by the metrics the authors reported. For the original classifiers I implement, I provide all the four metrics I mentioned above. Furthermore, I provide insightful analysis of the shortcomings of total accuracy and F1 metrics, since they are unable to communicate the incredible differences in performance among the four classifiers shown in Table 5. The analysis demonstrates an appreciation of the importance of choosing appropriate reporting metrics to convey the characteristics of a classifier fully.

#CS156classification. Similar to #VMnlp, this learning outcome also applies to the majority of this Capstone project. I will focus on highlighting different accomplishments than those described earlier, recognizing that all are similarly relevant. As a starting point, I describe in great depth the challenges of hate speech as a classification task in my literature review, addressing quantitative and qualitative methods of analysis, classical and cutting edge approaches, and shortcomings in this field of research. Furthermore, the extensive and non-trivial replications I perform collectively display a combination of methodological flexibility (given the vast array of techniques required) and deep expertise, demonstrated by the quality of implementation and accompanying analysis. Specifically, I employ both classical statistical approaches to NLP (such as TF-IDF weighting, denoising, tokenizing, lexical analysis) and more recent deep learning applications, including CNNs, LSTMs, their combinations and variants, transfer learning, and data augmentation.

To the best of my knowledge, this Capstone project represents the first application in academia of transfer learning to the realm of hate speech applications. Together with the implementation of Hemker's (2018) data augmentation technique, the technical achievements of this project are at the frontier of the field of NLP and, therefore, manifest deep expertise of classification techniques in this domain.

9.3 HCs — Product

#professionalism. I comply with the highest standards of professionalism. Following common standards, I provide a written report focused on a technical audience but nevertheless providing extensive background to enhance the accessibility of the paper. I strictly follow APA guidelines that, for example, require the listing of all authors' surnames for the first-time citation of papers with five authors or fewer. Furthermore, other features take the level of professionalism further. The glossary is a valuable addition that intends to help less technical readers with comprehension necessary technical terms. Finally, the accompanying GitHub repository is more easily digestible, organized, and replication-friendly than is standard in the field, fulfilling the vital role of modeling better practices for future research.

#organization. The organization of this paper follows the structure widely used in the field of computational sciences, where a technical and in-depth methodology section is preceded by a more accessible review of the literature and followed by a statement of the results achieved and their discussion. Hemker (2018) and Fortuna (2017) are two examples. Furthermore, I would also like to highlight the organization of the accompanying repository, which contains both a helpful README file to support replication attempts and a sensible internal structure. The repository contains easy-to-follow summary notebooks and folders containing information on the replications, coding environment metadata, and datasets used in this project. All of these features and their ease of access demonstrate deep carefulness towards organization at the service of future researchers.

#evidencebased. The primary application of evidence-based reasoning comes in my analysis section, where I distill the insights I gained in my replications efforts and synthesize my arguments over the opportunities for improvement in the literature. For example, I discuss how the lack of sufficient information about methodological choices can effectively hinder the reproducibility of previous research. As evidence, I mention the difficulties in performing the replications, ranging from Davidson et al. (2017) ambiguity over whether they built a character- or word-based model to Hemker's (2018) omission is providing information over several hyperparameters. Other examples are present to a lesser degree throughout the text, such as via my justification over the directions chosen for this research, where I leaned on building a high-quality implementation of more recent methodologies rather than running a qualitative analysis over previous work.

#sourcequality. As a relatively new field of research, the literature on hate speech classification has suffered from relatively uncoordinated and ad hoc analysis. Even so, I focused on drawing from the most prominent papers in the field, while also performing replication of seminal results. My descriptions of and motivations for methodological choices are, notwithstanding, grounded on reputable and widely acclaimed sources. To focus on a few examples, Rajaraman and Ullman (2011), Goodfellow et al. (2016), and Howard and Ruder (2018) are established references in the research of natural language processing and deep learning.

Concerning hate speech specifically, I took this HC further. I turned a critical eye on previous research, focusing on appraising, summarizing, and, in the case of three papers, reproducing its achievements. I have shown that even though both Davidson et al. (2017) and Badjatiya et al. (2017) can be seen as seminal papers in the literature with 411 and 242 citations, respectively, as of early 2020, both present an unacceptably high barrier to replication, with the latter containing unfortunate methodological errors. My dedication to the in-depth investigation of the results on which the literature stands and my contribution to a critical evaluation of their reproducibility represents a sophisticated application of this HC.

#responsibility. See #accountability. Generally, I have consistently stuck to proposed timelines while effectively responding to feedback, learnings, and new information. I succeeded in keeping myself accountable via dedicated days when I would work exclusively on Capstone. I also kept a clear list of immediate tasks that helped me prioritize my time. Finally, I tried to make consistent progress every day. One can see in Figure 15 that for the two weeks when I was working on the two replications that I mentioned above (September 29 to October 12), I submitted code on eleven out of fourteen days.

#algorithms. My technical implementation follows very high standards of coding quality and methodological correctness. Furthermore, to make this Capstone reproducible and comprehensible to future researchers, I have heavily modularized my code while also providing walkthrough notebooks to exemplify how I use the modules I have created. Take, for example, the Data Preprocessing and Training and Testing Classifier notebooks. Both are very short since they draw heavily from separate files. These files are utils.py and classifier_utils.py, which contain helper functions for data preprocessing and classifier training, respectively. Furthermore, I also devised unit tests for both of these files (test.py and classifier_test.py) so that I can immediately test whether changes in the implementation files unintentionally cause any core functionality to break. The use of unit tests is an underutilized best practice that I successfully implemented here.

Instances of successful implementation and debugging of advanced concepts abound. Take, for example, this commit where I change a single line in the code. The original code base, for an embedding model with random weights, was not reshuffling these weights across cross-validation folds. Hence, after the first fold, the embeddings were being trained on the entire dataset, instead of only the training set. This implementation issue can be solved in one single line, but recognizing this issue required a deep level of engagement with the codebase. Finally, I also exercised the best practice of small and frequent code committing throughout development. This Capstone project has accumulated 190+ commits on GitHub.

#studyreplication. I have applied this HC in three different ways: (1) Improving and redressing mistakes in the code base of the authors I replicate, (2) identifying irreplicable results and mistaken methodology in the literature, and (3) rendering my code base easily reproducible. For (1) and (2), take the work I did in replicating Badjatiya et al. (2017). First, I fixed two significant issues in their codebase. In one, the authors incorrectly removed all whitespace from every sentence in the data (see this commit). In the other, they did not restart the network weights when performing cross-validation (see this commit). In the other, they did not restart the network weights when performing cross-validation (see commit), thereby exposing the test sets of later folds to the network during training. Also, I raise these issues and link to my solutions in the original repository (see issue). Finally, I provide a summary of my replication in a notebook so that the results are easily retrievable and verifiable by others (see notebook). Similar examples can be brought forward concerning the replications of Davidson et al. (2017) and Hemker (2018). Generally, I display a high degree of conscientiousness in performing and rendering the results of my efforts available to other researchers.

For (3), see #algorithms. I compartmentalize my code base so that it can be easily tested, I provide comments on my files so that others can easily understand them, and I create notebooks which exemplify the utilization of these files while demonstrating the results I obtain.

#sampling. The data augmentation procedure motivates the application of this HC. Let us characterize the data generating procedure behind the tweet data. We model the tweet as a sequence of N words. When writing the sentence, the user decides at each time t which word she should write, considering the suitability of a few candidate words conditioned on what she has already written. For example, "I'm going home to see my ..." is a sentence that could be completed with the tokens *mother, father, boyfriend*, among others. The writer resolves this conflict by further conditioning on the intention of the text, i.e., what she means. For lexical analysis and hate speech classification, these tokens lead to the same label of non-offensive speech.

Hence, data augmentation can be interpreted as a resampling procedure where for each token of a sentence w_{old}, we consider alternative tokens w_{new} that fulfill the same part of speech while largely preserving semantic meaning. The distribution of this resampling procedure follows $p(w_{new}|w_{old})$: $\mathbb{1}(w_{new}, w_{old}) \wedge f(w_{new}, w_{old}) > c$, where $\mathbb{1}()$ is an indicator function that checks whether the two tokens have the same part of speech, and f outputs the cosine similarity between the two tokens, required to be larger than the threshold c. In Hemker's (2018) implementation, we resample only tokens that are nouns, set c to 0.75, and remove stochastic behavior by always selecting the word with the largest cosine similarity. This resampling strategy is appropriate here since the class imbalance prevents the classifier from learning from the minority class.

#confidence. My goal for this HC was to strike the proper tone concerning my analysis of the faults of previous research. The challenge is to avoid a sanctimonious tone while objectively discussing the problems of papers I replicated. I feel my approach is successful. I altogether avoid any implication of ill-intent and ground my criticism on hard evidence coming from publicly available codebases or the papers themselves. At the same time, I discuss through my own experience how these faults impede the progress of the field by rendering much of the research irreproducible. Furthermore, although it is my intent to role model the standards of quality that I wish to be commonplace, I avoid a holier-than-thou tone in favor of objectively discussing the benefits of exhaustive methodological instructions, complete codebases, and careful methodological execution.

#modeling. The HC also permeates the whole assignment, since it addressed the question of how we can effectively model and predict the phenomenon of hate speech. The *Methodology* section describes several approaches in NLP that shed light on the research question of language modeling. I consistently provide the shortcomings of earlier methods and how the field progressively evolved to model increasingly nuanced aspects of the inherent complexity of text data. For example, take the discursive flow from BOW to TF-IDF weighting to word embeddings. I discuss how TF-IDF builds upon simple BOW methods by creating a more useful measure of word importance that helps the model find the signal in the data. Furthermore, I describe how word embeddings are a significant breakthrough due to their ability to capture semantic relationships that were previously impossible to model.

The application of this HC is taken further since the detailed methodological discussion, which extends into very recent innovations, heavily informs my strategy on how to model hate speech. I discuss how transfer learning aims to deal with the problem of small data by training the classifier on a large corpus before seeing the hate speech data so that it can learn the structure of the English language beforehand. Although these examples are salient, the employment of data augmentation, the discussion of methodological faults of replicated papers, and the analysis of the results I attain are all evidence of a robust application of this HC.

9.4 HCs — Process

#rightproblem. My understanding of the goal of this Capstone project has changed over time, and I would like to characterize it under the umbrella of this HC. In the beginning, I was concerned with building a typology of performance metrics for a hate speech classifier, encompassing not only

the general recall and F1 scores but also considerations over robustness, interpretability, and bias. This holistic focus is manifest in my literature review, composed early on, which included a wide variety of concerns and approaches to this field of research.

As I progressed through my replications, however, it became evident that the field stood on shaky grounds due to a lack of reproducibility. There was also a methodological gap between the recent exponential advances of the broader field of NLP and the relatively simple architectures employed in the hate speech classification literature. With that in mind, I realized that producing in-depth qualitative analysis of the classifiers of current literature would be redundant since new architectures and training regimes would soon outperform them. Hence, I shifted my focus towards employing recent innovations in NLP *while* critiquing the reproducibility of earlier results, redressing their codebases where I could, and guaranteeing that my own set of results would be easily replicable. Doing so required the proper characterization of the current state of research and industry, along with my considerations over where my efforts would be most helpful to the field. This exercise represents an advanced and practical application of this HC.

#biasmitigation. Emotional bias was very relevant to the development of this project. I recognized its importance during the process of replicating the results of other researchers since that proved to be very disappointing. I remember that the first time I read Hemker (2018), I felt that my Capstone was almost redundant because of the brilliance I saw in the paper. Upon trying to replicate it, however, I saw through the lack of detail, inconsistencies, and, ultimately, the errors it contained. Eventually, I realized I was feeling angry and frustrated and that I was losing motivation because I could not see the point of working in this field of research "where people were so careless." Fortunately, I noticed that an emotional bias was clouding my judgment. Anger was causing me to "Perceive negative events as predictable, under human control, and brought about by others" (Lerner, Li, Valdesolo, & Kassam, 2015).

Recognizing it was the first step in mitigating it. My two key strategies revolved around journaling and discussing the issue with others. I journal very often, and it helps me to identify the issues I'm concerned about, realize their emotional charge, and paint a bigger picture that usually alleviates the problem. In this case, the first symptom was a sudden lack of motivation that, after some resistance, I managed to attribute to this feeling of frustration. Then, upon further reflection, both through journaling and in discussion with my advisor and friends, I had several helpful realizations. For example, whereas it was regrettable that the research I attempted to replicate was fallible, the anger created an illusion of certainty where I was failing to consider that these mistakes are caused not by ill-intent but rather by competing priorities and benevolent neglect. Furthermore, I also failed to appreciate that the recognition of these mistakes did not entail that my research was not worth pursuing, but rather that I could focus on redressing the failures I had identified and contributing with my set results that adhered to reproducibility best practices.

#purpose. I decided to work on this project because it spoke to three goals that I wanted to achieve. These are social impact, professional development, and intellectual growth. I spent the summer of 2016 working with recently settled refugees in The Netherlands and have focused on social impact to different extents throughout my life. Due to the dire consequences of online hate speech, I felt that this project was heavily connected to my sense of purpose due to its potential impact. At the same time, the project represented an intellectual challenge that I welcomed.

When I chose this topic, I was still very early in my path as a computer scientist, so I expected to learn a lot through it. Finally, since I wanted and will work as a data scientist, this project also serves as an opportunity to grow as a professional and add a vital piece of work to my portfolio.

Hence, the recognition of the vision I had for my Capstone project and the assessment of how different projects pertained to those motivated my choice to work with hate speech classification. The high degree of commitment and alignment helped me throughout its execution by powering me through the trying times and fueling my momentum throughout the past year. The seeking and maintenance of such alignment is representative of an in-depth application of this HC.

#scienceoflearning. The science of learning was fundamental to the task of learning the skills necessary (i.e., deep learning and NLP) to implement this project successfully. The two techniques I most often employed were practice testing and distributed practice, identified by Dunlosky et al. (2013) as the most effective out of a pool of 10 techniques researched. Let us consider each in turn. Practice testing lies at the core of Udacity's methodology. The curriculum is divided into chapters that are further broken down into lessons. These lessons contain a series of videos and articles, which are interspaced with non-trivial coding exercises related to the topic at hand. So, for example, the Convolutional Neural Networks chapter contains a lesson on transfer learning, whose practice exercise is to fill the gaps in the code implementation of the algorithm. These exercises are effective ways of prompting deliberate and substantial practice that renders the content much more memorable. My choice of this paid Nanodegree, instead of other free alternatives, was heavily informed by my trust in its effective implementation of practice testing.

Whereas I could rely on Udacity to take care of practice testing, I was still responsible for organizing my schedule around the notion of distributed practice. Luckily, since I took this Nanodegree in August/September 2019, I was far from feeling pressed by a looming deadline. Furthermore, since I was on school break for most of this time, I focused on spending about three hours a day on the program for five days a week. The combination of high consistency with distributed effort helped me learn the content very effectively, to the extent that I did not experience a significant technical block while working on this project. Furthermore, I organized the project itself around a distributed timeline of increasing methodological difficulty (i.e., replications from simple to complex and then my contributions), further enhancing my learning throughout. The analysis and implementation of these two learning techniques show a profound grasp of this HC and its application.

#constraints. I applied this HC most prominently by carefully discriminating between obstacles and constraints throughout the execution of this project. The key examples concern my approach to knowledge gaps and data quality issues. I started this project while knowing relatively little about machine learning. However, I did not attempt to box my methodological choices within the field of what I then knew. Instead, I focused on understanding what approaches I could feasibly learn within the timeline of this project that would also be particularly relevant to the task. Hence, I treated my lack of expertise as an obstacle to be surmounted rather than a constraint and leveraged the project as an opportunity to learn more about an exciting field of research such as deep learning and NLP. With that being said, I did take my time availability as an essential constraint in this decision process.

Dataset quality represents an example where the opposite dynamic between obstacles and constraints plays out. Very early in my literature review, I took note of the many issues of

publicly available datasets. Upon thinking about the issue, however, I decided to treat it as a constraint rather than an obstacle. Producing a labeled dataset of superior quality than the ones available is an incredibly challenging task. It would require HRB review, significant time and financial investment, and data mining skills that went more closely resembled a stage of a Ph.D. thesis rather than a Capstone project.

Furthermore, true to the nature of this HC, the constraint opens up opportunities for creative problem-solving. Both ambiguity and small data characterize many real-world problems. Both transfer learning and data augmentation are strategies relevant to these problems since they aim to increase the level of signal extracted from the data. Hence, both conceptually and in practice, I exercised thoughtfulness over the characterization of constraints and obstacles.

10 Appendix B — Capstone Hackathon

Hello everyone! This is your gentle reminder to send your "sorry I can't make it" to all the invitations you got for Saturday plans. I'm including a checklist and a recap of the run of the show below so that y'all are prepared. See you then!

Checklist

1. There is nothing on your agenda (or, at least, during the time you want to commit).
2. You know the place(s) you are going and it's got food, wifi, nice couches, stuff like that.
3. Prior to or first thing on that day, you've got a checklist of what you wish to accomplish. People will differ on how they'd like to go about this, but I'd rather use this day to power-through blocks and shit I really don't wanna do. A model is to have a list that you have 50/50 chances of finishing and then pretending it's a final project deadline and Reed will come at you if you don't "submit." The rationale is that it's easier to improve something later than getting it started in the first place.
4. Buy some nice wine, if you actually do it you deserve a pat on the back.

Run of the Show

1. Send your task list in this chat on Saturday morning (suggested time is 9 AM local). Treat it like those final project completion posts at the end of the semester where you update your percentages whenever you feel like it.
2. My guess is most people will want to mute the chat considering how our numbers, but try to take the time to read others' stuff when you feel like it. It's the place to ask and offer help!
3. We will have two quick check-in sessions for those who want to see other people "face to face" and share their goals and progress.
   (a) First at 10 AM PDT, 1 PM EDT, 7 PM CEST.
   (b) Second at 4 PM PDT, 7 PM EDT, 1 AM CEST.

(c) I am in PDT so it's hard for me to do anything earlier. If folks feel like having an earlier session, do it!

Whenever you decide to call it a day, take the take to share how your day was and how much you got done. It can serve as inspiration and marketing material for the next two hackathons :)

PS:

- If you change your mind about participating, feel free to leave the group!
- If I have forgotten anyone, please add them.